EKSKLUSIF

RESTful API

Konsep dan Implementasi

Pada Codeigniter 4

Membangun RESTful API

Admin Template Professional

Server dan Client

Agus Prawoto Hadi

Jagowebdev

RESTFul API
Konsep dan Implementasi
Pada Codeigniter 4

Penulis : Agus Prawoto Hadi

Edisi : I (Pertama)

Terbit : Juli 2023

Dimensi : 15,7 cm x 23 cm

Jumlah Halaman: xii + 316

Kata Pengantar

Alhamdulillah, penulis panjatkan kehadirat Allah S.W.T, karena dengan rahmat dan hidayah-Nya penulis dapat menyelesaikan buku "REST*Ful* API Konsep dan Implementasi Pada Codeigniter 4". Buku ini merupakan buah karya dari pengetahuan dan pengalaman kami dalam mengembangkan aplikasi RES*Ful* API.

Didunia serba teknologi saat ini, pengembangan aplikasi lintas platform sudah menjadi suatu keniscayaan, sehingga kebutuhan akan pengetahuan dan kemampuan pengembangan aplikasi RESTFul Api sudah menjadi suatu keharusan.

Buku ini akan memberi Anda pemahaman yang mendalam dan utuh tentang RESTFul API, tidak hanya membahas tentang konsep dan teori, tetapi buku ini juga mengajarkan bagaimana mengimplementasikan konsep tersebut dalam aplikasi nyata baik dari sisi client maupun server.

Penulis buku ini sudah sangat matang dan berpengalaan di dunia web development sehingga teori yang dibahas di buku ini merupakan teori yang sudah valid dan jamak di gunakan, implementasi coding yang di ajarkan juga senantiasa menerapkan best practice di dunia pemrograman.

Penulis menyadari bahwa buku ini masih jauh dari sempurna, untuk itu penulis menerima segala kritik dan saran yang membangun untuk perbaikan di versi berikutnya. Kritik dan saran dapat dikirim ke alamat email penulis di prawoto.hadi@gmail.com

Akhir kata semoga buku ini dapat membawa manfaat bagi pembaca.

Semarang, Juli 2023

Penulis

Source Code

Buku ini merupakan paket dari produk RESTFul Api Codeigniter 4. Dalam paket tersebut terdapat source code REST*Ful* API Admin Template Codeigniter 4 dan Aplikasi Client Admin Template Codeigniter 4. Keduanya dapat di download di halaman member Jagowebdev.com.

Kedua aplikasi tersebut telah dilakukan uji coba dan berjalan dengan baik pada lingkungan pengembangan PHP versi 8.2, database MariaDB versi 10.4, dan Apache 2.4. Pengujian dilakukan baik offline maupun online, dengan origin/domain yang berbeda.

Pada pengujian offline, kami menggunakan XAMPP 8.2 sedangkan pada pengujian online kami menggunakan shared hosting. Jika Anda menggunakan lingkungan pengembangan yang berbeda, misal menggunakan server Nginx atau menggunakan database MySQL mungkin aplikasi tidak dapat berjalan sebagaimana mestinya.

Daftar Isi

BAB 1 Install Aplikasi Server dan Client

Untuk mempermudah pembelajaran pada buku ini, terlebih dahulu perlu kita install Aplikasi API Server dan Aplikasi API Client yang disertakan pada paket buku ini.

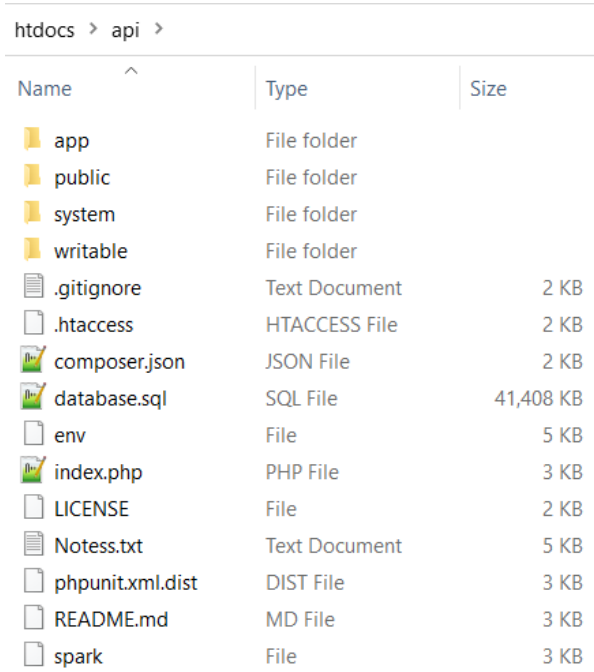
1.1. Install Aplikasi API Server

Aplikasi API Server merupakan aplikasi yang bertugas menyediakan data bagi client. Aplikasi API Server ini dibangun menggunakan Codeigniter 4 versi 4.3.5. Untuk cara instalasinya silakan ikuti langkah berikut:

a. Install PHP dan MySQL/MariaDB. Pada pengembangan yang kami lakukan kami menggunakan XAMPP 8.2.4 portable dimana didalamnya menggunakan PHP versi 8.2.4 dan database MariaDB versi 10.4, Anda dapat menggunakan database MySQL namun dengan beberapa penyesuaian. Untuk mengunduh XAMPP, dapat membuka tautan berikut:

   https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/

b. Selanjutnya ekstrak file api_server.zip dan copy file hasil ekstrak tersebut ke dalam folder htdocs, misal jika kita copy api folder htdocs/api maka struktur foldernya akan tampak seperti berikut:



c. Selanjutnya buat database dengan nama yang dikehendaki, misal api, kemudian load file database.sql yang disertakan pada file download ke database tersebut. Untuk keperluan ini bisa menggunakan database manager HeidiSQL

d. Edit file app/Config/App.php, pada bagian $baseURL isikan dengan url dimana aplikasi diinstall, jika kita ekstrak file ke folder htdocs/api seperti contoh diatas, maka $baseURL nya adalah http://localhost/api/ selanjutnya edit file app/Config/Database.php sesuai dengan konfigurasi database Anda. Untuk variabel lainnya bersifat opsional.

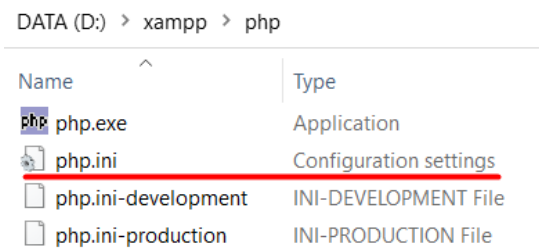
```
public $baseURL = 'http://localhost/api/';
public $jwtKeyAccessToken =
'652b0afc767d33dbe97f3677b70a40901f403f8867fe70b3a8dacc5f627
7e396';
```

```
public $jwtKeyRefreshToken =
'a49a2ab5935c1a084a799b4eca36b7be401b49ae0abfaeda6a32c4e3527f42a
a';
```

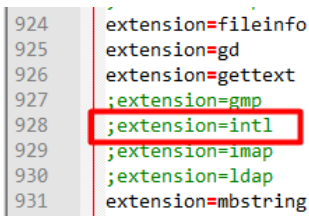
Note: base url harus diakhiri tanda slash (/)

Untuk property `$jwtKeyAccessToken` dan `$jwtKeyRefreshToken` bisa dibiarkan apa adanya atau bisa diganti dengan key Anda sendiri. Cara membuat key ini akan dibahas di bab berikutnya.

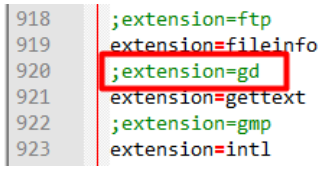
e. Codeigniter mensyaratkan beberapa library PHP diaktifkan, diantaranya library intl. Secara default **library ini tidak aktif** untuk mengaktifkannya buka file php/php.ini (dengan notepad/notepad++) bila ekstensi .ini tidak muncul, cari nama file php dengan type Configuration settings



kemudian cari baris ;extension=intl kemudian hilangkan titik koma yang ada didepan, misal sebagai berikut:



Selain intl agar aplikasi dapat berjalan dengan baik, library gd juga perlu diaktifkan, caranya sama seperti mengaktifkan intl yaitu hilangkan tanda titik koma (;) pada konfigurasi ekstensi gd (;extension=gd)



setelah selesai, simpan file dan **Restart Apache**

1.2. Install Aplikasi API Client

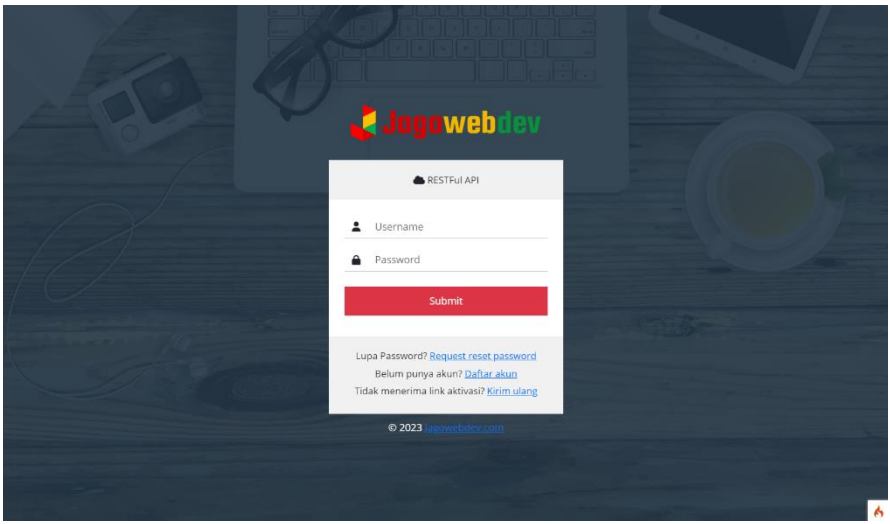
Aplikasi RESTFul API Server hanya bertugas mengelola data, untuk menampilkan data, perlu aplikasi tersendiri yaitu aplikasi client. Aplikasi client yang kami sertakan juga dibangun menggunakan Codeigniter 4 versi 4.3.5., nantinya aplikasi ini mengambil data dari apliaksi api yang telah kita install sebelumnya.

Untuk aplikasi client, cara installnya sama dengan aplikasi api, ekstrak file php ke dalam folder htdocs, misal htdocs/api-client kemudian sesuaikan property $baseURL dan $apiURL url yang ada di file app\Config\App.php. Pada aplikasi client ini kita tidak melakukan konfigurasi database karena data kita ambil dari aplikasi API. Berikut contoh configurasi nya:

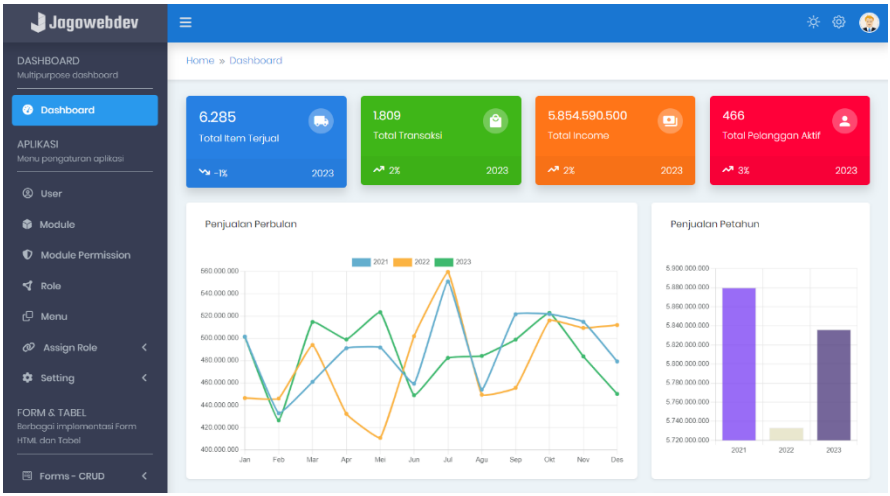
```
public $baseURL = 'http://localhost/api-client/';
public $apiURL   = 'http://localhost/api/';

// File Picker
public $filePickerServerURL = 'http://localhost/api/filepicker/';
public $filePickerIconURL    =
'http://localhost/api/public/images/filepicker_images/';
public $filePickerUploadURL =
'http://localhost/api/public/files/uploads/';
```

Pada aplikasi client ini, karena kita tidak mengambil data dari database maka folder app\Models tidak ada isinya/kosong.

Jika berhasil maka tampilan aplikasi client akan tampak seperti gambar berikut ini:



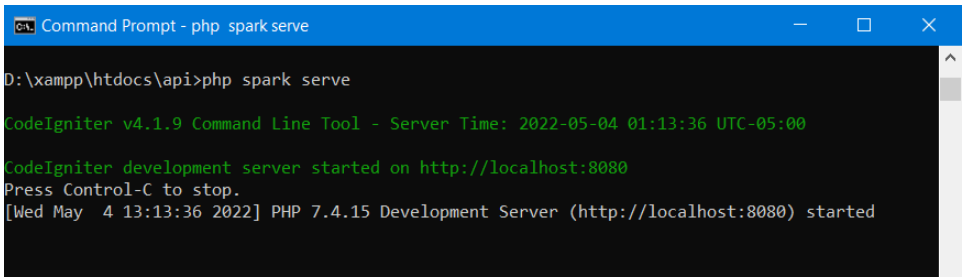
Untuk login dengan role admin gunakan username admin dan password admin, untuk role user gunakan username user password user, untuk login dengan role admin tampilan awalnya adalah sebagai berikut:



1.3. PHP Spark Serve (Not Recommended)

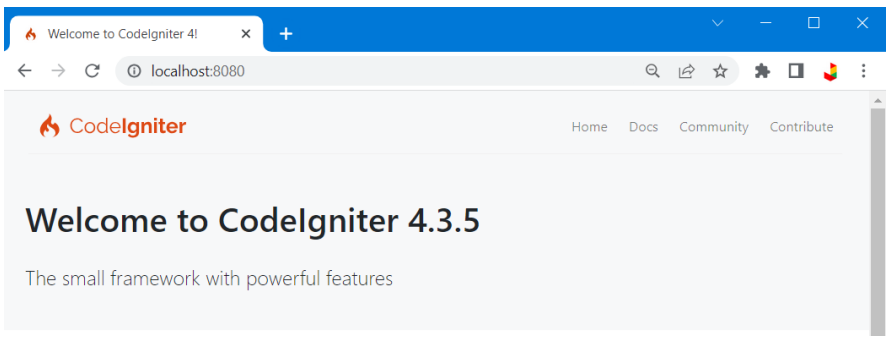
Dalam paket instalasinya Codeigniter menyediakan server lokal (local development server), dengan menggunakan server lokal ini, maka kita tidak perlu menggunakan Apache/Nginx atau server lainnya, namun demikian jika kita menggunakan database, kita tetap perlu menjalankan server database secara terpisah.

Untuk menjalankan server lokal pada Codeigniter, kita gunakan perintah php spark serve sebagai berikut:



Note: Untuk menghentikan server tekan Ctrl + C

Selanjutnya kita dapat mengakses halaman codeigniter dengan alamat http://localhost:8080, contoh tampilannya adalah sebagai berikut:



Penggunaan server spark ini terlihat simpel dan mudah, namun demikian menurut penuis terdapat kekurangan yang mendasar yaitu ketika menulis alamat resource static, sebagai contoh ketika kita ingin meload file site.css yang ada di folder public, maka alamat file tersebut adalah http://localhost:8080/site.css

Misal jika pada header kita tulis:

```
<html>
<head>
      <link rel="stylesheet" href="<?=$config->baseURL .
'css/site.css'?>"/>
</head>
```

Maka ketika dijalankan pada browser, akan berubah menjadi:

```
<<html>
<head>
      <link rel="stylesheet" href="http://localhost:8080/site.css"/>
</head>
```

selanjutnya ketika kita berpindah ke production server atau server hosting online dengan domain misal jagowebdev.com dan baseURL berubah menjadi https://jagowebdev.com maka ketika dijalankan pada browser alamat resource berubah menjadi:

```
<html>
<head>
      <link rel="stylesheet" href="https://jagowebdev.com/site.css"/>
</head>
```

Hal ini mengakibatkan error bahwa file site.css tidak ditemukan.

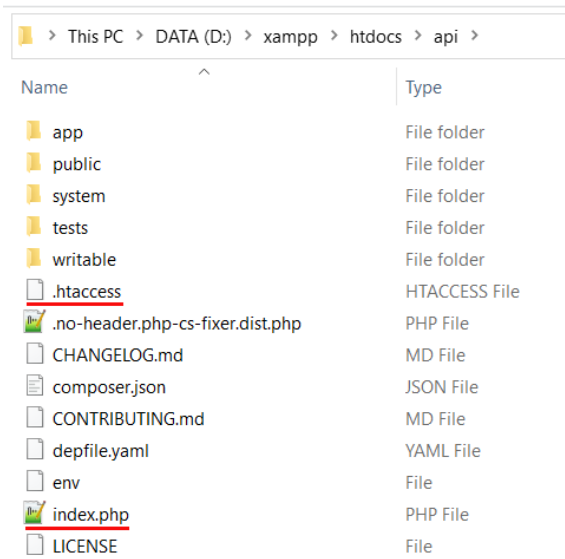
Kenapa demikian?

Hal ini disebabkan karena alamat localhost:8080 merujuk ke folder htdocs/public, sedangkan pada production server, https://jagowebdev.com merujuk ke folder htdocs bukan ke folder htdocs/public sehingga ketika baseURL nya berubah menjadi https://jagowebdev.com maka alamat css nya berubah menjadi https://jagowebdev.com/site.css, sehingga mengakibatkan muncul pesan error file tidak ditemukan, untuk itu, pada kesempatan kali ini kita tidak menggunakan server lokal bawaan Codeigniter melainkan menggunakan production server yaitu Apache.

1.4 Public Pada URL

Pada production server, untuk mengakses controller pada Codeigniter, kita harus menyertakan path public, misal http://localhost/api/public/user, hal ini tentu tidak efisien karena path public ini tidak berarti apapun, untuk itu kita perlu menghilangkan path public dari URL, caranya adalah sebagai berikut:

1. Copy file .htaccess dan index.php yang ada pada folder public ke folder api sebagai berikut:



2. Selanjutnya buka file index.php ubah $pathConfig dari:

```
$pathsConfig = FCPATH . '../app/Config/Paths.php';
```

menjadi:

```
$pathsConfig = FCPATH . 'app/Config/Paths.php';
```

BAB 2 REST dan RESTFul API

Ketika mengembangkan aplikasi API maka pasti kita akan bertemu dengan istilah REST, API, dan RESTFul API. Pada bab ini, kita akan membahas lebih jauh tentang ketiga istilah tersebut beserta hal hal terkait yang perlu diketahui.

2.1. REST, API, dan RESTFuAPI

2.1.1. REST

REST (Respresentational State Transfer) adalah sebuah desain arsitektur yang menerapkan prinsip prinsip/batasan batasan yang digunakan untuk mendesain sistem yang terdistribusi (distributed system).

REST ini diperkenalkan pertama kali oleh Roy Fielding dalam disertasinya yang berjudul "Architectural Styles and the Design of Network-based Software Architecture.". Arsitektur REST ini diperkenalkan untuk menjawab krisis scaleablity aplikasi/website (kemampuan aplikasi/web menangani permintaan dalam jumlah besar) yang terjadi di tahun 2000-an, dengan konsep REST ini, sebuah WEB dapat di scale lebih luas.

Meski begitu populer, REST tidak pernah ditetapkan sebagai standar oleh oraganisasi resmi, sehingga tidak ada implementasi yang baku, seperti tidak ada standar format data yang digunakan, tidak ada standar struktur data response, tidak ada standar penyusunan URI untuk identifier resource, tidak ada standar penamaan resource (plural/singular) dll, meski demikian REST dapat dikaitkan dengan berbagai standar lain sehingga REST dapat diimplementasikan dengan baik, seperti standar HTTP, standar versioning, standar format data, dll.

Pada disertasi fielding disebutkan bahwa arsitektur REST di implementasikan pada protokol HTTP, ya karena REST ini terinspirasi dari WWW (world wide web) yang menggunakan protokol HTTP, namun

demikian seperti yang telah disebutkan sebelumnya bahwa tidak ada standar formal untuk arsitektur REST, sehingga tidak ada standar baku protokol apa yang seharusnya digunakan, sehingga arsitektur REST ini bisa diterapkan di protokol apapun.

2.1.2. API

API (Application Programming Interface) merupakan istilah umum dalam dunia pemrograman, API bisa apa saja tidak hanya berhubungan dengan website dan jaringan, sebagai contoh, Anda mungkin tidak asing dengan jQuery, sebuah Javascript library untuk mempermudah pekerjaan ketika bekerja dengan javascript. jQuery menyediakan berbagai "API", missl add() untuk menambah element HTML, ajax() untuk mengirim request ke server, dll, dengan API ini, kita tidak perlu rumit menulis perintah di javascript.

Contoh lainnya adalah PHP, PHP menyediakan berbagai "API" baik berupa class maupun fungsi untuk mengakses data pada database seperti fungsi mysqli_query, atau class PDO, lebih lanjut dapat mengunjungi halaman PHP: Choosing an API - Manual yang dapat diakses melalui tautan https://www.php.net/manual/en/mysqlinfo.api.choosing.php

Selain builtin API (api bawaan) kita juga dapat membuat API sendiri yang dapat kita gunakan untuk melakukan pekerjaan tertentu, misal api untuk mendapatkan nama bulan sebagai berikut:

```
function nama_bulan($bulan) {

      if (!is_numeric($bulan))
            return false;

      if ($bulan < 1 && $bulan > 12)
            return false;

      $list_bulan = [1 => 'Januari', 'Februari', 'Maret',
'April', 'Mei', 'Juni', 'Juli', 'Agustus', 'September',
'Oktober', 'November', 'Desember'];

      return $list_bulan($bulan);
}

echo nama_bulan(5); // Menghasilkan Mei
```

nantinya untuk mendapatkan nama bulan kita cukup memanggil fungsi nama_bulan() tanpa perlu tahu kerumitan bagaimana coding untuk mendapatkan nama bulan.

Selain bentuk diatas, API juga dapat berbentuk alamat URI, bentuk inilah yang paling sering kita jumpai, sebagai contoh pada Github API, dengan mengakses alamat URI https://api.github.com/users kita mendapatkan resource daftar user Github.

2.1.3. REST API dan RESTFul API

Berdasarkan uraian diatas, maka dapt disimpulkan bahwa REST API adalah API yang menerapkan arsitektur REST dan RESTFul API merupakan istilah bagi aplikasi yang telah menerapkan arsitektur REST

Sebagaimana disebutkan sebelumnya bahwa REST utamanya menggunakan protokol HTTP sehingga pada REST API yang digunakan adalah protokol HTTP.

2.2. Istilah Penting

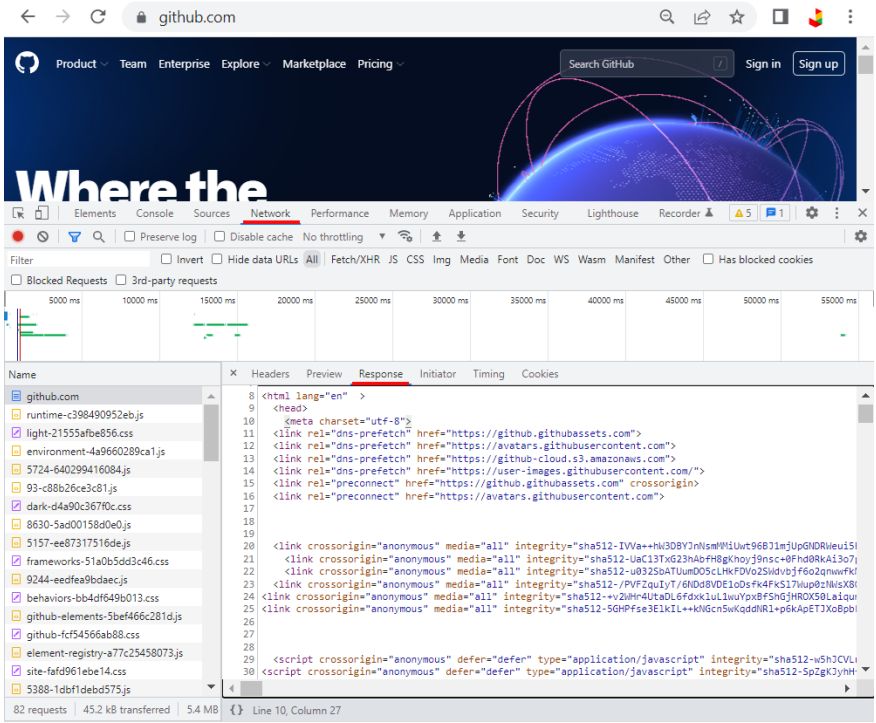
Sebelum membahas lebih jauh mengenai REST API, terdapat beberapa istilan yang perlu diketahui terkait REST

Resource dan Representasi (Representation)

Dalam arsitektur REST, komunikasi antara client dan server terkadang menyertakan data, data yang disertakan ini direpresentasikan dengan bentuk tertentu seperti JSON, XML, HTML dan lainnya, berikut contoh representasi dari resource dengan identifier https://api.github,com

Pada contoh diatas data direpresentasikan dalam format JSON.

Contoh lain adalah ketika kita membuka sebuah Website, misal: https://github.com, ketika kita mengakses url https://github.com maka browser mengirim request ke server untuk mendapatkan resource (data), server memproses request tersebut dan mengambil resource (data, database, file, dll) yang sesuai, kemudian mengolah resource untuk selanjutnya dikirim ke browser, pada contoh kali ini respon yang dikirim server berupa dokumen HTML seperti tampak pada tab Response pada Developer Tools browser



Oleh browser selanjutnya dokumen HTML tersebut di proses sehingga muncul halaman depan website.

Dari uraian diatas dapat disimpulkan bahwa resource adalah entitas/data berupa apapun yang disediakan oleh server yang dapat diidentifikasi (dengan URL), sedangkan representasi dapat disimpulkan sebagai bentuk/format penyajian dari suatu resource (format HTML, XML, JSON, dll), adapun URL dapat disimpulkan sebagai identifier dari resource.

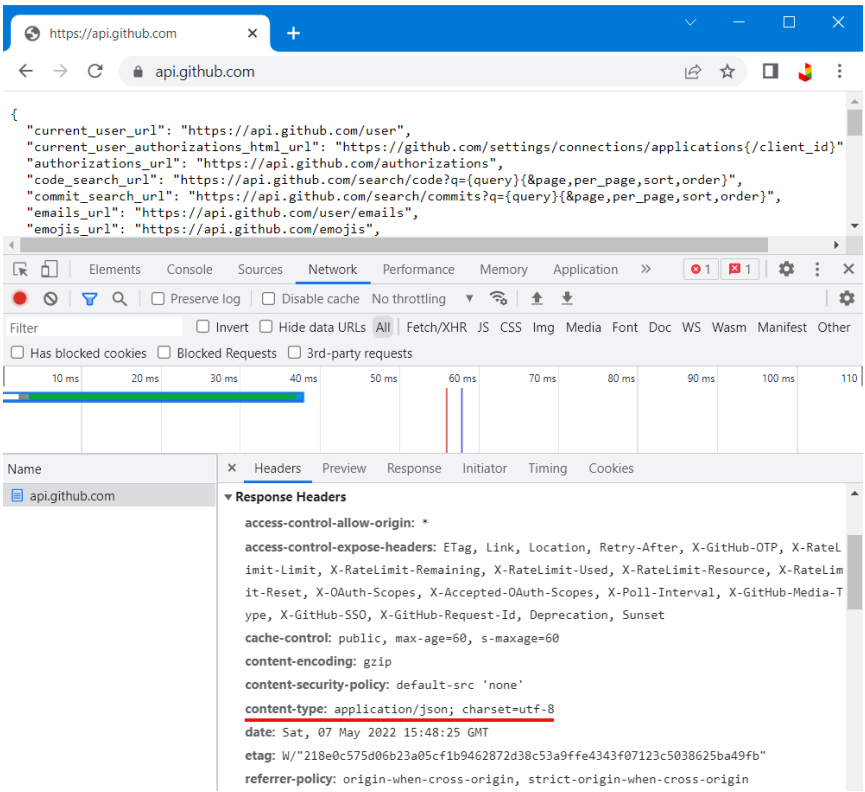
Identifier

Dalam termonologi REST, resource harus diberi identitas, identitas untuk resource ini disebut identifier, karena menggunakan protokol HTTP, maka bentuk identifier yang digunakan adalah URI, sebagi contoh resource user pada github diidentifikasikan dengan URI https://api.github.com/users.

Sebagaimana telah disebutkan sebelumnya bahwa tidak ada standar bagaimana mendesain REST termasuk identifier ini, REST hanya mensyaratkan bahwa identifier harus unik.

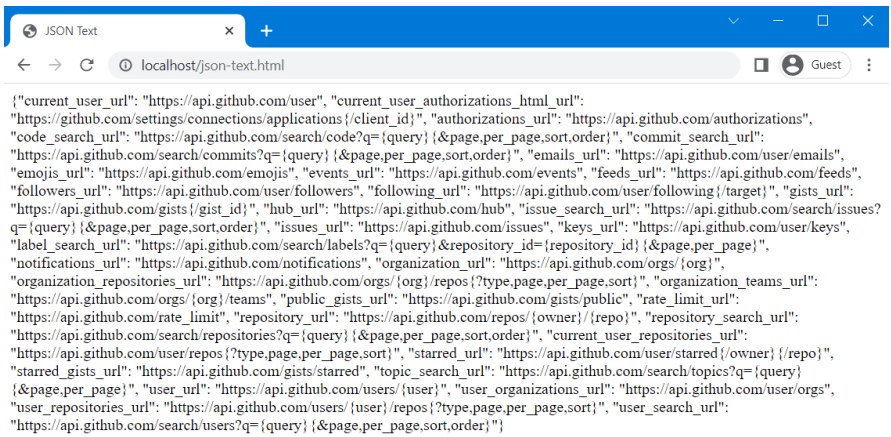
Media Type

Pada desertasi Fielding dijelaskan bahwa Media Type merupakan format data dari representasi, seperti JSON, XML, HTML, dll. Format data ini didefinisikan melalui header Content-Type, sebagai contoh ketika kita meng akses api github https://api.github.com melalui browser maka hasil yang kita peroleh adalah sebagai berikut:



Pada contoh diatas, dengan melihat apa yang ditampilkan oleh browser, kita tahu bahwa representasi data nya adalah JSON, namun demikian browser tidak bisa melihat seperti cara kita melihat, browser tahu bahwa

data direpresentasikan dalam bentuk JSON karena nilai Header Conten-Type nya adalah application/json, sehingga data yang ditampilkan ke kita berbentuk JSON yang sudah terformat, jika nilai Content-Type nya bukan application/json misal text maka browser akan menampilkan data apa adanya tidak terformat seperti JSON diatas, misal sebagai berikut:



Prinsip Adressability

Setiap URI mengidentifikasi hanya dan hanya satu resource. Jika sebuah website memiliki resource yang berbeda maka kita akan memiliki ekspektasi bahwa kita dapat mengakses resource tersebut dengan URI yang berbeda, misal resource yang memuat data semua user dan resource yang memuat data user tertentu harus memiliki identifier yang berbeda.

2.3. Karakteristik REST

Suatu aplikasi dapat dikatakan memenuhi REST jika aplikasi tersebut memenuhi kaidah kaidah yang telah disepakati, yaitu:

1. **Client Server Constrain**

Pada constrain (batasan) client server, diharuskan adanya pemisahan antara client dan server, client adalah aplikasi yang mengakses data/resource ke server, sedangkan server merupakan aplikasi yang

menyediakan resource, dengan pemisahan ini, maka aplikasi pada server dikembangkan secara independen tidak terikat pada client tertentu.

Sebagai contoh aplikasi Tokopedia dapat diakses dengan baik menggunakan browser maupun menggunakan aplikasi Android, jika aplikasi hanya dikembangkan untuk browser saja maka pengembang aplikasi mobil akan kesulitan mengelola data untuk ditampilkan ke user.

Bagaimana dengan Website? Seperti https://github.com? website hanya dikembangkan untuk keperluan ditampilkan pada client tertentu (browser) sehingga tidak memenuhi client server constrain.

Dengan adanya pemisahan ini maka perubahan pada aplikasi server tidak mempengaruhi aplikasi client, demikian juga perubahan pada aplikasi client tidak mempengaruhi aplikasi server, hal ini dapat meningkatkan skabilitas dan fleksibilitas aplikasi antar platform.

2. Stateless

Terjemahan bebas dari stateless adalah tanpa ikatan, artinya Aplikasi REST tidak mengingat/menyimpan kondisi (state) dari client, seperti apakah client berada pada posisi login atau tidak, pada prinsip stateless ini client lah yang harus menyimpan state, session, dan semua informasi yang dibutuhkan untuk mengakses resource pada server.

Selain tidak menyimpan kondisi client, server juga tidak menyimpan data request sebelumnya, session, maupun history, sehingga setiap request diperlakukan sebagai request baru yang independen/tersendiri, tidak berhubungan dengan request sebelumnya, sebagai contoh, pada request sebelumnya user telah berhasil login, kondisi login ini tidak diingat oleh aplikasi server, pada request berikutnya client dapat menyertakan Authorization header untuk memberitahu server bahwa user telah berhasil login. Authorization Header ini dibahas pada bab tersendiri.

Batasan/constrain stateless ini dapat meningkatkan visibility, reliaility, dan scaleability dari aplikasi api, namun demikian terdapat sisi negatif dari constarin ini, diantaranya dapat mempengaruhi network request, karena

data yang sama dikirim berulang kali (yaitu state client), seperti token yang selalu disertakan pada setiap request.

3. Cache

Constrain ini mengharuskan Web Server memberitahukan ke client apakah respon dapat di simpan di cache atau tidak, Dengan cache ini, maka dapat mengurangi waktu respon, trafik network, dan beban server karena client tidak perlu melakukan request ke server untuk data yang sama. Kekurangan constrain ini adalah adanya kemungkinan data yang ditampilkan oleh client tidak update/berbeda dengan yang ada di server.

4. Uniform Interface

Pada arsitektur REST aplikasi server dikembangkan tersendiri tanpa memperhatikan siapa pengguna resource tersebut, dengan model seperti ini, maka informasi/data dikirim dalam bentuk standar tidak spesifik ke format tertentu yang dibutuhkan aplikasi tertentu (seperti HTML untuk browser saja), dengan demikian arsitektur REST dikatakan memiliki interface yang Uniform (seragam).

2.4. Uniform Interface

Pada bagian sebelumnya telah dibahas sekilas mengenai Uniform Interface. Agar implementasi uniform interface ini dapat memenuhi kebutuhan client, maka REST menerapkan 4 batasan:

1. Identifikasi Resource (identification of resources)
2. Memanipulasi resource melalui representasi (manipulation of resources through representations)
3. Deskripsi pesan (Self descriptive messages), dan
4. hypermedia as the engine of application state.

Identifikasi Resource

Untuk menerapkan constrain uniform interface, yang pertama perlu diperhatikan adalah identifikasi resource. Berdasarkan desertasi Fielding,

semua yang memiliki nama dapat menjadi resource, sebuah dokumen, image, informasi cuaca (e.g. "cuaca hari ini di Los Angeles"), sekumpulan resource, non-virtual object (e.g. orang), dan lain lain.

Setiap resource harus diidentifikasi oleh identifier. Identifier ini harus permanen, tidak berubah ubah meskipun bentuk resource berubah. Jika resource pindah ke identifier lainnya, maka ketika ada request ke server maka server harus memberitahu bahwa identifier telah berubah.

Self-Descriptive Message

Self-Descriptive message ini maksudnya adalah representasi dari resource yang dikirim kepada client seharusnya disertai dengan informasi bagaimana memproses representasi tersebut.

Pada REST API, Content-Type header digunakan untuk menginformasikan baik pada server maupun client jenis media type yang ada pada HTTP body. Agar sesuai standar, sebaiknya penamaan media type ini memenuhi standar IANA https://www.iana.org/assignments/media-types/media-types.xhtml misal application/json., Media type non IANA juga dapat digunakan namun perlu di dokumentasikan dengan jelas agar dapat dipahami dengan baik oleh client.

Manipulation of resources through representations

Constrain ketiga ini memungkinkan client untuk memanipulasi resource melalui representasi yang dikirim ke server. Manipulasi ini dilakukan menggunakan HTTP Verb, yaitu POST, PUT/PATCH, dan DELETE

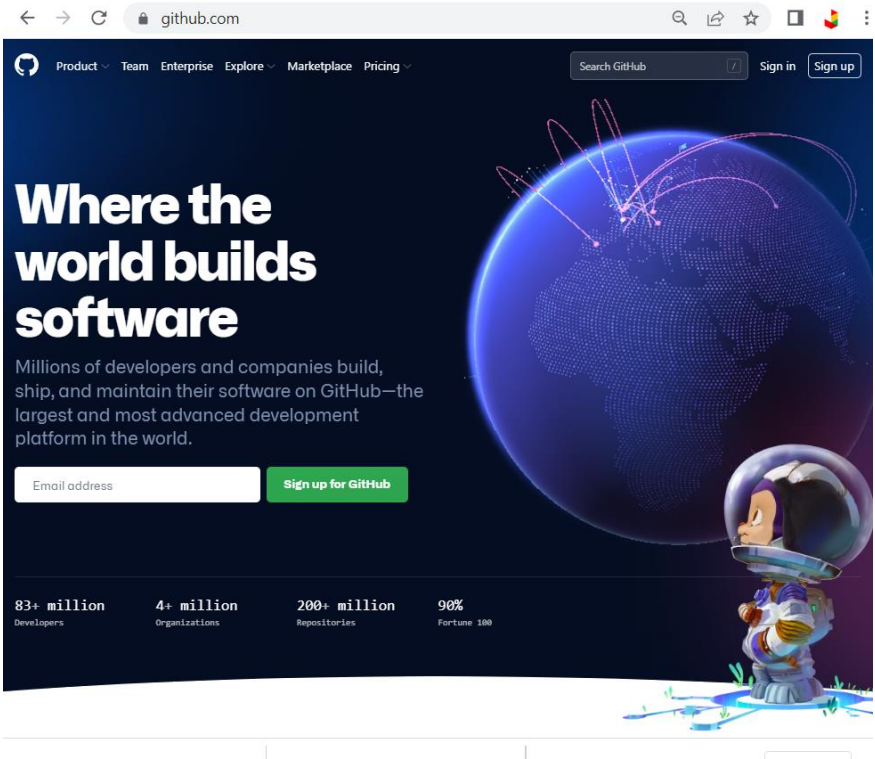
Hypermedia As The Engine Of Application State (HATEOAS)

Istilah diatas terlihat terlalu tinggi, mengawang awang, tapi sebenarnya simpel, hypermedia hanyalah text dengan hyperlink.

Sebuah halaman web merupakan application state (keadaan suatu aplikasi), dengan hypermedia, state tersebut dapat berpindah ke state yang lain, dengan kata lain dengan mengklik link, kita berpindah dari state (halaman) satu ke state yang lain, sehingga ketika kita menjelajah web kita

menggunakan Hypermedia sebagai alat/penggerak (As The Engine) untuk berpindah halaman web (Application State).

Contoh Hypermedia dalam praktik sehari hari adalah ketika kita membuka sebuah Website, misal https://github.com. Ketika browser mengirim



request ke server https://github.com, maka server mengirim representasi berupa dokumen HTML yang selanjutnya diproses oleh browser sehingga muncul halaman GitHub.

Didalam representasi yang dikirim server, terdapat beberapa link/anchor yang dapat digunakan oleh user untuk berpindah ke resource lainnya pada server, diantaranya link ke halaman Produk, Team, Enterprise, dll.

Dalam dokumen HTML link tersebut beruap tag Anchor (<a>), misal sebagai berikut:

```
<nav>
      <ul>
            <li><a href="/team">Team</a></li>
            <li><a href="/enterprise">Enterprise</a></li>
            <li><a href="/explore">Explore GitHub</a></li>
      </ul>
</nav>
```

Ketika link tersebut di klik, maka browser akan mengirim request ke server menggunakan url yang ada pada link tersebut dengan HTTP method GET (ketika membuka halaman website browser otomatis menggunakan HTTP method GET).

Selain link terdapat juga form login dan register dimana ketika kita isi form tersebut maka browser akan mengirim data ke server menggunakan method POST.

Dalam arsitektur REST implementasi hypermedia adalah server mengirim data ke client yang berisi informasi HTTP Request yang dapat dilakukan client kedepan setelah represntasi diterima.

"*Hypermedia is a way for the server to tell the client what HTTP requests the client might want to make in the future*."

Atau dengan kata lain hypermedia adalah strategi untuk menawarkan client untuk dapat mengakses resource lain yang ada pada server.

Dengan model seperti ini maka client otomatis dapat mengetahui tautan tautan yang disesiakan oleh server, berikut contoh response server yang mengandung hypermedia:

```
{
      id_user: 1,
      nama: "Agus Prawoto Hadi",
      links: {
            rel: "self", href: "https://jagowebdev.com/api/users/1",
            rel: "orders", href: "https://jagowebdev.com/api/users/1/orders"
      }
}
```

Pada contoh diatas, aplikasi client dapat langsung mengetahui alamat URI untuk detail pembelian yang dilakukan user, sehingga pengembang aplikasi client tidak perlu melihat dokumentasi atau menulis coding URL detail pembelian secara manual, dengan model seperti ini maka ketika sewaktu waktu URI tersebut berubah maka URI pada apliaksi client juga otomatis berubah sehingga aplikasi client tetap berjalan dengan baik.

Pada contoh diatas, penerapan hypermedia ini terlihat mudah, namun dikondisi lain bisa menjadi rumit. Seperti definisi sebelumnya bahwa hypermedia berisi informasi apa yang dapat dilakukan client kedepannya, "Apa yang dilakukan" ini bisa bermacam macam, tidak hanya mengambil data saja tapi bisa juga merubah atau menghapus data, sebagai contoh berikut ini response server yang mengandung hypermedia yang lebih kompleks:

```
{
      "id_user": 1,
      "nama": "Agus Prawoto Hadi",
      "links": {
            "rel": "self", "href": "https://jagowebdev.com/api/user/1",
            "rel": "orders", "href": "https://jagowebdev.com/api/user/1/order"
      },
      "actions": [
            {
                  "name":"update-user",
                  "title":"Update User",
                  "method":"PUT",
                  "href":"https://jagowebdev.com/user/1",
                  "type":"application/x-www-form-urlencoded",
                  "fields": [
                    {"name":"email","type":"email" },
                    {"name":"nama","type":"text" }
                  ]
            },
            {
                  "name":"delete-user",
                  "title":"Delete User",
                  "method":"DELETE",
                  "href":"https://jagowebdev.com/user/1",
                  "type":"application/json"
            }
      ]
```

```
}
```

Pada contoh diatas, server menyediakan berbagai informasi aksi (actions) yang dapat dilakukan client terhadap respurce user yaitu mengubah dan menghapus data user. Penggunaan kata actions beserta format didalam nya tidak ada standar baku yang mengatur struktur data actions ini sehingga implementasinya bisa bermacam macam, dan semakin banyak aksi yang dapat dilakukan, semakin rumit bentuk format data respon, karena kerumitan ini, tidak semua aplikasi server maupun client menerapkan Hypermedia, sehingga banyak yang menulis secara manual alamat URI pada aplikasi client, seperti URI untuk update atau hapus data.

BAB 3 Komunikasi Server dan Client

Sebelum membahas lebih jauh mengenai aplikasi api sebagai server dan client maka penting bagi kita untuk memahami bagaimana server dan client berkomunikasi.

3.1. Client dan Server

Ketika kita membuka halaman web misal https://jagowebdev.com menggunakan web browser, maka browser akan mengirim permintaan (request) ke server https://jagowebdev.com, selanjutnya browser melakukan tindakan sesuai dengan respon yang dikirim oleh server seperti menampilkan halaman web berupa teks dan gambar. Nah browser disini dapat diibaratkan aplikasi client dan https://jagowebdev.com adalah server nya.

Berikut ilustrasinya

3.2. HTTP Protocol

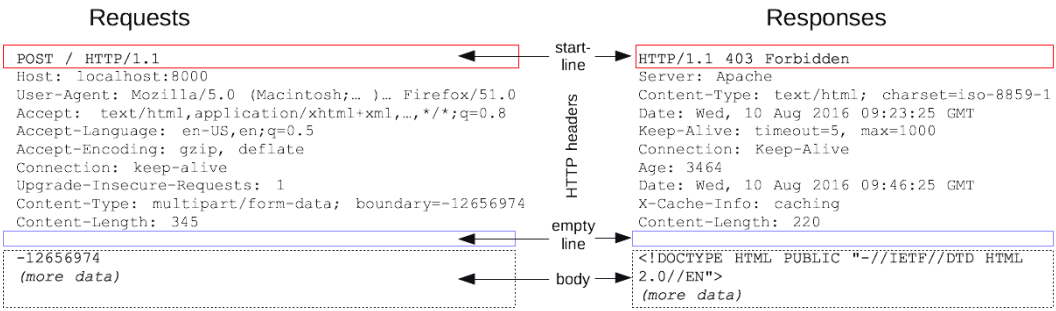
Pada pertukaran data melalui internet protokol yang digunakan bisa bermacam macam, pada contoh web browser sebelumnya, protokol yang digunakan adalah HTTP, hal ini ditandai dengan awalan http/https pada

permintaan yang dikirimkan (https://jagowebdev.com), selain http ada juga protokol lain seperti TCP.

Pada protokol HTTP permintaan (request) yang dikirim ke server disebut HTTP Request dan respon yang dikirim dari server ke client disebut HTTP Response. Isi dari HTTP Request dan HTTP Response ini disebut HTTP Message, HTTP Request merupakan HTTP Message yang dikirim ke server, sedangkan HTTP Respon merupakan HTTP Message yang dikirim dari server ke client.

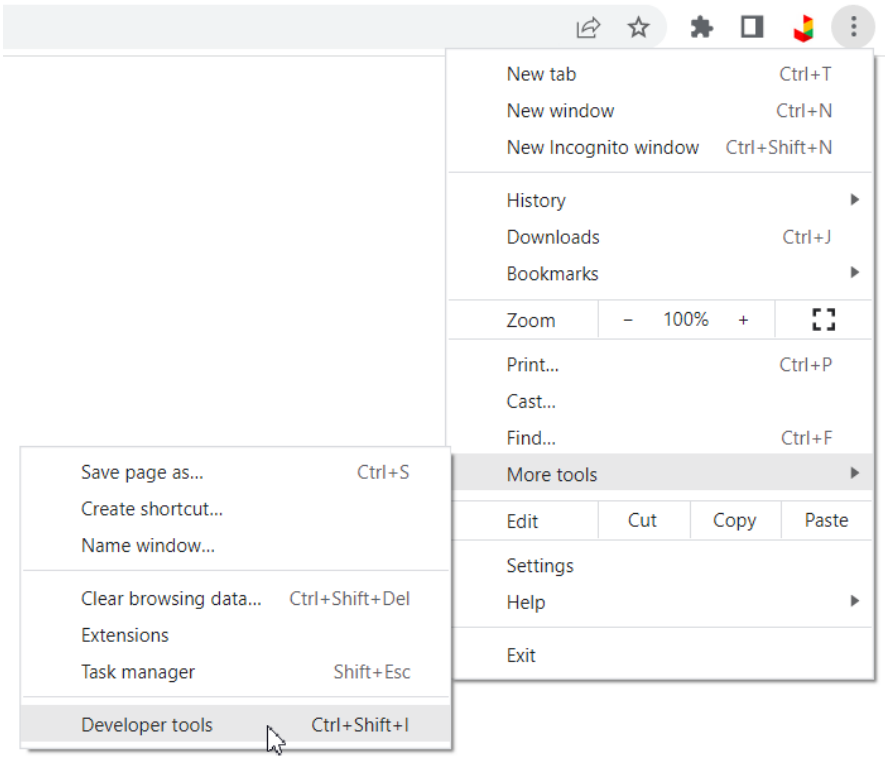
3.3. HTTP Message

Seperti disebutkan sebelumnya bahwa HTTP Message merupakan pesan yang dikirim baik dari client ke server maupun dari server ke client. HTTP Message ini terdiri dari dua bagian yaitu header dan body. Bagian header disebut HTTP Header dan bagian Body disebut HTTP Body, adapun format HTTP Message adalah sebagai berikut:

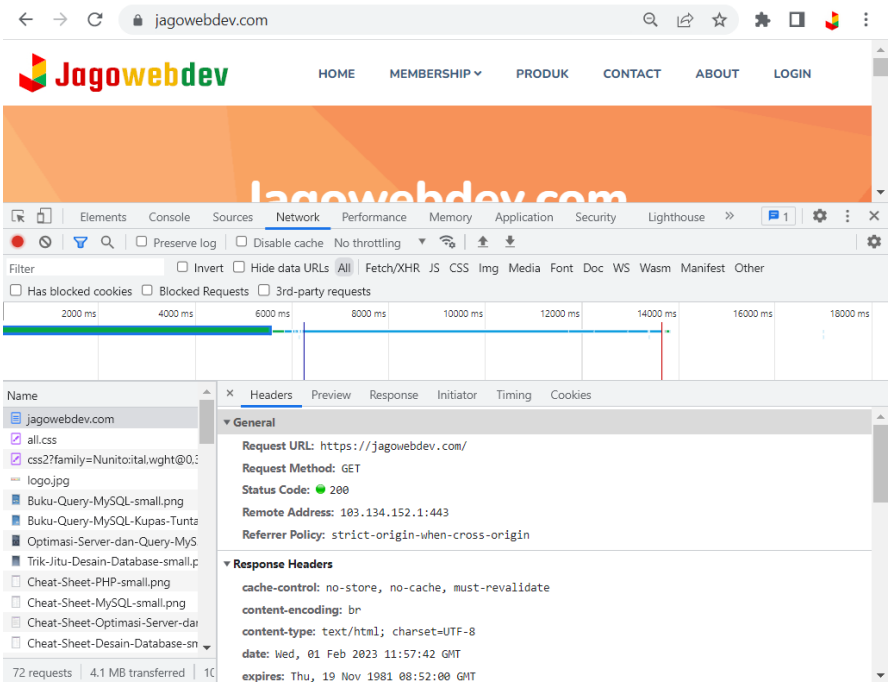


Sumber: https://developer.mozilla.org/

Pada browser, HTTP Message ini dapat dilihat di bagian Developer Tools (Ctrl + Shift + i) atau melalui menu More Tools > Developer Tools sebagai berikut:



Selanjutnya pada jendela Developer Tools yang muncul pilih tab Network, kemudian pada panel sebelah kiri pilih request yang ingin diinspect selanjutnya HTTP message akan ditampilkan di panel sebelah kanan, contoh sebagai berikut:



Pada browser, sebagaimana contoh gambar diatas, bentuk HTTP Meesage sudah dioleh sedemikian rupa sehingga lebih mudah dibaca dan lebih user friendly, disitu juga sudah dipisahkan antara header, response, dll, lebih lanjut mengenai HTTP Message ini dapat dibaca di halaman HTTP Message – HTTP| MDN yang bisa diakses melalui tautan https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

3.4. HTTP Header

HTTP Header dapat diibaratkan seperti meta data file (mime type, date modified file, dll). Ketika mengembangkan aplikasi REST API, kita akan banyak berkecimpung dengan HTTP Header ini, kita akan sering mendefinisikan secara manual informasi header yang akan kita kirim ke server maupun informasi header yang dikirim server ke client.

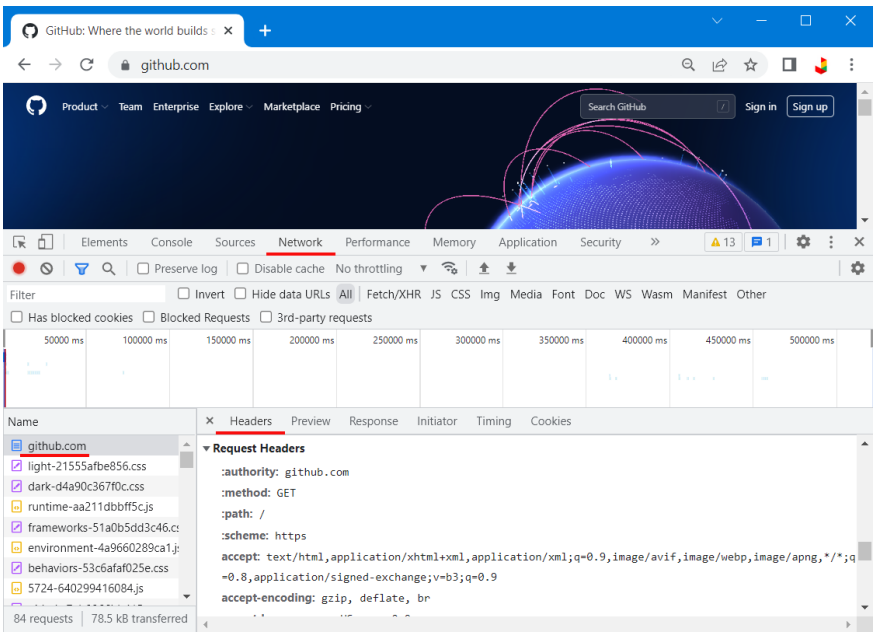
HTTP Header memuat banyak informasi yang dapat digunakan oleh server maupun client untuk melakukan suatu tindakan, misal ketika HTTP Header dari response server terdapat parameter Set-Cookie, maka browser akan menyimpan value cookie tersebut ke cookie browser, demikian juga ketika HTTP Header pada request yang dikirim browser terdapat "accept-encoding: gzip, deflate, br" maka server akan mengirim data pada HTTP Body terkompresi (gzip) sehingga transfer data dapat dilakukan dengan lebih cepat.

HTTP Header terdiri dari pasangan field dan velue, misal:

```
Content-Type: application/json
```

Pada contoh diatas, bagian field adalah Content-Type sedangkan bagian value adalah application/json

Untuk melihat HTTP Header pada browser buka Developer Tools kemudian klink tab Network, pada bagian panel sebelah kiri klik resource yang ingin di inspect kemudian klik tab Headers. Berikut contoh HTTP Header ketika kita membuka halaman https://github.com.



Pada contoh diatas, bagian Request Headers adalah HTTP Header yang dikirim oleh browser.

Penamaan header ini sudah ada standarnya, list lengkapnya dapat dilihat di tautan https://www.rfc-editor.org/rfc/rfc4229 atau di tautan https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers disana banyak sekali list HTTP header yang bisa digunakan, beberapa header yang penting untuk diketahui ketika mengembangkan aplikasi API adalah sebagai berikut:

Content-Type

Content type header digunakan untuk memberitahu format isi (HTTP Body) dari request/response.

Pada client, header ini digunakan untuk memberitahu server format data yang dikirimkan, misal client mengirim request dengan menyertakan Content-Type header berupa application/json, maka dari conten-type ini server akan tahu bahwa data yang akan diterima berupa JSON sehingga server tau apa yang akan dilakukan untuk membaca data tersebut.

Demikian juga dengan response, ketika server menyertakan header Content-Type pada response yang dikirim ke client, maka client akan tahu apa yang harus dilakukan untuk membaca data tersebut.

Contoh nilai Content-Type :

```
Content-Type: application/json
Content-Type: text/html; charset=utf-8
Content-Type: application/x-www-form-urlencoded
Content-Type: multipart/form-data
```

Accept

Accept menunjukkan tipe data yang dapat diterima oleh client. Dengan mendifinisikan Accept, maka server akan memberikan response sesuai dengan format yang dapat diterima oleh Accept tersebut.

Pada header Accept, client dapat mendefinisikan value menggunakan karakter asterik (*) atau secara spesifik mendefinisikan list format dengan

pemisah tanda koma, jika terdapat lebih dari satu format, maka yang menjadi prioritas adalah yang paling kiri dan yang spesifik (tidak mengandung asterik), jika sama sama spesifik maka dahulukan yang tidak memiliki q factor (q=), misal Accept Header memiliki nilai sebagai berikut:

```
text/html,application/xhtml+xml,application/xml;q=0.9,*/*
```

Maka urutan prioritas nya adalah:

- `text/html`
- `application/xhtml+xml`
- `application/xml;q=0.9`
- `*/*`

jika diterjemahkan kedalam bahasa umum maka dapat dikatakan seperti ini: Hi, Server, saya ingin mendapatkan resource dari server dan saya ingin resource tersebut dikirim dalam format HTML atau XHTML, jika tidak bisa, saya minta dikirim dalam format XML, jika tidak bisa juga saya akan menerima apapun yang Anda kirimkan.

Note: Jika Accept header tidak didefinisikan, maka server menganggap bahwa client dapat menerima semua jenis format, namun demikian, jika Accept Header didefinisikan namun server tidak dapat memenuhinya maka server akan memberikan respon 406 Not Acceptable.

Authorization

Authorization header digunakan oleh client untuk menyertakan informasi credential yang digunakan untuk mengakses resource pada server, misal JWT Access Token yang didapatkan setelah client login ke server.

Jika server memerlukan authentikasi dan Authorization Header tidak ditemukan atau Authorization Header ada namun nilainya tidak valid maka server akan mengirimkan respon 401 Uanauthorized, pada response tersebut disertakan parameter WWW-Authenticate yang berisi informasi credential yang diperlukan.

Berikut contoh Authorization Header

```
Authorization Bearer XXX
```

Cookie

Cookei header menunjukkan bahwa request yang dikirim oleh client ke server menyertakan cookie yang disimpan oleh browser. Cookie ini nantinya dapat digunakan oleh server untuk melakukan berbagai hal seperti authentikasi user, dll.

Cache Control

Cache Control Header memungkinkan server untuk memberitahu apakah respon yang dikirim dapat di cache atau tidak, jika dapat dicache maka berapa lama cache dapat disimpan. Berikut ini contoh cache control:

```
Cache-Control: max-age=3600
```

Max-age directive menunjukkan berapa lama client dapat menggunakan cache sebelum meminta request resource yang sama. Pada contoh diatas, jika sebelum satu jam (3600 detik) client meminta request resource yang sama maka response diambilkan dari cache dan tidak melakukan request ke server, dengan model seperti ini maka akan mengurangi jumlah request ke server dan data dapat ditampilkan dengan lebih cepat.

Contoh nilai Cache Control yang lain adalah no-cache.

```
Cache-Control: no-cache
```

Nilai ini menunjukkan bahwa response tidak boleh dicache, setiap permintaan resource harus dilakukan melalui request ke server.

Setting nilai Cache-control merupakan personal judgemen, Anda/Tim Anda sendiri yang menentukan kapan representasi diperbarui, Jika penentuan tidak tepat maka client akan memperoleh data yang out of date.

WWW-Authenticate

Ketika server mengirim respon dengan status code 401 Unauthorized maka server juga menyertakan WWW-Authenticate Header, misal

```
401 Unauthorized HTTP/1.1
WWW-Authenticate: Basic
```

Header WWW-Authenticate menunjukkan jenis authorization apa yang dapat diterima oleh server. Pada contoh ini, server mensyaratkan client untuk menggunakan Basic Authentication, ketika menerima response seperti ini maka seharusnya client mengirim ulang data authorization dengan menyertakan header Authorization.

Custom Header

Selain HTTP Header yang telah didefinisikan pada standar, kita juga dapat mendefinisikan HTTP Header sendiri, misal untuk menyertakan versi api pada request yang dikirim client ke server. Contoh lain adalah memberikan informasi versi PHP yang digunakan, misal sebagai berikut:

Naming Convention

Berikut aturan penamaan field HTTP Header:

- HTTP Header bersifat case insensitive artinya tidak memperhatikan huruf besar maupun huruf kecil, misal: content-

type, Content-type, Conten-Type, namun demikian best practice nya adalah menggunakan huruf besar disetiap awal kata, misal Content-Type

- Untuk custom header, meskipun bisa menggunakan prefix X-, seperti x-powered-by, x-token, dll, namun penggunaan prefik ini sudah tidak disarankan, gunakan saja nama yang unik yang sekiranya tidak akan bentrok dengan standar header yang ada.
- Gunakan hypen (-) untuk pemisah antar kata.

3.5. HTTP Status Code

Ketika membangun aplikasi API maka HTTP Status Code merupakan satu hal yang wajib Anda kuasai karena setiap respon yang dikirim server ke client akan menyertakan status code dan status code ini akan digunakan oleh client untuk memvalidasi apakah requst yang dikirim sukses atau gagal.

HTTP Status code sendiri merupakan kode yang dikirim server ke client sebagai respon atas request yang dikirim client. HTTP Status Code terdiri dari tiga digit integer, dimana digit pertama menunjukkan response class. HTTP Status Code ini dimaintain oleh IANA (Internet Assigned Number Authority), daftar lengkap status code dapat dilihat pada halaman Hypertext Transfer Protocol (HTTP) Status Code Registry yang dapat diakses melalui tautan https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

HTTP Status Code dikelompokkan menjadi 5 class (kategori) sebagai berikut:

- 1xx. Informasional Response
- 2xx. Success
- 3xx. Redirection
- 4xx. Client Error
- 5xx. Server Error

Penting: Untuk status code Error, perlu ditambahkan response payload yang menjelaskan detail error yang terjadi, karena bisa jadi server menghasilkan status code yang sama, misal 400, namun errornya berbeda beda.

3.5.1. Response Code Penting

Dalam mengembangkan REST API, tidak semua response code yang ada pada standar digunakan. Agar lebih fokus, berikut beberapa response code yang penting untuk diketahui:

200 OK

- Status code ini digunakan untuk response sukses namun tidak spesifik pada response sukses tertentu, misal request GET atau PUT yang sukses dieksekusi.
- Status code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET, PUT	Request sukses dieksekusi.

201 Created

- Status code ini digunakan jika proses penambahan resource berhasil.
- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
POST	Penambahan resource sukses

204 Created

- Status code ini digunakan untuk response sukses namun tidak memberikan data, misal ketika kita merequest data user dengan id 232 namun demikian user dengan id tersebut tidak ditemukan di database.

- Status code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET	Request sukses dieksekusi.

400 Bad Request

- Status code ini digunakan jika error tidak spesifik ke jenis error tertentu, misal (1) user tidak menyertakan informasi credential ketika menambah data, (2) ketika ingin mengakses resource, parameter tidak lengkap, (3) terdapat field yang tidak terkirim, dll

- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET	Parameter tidak lengkap, misal: URL seharusnya: /products?title=php&tahun_publikasi=2022 namun client tidak menyertakan title sehigga parameter tidak lengkap, misal /products?tahun_publikasi=2022
POST, PUT, DELETE	Parameter/input yang dikirim tidak lengkap

401 Unauthorized

- Status code ini diberikan jika client ingin mengakses resource namun tidak menyertakan informasi credentials seperti token authentication.

- Status code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET, POST, PUT, DELETE	Client ingin mengakses data (GET), menambah data (POST), mengupdate data (PUT), menghapus data (DELETE) namun tidak menyertakan informasi credentials

403 Forbidden

- Status code 403 digunakan karena akses pada resource tidak diperkenankan, misal user dengan hak akses "Guest" ingin mengakses resource yang khusus diperuntukkan untuk admin. Contoh lain, client mengakses resource berkali kali melebihi limit yang telah ditetapkan.
- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET	Client ingin mengakses data namun tidak diperkenankan.
PUT, DELETE	Client ingin mengupdate (PUT) resurce, menghapus (DELETE) resource namun akses terhadap resource tersebut tidak diperkenankan

404 Not Found

- Status Code ini digunakan jika alamat URI tidak sesuai dengan yang telah ditetapkan atau ketika resource tidak ditemukan, misal ketika client merequest data user tertentu, namun data user tersebut tidak ditemukan di database.
- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
GET, PUT, DELETE	Client ingin mengakses (GET), mengupdate (PUT), atau menghapus (DELETE) resource, namun resource tersebut tidak ditemukan.

409 Conflict

- Status code ini digunakan ketika ada data yang konflik, misal, ketika menerima data registrasi user, terdapat username atau email yang sama dengan yang sudah terdaftar. Status code ini juga digunakan ketika user melakukan request POST berkali kali.
- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
POST, PUT	Data yang dikirim bentrok dengan data yang sudah ada

422 Unprocessable Entity

- Status code ini digunakan ketika data yang dikirim client tidak dapat diproses lebih lanjut karena suatu hal, misal, input yang tidak lolos validasi, seperti username yang seharusnya lebih dari 4 karakter, terdapat data input yang masih kosong, dll
- Status Code ini dapat digunakan pada HTTP Method:

Method	Keterangan
POST, PUT	Data yang dikirim tidak dapat diproses

500 Internal Server Error

- Status code ini digunakan ketika terjadi kesalahan pada server seperti kesalahan penulisan syntax.
- Status code ini dapat digunakan pada HTTP Method:

Method	Keterangan
Semua Method	Terjadi error pada server ketika melakukan pemrosesan request dari client

Berikut resume dari penggunaan status code:

HTTP Method	Response Code	
	Sukses	Error
GET	200	400, 401, 403, 404, 500
POST	201	400, 401, 409, 422, 500
PUT	200	400, 401, 403, 409, 422, 500
DELETE	200	400, 401, 403, 422, 500

Selanjutnya berikut beberapa contoh response header dengan status code 4xx beserta pesan errornya:

Status code 403

```
HTTP/2 403
{
      "message": "Maximum number of login attempts exceeded. Please try
again later.",
      "documentation_url": "https://docs.github.com/rest"
}
```

Status code 401

```
HTTP/2 401
{
      "message": "Bad credentials",
      "documentation_url": "https://docs.github.com/rest"
}
```

Status code 422

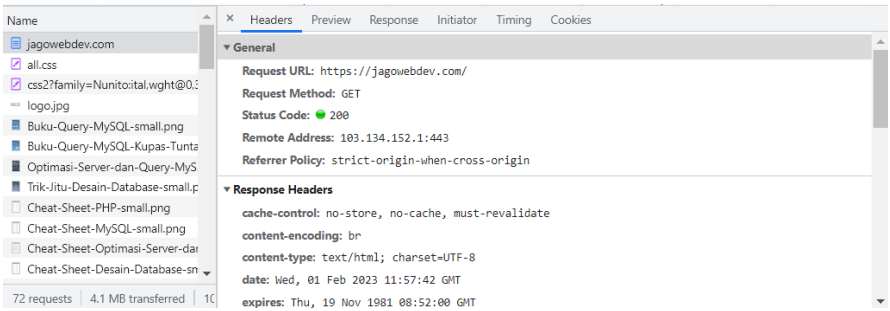
```
HTTP/2 422 Unprocessable Entity
Content-Length: 149

{
      "message": "Validation Failed",
      "errors": [
            {
                  "resource": "Issue",
                  "field": "title",
                  "code": "missing_field"
            }
      ]
}
```

Sumber: https://docs.github.com/en/rest/overview/resources-in-the-rest-api

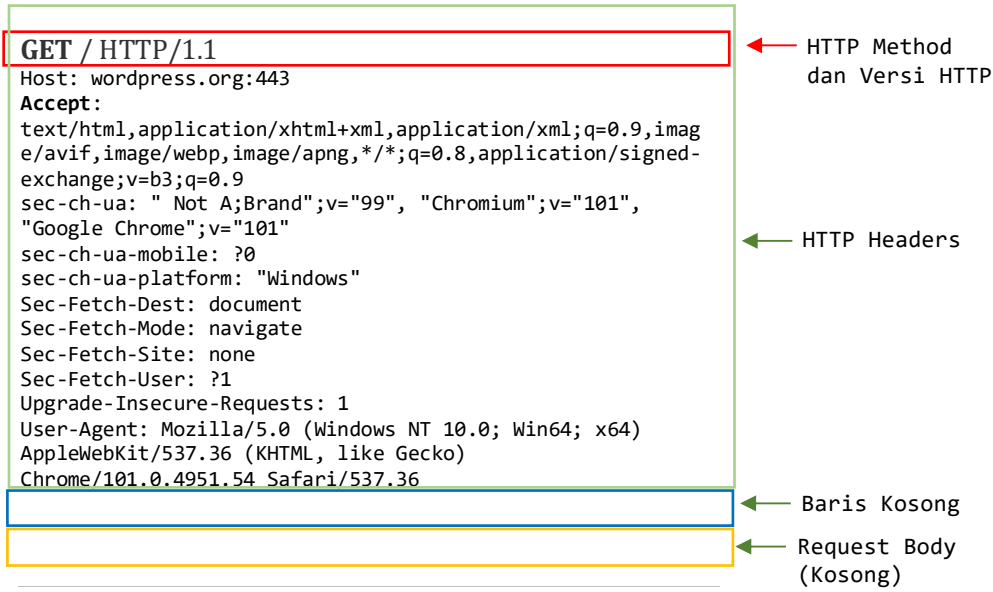
3.6. HTTP Request

HTTP Request merupakan permintaan (request) yang dikirim oleh client ke server. Sebagai contoh ketika kita mengakses URL https://jagowebdev.com menggunakan browser maka dibelakang layar

browser akan mengirim permintaan (request) ke server https://jagowebdev.com. Pada browser Chrome HTTP Request ini dapat dilihat melalui Developer Tools, tab Netwok sebagai berikut:

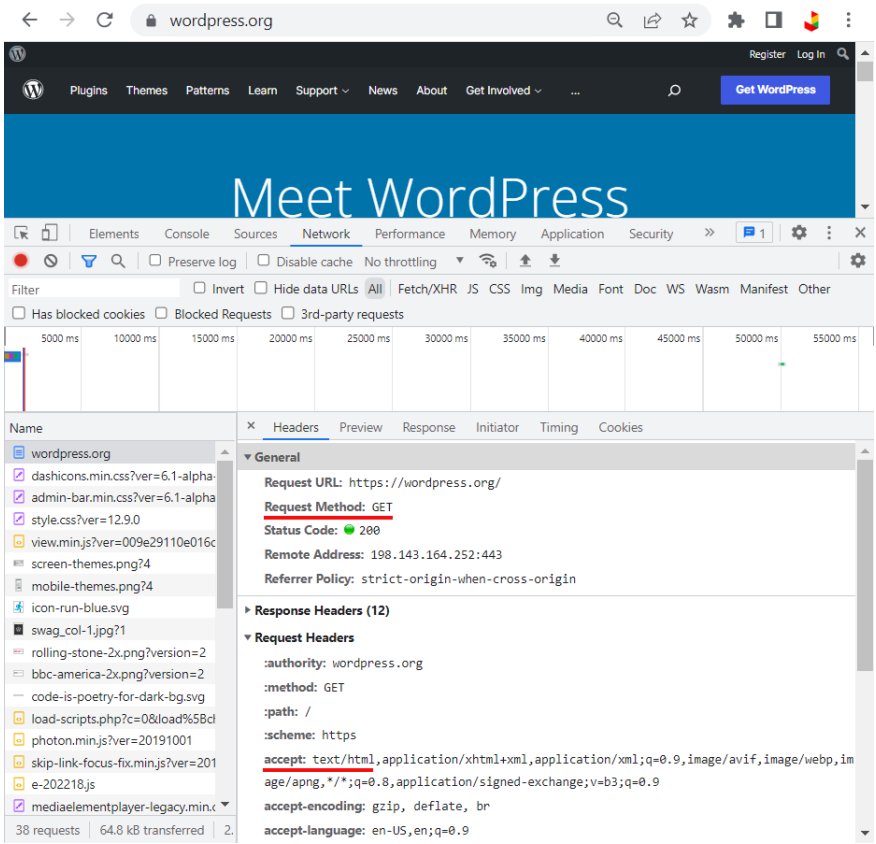


Contoh lain adalah ketika kita mengakses url https://wordpress.org, maka browser (client) akan mengirim request ke https://wordpress.org dengan HTTP Request Message sebagai berikut:

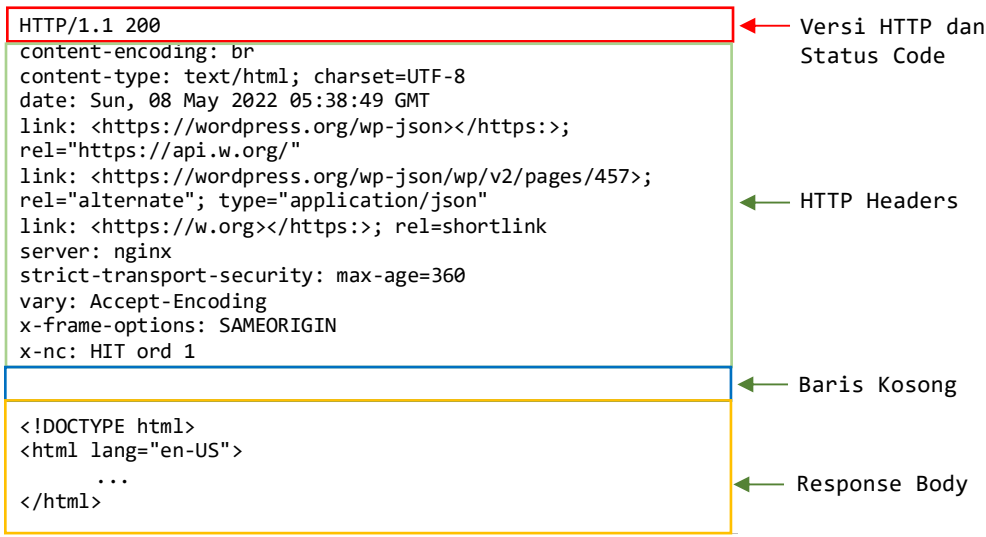


pada contoh diatas bagian Request Body kosong karena pada method GET tidak ada data yang dikirim, pada method POST, Request Body ini akan berisi data form yang dikirim ke server.

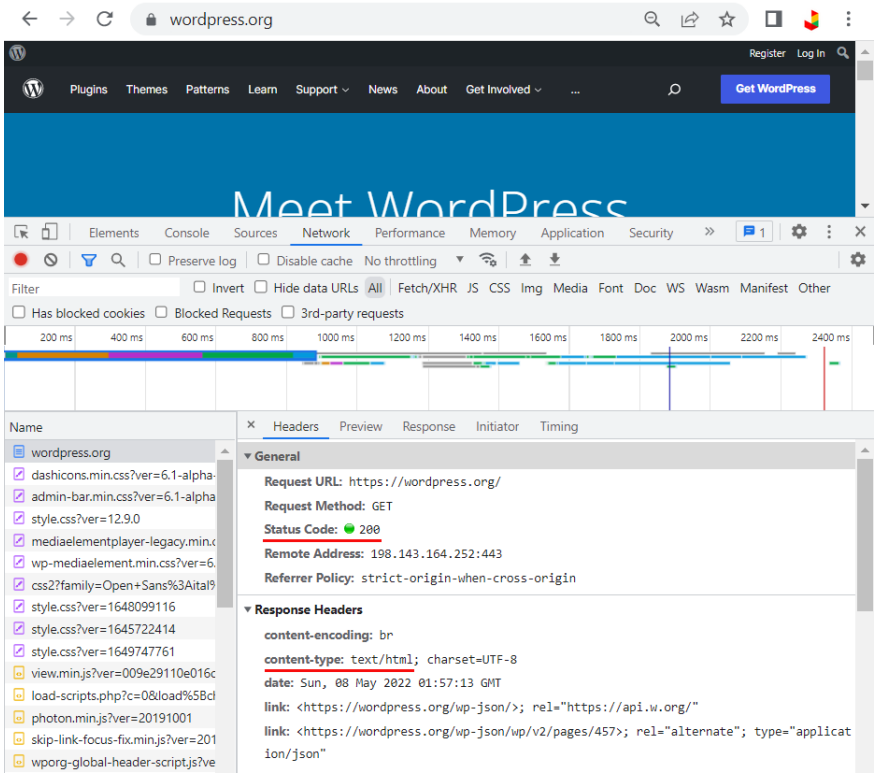
Pada browser data HTTP Request diatas dikelola sedemikian rupa sehingga informasi mudah di gunakan, berikut contoh pada browser Chrome:



Berdasarkan request yang dikirim oleh client, maka server memproses request tersebut kemudian memberikan respon sesuai dengan informasi yang dikirim oleh client. Sesuai standar yang ada, bentuk HTTP Response Message yang dikirim server adalah sebagai berikut:



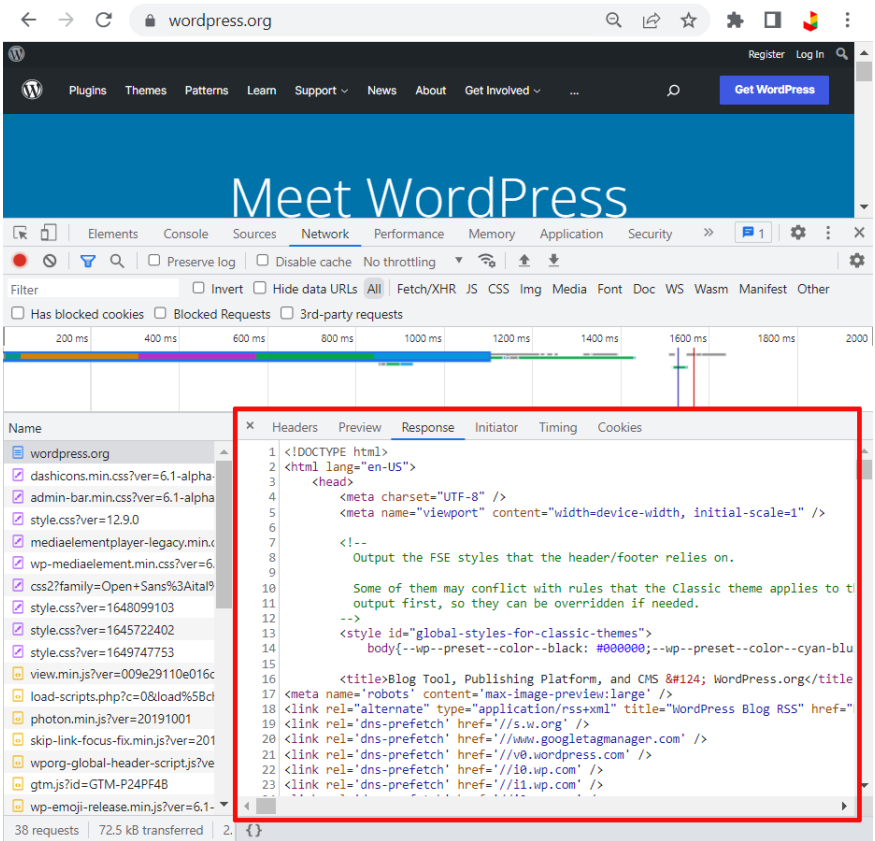
Sama seperti HTTP Request, pada browser infomasi HTTP Response ini dikelola sedemikian rupa sehinga mudah untuk di inspect, berikut contoh pada browser Chrome:



Pada proses diatas terlihat bahwa wordpress.org mengirim respon dengan Status Code 200 dan Content-Type Header : text/html, hal ini karena pada Request Header Accept (pada contoh sebelumnya), nilai paling kiri adalah text/html:

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
```

sehingga response yang dikirim adalah dokumen HTML. Adapun representasi nya adalah HTML sebagai berikut:



HTTP Response Body tersebut selanjutnya di proses oleh browser sehingga muncul tampilan halaman wordpress.org.

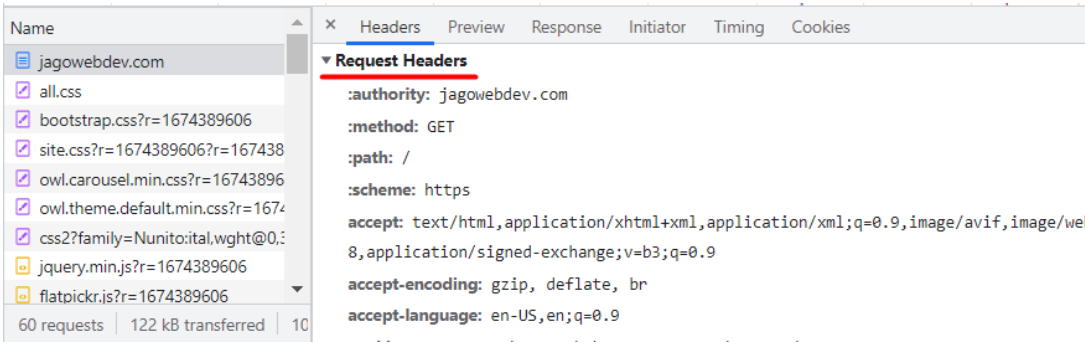
Setiap request yang dilakukan oleh client (browser) selalu disertai dengan HTTP Method, jika request ditujukan untuk mendapatkan data, maka yang digunakan GET, pada kondisi lain, misal mengirim data melalui form, maka HTTP Method yang digunakan adalah POST (browser hanya mengenal GET dan POST). Pada aplikasi API yang kita kemangkan nantinya HTTP Method yang digunakan lebih banyak lagi seperti PUT, DELETE, OPTIONS.

3.6.1. HTTP Request Header

HTTP Request Header adalah HTTP Header yang dikirim client kepada server yang berisi berbagai informasi yang dapat digunakan server untuk memproses request dari client.

Request Header yang penting untuk diperhatikan adalah Accept yang menunjukkan bentuk respon apa yang diharapkan client. Pada contoh sebelumnya (http://wordpress.org), response yang diharapkan adalah HTML: Accept: text/html

HTTP Request Header ini dapat dilihat melalui developer tools browser seperti pada contoh berikut:



Pada browser, http request header ini otomatis di definisikan oleh browser sesuai dengan kemampuan browser, misal pada gambar diatas, header yang dikirim browser berisi accept-encoding: gzip, deflate, br yang artinya browser dapat menerima content yang di kompres (seperti gzip) yang nantinya oleh browser konten yang di kompres ini akan di ekstrak dan di tampilkan oleh browser, dengan mengetahui kemampuan ini, maka server akan mengirim data dalam bentuk terkompres. Dengan mengirim konten yang di kompres, maka akan memperkecil bandwith dan meningkatkan kecepatan akses data.

Layaknya browser, ketika membangun aplikasi client maka kita perlu mendefinisikan beberapa parameter header secara manual sesuai dengan kebutuhan sehingga server paham apa yang kita inginkan, maka penting

bagi kita untuk mengetahui berbagai parameter yang ada pada HTTP Header ini.

Terkait dengan aplikasi yang kita bangun, HTTP Header yang penting untuk diketahui adalah Authorization Header. Header ini nantinya kita gunakan untuk keperluan otorisasi akses ke resource yang ada pada server, lebih lanjut mengenai header Authorization ini dapat dibaca di halaman Authorization – HTTP | MDN yang dapat diakses melalui tautan https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization.

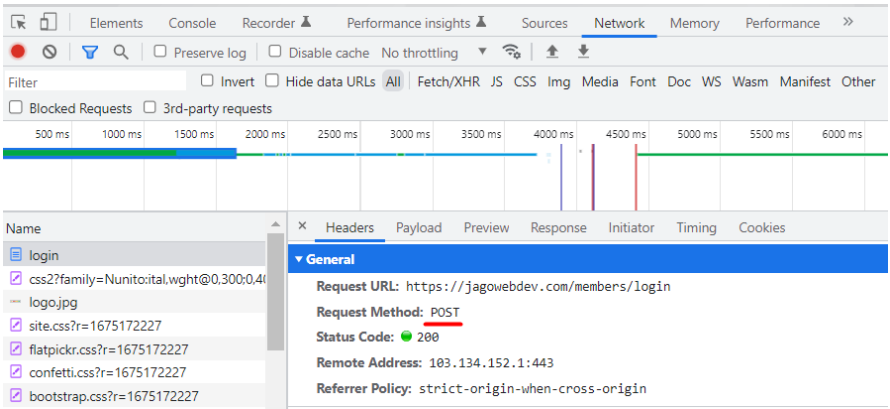
Header lain yang penting diketahui adalah Content-Type. Seperti disebutkan pada sub bab HTTP Header, header ini akan memberitahu baik server maupun client data apa yang dikirimkan, misal ketika client mengirim data ke server dengan header Content-Type application/json atau ketika server mengirim data ke client dengan Content-Type application/json, maka masing masing akan tahu bahwa data yang dikirim berbentuk JSON.

Lebih jauh mengenai header Content-Type dapat dibaca di halaman Content-Type – HTTP | MDN yang dapat diakses melalui tautan https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type

3.6.2. HTTP Request Body

HTTP Request Body berisi data yang dikirim client ke server, pada request dengan method get, tidak ada body yang dikirim, kenapa? Karena semua parameter/data dikirim melalui url.

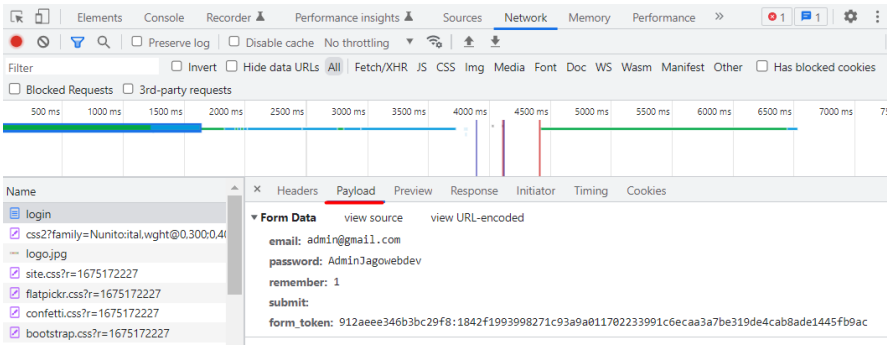
HTTP Request Body ini dapat dilihat dibagian Developer Tools bagian Network, tab Payload. Sebagai contoh ketika kita login dihalaman member Jagowebdev di https://jagowebdev.com/members/login maka ketika form disubmit browser akan mengirim request dengan method post,



kenapa method post? Karena atribut method pada form html bernilai post yaitu sebagai berikut:

```
<form method="post" action="">
      ...
</form>
```

Selanjutnya, pada bagian body dari HTTP Message (payload) akan terlihat data username dan password, serta inputan lainnya yang kita kirim ke server sebagai berikut:



Note: pada request dengan method get, tab Payload ini tidak muncul karena pada method get tidak ada data yang dikirim, body dari HTTP Message kosong.

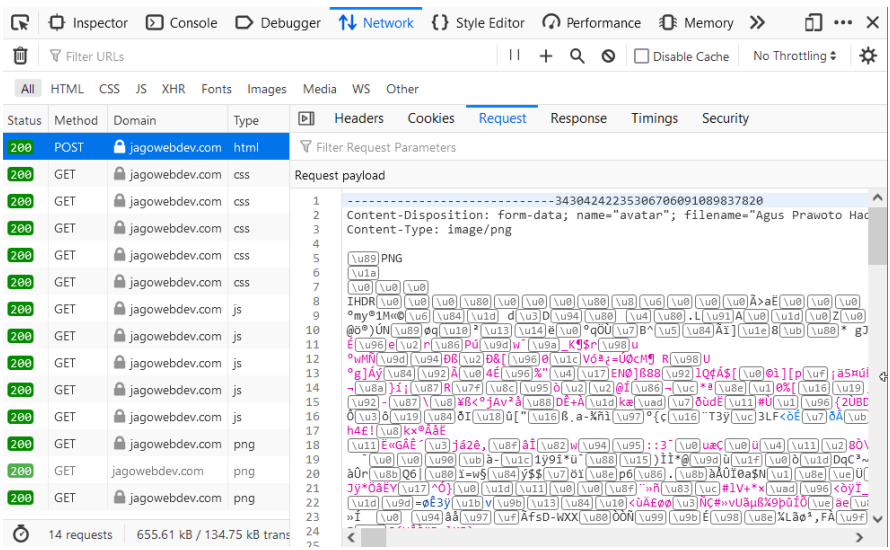
Pada gambar diatas, pada tab Payload, jika kita klik view source, maka akan terlihat data raw (mentah) dari http request body sebagai berikut:

```
email=admin%40gmail.com&password=AdminJagowebdev&remember=1&submi
t=&form_token=08b6c075f8da47f5b1%3A4fc55ca46693a3a04ac0a81decd114
97c7006f81f206fb68fe27e8720c3a402363
```

Bentuk data RAW inilah yang dikirim client ke server, jika server berbasis PHP maka secara otomatis data ini disimpan pada variabel $_POST (karena http method yang digunakan POST) dalam bentuk array.

Note: memahami bentuk raw HTTP Body ini sangat penting, kenapa? Karena ketika membangun Aplikasi Client dan ingin mengirim data ke server, maka bentuk data yang dikirim ke server salah satunya berbentuk data raw seperti diatas.

Contoh lain ketika kita mengirim data ke server yang berisi data binary, misal kita mengirim data image ke server, maka bentuk payloadnya adalah sebagai berikut:



Note : Pada contoh diatas penulis menggunakan browser Firefox, karena pada Chrome, request yang menyertakan data binary data Payload nya tidak muncul.

Dari gambar diatas terlihat bahwa bentuk request method nya berupa POST dan data raw nya berbentuk binary. Pada server dengan basis PHP maka data raw file ini akan otomatis disimpan pada variabel $_FILES sedangkan data lainnya akan disimpan dalam variabel $_POST.

Note: Jika request method bukan post (misal put) maka pada PHP data raw ini tidak akan disimpan pada variabel $_POST maupun $_FILES, inilah kenapa pada aplikasi api yang kita bangun nantinya jika akan melakukan update data yang menyertakan binary file, kita menggunakan HTTP method post bukan put.

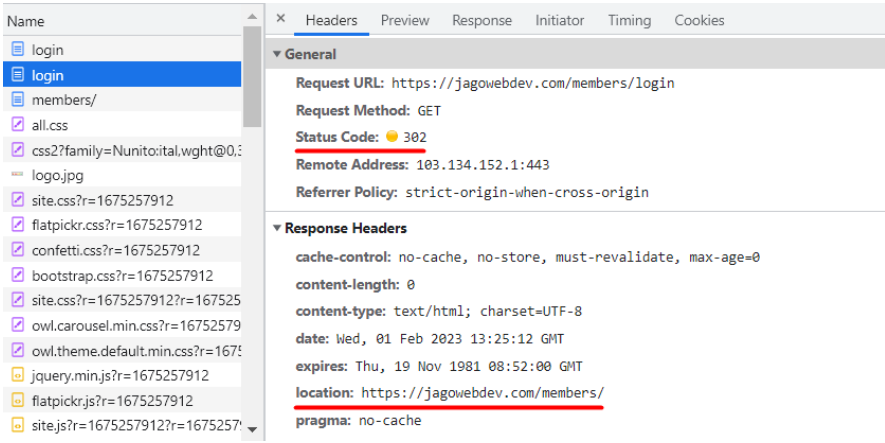
3.7. HTTP Response

Kebalikan dari HTTP Request, HTTP Response merupakan respon (HTTP Message) yang dikirim oleh server ke client. Ketika mengembangkan aplikasi api baik server maupun client, penting untuk memahami tentang HTTP Response ini.

3.7.1. HTTP Response Header

HTTP Response Header merupakan header dari HTTP Message yang dikirim dari server ke client. Pada response header ini terdapat berbagai value yang dapat digunakan oleh client untuk melakukan suatu tindakan, misal response header dengan kode 302 akan memberitahu client bahwa halaman yang diminta sementara berpindah sehingga ketika client menemui response code ini maka seharusnya client melakukan redirect ke halaman sesuai url yang disertakan pada response header.

Sebagai contoh ketika kita berhasil login ke halaman jagowebdev: https://jagowebdev.com/members/login, maka server akan memberikan response code 302 dan memberikan data url pada parameter location pada response header sebagai berikut:



Ketika menerima response tersebut maka browser akan melakukan redirect ke url https://jagowebdev.com/members/

Note: Header location tersebut muncul karena pada server terdapat script redirect `header('location: https://jagowebdev.com/members/')`

Contoh berikutnya adalah Header Content-Type. Ketika Header Content-Type bernilai application/json, maka browser akan menampilkan data JSON dengan format yang mudah dibaca, misal sebagai berikut

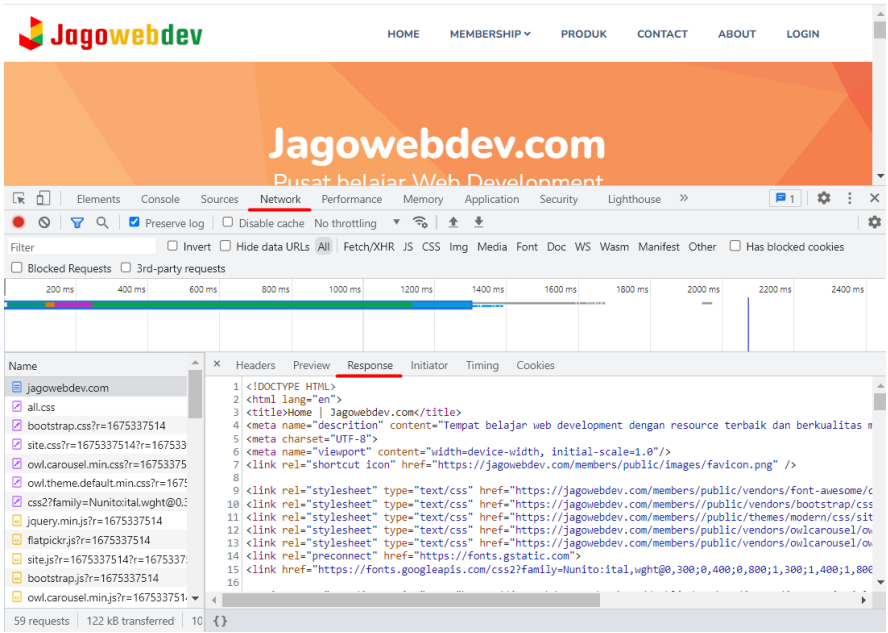
Sama seperti browser, ketika kita mengembangkan Aplikasi Client, maka kita harus dapat menterjemahkan dengan baik HTTP Response Header yang dikirim oleh server sehingga aplikasi client yang kita bangun dapat memberikan respon yang tepat, misa jika parameter Content-Type pada header yang dikirim server bernilai application/json maka response body

nya pasti berbentuk data json sehingga kita dapat langsung memparsing data json tersebut.

3.7.2. HTTP Response Body

HTTP Response Body merupakan body dari HTTP Message yang dikirim dari server ke client. Bentuk data response body ini juga bisa bermacam macam, bisa html, teks, json, binary (misal file image, pdf, excel), dll. Bentuk data ini tercermin dari header Content-Type.

HTTP Response body ini dapat dilihat pada developer tools tab Network bagian response, misal sebagai berikut:



Pada contoh diatas, HTTP Response Body berupa dokumen HTML, hal in tercermin dari Content-Type pada header yaitu text/html.

3.7.3. Content Negotiation

Content Negotiation merupakan mekanisme dimana Server API memberikan representasi sesuai dengan request yang dikirim oleh user,

pada protokol HTTP, content negotiation ini dilakukan melalui HTTP Header.

HTTP Header yang digunakan untuk melakukan content negotiation adalah Accept dan Accept-Language, sebagai contoh ketika kita mengakses url menggunakan browser, maka browser akan mengirim request dengan accept header berupa list format dokumen yang dapat diterima oleh browser:

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
```

karena fungsi utama browser adalah menampilkan dokumen HTML, maka prioritas pada list tersebut adalah text/html.

Selain Accept Header, browser juga mengirim Accept-Language Header sebagai berikut:

```
Accept-Language: en-US,en;q=0.9
```

Pada Accept-Language Header diatas, browser meminta agar server mengirim resource dalam bahasa American English.

Contoh content negotiation dari sisi server:

Sebuah server memiliki reource berupa data user, untuk mengakses resource ini disediakan identifier berupa alamat URI sebagai berikut: https://jagowebdev.com/api/v1/users

Selanjutnya ketika client ingin mengambil data resource ini, maka server akan mengidentifikasi representasi apa yang diinginkan oleh client. Setiap client memiliki kebutuhan yang berbeda beda, jika yang diminta adalah dokumen HTML (Accept:text/html), maka server akan memberikan output HTML, misal:

```
<html>
<head>
```

```
      <title>Data User</title>
</head>
<body>
      <table>
            <thead>
                  <tr>
                        <th>Nama</th>
                        <th>Email</th>
                        <th>Website</th>
                  </tr>
            </thead>
            <tbody>
                  <tr>
                        <th>Agus Prawoto Hadi</th>
                        <th>prawoto.hadi@gmail.com</th>
                        <th>https://jagowebdev.com</th>
                  </tr>
            </tbody>
      </table>
</body>
</html>
```

Jika client menginginkan data tersebut dikirim dalam bentuk format JSON (Accept: application/json), maka server akan mengirim data JSON, misal:

```
{
      "nama" : "Agus Prawoto Hadi",
      "email" : "prawoto.hadi@gmail.com",
      "website" : "https://jagowebdev.com"
}
```

Pada contoh diatas terlihat bahwa dengan satu identifier, representasi dari resource dapat bermacam macam sesuai dengan kebutuhan client.

3.7.4 Sinkronisasi HTTP Response Header dan HTTP Response Body

Pada pembahasan sebelumnya telah dijelaskan pentingnya informasi HTTP Header baik bagi aplikasi server maupun aplikasi client salah satunya karena isi dari HTTP Body tercermin dari informasi Content-Type yang ada pada HTTP Header, untuk itu ketika mengembangkan aplikasi api

baik server maupun client, penting bagi kita untuk selalu memperhatikan sinkronisasi antara HTTP Header dan HTTP Body.

Ketidak sinkronan antara HTTP Header dan HTTP Body dapat menyebabkan client salah dalam menterjemahkan response dari server, misal pada response yang diterima client header Content-type nya bernilai application/json namun pada bagian body berupa data binary maka client akan menganggap response dari server adalah data json, sehingga ketika mencoba melakukan decode akan mendapati error.

3.8. HTTP Method (Verbs)

Ketika mengembangkan aplikasi api maka kita harus memahami dengan baik HTTP Method. Kenapa demikian? Karena REST menggunakan HTTP Method untuk menterjemahkan permintaan yang dikirim oleh client.

Setiap HTTP Method memiliki fungsi tertentu, seperti POST untuk menambah data, GET untuk mendapatkan data, dll, dengan menyertakan HTTP Method pada setiap request, maka tujuan request menjadi jelas, sehingga nantinya dapat diperkirakan hasil apa yang akan didapatkan.

HTTP Method yang sering digunakan ketika mengembangkan aplikasi API adalah GET, PUT, POST, DELETE, dan OPTIONS. List lengkap beserta penjelasan mengenai HTTP Method dapat dibaca di https://www.rfc-editor.org/rfc/rfc9110.html#name-method-definitions dan https://www.rfc-editor.org/rfc/rfc7231#page-59

3.8.1. Idempoten

Sebelum membahas lebih lanjut mengenai HTTP Method penting bagi kita untuk memahami tentang idempoten.

Apa itu idempoten?

Seperti yang disebutkan pada standar yang ada (https://www.rfc-editor.org/rfc/rfc9110.html#name-idempotent-methods) idempoten adalah properti HTTP Method.

Lantas apa maksud idempoten?

Idempoten berarti jika request identik dijalankan berulang kali maka akan menghasilkan efek yang sama (pada server) seperti ketika hanya satu request dijalankan.

Dengan kata lain, HTTP Method bersifat idempoten jika HTTP Method tersebut dijalankan berulang kali (dengan input yang sama) maka efek yang terjadi pada server akan sama seperti ketika hanya satu request saja yang dijalankan, jadi tidak peduli request dijalankan satu atau seratus kali, efek yang terjadi pada server akan sama.

Sebagai contoh HTTP Method GET bersifat idempoten karena berapapun request dijalankan akan memberikan hasil yang sama dan data pada server tidak berubah, misal ketika kita membuka halaman web https://jagowebdev.com maka kondisi data pada server tidak berubah.

Mengapa memahami idempoten ini pemning?

Dengan memahami idempoten ini, maka ketika mengembangkan aplikasi api kita dapat memetakan idempotensi HTTP Method sehingga dapat menentukan langkah yang tepat.

Sebagai contoh pada aplikasi api untuk transaksi keuangan, ketika user melakukan request penambahan dana maka server akan otomatis menambah saldo dana nasabah tersebut, namun bagaimana jika saldo sudah bertambah namun ternyata request dari user tersebut timeout dan user mengulang request tersebut? Jika hal ini tidak diantisipasi maka akan terjadi transaksi yang tidak sah, saldo akan terus bertambah.

Dengan memahami bahwa request tersebut tidak idempoten, maka pengembang dapat menentukan langkah langkah untuk mencegah terjadinya transaksi yang tidak sah, diantaranya menambahkan key unik pada header sebagai identifier dari request, misal sebagai berikut:

```
POST / HTTP/1.1
Host: jagowebdev.com:443
Accept: application/json,application/xml
Idempoten-key: AgpX9OSikjUjahPka
```

```
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54
Safari/537.36
```

jika request berhasil maka simpan key tersebut pada database dan tandai bahwa request dengan identifier tersebut telah berhasil dieksekusi sehingga jika terdapat request dengan identifier yang sama maka tolak request tersebut dan beri response ke client bahwa request sebelumnya telah berhasil dieksekusi.

Berikut ini daftar HTTP Method beserta status idempoten nya.

HTTP Method	Idempoten
GET	YA
POST	TIDAK
PUT	YA
PATCH	TIDAK
DELETE	YA
OPTION	YA
HEAD	YA

3.8.2. Safe Method

Selain Idempoten, istilah berikutnya yang perlu kita pahami adalah Safe Method.

Apa itu Safe Method?

Sama seperti Idempoten, Safe Method adalah properti HTTP Method (https://www.rfc-editor.org/rfc/rfc9110.html#name-safe-methods)

Lantas apa maksudnya Safe Method?

Safe method berarti bahwa HTTP Method yang ketika dieksekusi tidak merubah resource/data pada server.

HTTP Method yang termasuk kategori Safe Method adalah GET, HEAD, dan OPTIONS.

Mengapa perlu mengkategorikan HTTP Method sebagai Safe Method?

Dengan mengkategorikan HTTP Method sebagai Safe Method akan membantu kita melakukan tindakan yang tepat, diantaranya:

- Cache, pada HTTP Method dengan kategori safe method maka server dapat mengijinkan client untuk mencache response sehingga dapat meningkatkan performa aplikasi dan mengurangi trafik network.
- Request retry. Pada HTTP Method dengan kategori safe method kita tidak perlu memproteksi resource server karena berapa kali pun client melakukan request yang sama tidak akan merubah data pada server.
- Tidak digunakan untuk merubah data. Method dengan kategori Safe Method tidak digunakan untuk merubah data, sebagai contoh method GET terkadang digunakan untuk menghapus data sebagai berikut: https://jagowebdev.com/api/user?id=1&action=delete dengan mengetahui bahwa GET adalah safe method maka kita perlu memproteksi server agar perintah tersebut tidak dieksekusi dan tidak seharusnya ketika kita mengembangkan aplikasi api server mengijinkan Safe Method digunakan untuk melakukan perubahan data.

Berikut ini daftar HTTP Method beserta status idempoten dan safe method nya

HTTP Method	Idempoten	Safe Method
GET	YA	YA
POST	TIDAK	TIDAK
PUT	YA	TIDAK
PATCH	TIDAK	TIDAK
DELETE	YA	TIDAK

HTTP Method	Idempoten	Safe Method
OPTION	YA	YA
HEAD	YA	YA

3.8.3. GET

Pada standar yang ada, method GET digunakan untuk mendapatkan resource/data/informasi dari server dan tidak digunakan untuk mengubah data sehingga method GET ini bersifat idempoten, selain itu method ini juga bersifat safe karena tidak merubah data pada server.

Method GET ini menghasilkan response code sebagai berikut:

- Jika request menghasilkan data maka response yang dikirim menggunakan response code 200 OK dengan body berupa data.
- Jika data tidak ditemukan, maka response code yang diberikan adalah 404 Not Found.
- Jika parameter tidak benar/tidak lengkap maka response code yang dihasilkan adalah 400 Bad Request.

3.8.4. POST

Method POST digunakan untuk menambahkan data pada server. Data yang akan ditambahkan ada di request body, method ini bersifat tidak aman (unsafe) karena mengubah data pada server dan tidak idempoten karena request yang sama yang dikirim ke server akan menghasilkan result yang berbeda, misal ketika ingin menambahkan data user, maka setiap request yang dikirim akan memberikan dampak pada server yaitu penambahan data user yang sama berkali kali.

Pada standar yang telah ditetapkan (https://www.rfc-editor.org/rfc/rfc9110.html#name-post) jika method POST berhasil dieksekusi maka response code yang dihasilkan adalah 201 (Created) dan response header yang dikirim ke server mengandung alamat dari resource yang baru ditambahkan tersebut, misal sebagai berikut:

```
201 Created
Location: https://jagowebdev.com/api/user/1
```

Jika terjadi error, maka response code bisa bermacam macam tergantung jenis errornya, hal ini tidak diatur di standar, jika error disebabkan karena kelasahan client, seperti isian tidak lengkap, isian tidak sesuai kriteria, dll response code yang digunakan adalah 400.

3.8.5. PUT

Method PUT digunakan untuk memperbarui (mengupdate/mengedit) resource yang sudah ada, pada method PUT ini data yang diganti adalah semua field/data, misal pada resource user terdapat field nama, email, dan password sebagai berikut:

```
nama: Agus Prawoto Hadi
email: prawoto.hadi@gmail.com
password: password123
```

maka dengan method put ini, jika ingin mengganti data email, maka data nama dan password juga harus disertakan, jika tidak maka akan diberi nilai null, misal jika parameter yang dikirim hanya email sebagai berikut:

```
PUT /user/1
{
      email: agusph@jagowebdev.com
}
```

Maka pada server data user akan berubah menjadi:

```
nama: null
email: agusph@jagowebdev.com
password: null
```

Method ini termasuk kriteria method yang tidak aman karena digunakan untuk mengubah data pada server, namun demikian, method ini bersifat idempoten karena meskipun request dilakukan berkali kali, data hanya akan berubah sekali saja, misal ketika mengupdate data nama pada user

dengan ID 1 dari Agus menjadi Hadi, maka berapa kalipun request dikirim maka nama tetap akan berubah menjadi Hadi.

Berdasarkan standar yang ada (https://www.rfc-editor.org/rfc/rfc9110.html#name-put) jika method ini sukses dieksekusi maka response code yang dikirim ke client adalah 200, jika terjadi error maka seperti method POST, response code yang dihasilkan tergantung jenis error nya, jika error yang disebabkan karena inputan user, maka status code yang dihasilkan adalah 400.

Pada standar yang ada jika resource yang diupdate tidak ada, maka server **dapat** membuat resource baru kemudian mengirim response 201 (Created) ke client, sama seperti method POST. Hal ini bersifat optional, dalam praktik penggunakan PUT seperti ini akan membuat rancu dan membarikan hasil yang tidak dapat diprediksi sehingga hindari penggunakan PUT untuk menambah data.

3.8.6. PATCH

Method Patch sama seperti method PUT yaitu digunakan untuk memperbaruai/mengupdate resource pada server, bedanya PATCH digunakan untuk mengupdate bagian tertentu/field tertentu dari data, misal hanya data nama atau email saja. Method PATCH ini baru ada pada tahun 2010.

Pada standar yang ada, method PATCH ini bersifat tidak aman (unsafe) karena method ini merubah data pada server, selain itu method ini juga tidak idempoten.

Dalam praktik, di hampir pada semua kasus, method ini bersifat idempoten karena sifatnya merip dengan method PUT, sebagai contoh ketika kita melakukan update data email user sebagai berikut:

```
PATCH /user/1
{
      email: prawoto.hadi@gmail.com
}
```

maka berapakalipun request tersebut diulang, hasilnya akan tetap sama, yaitu nilai email akan menjadi: prawoto.hadi@gmail.com, namun demikian dalam standar yang ada (https://www.rfc-editor.org/rfc/rfc5789), PATCH tidak hanya digunakan untuk update data, tetapi memiliki makna yang lebih luas, diantaranya dapat digunakan untuk menambahkan resource baru:

"With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version

The PATCH method affects the resource identified by the Request-URI, and it also MAY have side effects on other resources; i.e., ***new resources may be created****, or existing ones modified, by the application of a PATCH."*

Terjemahan bebasnya adalah sebagai berikut:

Dengan PATCH, entitas terlampir (pada request body) berisi serangkaian instruksi yang menjelaskan bagaimana sumber daya yang saat ini berada di server dimodifikasi untuk menghasilkan versi baru.

Method PATCH mempengaruhi resource yang diidentifikasi pada request URI, dan mengakibatkan efek samping pada resource lain seperti: ***penambahan resource baru*** *(data baru) atau respurce yang ada dimodifikasi ketika menerapkan method PATCH ini.*

Pada deskripsi diatas juga disebutkan bahwa request URI dengan method PATCH dapat membawa entitas (pada request body) yang berisi serangkaian instruksi untuk melakukan operasi pada resource, yang bisa jadi mengakibatkan penambahan resource baru.

Sebagai contoh pada RFC6902 (https://www.rfc-editor.org/rfc/rfc6902) tentang "PATCH Method for HTTP" dicontohkan penggunaan Patch untuk mengubah dokumen sebagai berikut:

```
PATCH /my/data HTTP/1.1
Host: example.org
Content-Length: 326
Content-Type: application/json-patch+json
```

```
If-Match: "abc123"

[
      { "op": "test", "path": "/a/b/c", "value": "foo" },
      { "op": "remove", "path": "/a/b/c" },
      { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ]
},
      { "op": "replace", "path": "/a/b/c", "value": 42 },
      { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
      { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

atau dalam contoh operasi pada database:

```
PATCH /user/1 HTTP/1.1
Host: jagowebdev.com
Content-Length: 191
Content-Type: application/json-patch+json

[
      { "op": "replace", "field": "email", "value":
"budi@gmail.com"},
      { "op": "add", "field": "alamat", "value": "Jl. Pemuda N0.
02, Semarang" },
      { "op": "remove", "field": "kode_pos"},
]
```

Pada contoh diatas, terlihat bahwa dengan method patch kita dapat menambah, mengubah, maupun menghapus data/field.

Berdasarkan uraian diatas maka jelas bahwa method PATCH ini bersifat tidak idempoten.

3.8.7. PUT Vs PATCH - Real Life

Pada uraian diatas, maka dapat disimpulkan bahwa perbedaan PUT dan PATCH adalah jika pada PUT yang diupdate adalah semua data/field sedangkan pada PATCH yang diupdate hanya data/field tertentu.

Pada method PUT, karena yang diupdate adalah semua data/field, maka request yang dikirim harus menyertakan semua data, jika tidak, maka akan diberi nilai null, sedangkan pada PATCH hanya perlu mengirim data yang berubah saja.

Terkait dengan penerapan pada pengembangan aplikasi api, ketika kita menyediakan form untuk update data, semua data yang ada di form tersebut kita kirim ke server, terlepas apakah data tersebut di ubah user atau tidak, sehingga bisa jadi hanya beberapa field saja yang diupdate atau bisa jadi semua field diupdate dan terlalu kompleks jika kita mengirim data yang berubah saja, untuk itu pada aplikasi api yang kita kembangkan, kita menggunakan method PUT untuk update data, berapapun data/field yang diupdate.

3.8.8. DELETE

Method DELETE digunakan untuk menghapus resource pada server. Method ini berifat idempoten karena berapa kalipun request diulang akan memberikan hasil yang sama dan method ini bersifat tidak aman (unsafe) karena merubah resource pada server.

Berdasarkan standar yang ada (https://www.rfc-editor.org/rfc/rfc7231#page-29) jika eksekusi method ini berhasil dan tidak ada informasi lebih lanjut yang diberikan (response body kosong) maka response code yang digunakan adalah 204 (No Content), namun jika server menyertakan response body (yang menggambarkan status) maka response code yang digunakan adalah 200 (OK).

3.8.9. OPTIONS

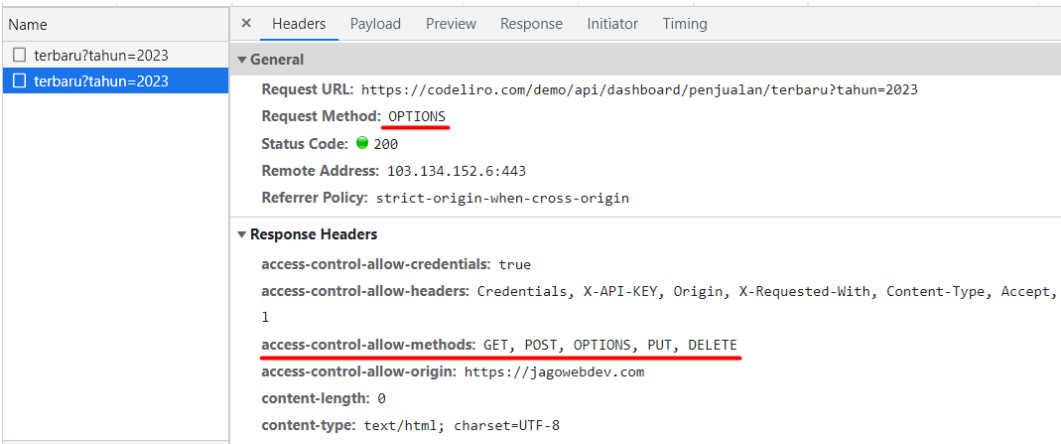
Method OPTIONS ini digunakan untuk melakukan pengecekan HTTP Method apa saja yang diperbolehkan oleh server, dengan pengecekan ini client dapat mengetahui apakah method yang akan dikirim ke server di perbolehkan atau tidak.

Method Options ini bersifat idempoten dan safe, namun demikian hasil dari method ini tidak boleh di cache, kenapa? Karena hasil nya bersifat dinamis bisa berubah sewaktu waktu.

Ketika mengirim request dengan method ini, tidak ada operasi CRUD yang dilakukan pada server, jadi server benar benar hanya memberikan

informasi terkait method yang diperbolehkan, jika request berhasil dieksekusi maka response code yang digunakan adalah 204 (No Content).

Sebagai contoh, pada browser, ketika kita akan mengirim request menggunakan method PUT, maka sebelumnya browser akan mengirim request pada url yang sama dengan method OPTIONS sebagai berikut:



Pada contoh diatas server memberikan response dengan parameter header access-control-allow-methods: GET, POST, OPTIOS, PUT, DELETE karena method PUT ada pada parameter access-control-allow-methods, maka browser akan mengeksekusi url yang sebelumnya akan dikirim ke server.

Request yang dilakukan oleh browser ini disebut CORS preflight request, yaitu request yang digunakan browser untuk melakukan pengecekan apakah server memperbolehkan CORS dan apakah server memperbolehkan method yang akan dikirim, lebih lanjut mengenai preflight request ini dapat dibaca di halaman Preflight request - MDN Web Docs Glossary: Definitions of Web-related terms | MDN yang dapat diakses melalui tautan https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request .

3.8.10. HEAD

Method Head ini sama seperti method GET bedanya response body pada method HEAD kosong, sehingga hanya header saja yang dikirim oleh server. Method ini biasanya digunakan untuk mendapatkan metadata dari suatu resource.

Selain mengetahui metadata dari suatu resource, method ini juga dapat digunakan untuk: (1) mengetahui kapan perubahan terakhir pada resource (header If –Modified-Since), (2) untuk mengetahui ukuran file (header Content-Length), (3) melakukan testing apakah resoure eksis (response code 200 atau 404), dll.

Halaman ini sengaja dikosongkan
Jagowebdev.com

BAB 4 Authentikasi dan Authorisasi

Authentikasi dan Authorisasi merupakan sesuatu yang penting untuk dipahami ketika kita membangun suatu sistem yang terproteksi, salah satunya Aplikasi Api. Meskipun sering membaca atau mendengar kedua istilah tersebut, masih banyak yang belum memahaminya dengan baik.

Authentikasi adalah proses memverifikasi identitas untuk memastikan bahwa pengguna memiliki ijin untuk mengakses sistem. Contoh mudahnya adalah SIM, dengan memiliki SIM berarti kita memiliki ijin untuk mengemudikan kendaraan, atau tiket pertunjukan, dengan memiliki tiket maka kita memiliki ijin untuk menonton pertunjukan.

Adapun Authorisasi adalah proses pemberian hak akses kepada pengguna yang telah terverifikasi melalui proses authentikasi. Tujuan dari authorisasi ini adalah untuk memastikan bahwa data atau resource diakses oleh pengguna yang berhak, sebagai contoh tidak semua yang memiliki SIM boleh mengakses/melewati jalan tertentu misal jalan satu arah, contoh lain adalah, jika kita memiliki tiket pertunjukan kelas festival, maka kita hanya diperbolehkan menempati tempat duduk kelas festival dan tidak diperbolehkan menempati tempat duduk kelas VIP.

4.1. Enkripsi, Encode, dan Hash

Sebelum membahas lebih jauh mengenai authentikasi dan authorisasi ada baiknya Anda memahami istilah enkripsi, encode, dan hash, kenapa? karena pada bagian berikutnya atau dalam praktik Anda akan sering bertemu dengan ketiga istilah tersebut.

Pada prinsipnya enkripsi, encode, dan hash sama-sama digunakan untuk mengubah data/string menjadi karakter acak, bedanya pada enkripsi dan encode karakter acak tersebut dapat dikembalikan ke data/string awal sedangkan hash tidak, sehingga sekali data/string dihash maka tidak mungkin diketahui string asalnya.

4.1.1. Enkripsi

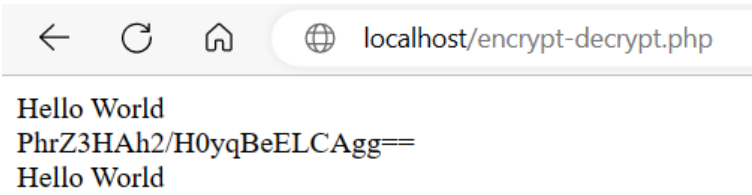
Pada enkripsi untuk menghasilkan karakter acak diperlukan key, key ini nantinya juga digunakan untuk mengembalikan karakter acak ke data awal (decrypt), dengan demikian hanya yang melakukan enkripsi/pemilik key yang dapat membaca isi dari karakter acak tersebut.

Proses enkripsi dapat dilakukan menggunakan berbagai macam algoritma, salah satunya adalah algoritma yang "military grade" seperti AES256, dengan algoritma ini hasil enkripsi ini dapat dipastikan sangat aman.

Contoh penerapan enkripsi pada PHP adalah sebagai berikut:

```
$text = 'Hello World';
$key = '31248275cc4b34882e03ad1af7d38bf5';
$iv = '5fe343def475f751';

$encrypt = openssl_encrypt($text, 'AES-256-CBC', $key, 0, $iv);
$decrypt = openssl_decrypt($encrypt, 'AES-256-CBC', $key, 0, $iv);

echo $text . '<br/>';
echo $encrypt . '<br/>';
echo $decrypt . '<br/>';
```

Ketika kita jalankan pada browser, hasil yang kita peroleh adalah sebagai berikut:



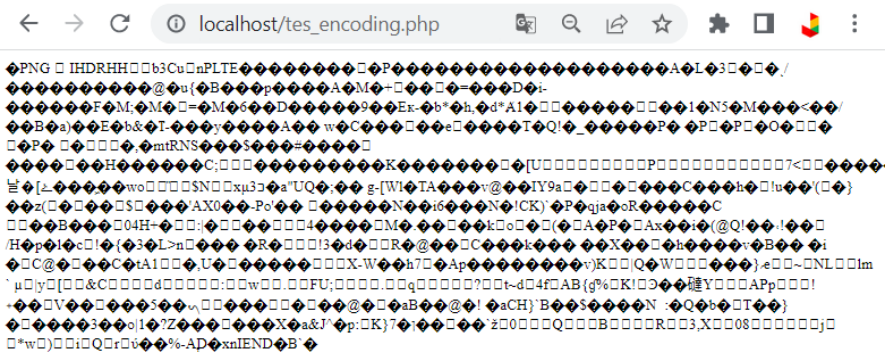
4.1.2. Encode

Berbeda dengan enkripsi, pada encode untuk menghasilkan karakter acak tidak memerlukan key, tujuan utama dari encode ini adalah untuk mengubah data binary menjadi karakter ASCII sehingga dengan mudah dapat dikirim melalui jaringan/network atau disimpan pada database.

Algoritma yang umum digunakan untuk melakukan encoding adalah base64, sebagai contoh kita mengambil data image sebagai berikut:

```
$image = file_get_contents('favicon.png');
echo $image;
```

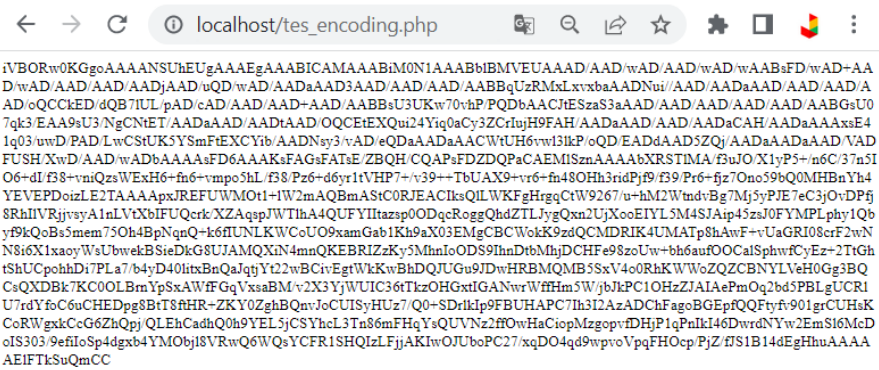
Ketika dijalankan pada browser hasil yang kita peroleh adalah data binary sebagai berikut:



Data binary ini tentu saja tidak bisa dikirim melalui network maupun disimpan pada database, untuk itu kita perlu mengubahnya menjadi data string seperti ASCII menggunakan encode base64, sebagai contoh:

```
$image = file_get_contents('favicon.png');
$encode = base64_encode($image);
echo $encode;
```

Jika dijalankan pada browser, hasil yang kita peroleh adalah sebagai berikut:



Jika diperlukan, data hasil encode ini dapat dikembalikan ke bentuk asal (didecode), misal:

```
$image = file_get_contents('favicon.png');
$encode = base64_encode($image);
$decode = base64_decode($encode);
file_put_contents('favicon_new.png', $decode);
```

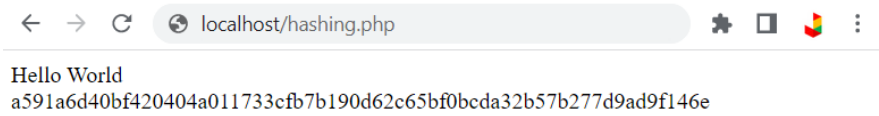
Ketika dijalankan maka akan terbentuk file baru dengan nama favicon_new.png

4.1.3. Hash

Pada hash, karakter acak yang dihasilkan tidak dapat dikembalikan ke bentuk awal, metode ini biasanya digunakan ketika menyimpan password pada database, hashing ini dapat dilakukan dengan berbagai algoritma diantaranya SHA-256 yang bersifat cryptographic sehingga sangat aman dan sangat sulit ditebak, contoh:

```
$text = 'Hello World';
$hash = hash('sha256', $text);
echo $text . '<br/>';
echo $hash;
```

Hasil yang diperoleh ketika dijalankan pada browser adalah sebagai berikut:



4.2. Jenis Authentikasi

Saat ini terdapat banyak jenis authentikasi, seperti authentikasi berbasis password, authentikasi berbasis biometrik (sidik jari/pengenal wajah), authentikasi berbasis token, passwordless authentication, dll. Terkait Aplikasi API yang kita kembangkan, terdapat dua jenis authentikasi yang perlu kita ketahui yaitu:

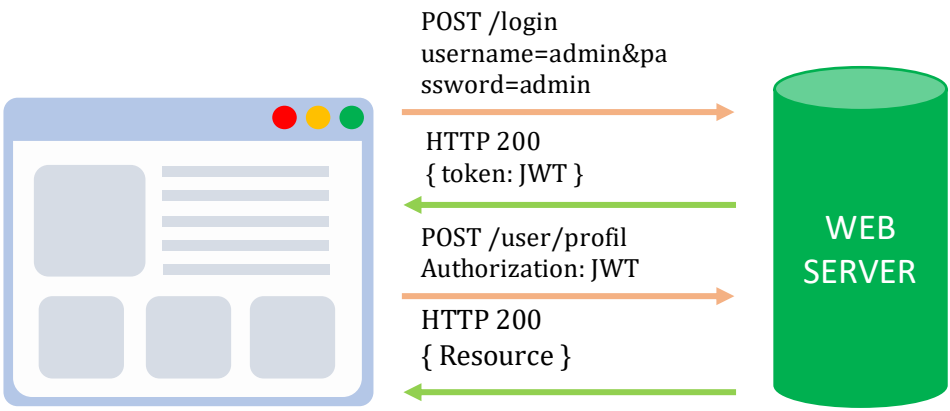
1. **Authentikasi Berbasis Password**. Metode ini merupakan metode konvensional dimana client mengirim username dan password ke server, meskipun konvensional, hingga saat ini metode ini merupakan metode yang paling banyak digunakan. Metode ini biasanya digunakan pada form login baik pada aplikasi desktop maupun mobile dimana ketika ingin login ke aplikasi, kita diminta untuk mengisi username/email dan password.

   Dalam konteks API, penggunaan metode ini pun juga banyak digunakan karena implementasinya yang mudah dan simpel, dengan protokol HTTPS penggunaan metode ini menjadi lebih aman karena data username dan password yang dikirim (dari client (browser) ke server) terenkripsi.

2. **Authentikasi Berbasis Token**, Token-Based Authentication (TBA). Pada metode ini, proses authentikasi dilakukan menggunakan token, token sendiri dapat didefinisikan sebagai segala string yang dapat digunakan untuk mengidentifikasi user.

   Pada metode ini, alur yang digunakan adalah: pertama tama user login ke server menggunakan username dan password (menggunakan HTTP POST), setelah berhasil login, maka server akan menggenerate token untuk dikirim ke user, pada request berikutnya user

menggunakan token tersebut untuk mengakses resource pada server, tanpa perlu mengirim ulang username dan password. Ketika server menerima token, server akan mengidentifikasi user berdasarkan token tersebut, ilustrasinya adalah sebagai berikut:



Contoh implementasi TBA ini bisa dibaca dihalaman "Token Based Authentication - Implementation Demonstration" yang bisa diakses melalui tautan https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/

4.3. Token Based Authentication and Authorization

Ketika membangun Aplikasi API kita dihadapkan banyak pilihan metode Authentikasi dan Authorisasi, diantaranya seperti yang telah disebutkan sebelumnya yaitu menggunakan username password dan token. Dari berbagai pilihan yang ada, pilihan yang paling banyak digunakan adalah menggunakan token (Token Based Authentication) karena lebih aman dan lebih fleksibel.

Pada bagian ini kita akan membahas berbagai teknik token based authentication dan authorization yang saat ini banyak digunakan untuk mengembangkan Aplikasi Api sehingga Anda akan memiliki gambaran dan referensi yang utuh mengenai implementasi TBA, dengan demikian ketika

nantinya menerapkan TBA ini, Anda lebih yakin dan mantab dengan metode yang dipilih.

4.3.1. OAuth

Ketika berbicara tentang authentikasi dan authorisasi, maka tidak terlepas dari istilah OAuth.

Apa itu OAuth?

OAuth merupakan framework yang digunakan untuk proses authorisasi, hal ini sebagaimana disebutkan pada judul standar yang mendefinisikan OAuth yaitu "The OAuth 2.0 Authorization Framework", yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc6749. Pada standar tersebut dijelaskan secara lengkap dan terperinci bagaimana menggunakan OAuth.

Note: Pada standar OAuth diatas, baik pada judul maupun isi, disebutkan bahwa OAuth ini merupakan protokol authorization bukan protokol authentication, sehingga tujuan OAuth ini adalah untuk memberikan akses pada resource tertentu, bukan untuk melakukan authentikasi.

Contoh penggunaan OAuth adalah ketika kita login ke suatu aplikasi menggunakan akun dari aplikasi lain. Sebagai contoh ketika kita ingin login ke aplikasi Facebook menggunakan akun Google, maka kita terlebih dahulu diminta untuk login ke akun Google, setelah berhasil login, maka Google akan mengirim token (ID Token) ke Facebook yang berisi informasi profil user. ID Token ini sebagai bukti bahwa user telah melalui proses authentikasi. Pada proses ini Google tidak mengirim credentials (username dan password) kita ke Facebook, melainkan hanya mengirim token, token inipun tidak berisi credential kita, hanya berisi informasi profil user.

Pada contoh diatas, selain ID Token, Google juga dapat mengirim access token dan refersh token. Access token ini nantinya digunakan Facebook untuk mengakses lebih jauh data user pada Google, seperti data calendar, email, dll tergantung apa yang ingin diintegrasikan user ke aplikasi Facebook, misal jika user ingin mensinkronisasikan data Google Calendar

pada Facebook Events, maka Facebook akan mengakses data Google Calendar user menggunakan access token yang telah diberikan.

Token yang digunakan dalam OAuth dapat bermacam macam, bisa menggunakan "opaque" token, opaque berarti karakter acak yang tidak memiliki arti yang hanya dimengerti oleh sistem OAuth, bisa juga menggunakan token yang terstruktur seperti Json Web Token (JWT), OAuth sendiri tidak mensyaratkan token harus opaque atau JWT.

4.3.2. ID Token

ID Token merupakan token yang digunakan untuk proses authentikasi aplikasi client ke server, sebagaimana dicontohkan sebelumnya ketika kita login ke aplikasi Facebook dengan akun Google, Google akan mengirim ID Token ke Facebook.

ID Token ini diperkenalkan oleh OpenID Connect (OIDC), spesifikasi dan deskripsi lengkapnya dapat dibaca pada halaman "OpenID Connect Core 1.0 incorporating errata set 1" yang dapat diakses melalui tautan https://openid.net/specs/openid-connect-core-1_0.html, dalam spesifikasi tersebut disebutkan bahwa ID Token ini dibangun menggunakan framework OAuth.

ID Token ini berbentuk Json Web Token (JWT – dibahas kemudian), dengan bentuk JWT maka kita dapat menginspect data yang ada pada ID Token (Payload), ID token biasanya berisi informasi personal user seperti nama, email, nomor telepon, dll yang tersimpan di server, contoh ID Token adalah sebagai berikut:

```
eyJraWQiOiIxZTlnZGs3IiwiYWxnIjoiUlMyNTYifQ.ewogImlzcyI6ICJodHRwOi8
vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5NzYxMDAxIiwKICJhd
WQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZfV3pBMk1qIiwKICJleHA
iOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5NzAsCiAibmFtZSI6ICJKYW5lI
ERvZSIsCiAiZ2l2ZW5fbmFtZSI6ICJKYW5lIiwKICJmYW1pbHlfbmFtZSI6ICJEb2U
iLAogImdlbmRlciI6ICJmZW1hbGUiLAogImJpcnRoZGF0ZSI6ICIwMDAwLTEwLTMxI
iwKICJlbWFpbCI6ICJqYW5lZG9lQGV4YW1wbGUuY29tIiwKICJwaWN0dXJlIjogImh
0dHA6Ly9leGFtcGxlLmNvbS9qYW5lZG9lL21lLmpwZyIKfQ.rHQjEmBqn9Jre0OLyk
YNnspA10Qql2rvx4FsD00jwlB0Sym4NzpgvPKsDjn_wMkHxcp6CilPcoKrWHcipR2i
AjzLvDNAReF97zoJqq880ZD1bwY82JDauCXELVR9O6_B0w3K-
E7yM2macAAgNCUwtik6SjoSUZRcf-
```

```
O5lygIyLENx882p6MtmwaL1hd6qn5RZOQ0TLrOYu0532g9Exxcm-
ChymrB4xLykpDj3lUivJt63eEGGN6DH5K6o33TcxkIjNrCD4XB1CKKumZvCedgHHF3
IAK4dVEDSUoGlH9z4pP_eWYNXvqQOjGs-
rDaQzUHl6cQQWNiDpWOl_lxXjQEvQ&state=af0ifjsldkj
```

Sumber: https://openid.net/specs/openid-connect-core-1_0.html#id_tokenExample

Pada token diatas, bagian Payload berisi claims sebagai berikut:

```
{
  "iss": "http://server.example.com",
  "sub": "248289761001",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "gender": "female",
  "birthdate": "0000-10-31",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

Note: Json Web Token (payload, claims, dll) akan dibahas di bagian sub bab 4.3. Json Web Token.

Dari claims diatas, claims yang paling penting adalah claims “aud". Claims ini menunjukkan identias aplikasi yang menggunakan token (pada contoh sebelumnya, ID Token yang dikirim Google berisi claims aud yang menandai aplikasi Facebook sebagai pengguna token), dengan claims aud ini, nantinya server akan melakukan pengecekan apakah aplikasi berhak menggunakan token tersebut, jika tidak berhak maka akses akan ditolak.

Dengan claims yang ada pada ID Token, aplikasi (pada contoh sebelumnya Facebook) dapat menampilkan data user seperti nama user, email, profile picture, dll sesuai dengan data yang disertakan pada claims.

ID Token ini digunakan oleh berbagai perusahaan besar, diantaranya Google: https://developers.google.com/identity/openid-connect/openid-

connect#obtainuserinfo, Meta (Facebook), dan Microsoft: https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-protocols-oidc

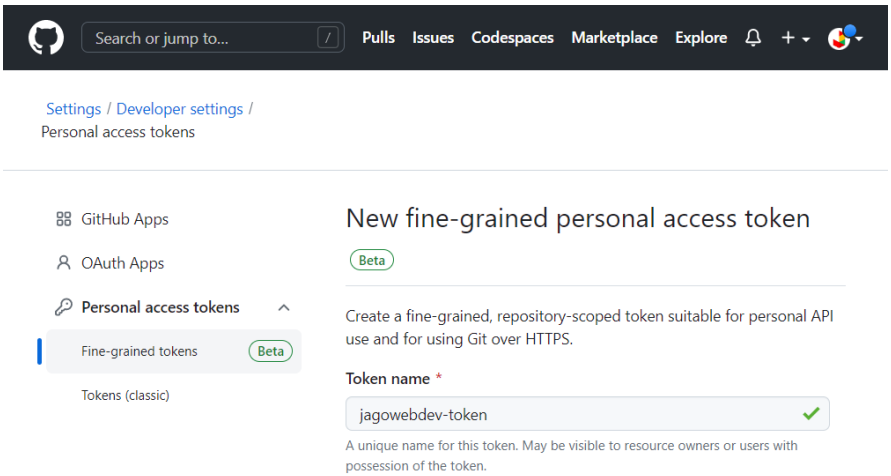
4.3.3. Access Token

Access Token (Akses Token) merupakan token yang digunakan untuk mengakses resource tertentu yang terproteksi pada server seperti contoh pada bagian sebelumnya dimana Facebook mengakses data Google Calendar menggunakan Access Token.

Access token ini telah didefinisikan pada standar OAuth 2.0 "The OAuth 2.0 Authorization Framework" section 1.4, standar ini didokumentasikan pada RFC 6749 yang dapat diakses melalui tautan https://datatracker.ietf.org/doc/html/rfc6749#section-1.4, dengan adanya standar ini maka seharusnya penggunaan Access Token ini memenuhi standar yang telah didefinisikan tersebut.

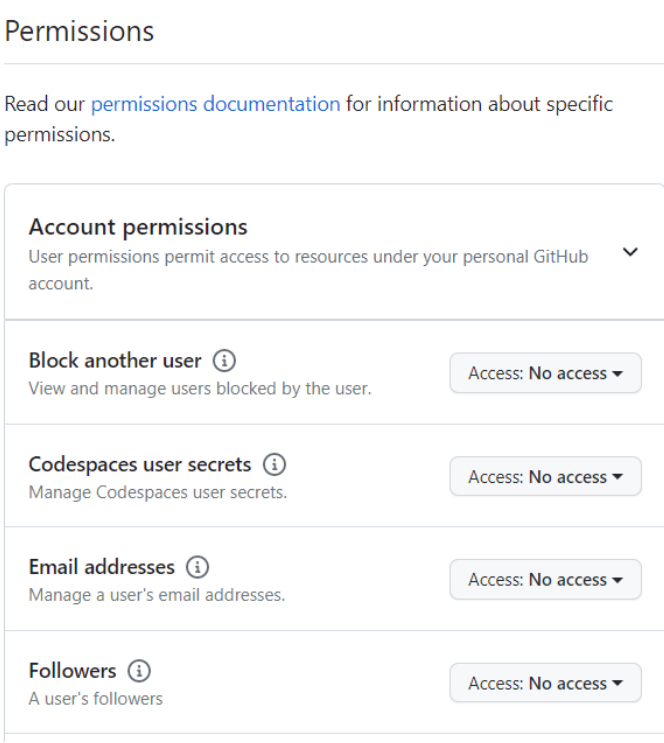
Access token biasanya berusia pendek seperti 15 atau 30 menit, namun demikian hal ini tidak disebutkan di standar, hanya disebutkan bahwa access token usianya lebih pendek dari refresh token. Usia access token yang pendek ini bertujuan untuk meningkatkan keamanan, karena jika access token ini jatuh ke tangan yang tidak berhak, maka mereka hanya memiliki waktu yang pendek sebelum token ini expired.

Access Token ini biasanya berisi informasi yang yang dapat digunakan oleh Server API untuk mengidentifikasi aksi apa yang bisa dilakukan oleh pengguna token, apakah membaca, menambah, mengubah, atau menghapus resource (informasi aksi ini bisa disertakan didalam token atau disimpan pada database). Sebagai contoh, berikut ini proses membuat Access Token pada Github



Sumber: https://github.com/settings/personal-access-tokens/new

Pada halaman tersebut diatas, kita dapat memilih permission (aksi) yang dapat dilakukan pengguna token sebagai berikut:



Nantinya pengguna token akan dapat melakukan aksi pada resource sesuai dengan yang telah didefinisikan diatas.

Access token ini dapat berupa (1) string acak (opaque), string acak ini nantinya digunakan untuk mengambil informasi authorisasi (permission yang dimiliki) user dari database, (2) string yang terstruktur, seperti JWT, yang didalamnya berisi informasi authorisasi, dengan menyimpan informasi authorisasi pada token, aplikasi tidak perlu melakukan query pada database.

Penggunaan token opaque dan JWT ini sesuai dengan yang dijelaskan pada standar sebagai berikut:

"*The token may denote an identifier used to* ***retrieve the authorization information*** *or may* ***self-contain*** *the authorization information in*

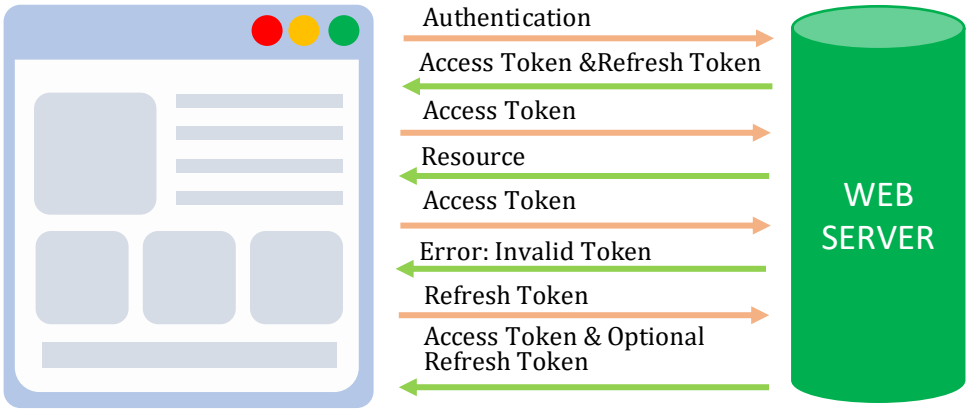
verifiable manner (i.e., a token string consisting of some data and a signature)."

Pada contoh Github diatas, Github menggunakan opaque token sebagai access token, selanjutnya informasi permission user diambil dari database. Contoh lain adalah Google, Google juga menggunakan opaque token (https://cloud.google.com/docs/authentication/token-types#access-contents), namun demikian meski digunakan oleh perusahaan besar, Access Token tidak harus berbentuk opaque, bisa juga JWT.

4.3.4. Refresh Token

Refresh token merupakan token yang digunakan untuk mendapatkan akses token. Refresh token diterbitkan oleh server api ketika access token expired atau invalid. Refresh token ini bersifat opsional, jika refresh token diterbitkan maka pengirimannya dilakukan bersamaan dengan pengiriman access token.

Adapun alur refresh token adalah sebagai berikut:



Pada alur diatas pertama tama client mengirim credentials ke server untuk keperluan authentication, jika berhasil maka server mengirim access token dan refresh token ke client, selanjutnya jika client mengirim request ke server dan mendapat pesan error bahwa token invalid maka client akan mengirim refresh token, selanjutnya server akan menggenerate dan mengirim access token yang baru dan opsional refresh token yang baru,

jika server tidak mengirim refresh token yang baru, maka client akan menggunakan refresh token yang lama.

Refresh token ini didokumentasikan pada standar "OAuth 2.0 Authorization Framework" yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc6749#page-10

4.3.5. API Key

Api Key merupakan kode unik yang dapat digunakan untuk melakukan authentikasi maupun authorisasi pengguna/user atau aplikasi. API Key hanya berupa karakter acak (opaque) yang tidak memiliki arti, selain itu juga tidak ada informasi yang tersimpan pada key. API Key ini bisa diibaratkan semacam ID yang tidak memiliki arti. Untuk mendapatkan API Key ini biasanya user login terlebih dahulu ke portal server kemudian menggenerate API Key.

Pada awal berkembangnya web API, API Key ini sangat populer digunakan untuk proses authentikasi dan authorisasi, hal ini dikarenakan penerapannya yang simpel dan mudah, saat ini sudah banyak alternatif yang lebih baik sehingga penggunaan API Key untuk proses authentikasi dan authorisasi sudah mulai ditinggalkan, meskipun demikian masih banyak perusahaan besar yang masih menggunakan API Key diantaranya Stripe, Twillo, dan SendGrid.

API Key ini digunakan ketika aplikasi client ingin mengakses resource server, biasanya api key disertakan dalam query string url, misal:

```
https://jagowebdev.com/api/user/1/profile?api_key=bda6f8b71e10d818
```

atau pada HTTP header:

```
GET https://jagowebdev.com/api/user/1/profile HTTP/1.1
X-API-KEY:  bda6f8b71e10d818
```

API Key ini tidak terstandar sehingga bentuknya tidak seragam, ada yang menggunakan hasing saja, ada yang menggunakan enkripsi, dll, selain itu pengimplementasian pada aplikasi juga bermacam macam.

Authentikasi dengan API Key

Penerapan authentikasi dengan API Key adalah dengan menyimpan API key pada database yang merujuk pada user tertentu, nantinya ketika API Key ini digunakan untuk mengakses apikasi API maka aplikasi akan melakukan pengecekan user pada database berdasarkan API Key tersebut.

Pada database Api Key disimpan dalam bentuk terenkripsi atau dalam bentuk yang sudah dihash, dengan bentuk seperti ini akan membuat API Key menjadi lebih aman karena jika database jatuh ke tangan yang tidak berhak mereka tidak akan tahu API Key yang sebenarnya.

Api key yang digunakan client tidak terenkripsi juga tidak memiliki masa expired, sehingga siapapun yang memiliki api key ini dapat langsung menggunakannya tanpa batasan waktu, hal ini membuat API Key tidak aman, sehingga agar lebih aman, perlu dilakukan beberapa tambahan pengamanan pada API Key. Karakteristik API Key yang tidak aman ini membuat sebagain vendor hanya menggunakan API Key untuk membaca data, tidak untuk mengubah atau menghapus data.

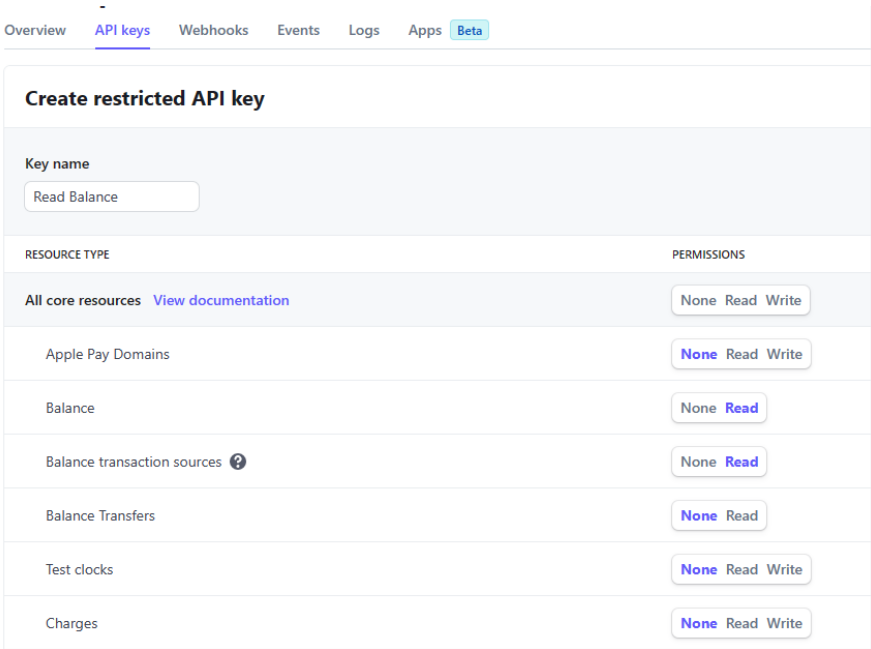
Pengamanan Api Key dapat dilakukan diantaranya menggunakan protokol HTTPS pada server, dengan HTTPS ini maka semua data yang dikirim dari client ke server terenkripsi, termasuk data api key ini, selain itu perlu ditambahkan juga informasi terkait API Key ini (informasi ini bisa disimpan di database), seperti tanggal API Key dibuat, waktu expired nya, dll, hal ini dapat mempermudah ketika ada pergantian key/rolling key.

Dengan berbagai tambahan ini, maka akan menambah beban pada server untuk melakukan pengecekan, selain itu, jika jumlah API Key banyak, maka maintenance nya akan menjadi rumit.

Authorisasi dengan API Key

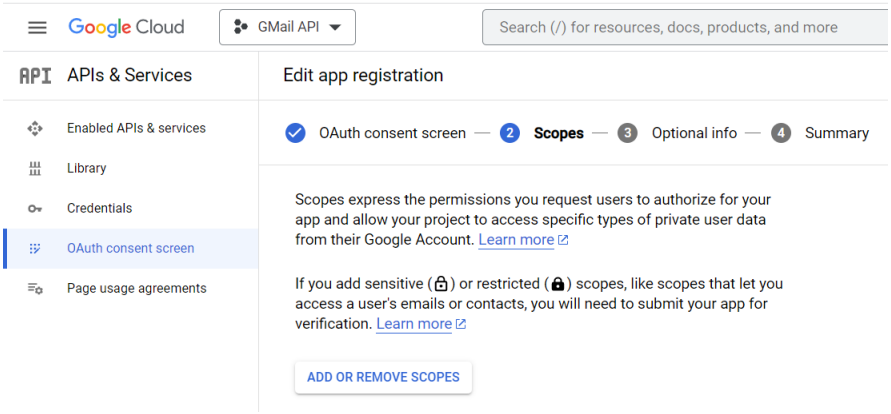
Ketika user menggenerate/membuat API Key, biasanya aplikasi api memberikan opsi resource apa saja yang bisa diakses dan aksi apa saja

yang bisa dilakukan dengan API Key tersebut, opsi yang dipilih ini nantinya disimpan pada database, sebagai contoh berikut ini opsi pada aplikasi Stripe ketika user membuat API Key:



Google

API Key oleh Google digunakan untuk mengidentifikasi project/aplikasi client bukan untuk mengidentifikasi user dan oleh Google, API Key ini digunakan hanya untuk melakukan monitoring request, seperti mengetahui jumlah akses (quota) ke resource, membatasi limit request, membatasi alamat IP yang diperbolehkan mengakses resource, mengecek trafik dari API Key, dll, sedangkan untuk scope akses yang dapat dilakukan user diatur menggunakan OAuth, berikut contoh halaman Scopes pada Google:



Dengan menggunakan API Key sebagai identifikasi project/aplikasi, akan memungkinkan satu user memiliki tiga api key yang berbeda untuk satu alamat API, yaiitu API Key untuk aplikasi desktop, API Key untuk aplikasi Android, dan API Key untuk aplikasi IOS.

4.3.6. Json Web Token

Json Web Token (JWT) adalah token yang terstruktur yang berisi informasi tertentu yang dibutuhkan oleh server, sehingga isi dari JWT ini bermacam macam sesuai dengan kebutuhan server, hal ini membuat ukuran token ini bisa bervariasi tergantung banyakanya informasi (claims) yang disimpan pada token.

JWT ini tidak terikat pada token tertentu, penggunaannya bisa bermacam macam, pada penjelasan sebelumnya, JWT bisa digunakan untuk access token, refresh token, ID token, dll.

4.3. Json Web Token (JWT)

Dari penjelasan sebelumnya, dapat kita simpulkan bahwa terdapat dua jenis token yang dapat digunakan untuk melakukan authentikasi dan authorisasi, yaitu opaque token (token berupa karakter acak) dan JWT

token. Pada aplikasi API yang kita kembangkan, kita menggunakan JWT token untuk keperluan authentikasi dan authorisasi.

4.3.1. Kelebihan JWT

Berikut ini beberapa kelebihan menggunakan token JWT:

1. Industri Standar, didokumentasikan pada Standar RFC 7519 (https://www.rfc-editor.org/rfc/rfc7519).
2. Dukungan yang luas. JWT didukung oleh banyak bahasa pemrograman, terdapat banyak sekali framework dan library dari berbagai bahasa pemrograman yang dapat memudahkan kita bekerja dengan JWT, daftar lengkapnya dapat dibaca di halaman "JSON Web Token Libraries" yang dapat diakses melalui tautan https://jwt.io/libraries.
3. Aman. JWT memiliki signature, dengan signature ini akan memastikan validitas JWT. Signature ini dienkripsi dengan metode enkripsi yang terbukti sangat aman, sehingga token akan tetap aman ketika disimpan di sisi client.
4. Lintas platform. JWT dapat diimplementasikan lintas platform baik web, mobile, maupun desktop.
5. Stateles. Dengan JWT informasi client disimpan pada token, server tidak menyimpan informasi apapun terkait client, baik data maupun request yang dilakukan client sehingga beban server menjadi lebih ringan.
6. Efisien. Ukuran token JWT yang kecil akan (1) memudahkan penyimpanan token, misal menyimpan token pada cookie, menempelkan token pada dokumen HTML, dll (2) memudahkan pengiriman token, misal, mengirim token melalui HTTP header, melalui HTTP Body, melalui URI, dll.

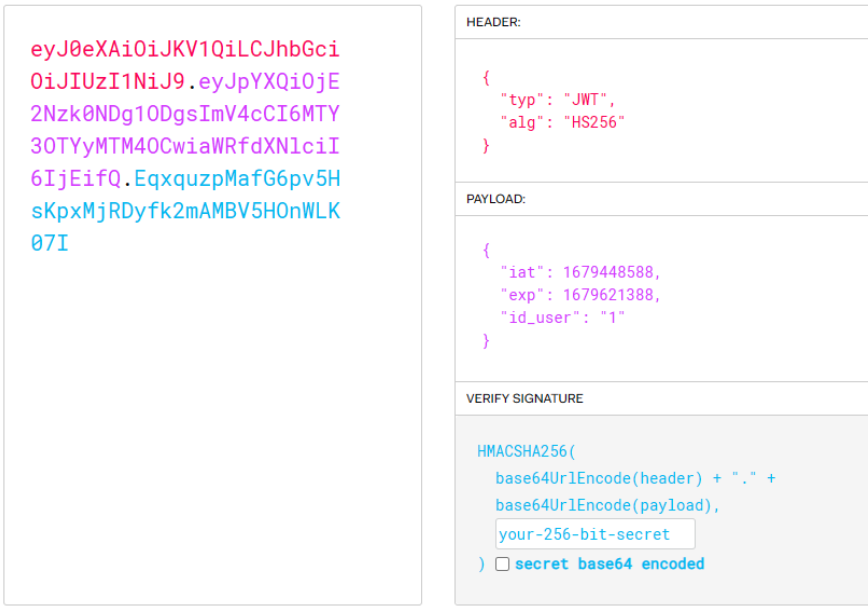
4.3.2. Struktur Token JWT

Token JWT memiliki struktur standar yaitu header, payload, dan signature, ketiganya dipisahkan dengan tanda titk (.) contoh sebagai berikut:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2Nzk0NDg1ODgsImV4
cCI6MTY3OTYyMTM4OCwiaWRfdXNlciI6IjEifQ.WPEeF3kv2RSA83Wyq5VbiCCcHy
CGgt_zcQ-TH-HtVAU
```

Note: pada token diatas, dengan dipisah karakter titik (.), bagian pertama adalah header, kedua adalah payload, dan terakhir ada signature.

Jika token diatas didecode (misal menggunakan https://jwt.io) maka hasil yang kita peroleh adalah sebagai berikut:



Header

Header merupakan bagian pertama token. Bagian header ini berisi dua komponen yaitu tipe token dan algoritma yang digunakan pada signature, tipe token selalu bernilai "JWT", sedangkan algoritma yang digunakan bisa

bermacam macam, bisa RS256 atau RSA, HS256 atau HMAC, atau yang lain. Contoh header seperti tampak pada gambar diatas:

```
{
      "typ" : "JWT",
      "alg" : "HS256"
}
```

Payload

Bagian kedua token adalah payload. Bagian payload ini berisi data/informasi dari token, data/informasi ini disebut "claims", hal ini sesuai dengan yang di sebutkan pada standar "JSON web Token" section 4 yang bisa diakses melalui tautan https://www.rfc-editor.org/rfc/rfc7519#section-4

Claims sendiri dibagi menjadi tiga yaitu: registerd, public, dan private claims.

- Registered claims merupakan predefined claims, yaitu claims yang terdaftar (registered) sebagai standar claims, contoh iat, iss, exp, dll, list lengkap registered claims dapat dibaca di halaman RFC 7519: JSON Web Token (JWT) yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc7519#page-9. Penggunaan registered claims ini harus dimaksudkan sesuai dengan standar tersebut.

  Kenapa kita perlu mengikuti standar ini? Karena selain best practice juga akan memudahkan komunikasi kita dengan pengembang lain maupun code kita dengan code orang lain, lebih spesifiknya dengan menerapkan registered claims, pengembang lain yang membaca code kita langsung tahu maksud dari claims tersebut, selain itu jika kita menggunakan aplikasi/library untuk mengelola jwt maka library tersebut akan langsung memperlalukan registered claims tersebut sesuai dengan standar yang ada.

- Public claims merupakan claims yang tidak ada pada standard tapi terdaftar di spesifikasi JSON Web Token yang di didokumentasikan

oleh IANA yang dapat diakses melalui tautan https://www.iana.org/assignments/jwt/jwt.xhtml#claims. Seperti registered claims, penggunaan claims ini seharusnya dimaksudkan sesuai dengan yang ada pada spesifikasi tersebut. Sebagai contoh ketika kita menggunakan claims "profile" maka seharusnya berisi url halaman profile.

- Private claims merupakan claims yang tidak ada pada registered claims maiupun public claims. Claims ini digunakan untuk keperluan internal yang telah disepakati oleh pengguna aplikasi, contohnya adalah claims id_user.

Payload tidak harus mengandung claims registered maupun claims public, kita bebas menggunakannya sesuai kebutuhan. Pada aplikasi api yang kita kembangkan, claims yang kita gunakan adalah:

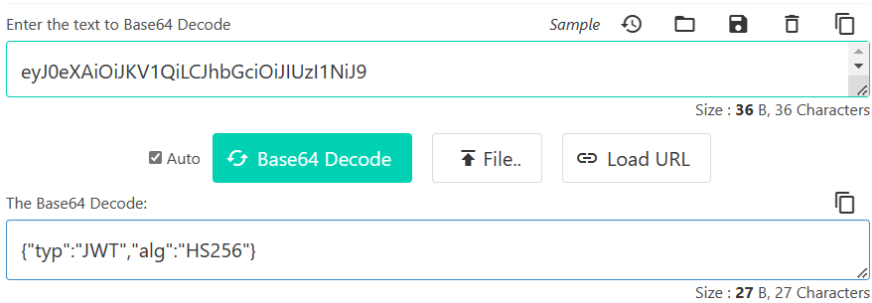
- iat (claims registered). iat merupakan kependekan dari issued at, yang artinya kapan token diterbitkan.
- exp (claims registered). exp berarti informasi waktu kapan token expired.
- id_user (claims private). Id user ini berisi id yang merujuk pada user tertentu.

Signature.

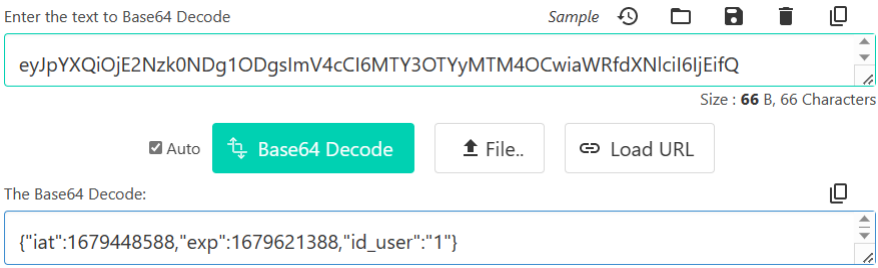
Signature ini merupakan string yang digunakan untuk memvalidasi token JWT. Bagian header dan payload token JWT tidak dienkripsi hanya di encode dengan base64, sehingga sebenarnya kita dapat dengan mudah membaca isi dari header dan payload token, misal pada contoh token sebelumnya:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2Nzk0NDg1ODgsImV4
cCI6MTY3OTYyMTM4OCwiaWRfdXNlciI6IjEifQ.WPEeF3kv2RSA83Wyq5VbiCCcHy
CGgt_zcQ-TH-HtVAU
```

Pada token diatas jika bagian header kita decode menggunakan base64 decoder: https://codebeautify.org/base64-decode maka hasil yang kita peroleh adalah sebagai berikut:



Demikian juga pada bagian payload



untuk itu, untuk memastikan bahwa token tersebut merupakan token yang kita terbitkan maka kita gunakan signature untuk memvalidasi token.

Signature ini tidak diencode, melainkan dienkripsi, algoritma yang digunakan untuk mengenkripsi sesuai dengan yang didefinisikan pada header yang pada contoh diatas adalah HS256. Untuk mengenkripsi signature ini diperlukan key, key tersebut nantinya juga digunakan untuk mendecrypt token, sehingga penting bagi kita untuk membuat key ini seaman mungkin baik bentuk key nya yang sulit ditebak maupun tempat penyimpanannya.

4.3.3. Karakteristik JWT

Berdasarkan penjelasan diatas, JWT memiliki karakteristik sebagai berikut:

1. **Stateles** yang artinya server tidak menyimpan informasi tentang client, semua informasi disediakan oleh token.
2. **Tidak terenkripsi namun aman.** Sebagaimana dibahas pada bagian sebelumnya bahwa pada Token JWT, yang terenkripsi hanya bagian signature saja, sehingga siapa saja yang memiliki token dapat dengan mudah melihat apa isi dari token tersebut (payload), namun demikian dengan adanya signature, membuat token ini manjadi aman.
3. **Fleksibel.** JWT dapat diisi berbagai macam data/claims sesuai dengan kebutuhan.
4. **Efisien.** Ukuran JWT yang kecil membuatnya efisian untuk di kirim melalui network, selain itu penggunaan JSON sebagai format data membuatnya mudah untuk di parse dan telah didukung oleh berbagai bahasa pemrograman.

4.3.4. JWT Secret Key

Pada pembahasan sebelumnya telah kita bahas pentingnya secret key pada JWT untuk mengenkripsi signature. Untuk menggenrate key yang aman, php telah menyediakan fungsi untuk menggenerate karakter acak secara cryptographic, yaitu menggunakan fungsi random_bytes(), agar lebih aman, jumlah karakter acak ini sebaiknya minimal 16 digit.

Fungsi random_bytes() ini menghasilkan digit binary sehingga tidak bisa dicetak, untuk itu kita perlu mengubahkannya menjadi string, misal ke bentuk hexadesimal. PHP sendiri telah menyediakan fungsi untuk mengubah binary ke hexadesimal yaitu menggunakan fungsi bin2hex().

Contoh script PHP untuk menggenate key adalah sebagai berikut:

```
echo bin2hex(random_bytes(32));
```

Contoh hasilnya adalah sebagai berikut:

```
0d383aa0129cebb6f6f5ef5c3b73968da2c6ccb52cf0e7e4225835df1dc66eba
```

4.4 Skema Authentikasi

Setelah kita memahami jenis-jenis authentikasi, selanjutnya mari kita bahas mengenai skema authentikasi. Saat ini terdapat berbagai jenis skema authentikasi, skema authentikasi yang dapat digunakan untuk jenis authentikasi berbasis password (password based authentication) dan authentikasi berbasis token (token based authentication) adalah skema authentikasi basic, digest, dan bearer.

4.4.1. Basic

Pada skema authentikasi basic, proses authentikasi dilakukan dengan cara client mengirim data credentials (username dan password) ke server, format data credentials yang dikirim adalah username:password yang diencode dengan algoritma base64, data credentials ini dikirim melalui http header Authorization, contoh sebagai berikut:

```
GET /api/dashboard/penjualan/terbaru?tahun=2023 HTTP/1.1
Host: https://jagowebdev.com
Authorization: Basic YWRtaW46YWRtaW4=
```

Pada contoh diatas string `YWRtaW46YWRtaW4=` merupakan string admin:admin (username admin, password admin) yang diencode base64, nantinya ketika server menerima token tersebut, server akan men-*decode* token dan melakukan pengecekan username dan password pada database.

Penerapan skema ini salah satunya adalah client menyediakan form login kepada user, setelah user mengisi username dan password dan mensubmit form, selanjutnya client akan mengirim data username dan password tersebut menggunakan format username:password yang diencode base64.

Skema authentikasi ini aman jika digunakan pada protokol HTTPS, hal ini dikarenakan username dan password dikirim apa adanya (dapat dengan

mudah di decode) sehingga bisa jadi ditengah jalan data tersebut diambil oleh orang yang tidak berhak (Man in the middle attack - MITM), hal ini seperti yang disebutkan pada standar skema authentikasi basic "The Basic HTTP Authentication Scheme".

"*This scheme is not considered to be a secure method of user authentication* ***unless used in conjunction with some external secure system such as TLS*** *(Transport Layer Security, [RFC5246]),* ***as the user-id and password are passed over the network as cleartext.***"

Standar ini dapat diakses melalui tautan: https://www.rfc-editor.org/rfc/rfc7617

Meskipun termasuk skema authentikasi yang konvensional, namun hingga saat ini sekma ini merupakan skema yang paling banyak digunakan karena penerapannya yang simpel dan mudah, sebagai contoh form login pada website atau aplikasi mobile, pada form login tersebut data username dan password dikirim ke server apa adanya melalui HTTP Body menggunakan method POST, meskipun dikirim pada bagian body HTTP Request, tidak di pada bagian header, namun pada prinsipnya sama, username dan password dikirim apa adanya.

Dalam konteks API, ketika menggunakan skema ini maka setiap kali melakukan request ke server, aplikasi client akan mengirim username dan password user, hal ini tentu akan merepotkan jika user harus memasukkan username dan password, alternatif lain adalah username dan password disimpan di aplikasi client, seperti cookie, local storage, dll, sehingga setiap kali melakukan request, username dan password ini otomatis langsung dikirim ke server, meskipun lebih praktis namun cara ini sangat riskan, username dan password rawan terekpose.

4.4.2. Digest

Skema authentikasi berikutnya adalah Digest. Pada skema Digest ini, pertama tama aplikasi client mengirim request authentikasi ke server, selanjutnya server memberikan response dengan mengirim nonce (karakter acak unik sekali pakai) ke client, nonce ini berfungsi untuk

mencegah reply attack, nantinya oleh client nonce ini dikirim bersama username dan password user. Contoh request authentikasi pada skema Digest adalah sebagai berikut:

```
GET /api/dashboard/penjualan/terbaru?tahun=2023 HTTP/1.1 401
Unauthorized
WWW-Authenticate: Digest
      realm = "dashboard”
      nonce = "827ccb0eea8a706c4c34a16891f84e7b"
      opaque = "1a18886587c2efa7b720554ff646d482"
```

Note: realm disini dapat diibaratkan suatu area yang di proteksi oleh credentials.

Setelah mengirim request authentikasi, selanjutnya client menampilkan form login ke user, ketika user mengisi username dan password dan mensubmit form, maka client akan mengirim username dan pasword ke server. Pada request yang dikirim, username dan password ini berada pada bagian Authorization Header dan ditulis menggunakan format tertentu, contoh sebagai berikut:

```
Authorization: Digest username="admin"
      realm = "dashboard”
      nonce = "827ccb0eea8a706c4c34a16891f84e7b"
      uri = "/api/dashboard/penjualan/terbaru?tahun=2023"
      response="aca15130233cb2d50b9fc1eae47b9e8d"
      opaque = "1a18886587c2efa7b720554ff646d482"
```

Pada contoh diatas, username dan password berada di dalam parameter response. Parameter response sendiri merupakan hash MD5 dari HA1, nonce, dan HA2, adapun HA1 dan HA2 adalah sebagai berikut:

- HA1 ini merupakan hash MD5 dari username, realm, dan password, adapun formatnya adalah md5(username:realm:password).
- HA2 merupakan hash MD5 dari method dan digestUri. Adapun formatnya adalah: md5(method:digestUri).

Berikut ini contoh membuat parameter response dengan PHP:

```
// $ha1 = md5(username:realm:password);
$ha1 = md5('admin:dashboard:1234');

// $ha1 = md5(request_method:digestURI);
$ha2 = md5('GET:/api/dashboard/penjualan/terbaru?tahun=2023');

// $response = md5(ha1:nonce:ha2)
$response = md5($ha1 . ':827ccb0eea8a706c4c34a16891f84e7b:' .
$ha2);

echo $ha1 . '<br/>';
echo $ha2 . '<br/>';
echo $response;
```

Hasil yang kita peroleh adalah sebagai berikut:

```
7885a4e393efd8bcf26b1b1e7360dbac
8b22f92bff08788136385f5541cdc2bc
aca15130233cb2d50b9fc1eae47b9e8d
```

Jika pada response server menyertakan qop (quality of protection) dengan nilai auth, misal:

```
GET /api/dashboard/penjualan/terbaru?tahun=2023 HTTP/1.1 401
Unauthorized
WWW-Authenticate: Digest
      realm = "dashboard”
      nonce = "827ccb0eea8a706c4c34a16891f84e7b"
      qop: "auth"
      nc="00000003"
      opaque = "1a18886587c2efa7b720554ff646d482"
```

maka susunan response menjadi: HA1, nonce, nc (nonce count), cnonce (client nonce), qop, dan HA2, berikut ini contoh penerapannya pada PHP:

```
// $ha1 = md5(username:realm:password);
$ha1 = md5('admin:dashboard:1234');

// $ha1 = md5(request_method:digestURI);
$ha2 = md5('GET:/api/dashboard/penjualan/terbaru?tahun=2023');

$cnonce = bin2hex(random_bytes(8));

// $response = md5(ha1:nonce:nc:cnonce:qop:ha2)
```

```
$response = md5($ha1 .
':827ccb0eea8a706c4c34a16891f84e7b:00000003:' . $cnonce . ':auth:'
. $ha2);

echo $ha1 . '<br/>';
echo $ha2 . '<br/>';
echo $response;
```

Hasil:

```
7885a4e393efd8bcf26b1b1e7360dbac
8b22f92bff08788136385f5541cdc2bc
89d120b4c9b289c49903770bb443cfd1
```

dengan parameter response diatas, maka bentuk HTTP header yang dikirim client ke server adalah sebagai berikut:

```
Authorization: Digest username="admin"
      realm = "dashboard”
      nonce = "827ccb0eea8a706c4c34a16891f84e7b"
      uri = "/api/dashboard/penjualan/terbaru?tahun=2023"
      qop="auth"
      opaque = "1a18886587c2efa7b720554ff646d482"
      cnonce="f8db26543c1cecd854329c5f78991908"
      nc="00000003"
      response="89d120b4c9b289c49903770bb443cfd1"
```

Pada pembahasan diatas telah disebutkan bahwa pada skema ini client mengirim username dan password ke server, prinsipnya sama seperti skema basic, hanya bentuknya saja yang berbeda.

Dalam konteks API, sama seperti skema basic, setiap kali melakukan request ke server, client harus mengirim username dan password, hal ini tentu akan merepotkan karena user harus memasukkan username dan password, alternatif lain yaitu menyimpan username dan password di aplikasi client, seperti cookie, localstorage, dll, namun cara ini sangat riskan, username dan password rawan terekpose.

4.4.3. Bearer

Bearer merupakan skema authentikasi berbasis token yang digunakan untuk mengakses resource yang terproteksi. Token pada bearer ini biasanya diberikan oleh aplikasi API ketika client berhasil login. Token ini bisa berbentuk random string (opaque) seperti API Key, bisa juga berbentuk token terstruktur seperti JWT (Access Token, Refresh Token, dll), contoh skema bearer adalah sebagai berikut:

```
GET /api/dashboard/penjualan/terbaru?tahun=2023 HTTP/1.1
Host: https://jagowebdev.com
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2Nzk0NDg1ODgsImV4cC
I6MTY3OTYyMTM4OCwidHlwZSI6InByaXZhdGUiLCJpZF91c2VyIjoiMSJ9.WPEeF3kv
2RSA83Wyq5VbiCCcHyCGgt_zcQ-TH-HtVAU
```

Token yang digunakan pada skema bearer ini juga sering disebut bearer token. Bearer sendiri dapat diterjemahkan sebagai pembawa (carier), sehingga skema bearer dapat dipahamai sebagai “Pembawa (bearer) token yang sebelumnya telah diberikan oleh server API".

Salah satu penerapan skema ini adalah pertama tama aplikasi client menampilkan form login kepada user, setelah user mengisi username dan password dan mensumbit form, maka aplikasi client mengirim username dan password tersebut ke server untuk dilakukan proses authentikasi. Setelah proses authentikasi berhasil maka server akan mengirim token. Token ini nantinya digunakan client untuk melakukan request resource ke server.

Dalam konteks API, maka ketika melakukan request ke server, client hanya perlu mengirim token (yang didapat setelah proses authentikasi), tidak perlu mengirim username dan password, hal ini berbeda dengan skema basic dan digest. Dengan menggunakan token, maka penyimpanan pada sisi client menjadi lebih aman, karena username dan password tidak terekspose, selain itu token juga dapat dapat diatur waktu expired nya, sehingga jika token ini digunakan oleh orang yang tidak berhak maka mereka hanya memiliki waktu yang sangat terbatas sebelum token expired.

Catatan:

Pada berbagai contoh HTTP Request Header diatas, kita selalu menulis nama skema di awal string/token, misal Authorization: Basic XXX, Authorization: Digest. dan Authorization: Bearer XXX.

Mungkin Anda bertanya tanya kenapa perlu menyertakan nama skema di awal string/token?

Hal ini dilakukan untuk memberitahu server skema apa yang digunakan untuk proses authentikasi sehingga server dapat menentukan proses authentikasi seperti apa yang akan dilakukan.

Penyertaan nama skema ini bermanfaat salah satunya jika server memiliki lebih dari satu skema authentikasi, misal resource tertentu diproteksi menggunakan skema bearer sedangkan resource lain menggunakan skema digest. Dengan menyertakan nama skema didepan string/token, maka server akan langsung mengetahui langkah yang harus dilakukan, misal jika client menggunakan authentikasi basic maka server perlu mendecode username dan password, dll.

Mungkin Anda juga bertanya tanya, lantas jika server hanya menggunakan satu skema authentikasi, bolehkah tidak menyertakan nama skema authentikasi di depan token?

Ya, boleh boleh saja, misal kita membuat aplikasi yang hanya kita gunakan sendiri.

Meskipun boleh, namun hal ini bukan merupakan best practice karena bagi orang lain yang membaca code kita, mereka tidak paham skema apa yang kita gunakan, untuk itu sebaiknya selalu sertakan nama skema authorisasi di depan token.

4.5. Implementasi Token

Setelah kita mempelajari berbagai jenis teknik dan metode authentikasi dan authorisasi, pada bagian ini kita akan membahas bagaimana

implementasi authentikasi dan authorisasi pada aplikasi API yang akan kita kembangkan.

4.5.1. Authentikasi

Pada aplikasi api yang kita kembangkan jenis authentikasi yang kita gunakan adalah token based authentication sehingga skema authentikasi yang kita gunakan adalah bearer. Adapun alurnya adalah sebagai berikut: pertama tama user login menggunakan form login, setelah berhasil login maka server akan menggenerate dan mengirim access token dan refresh token ke client, access token dan refresh token ini berbentuk JWT, selanjutnya dengan access token tersebut client melakukan request resource ke server.

Ketika access token expired maka dengan refersh token client meminta access token yang baru ke server, server merespon dengan menggenerate access token dan refresh token yang baru dan mengirimmnya ke client. Ketika menggenerate access token yang baru aplikasi api juga menggenerate refresh token yang baru, hal ini diperlukan untuk meningkatkan keamanan (token rotation).

4.5.2. Authorisasi

Pada standar yang ada, karekteristik JWT adalah stateless yang artinya semua informasi yang diperlukan server disimpan di token (payload), termasuk authorisasi user, untuk keperluan tersebut, kita dapat menambahkan berbagai informasi/claims pada payload, seperti data hak akses yang dimiliki oleh user, misal sebagai berikut:

```
{
    "iat": 1679448588,
    "exp": 1679621388,
    "user": {
        "id_user": 1,
        "role": [
            "admin"
        ],
        "permission": [
            {
```

```
                "id_module": 1,
                "nama_module": "user",
                "permission": [
                    "create",
                    "read_all",
                    "delete_all",
                    "update_all"
                ]
            },
            {
                "id_module": 1,
                "nama_module": "menu",
                "permission": [
                    "create",
                    "read_all",
                    "delete_all",
                    "update_all"
                ]
            }
        ]
    }
}
```

Dengan model diatas ketika server membaca payload token maka akan langsung tahu hak akses user tanpa melakukan pencarian di database, sehingga dapat mengurangi beban server.

Kondisi diatas terlihat ideal, namun demikian, dalam praktik hal ini tidak dapat diterapkan karena hak akses bersifat dinamis, sewaktu waktu dapat berubah, sedangkan token bersifat statis, tidak berubah, sehingga untuk mengetahui hak akses user tetap harus dilakukan melalui query database, hal ini juga berlaku untuk data dinamis lainnya seperti data profil user (misal jika user tiba tiba di blokir), data pembelian user, dll, itulah kenapa nantinya pada aplikasi API yang kita bangun private claims pada JWT hanya berisi id_user, data id_user ini nantinya digunakan untuk mengambil data-data terkait user tersebut melalui query database.

BAB 5 Mendesain Resource

Setelah kita membahas berbagai teori tentang REST API pada bagian ini kita akan membahas babagaimana mendesain resource pada REST API, yaitu mendefinisikan URI sebagai identifier resource, menentukan format data, dan menentukan response code yang sesuai.

5.1. Mendesain URI

Sebagaimana telah dibahas sebelumnya, tidak ada standar dalam mendesain URI pada REST API, REST hanya mensyaratkan bahwa identifier reource harus unik, ya itu saja, namun demikian seiring berjalannya waktu, dengan berbagai macam pengalaman implementasi, akhirnya mengerucut ke suatu pola tertentu, seperti penamaan resource, struktur URI, dll.

5.1.1. URI, URL, dan Endpoint

Sebelum mendesain URI, ada baiknya Anda tahu istilah URI, URL, dan Endpoint, kenapa? Karena dalam pembahasan buku ini atau dalam praktik, Anda akan sering menemui ketiga istilah tersebut.

Apa perbedaan ketiganya?

URI (Universal Resource Identifier) merupakan identifikasi dari resource (identifier), sedangkan URL (Universal Resource Locator) merupakan alamat yang menunjukkan resource tersebut (locator/lokasi), URI dapat diibaratkan seperti kita menyebut nama bengkel AHASS 02575 namun kita tidak menunjukkan lokasi/jalan/alamat menuju bengkel tersebut, disini, nama bengkel merupakan identifier. Selanjutnya jika kita menyebut alamat bengkel, misal: Jl. Dr. Rajiman No.415, Penumping, Kec. Laweyan, Kota Surakarta, Jawa Tengah 57149, maka alamat ini termasuk URL karena menunjuk lokasi bengkel, selain URL alamat ini juga termasuk URI karena alamat tersebut juga dapat digunakan sebagai identifikasi/identifier bengkel.

Dalam dunia web, tentu kita sudah familiar dengan istilah URL karena kita pasti menggunakan istilah tersebut untuk mengidentifikasi alamat web, misal: https://jagowebdev.com sama seperti analogi bengkel sebelumnya, ketika kita membicarakan URI, maka URL ini juga bisa disebut URI, karena alamat tersebut merupakan identifier dari suatu resource, sehingga dapat disimpulkan bahwa URL adalah URI. Hal ini sesuai dengan yang disebutkan pada standar "Uniform Resource Identifier (URI): Generic Syntax":

"*A URI can be further classified as a locator, a name, or both. The term* ***"Uniform Resource Locator" (URL) refers to the subset of URIs*** *that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism.*"

Standar tersebut dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc3986#section-1.1.3

Pada standar diatas jelas disebutkan bahwa URL merupakan bagian (subset) dari URI. Meskipun URL merupakan bagian dari URI, namun tidak semua URI merupakan URL karena URI bersifat umum sedangkan URL spesifik, URI juga bisa mencakup URN, misal ISBN: URN:ISBN:0-395-36341-1 (https://www.rfc-editor.org/rfc/rfc3187#section-5).

Dalam konteks REST API, kita menggunakan istilah URI karena kita berbicara tentang resource identifier meskipun bentuk URI dalam REST API umumnya sama dengan bentuk URL.

Selain URI, dalam REST API kita juga mengenal istilah endpoint. Endpoint merupakan lokasi dari resource beserta aktivitasnya, sebagai contoh resoure user diidentifikasikan dengan URI https://jagowebdev.com/api/user dalam konteks endpoint, maka URI ini juga merupakan endpoint dengan aktivitas mengambil data (GET), selain GET, juga terdapt aktivitas lain seperti POST, PUT, DELETE, sehingga endpoint nya menjadi:

```
POST https://jagowebdev.com/api/user
PUT https://jagowebdev.com/api/user/1
DELETE https://jagowebdev.com/api/user
```

Dalam buku ini kita menggunakan URI dan endpoint secara bergantian, keduanya merujuk pada maksud yang sama yaitu sebagai resource identifier (beserta aksinya).

5.1.2. Resource Archetype

URI sebagai identifier resource, didalamnya mengandung nama resource, sehingga sebelum mendesain URI sebaiknya kita mengetahui tentang Resource Archetype sehingga nantinya penamaan resource bisa menjadi lebih terstruktur.

Resource Archetype merupakan istilah untuk penamaan resource. Resource Archetype ini bukan merupakan standar, Fielding dalam disertasinya juga tidak mengatur hal ini dan tidak menggunakan istilah "archetype", pengelompokan archetype ini hanya sebagai guideline agar penamaan URI menjadi lebih **tertata dan konsisten**.

Dalam REST API, resource dibagi menjadi empat jenis resource archetype yaitu: Document, Collection, Store, dan Controller, setiap URI yang kita desain hendaknya hanya mengandung salah satu jenis archetype saja.

Document

Document ini seperti ibarat object instance atau sebuah record/baris data pada database, document archetype dapat dikategorikan lebih spesifik menjadi collection atau store, contoh Document adalah sebagai berikut:

```
https://jagowebdev.com
https://jagowebdev.com/users/
https://jagowebdev.com/users/1332
https://jagowebdev.com/users/1332/articles
```

URI pertama pada contoh diatas menunjukkan doc-root atau parent resources.

Collection

Collection merupakan resource berupa direktori resource yang dikelola oleh server, jika server mengijinkan, client dapat menambahkan resource

baru, tetapi keputusan penambahan resource baru tersebut ada pada server, collection biasanya berupa nama benda, contoh collection adalah sebagai berikut:

```
https://jagowebdev.com/user
https://jagowebdev.com/user/1332/artikel
https://jagowebdev.com/user/1332/artikel/kategori
```

Pada contoh diatas collection user diidentifikasikan dengan URI `https://jagowebdev.com/user` , pada collection ini client/user dapat menambah resource baru, misal melalui sistem registrasi.

Store

Store adalah resource yang sepenuhnya dikelola oleh client, client memiliki kontrol penuh pada resource seperti menambah, mengubah, maupun menghapus resource.

Contoh resoure store adalah resource produk favorit pada REST API e-commerce, pada resource produk favorit tersebut setiap user dapat menambah maupun menghapus produk favorit, sebagai contoh request berikut ini akan menambahkan hp-samsung pada resource favorite user dengan id 1234

```
POST /users/1234/favorite/hp-samsung
```

Adapun contoh URI untuk resource produk favorite yang dimiliki user dengan id 1234 adalah sebagai berikut:

```
https://jagowebdev.com/api/user/1234/favorite
```

URI pada resource stores bersifat tetap tidak berubah, baik ketika ada penambahan maupun pengurangan resource, seperti contoh resource produk favorite diatas, URL nya tetap `https://jagowebdev.com/user/1234/favorite` berapapun jumlah data yang ada pada resource.

Controller

Controller merupakan function yang bertugas untuk melakukan tindakan khusus dimana tindakan tersebut tidak dapat dipetakan pada HTTP method selain itu juga tindakan tersebut tidak dapat dipetakan aksinya apakah create, read, update, atau delete (CRUD).

Nama controller ini biasanya berada di bagian paling akhir dari URI, berikut ini adalah contoh controller yang memungkinkan user untuk untuk mengirim ulang notifikasi:

```
POST /notifikasi/3453/resend
```

Contoh lain controller adalah controller pada github untuk me-*rename* branch:

```
POST
https://api.github.com/repos/OWNER/REPO/branches/BRANCH/rename
```

Note: controller ini seharusnya tidak mengandung http verb seperti get-notifikasi, delete-notifikasi, dll

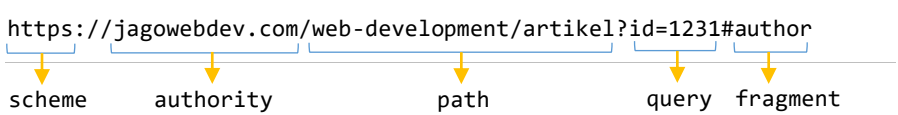
5.1.3. Mendefinisikan URI

Ketika mendesain URI maka kita perlu mendesainnya sedemikian rupa sehingga simpel, jelas, dan mudah dipahami, sehingga dapat mencerminkan resource yang dirujuk.

Format URI sendiri telah dietapkan pada standar RFC 3986 dengan judul "Uniform Resource Identifier (URI): Generic Syntax" yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc3986, pada standar tersebut format URI ditetapkan sebagai berikut:

```
scheme ":" authority "/" path [ "?" query ] [ "#" fragment ]
```

Sebagai contoh:



Berikut beberapa prinsip (Guideline) yang dapat digunakan untuk mendefinisikan URL

1. Gunakan kata benda (noun) atau controller

Resource sebaiknya menggunakan kata benda, misal user, artikel, kategori, produk, dll, atau contoller (seperti dibahas pada resource archetype)

2. Slash (/)

Tanda slash (/) digunakan untuk menunjukkan hierarikal relationship dari resource, atau mudahnya slash menunjukkan sub resources/nested resource dari suatu resource, misal:

```
https://jagowebdev.com/user/{id_user}/artikel/idartikel
```

Note: tanda kurung menunjukan bahwa nilai resource dinamis yang pada contoh diatas, nilai id_user dapat berubah ubah.

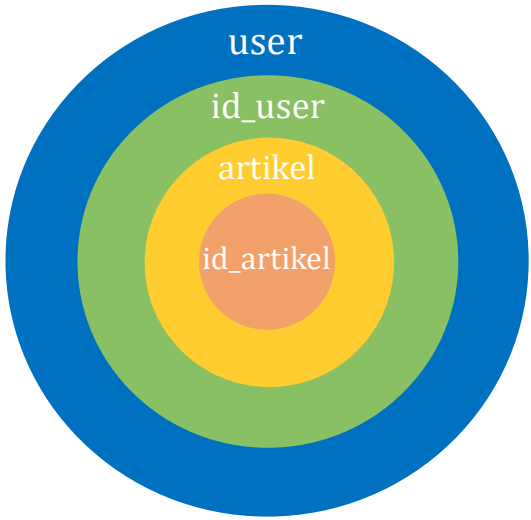
Pada URI diatas resource utamanya adalah users, resource id_user (user dengan id tertentu) adalah sub resource dari resource users, resource articles adalah sub resource dari id_user (user dengan id tertentu) dan seterusnya.

Konsep Subresource

Sub resource digunakan untuk mengidentfikasi resource tertentu dari resource parent, pada contoh diatas:

```
https://jagowebdev.com/user/agusph/artikel/{id_artikel}
```

resource agusph digunakan untuk mengidentifikasikan resource tertentu dari user, resource artikel mengidentifikasikan resource tertentu dari user agusph (artikel yang diterbitkan user agusph), dan id artikel digunakan untuk mendefinisikan reource tertentu (artikel tertentu) dari artikel yang diterbitkan oleh agusph, jika diilustasikan dengan gambar maka bentuknya adalah sebagai berikut:



Agar lebih paham mengenai sub resource, perhatikan beberapa contoh kasus berikut:

Contoh 1:

Misal ada keperluan untuk mengambil resource artikel, maka kita definisikan URI nya sebagai berikut:

```
https://jagowebdev.com/articles/
```

Selanjutnya ada keperluan untuk mengidentifikasi artikel berdasarkan idartikel, untuk keperluan tersebut kita bisa membuat sub resource dari resource artikel karena idartikel berupakan bagian (sub resource) dari artikel, sehingga susunan URI sebagai berikut:

```
https://jagowebdev.com/articles/idartikel
```

Berikutnya kita ingin mengidentifikasi resource artikel yang ditulis oleh user tertentu, untuk keperluan tersebut kita tidak bisa membuat sub

resource dari URI sebelumnya, sehingga kita perlu mendefinisikan resource baru sebagai berikut:

```
https://jagowebdev.com/users/iduser/articles
```

Misal selanjutnya kita ingin menampilkan artikel dengan kategori tertentu, maka kita tidak bisa menggunakan articles sebagai resource utama karena kategori artikel bukan merupakan sub resource dari resource artikel, melainkan harus membuat resource utama baru yaitu resource kategori artikel sebagai berikut :

```
https://jagowebdev.com/kategori-artikel/{idKategori}/artikel
```

Contoh 2:

Perhatikan URI berikut:

```
https://jagowebdev.com/articles/idartikel/users/agusph
```

Pada URI diatas, resource utamanya adalah artikel (articles) sedangkan id artikel merupakan sub resource dari artikel, dst. Pada URI tersebut dapat dibaca bahwa idartikel merupakan resource tertentu (artikel tertentu) dari resource artikel, selanjutnya resource users merupakan sub resource dari resource idartikel.

Wait… apakah Anda merasa Aneh?

Ya aneh, bisakah users menjadi sub resource idartikel? Menurut saya tidak, karena pada endpoiint id artikel, data user (penulis artikel) sudah dapat ditampilkan, yaitu user penulis artikel tersebut.

Jadi sebenarnya mendesain URI ini adalah seni, tidak ada pedoman yang baku, semuanya disesuaikan kebutuhan, yang penting konsep hierarki/sub resource nya jalan.

Contoh 3:

Misal kita ingin menampilkan semua komentar pada suatu artikel, maka kita dapat membuat URI sebagai berikut:

```
https://jagowebdev.com/comments
```

Pertanyaan selanjutnya adalah, jika ingin menampilkan komentar artikel tertentu maka kita tidak dapat membuat subresource dari resource comments tersebut melainkan harus membuat resource utama yang baru:

```
https://jagowebdev.com/articles/idarticle/comments
```

Contoh 4:

Kita ingin mendefinisikan resource gallery, URI yang kita definisikan adalah sebagai berikut:

```
https://jagowebdev.com/galleries
```

Selanjutnya kita ingin mengidentifikasi resource kategori gallery, untuk keperluan tersebut maka kita tidak dapat membuat subresource dari resource gallery melainkan perlu mendefinisikan resource utama baru berupa kategori gallery misal sebagai berikut:

```
https://jagowebdev.com/gallery-category
```

3. Hindari Trailing Forward Slash (/)

Forward slash di akhir URI tidak memiliki arti dan kadang membuat kebingungan, misal apakah ada kesalahan penulisan pada URI dimana dibelakang slash seharusnya ada karakter lainnya, berikut contoh penulisan yang benar:

```
https://jagowebdev.com/articles/idArtikel
```

sedangkan berikut contoh penulisan yang salah

```
https://jagowebdev.com/articles/idArtikel/
```

4. Gunakan hypen (-) sebagai pemisah antar kata

Agar URI mudah dibaca, maka gunakan hypen untuk pemisah antar kata, jangan gunakan spasi atau underscore(_) kenapa? Karena di URI, spasi akan di encode menjadi "%20" sedangkan underscore adalah bagian dari kata, selain itu biasanya tampilan URI akan diberi garis bawah sehingga penggunaan underscore menjadi tidak jelas.

Bagaimana dengan query string? Untuk query string tidak ada standar maupun rekomendasi penamaan, penulis lebih memilih menggunakan underscore untuk Query String, misal:

```
https://jagowebdev.com/comments?id_artikel = id
```

Kenapa demikian, karena model ini inline dengan form HTML yang disubmit, sebagai berikut:

```
<form method="get" action="https://jagowebdev.com/comments">
      <input type="text" name="id_artikel" value="1132"></input>
      <button type="submit">Submit</button>
</form>
```

Maka ketika form tersebut disubmit, URI akan berubah menjadi:

```
https://jagowebdev.com/comments?id_artikel=1132
```

Penggunaan underscore ini juga akan memudahkan aplikasi client yang menggunakan bahasa HTML (aplikasi berbasis browser)

5. Selalu gunakan lowercase

Usahakan URI menggunakan huruf kecil semua, hal ini dilakukan agar URI lebih mudah dan lebih nyaman dibaca, selain itu terkadang penggunaan huruf kapital menyebabkan error. Pada standar URI “Uniform Resource Identifier (URI): Generic Syntax" yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc3986 disebutkan bahwa URI bersifat Case Sensitive yang artinya penggunaan huruf besar dan kecil

berpengaruh, kecuali untuk schema dan host (authority). Jika Anda mengetikkan URI berikut, maka hasilnya akan sama saja:

```
https://jagowebdev.com
HTTPS://jagowebdev.com
https://Jagowebdev.com
```

Ketika URI tersebut dijalankan pada browser, maka browser akan otomatis mengubahnya menjadi huruf kecil semua.

Berbesa dengan sebelumnya, URI berikut memberikan hasil yang berbeda:

```
https://jagowebdev.com/Members/produk
HTTPS://jagowebdev.com/members/produk
```

Ketika URI dijalankan pada browser, maka browser tidak akan mengubah Members menjadi huruf kecil, alhasil url pertama mengakibatkan halaman tidak ditemukan, di kondisi lain terkadang mencampur huruf besar dan kecil tidak masalah, namun demikian untuk menghindari risiko error, maka sebaiknya gunakan lowercase semua.

6. URL sebaiknya tidak menyertakan file ekstensi

Umumnya period/dot/titik digunakan untuk memisahkan antara nama file dan ekstensi file, misal foto.jpg. Dalam praktik penggunaan period pada URI digunakan untuk menunjukkan output format yang diinginkan client, misal:

```
https://jagowebdev.com/artikel/1132.json
https://jagowebdev.com/artikel/1132.pdf
```

Pada contoh diatas pada URI pertama server akan mengirim respon format JSON sedangkan pada URI kedua format yang dikirim adalah pdf.

Penggunaan model seperti ini tidak disarankan, untuk memberitahukan format yang diinginkan, direkomendasikan menggunakan Accept Header, misal:

```
GET https://jagowebdev.com/api/artikel/1
Accept: application/json
```

selain menggunakan Accept Header dapat menggunakan query string, misal sebagai berikut:

```
https://jagowebdev.com/artikel/1132?format=json
https://jagowebdev.com/artikel/1132?format=pdf
```

Dari kedua cara diatas, lebih disarankan menggunakan Accept Header, hal ini sesuai dengan peruntukan Accept Header, selain itu dengan menggunakan Accept Header, URI menjadi lebih bersih dan rapi.

7. Gunakan nama spesifik untuk URI Authority

Layanan api disarankan menggunakan subdomain dengan nama api, contoh sebagai berikut:

```
https://api.github.com
https://api.twitter.com
https://api.sunrise-sunset.org
```

penggunaan subdomain ini tidak wajib, Anda bebas menggunakan identifikasi lainnya seperti menggunakan path api misal: https://swapi.dev/api/ atau https://partner.shopeemobile.com/api/ yang penting dapat diketahui dengan mudah bahwa alamat URI merupakan alamat api.

Selanjutnya jika aplikasi API menyediakan protal untuk developer (*develper portal*) yang biasanya berisi dokumentasi, penyediaan API key, forum, dll, maka sebaiknya alamat portal ini menggunakan subdomain developer, misal:

```
https://developer.twitter.com
https://developer.tokopedia.com
```

Dengan model sepert ini, maka yang membaca akan langsung tahu bahwa halaman tersebut ditujukan untuk developer aplikasi (programmer)

sehingga lebih "friendly", namun demikian, Anda bebas menggunakan identikasi lainya seperti menggunakan kata docs sebagaimana digunakan oleh Github pada https://docs.github.com/en/rest

Resume URI Design:

Berdasarkan uraian diatas, berikut ini checklist hal-hal yang perlu diperhatikan ketika medesain URI:

No	Checklist
1	Menggunakan nama benda untuk resource
2	Nested Reationship
3	Hindari trailing slash
4	Gunakan Hypen (-) untuk pemisah antar kata
5	Gunakan lowercase
6	Hindari penggunaan ekstansi
7	Hindari penggunaan HTTP Method pada URI, kecuali untuk method diluar HTTP Verb

5.1.4. Query String

Pada kondisi tertentu, tidak semua resource dapat dipetakan dengan URI, misal ketika kita ingin mendapatkan resource artikel yang terbit pada rentang tanggal tertentu, maka tidak pas jika kita menggunakan model resource-sub resource karena resource yang kita iginkan tersebut bukan sub resource dari artikel melainkan resource dengan kriteria tertentu, untuk itu lebih tepat jika kita menggunakan filter. Jadi dapat disimpulkan bahwa fileter digunakan untuk mendapatkan resource dengan kriteria tertentu.

Filter ini dilakukan menggunakan query string, sebagai contoh untuk mendapatkan resource artikel dengan tanggal tertentu, maka kita gunakan query string sebagai berikut:

```
http://localhost/api/artikel?start_date=2023-01-01&end_date=2023-12-31
```

Contoh lain adalah pagination, misal mendapatkan resource artikel halaman satu, dimana jumlah artikel setiap halaman adalah 25 artikel, maka query string yang kita gunakan adalah sebagai berikut:

```
http://localhost/api/artikel?page=1&length=25
```

Contoh berikutnya misal kita ingin mencari data user dengan id_permission 1, query string yang digunakan adalah:

```
http://localhost/api/user?id_permmission=1
```

Pada contoh permission ini, bisakah kita menggunakan model resource-sub resource?

Jika kita menggunakan model resource-sub resource maka alamat URI menjadi:

```
http://localhost/api/user/1
```

Menurut saya hal ini tidak pas karena id_permission bukan sub resource dari user. Kenapa? Karena id_permission **tidak dapat digunakan untuk mengidentifikasikan resource user secara unik**, hal ini berbeda dengan id_user, id user bersifat unik yang dapat mengidentifikasikan setiap user pada resource user.

5.2. Mendesain Response Body

Setelah kita membahas bagaimana mendesain URI sebagai identifier resource, selanjutnya mari kita bahas bagaimana mendesain response body atau bagaimana resource direpresentasikan.

Pada bab awal telah kita bahas bahwa tidak ada standar format data untuk aplikasi REST API, kita bisa menggunakan JSON, XML, HTML, dll. Pada aplikasi api yang kita kembangkan kita menggunakan JSON sebagai

representasi format resource, selain JSON, sesekali kita juga menggunakan HTML dan BLOB.

5.2.1. JSON (Javascript Object Notation)

Saat ini JSON merupakan format yang paling populer untuk merepresentasikan resource, mungkin hampir semua aplikasi api menggunakan format ini, sebelumnya ketika SOAP masih populer digunakan dalam mengembangkan api, format XML lah yang populer digunakan karena SOAP mensyaratkan menggunakan format ini untuk representasi resource, meski demikian baik JSON maupun XML keduanya merupakan format yang populer dan banyak digunakan karena keduanya human and machine readable yang artinya mudah dibaca oleh manusia maupun mesin.

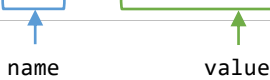
JSON didokumentasikan pada Standar RFC 8259 "The JavaScript Object Notation (JSON) Data Interchange Format" yang dapat diakses melakui tautan https://www.rfc-editor.org/rfc/rfc8259. Pada standar tersebut, JSON merupakan format standar pertukaran data (Data Interchange Format) yang bersifat lightweight, text-based, *language-independent*, yang artinya JSON bersifat ringan, berbasis text, dan tidak terikat pada bahasa pemrograman manapun.

Meskipun JSON menggunakan kata "Javascript" ya karena JSON ini awalnya memang format data dari Javascript, namun ketika dijadikan standar pertukaran data maka JSON ini tidak terikat pada bahasa pemrograman manapun atau bersifat *language-independen*, banyak bahasa pemrograman yang telah mendukung dengan penuh format data ini seperti PHP yang menyediakan fungsi json_encode() dan json_decode() untuk mempermudah kita bekerja dengan JSON.

Format Penulisan JSON

JOSN berbentuk pasangan name value (name value pair). Name harus berbentuk string dengan pembuka dan penutup double quote (") bukan single quote (') dan harus unik, sedangkan value dapat berupa object, array, string, dan number. Berikut contoh JSON

```
{ "nama" : "Agus Prawoto Hadi"}
```



Pada contoh diatas, value berupa string, contoh lainnya adalah sebagai berikut:

```
{
      "id_user" : 1,
      "nama" : "Agus Prawoto Hadi",
      "alamat" : {
            "jalan" : "Jl. Kencur No. 19",
            "kabupaten" : "Sukoharjo"
      },
      "bahasa_pemrograman" : ["PHP", "MySQL", "Javascript"]
}
```

Pada contoh JSON diatas, bentuk/tipe data value bermacam macam, baris pertama berbentuk number, baris kedua string, baris ketiga object, dan baris keempat adalah array.

Penamaan name pada JSON tidak memiliki standar, Anda bisa menggunakan spasi, dash(-), underscore (_), dll, namun demikian terdapat beberapa hal yang dapat dipertimbangkan ketika membuat nama pada name ini:

1. Menggunakan huruf kecil semua.

Kenapa?

Karena dibanyak bahasa pemrograman, bagian name ini nantinya akan menjadi key, misal pada PHP akan menjadi array key misal $user['id_user'] atau property object misal $user->id_user, di javascript name ini juga akan menjadi property dari Object, misal user.id_user, key dan property ini bersifat case sensitive, sehingga dengan mencampur antara huruf besar dan kecil kemungkinan terjadinya error pada program akan sangat tinggi.

2. Pemisah kata sebaiknya menggunakan underscore (_). Selain underscore, Anda juga dapat menggunakan camelCase, misal: idUser.

Hindari penulisan name menggunakan spasi atau dash(-). Kenapa? Seperti pada pembahasan nomor 1, bahwa ketika di parsing, JSON name ini akan menjadi key pada array dan property pada object, misal (1) pada PHP: $user->id_user, (2) pada javascript: user.bahasa_pemrograman atau user.bahasaPemrograman, jika menggunakan dash maka pada PHP property ini akan mengakibatkan error, karena PHP tidak memperbolehkan penulisan property object menggunakan dash seperti $user->id-user, sedangkan pada javascript, penggunaan spasi atau dash ini akan merepotkan karena untuk memanggilnya kita tidak bisa menggunakan notasi dot (.) melainkan harus menggunakan string, misal property bahasa-pemrograman tidak bisa dipanggil dengan user.bahasa-pemrograman melainkan harus user['bahasa-pemrograman'].

Intinya penulisan nama sebaiknya disesuikan dengan bahasa pemrograman yang akan digunakan meskipun tidak harus, karena memang pada dasarnya JSON format *language-independen*, tidak terikat pada bahasa pemrograman apapun.

5.2.2. Struktur JSON Untuk Representasi

Meskipun hampir semua aplikasi api menggunakan JSON untuk merepresntasikan resource, namun hingga saat ini tidak ada format baku struktur JSON seperti apa yang seharunya digunakan, masing masing pengembang api memiliki format sendiri sendiri, meskipun ada beberapa yang menggunakan format baku (menggunakan standar yang ada).

5.2.2.1. Response Sukses

Ketika request yang dikirim client berhasil dieksekusi oleh server, maka server akan mengirim data ke client sesuai dengan request client tersebut, agar format data yang dikirim seragam, ada yang membuat standar format data yang disajikan.

JSON Api

JSON API (https://jsonapi.org) membuat standar format data JSON yang di kirim ke client, agar client tahu bahwa format data yang digunakan adalah JSON API, maka JSON API menggunakan Header Content-Type:

application/api+json. Bentuk format data yang didefinisikan oleh JSON API adalah sebagai berikut:

```
{
  "links": {
    "self": "http://example.com/articles",
    "next": "http://example.com/articles?page[offset]=2",
    "last": "http://example.com/articles?page[offset]=10"
  },
  "data": [
    {
      "type": "articles",
      "id": "1",
      "attributes": {
        "title": "JSON:API paints my bikeshed!"
      },
      "relationships": {
        "author": {
          "links": {
            "self": "http://example.com/articles/1/relationships/author",
            "related": "http://example.com/articles/1/author"
          },
          "data": {
            "type": "people",
            "id": "9"
          }
        },
        "comments": {
          "links": {
            "self":
"http://example.com/articles/1/relationships/comments",
            "related": "http://example.com/articles/1/comments"
          },
          "data": [
            {
              "type": "comments",
              "id": "5"
            },
            {
              "type": "comments",
              "id": "12"
            }
          ]
        }
      },
      "links": {
        "self": "http://example.com/articles/1"
      }
```

```
    }
  ],
  "included": [
    {
      "type": "people",
      "id": "9",
      "attributes": {
        "firstName": "Dan",
        "lastName": "Gebhardt",
        "twitter": "dgeb"
      },
      "links": {
        "self": "http://example.com/people/9"
      }
    }
  ]
}
```

Collection JSON

Selain JSON API ada juga yang membuat standar lain, yaitu Collection JSON yang dapat diakses melalui tautan http://amundsen.com/media-types/collection/format/ standar ini menggunakan Header Content-Type application/vnd.collection+json, adapun contoh formatnya adalah sebagai berikut:

```
{
  "links": {
    "self": "http://example.com/articles",
    "next": "http://example.com/articles?page[offset]=2",
    "last": "http://example.com/articles?page[offset]=10"
  },
  "data": [
    {
      "type": "articles",
      "id": "1",
      "attributes": {
        "title": "JSON:API paints my bikeshed!"
      },
      "relationships": {
        "author": {
          "links": {
            "self": "http://example.com/articles/1/relationships/author",
            "related": "http://example.com/articles/1/author"
          },
```

```
          "data": {
            "type": "people",
            "id": "9"
          }
        },
        "comments": {
          "links": {
            "self":
"http://example.com/articles/1/relationships/comments",
            "related": "http://example.com/articles/1/comments"
          },
          "data": [
            {
              "type": "comments",
              "id": "5"
            },
            {
              "type": "comments",
              "id": "12"
            }
          ]
        }
      },
      "links": {
        "self": "http://example.com/articles/1"
      }
    }
  ],
  "included": [
    {
      "type": "people",
      "id": "9",
      "attributes": {
        "firstName": "Dan",
        "lastName": "Gebhardt",
        "twitter": "dgeb"
      },
      "links": {
        "self": "http://example.com/people/9"
      }
    }
  ]
}
```

Baik JSON API maupun Collection JSON secara resmi terdaftar sebagai standar Media Type, selengkapnya dapat di baca di halaman "Media Types"

yang dapat diakses melalui tautan https://www.iana.org/assignments/media-types/media-types.xhtml

Meskipun telah ada standar, dalam praktik, masing masing vendor lebih memilih menggunakan format sendiri sendiri, hingga saat ini masih sedikit yang menggunakan standar yang ada, misal pada GitHub langsung menyajikan list data secara langsung sedangkan Twitter memasukkan nya pada atribut data. Sebagai gambaran berikut contoh format data JSON dari beberapa vendor:

1. **Twiter**

```
{
    "data": {
        "attachments": {
            "media_keys": [
                "7_1138489597158199298"
            ]
        },
        "author_id": "2244994945",
        "context_annotations": [{}],
        "created_at": "2019-06-11T17:59:13.000Z",
        "entities": {},
        "id": "1138505981460193280",
        "lang": "en",
        "public_metrics":{"retweet_count": 107, "reply_count": 31},
        "source": "Twitter Web Client",
        "text": "ʊ̟̈da-dada-DAH! We’re introducing the first Twitter
Developer Labs endpoints: \n\n✨GET/users and GET/tweets ✨\n\nLabs
is now open to all developers to start experimenting today
ɻɠhttps://t.co/eNx4Wc3Qwj https://t.co/ucmZrJAYjk"
    },
    "includes": {
        "media": [
            {
                "media_key": "7_1138489597158199298",
                "type": "video"
            }
        ]
    }
}
```

Referensi: https://developer.twitter.com/en/docs/labs/tweets-and-users/quick-start/get-tweets

2. Facebook

```
"feed": {
  "data": [
    {
      "created_time": "2021-12-12T01:24:21+0000",
      "message": "This picture of my grandson with Santa",
      "id": "POST-ID"
    },
    {
      "created_time": "2021-12-11T23:40:17+0000",
      "message": ":)",
      "id": "POST-ID"
    },
    {
      "created_time": "2021-12-11T23:31:38+0000",
      "message": "Thought you might enjoy this.",
      "id": "POST-ID"
    }
  ],
  "paging": {
    "previous": "https://graph.facebook.com/v8.0/USER-
D/feed?format=json&limit=3&since=1542820440&access_token=ACCESS-
OKEN&__paging_token=enc_AdC...&__previous=1",
    "next": "https://graph.facebook.com/v8.0/USER-
D/feed?format=json&limit=3&access_token=ACCESS-
OKEN&until=1542583212&__paging_token=enc_AdD..."
  }
},
"id": "USER-ID"
```

Referensi: https://developers.facebook.com/docs/graph-api/guides/field-expansion

3. Google

```
{
  "kind": "books#volumes",
  "totalItems": 489,
  "items": [
    {
      "kind": "books#volume",
      "id": "4lZcsRwXo6MC",
      "etag": "fhi1pWZ1HMM",
```

```
      "selfLink":
"https://www.googleapis.com/books/v1/volumes/4lZcsRwXo6MC",
      "volumeInfo": {
        "title": "REST API Design Rulebook",
        "authors": [
          "Mark Masse"
        ],
        "publisher": "\"O'Reilly Media, Inc.\"",
        "publishedDate": "2011-10-25",
        "description": "The basic rules of REST APIs - \"many nouns,
few verbs, stick with HTTP\" - seem easy, but that simplicity and
power require discipline to work smoothly. This brief guide provides
next steps for implementing complex projects on simple and extensible
foundations.",
      },
      "saleInfo": {
        "country": "ID",
        "saleability": "NOT_FOR_SALE",
        "isEbook": false
      },
      "accessInfo": {
        "country": "ID",
        "viewability": "PARTIAL",
        "embeddable": true
      },
      "searchInfo": {
        "textSnippet": "This brief guide provides next steps for
implementing complex projects on simple and extensible foundations."
      }
    }
  ]
}
```

Referensi: https://www.googleapis.com/books/v1/volumes?q=restapi

4. Tokopedia

```
{
    "header": {
        "process_time": 5.871520385,
        "messages": "Your request has been processed successfully"
    },
    "data": [
        {
            "basic": {
                "productID": 15245228,
                "shopID": 480829,
```

```
                "status": 1,
                "name": "hxh wallpaper best la zzzz",
                "condition": 1,
                "childCategoryID": 1828,
                "shortDesc": "Best wallpaper for hxh"
            },
            "price": {
                "value": 3000,
                "currency": 1,
                "LastUpdateUnix": 1557981264,
                "idr": 3000
            },
            "weight": {
                "value": 5,
                "unit": 1
            }
        }
    ]
}
```

Referensi:
https://developer.tokopedia.com/openapi/guide/?msclkid=fdba8c60cf3211ec908ef885ece77fe9#/product/getproduct

5. Github

```
[
  {
    "id": 2126244,
    "node_id": "MDEwOlJlcG9zaXRvcnkyMTI2MjQ0",
    "name": "bootstrap",
    "full_name": "twbs/bootstrap",
    "private": false,
    "owner": {
      "login": "twbs",
      "id": 2918581,
      "node_id": "MDEyOk9yZ2FuaXphdGlvbjI5MTg1ODE=",
      "avatar_url": "https://avatars.githubusercontent.com/u/2918581?v=4"
    },
    "html_url": "https://github.com/twbs/bootstrap",
    "description": "The most popular HTML, CSS, and JavaScript framework
for developing responsive, mobile first projects on the web.",
    "fork": false,
    "url": "https://api.github.com/repos/twbs/bootstrap",
    "created_at": "2011-07-29T21:19:00Z",
    "updated_at": "2022-05-09T01:33:27Z",
```

```
    "allow_forking": true,
    "is_template": false
  },
  {
    "id": 8545220,
    "node_id": "MDEwOlJlcG9zaXRvcnk4NTQ1MjIw",
    "name": "bootstrap-expo",
    "full_name": "twbs/bootstrap-expo",
    "private": false,
    "owner": {
      "login": "twbs",
      "id": 2918581,
      "node_id": "MDEyOk9yZ2FuaXphdGlvbjI5MTg1ODE=",
      "avatar_url": "https://avatars.githubusercontent.com/u/2918581?v=4"
    },
    "html_url": "https://github.com/twbs/bootstrap-expo",
    "description": "Beautiful and inspiring uses of Bootstrap.",
    "fork": false,
    "url": "https://api.github.com/repos/twbs/bootstrap-expo",
    "created_at": "2013-03-04T00:33:43Z",
    "updated_at": "2022-04-11T11:15:58Z",
    "allow_forking": true,
    "is_template": false
  }
]
```

Referensi: https://api.github.com/users/twbs/repos atau https://docs.github.com/en/rest

Dengan menempatkan list data ke atribut data, seperti pada Twitter, atau items pada Google, informasi yang disertakan bisa menjadi lebih fleksibel, misal jika ingin menyertakan Hypermedia berupa pagination, maka format yang digunakan bisa seperti berikut:

```
{
    "items": {[
        ...
    ]},
    "navigation": {
           "next" : "",
           "prev" : "",
           "first" : "",
           "last" : "",
    }
}
```

Lebih jauh lagi, data juga dapat kita lebih rinci lagi sehingga dapat memasukkan lebih banyak attribute, misal:

```
{
    "items": {[
        {
                "data" : {
                      ...
                },
                "links" : {
                      "self": ""
                },
                "actions" : {
                      "edit" : {
                            "url" : "",
                            "method" : "PUT",
                            fields: {}
                      },
                      "delete" : {
                            "url",
                            "method" : "POST"
                      }
                }
            },
            {
                  ...
            }
    },
    "navigation": {
        "next" : "",
            "prev" : "",
            "first" : "",
            "last" : ""
    }
}
```

Pada model yang punulis kembangkan, penulis menggunakan model yang terakhir ini karena lebih fleksibel.

5.2.2.2. Error Response

Selain response sukses, aplikasi juga dapat memberikan response error. Ketika mendesain response error, maka pastikan response error tersebut jelas, isi response hendaknya memuat beberapa hal penting yaitu:

- Memuat "Human Readable Response", response singkat yang dapat dibaca oleh user, misal "Request melebihi limit yang ditetapkan". Pesan ini tidak perlu panjang, cukup singkat saja yang penting informatif.
- Memuat "Machine readable code" yaitu kode tertentu yang merujuk spesifik error dimaksud, misal Github menggunakan data JSON {"code" : "missing_field"} jika ada parameter yang kurang lengkap, atau Google menggunakan data JSON {"reason": "invalidParameter"} untuk invalid value. Kode ini sifatnya tetap, tidak berubah meskipun deskripsi errornya berubah.

  Kode ini diperlukan karena HTTP Status Code belum menggambarkan error secara spesifik, sebagai contoh pada HTTP Status Code 401 Unauthorized, maka bisa terjadi beberapa kemungkinan, bisa jadi user tidak memiliki permission untuk mengakses resource, bisa juga jumlah request yang dilakukan user telah melebihi limit harian, dll. Selain itu hanya dengan pesan error biasa, misal "Request melebihi limit" tidak bisa menjadi patokan, karena pesan error tersebut dapat berubah.

  Dengan kode ini maka akan memudahkan user untuk menemukan masalah secara spesifik yang ada di dokumentasi API, selain itu kode ini juga dapat digunakan oleh client (aplikasi) untuk melakukan tindakan tertentu, misal:

```
if ($response['error_code'] == 'missing_field') {
      // redirect to
}
```

  Bandingkan jika menggunakan message

```
if ( $response['message'] == 'Validation failed') {
      // redirect to
}
```

Dengan model terakhir diatas, maka jika message berubah misal menjadi "Missing input field" maka script diatas tidak dapat digunakan lagi.

- Dokumentasi URL. Pesan error hendaknya disertakan link dokumentasi atas error tersebut sehingga memudahkan user untuk mencari rujukan untuk mengatasi error dimaksud.

Format response error sendiri telah ada standarnya yaitu pada RFC 7807 yang dapat diakses melalui tautan https://datatracker.ietf.org/doc/html/rfc7807, standar ini menggunakan Header Content-Type application/problem+json, adapun contoh format representasinya adalah sebagai berikut:

```
{
    "type": "https://example.com/probs/out-of-credit",
    "title": "You do not have enough credit.",
    "detail": "Your current balance is 30, but that costs 50.",
    "instance": "/account/12345/msgs/abc",
    "balance": 30,
    "accounts": ["/account/12345",
                 "/account/67890"]
}
```

Contoh lain adalah:

```
{
   "type": "https://example.net/validation-error",
   "title": "Your request parameters didn't validate.",
   "invalid-params": [ {
                         "name": "age",
                         "reason": "must be a positive integer"
                       },
                       {
                         "name": "color",
                         "reason": "must be 'green', 'red' or
'blue'"}
                     ]
}
```

Meskipun telah ada standarnya, dalam praktik hanya sedikit yang menggunakan standar tersebut, masing masing vendor mendefinisikan

pesan error sendiri sendiri, atau mengadopsi hanya untuk bagian tertentu saja, tidak semuanya, berikut beberapa contoh format JSON untuk response error dari berbagai vendor

1. Twitter

```
{
  "errors": [
    {
      "parameters": {
        "end_time": [
          "2026-10-31T23:59Z"
        ]
      },
      "message": "Invalid 'end_time':'2026-10-31T23:59Z'. 'end_time'
must be a minimum of 10 seconds prior to the request time."
    }
  ],
  "title": "Invalid Request",
  "detail": "One or more parameters to your request was invalid.",
  "type": "https://api.twitter.com/2/problems/invalid-request"
}
```

Referensi: https://developer.twitter.com/en/support/twitter-api/error-troubleshooting

2. Facebook

```
{
      "error": {
            "message": "Message describing the error",
            "type": "OAuthException",
            "code": 190,
            "error_subcode": 460,
            "error_user_title": "A title",
            "error_user_msg": "A message",
            "fbtrace_id": "EJplcsCHuLu"
      }
}
```

Referensi: https://developers.facebook.com/docs/graph-api/guides/error-handling

3. Google

```
{
  "error": {
    "code": 503,
    "message": "Service temporarily unavailable.",
    "errors": [
      {
        "message": "Service temporarily unavailable.",
        "domain": "global",
        "reason": "backendFailed"
      }
    ]
  }
}
```

Referensi: https://www.googleapis.com/books/v1/volumes/1 atau https://cloud.google.com/apis/design/errors

4. Tokopedia

```
{
    "header": {
        "process_time": 0.99861021,
        "messages": "Our server encounters an error, please try again
later",
        "reason": "Failed To Scan DB Data",
        "error_code": "OPENAPI_DB_002"
    },
    "data": null
}
```

Referensi: https://developer.tokopedia.com/openapi/guide/?msclkid=fdba8c60cf3211ec908ef885ece77fe9#/product/getproduct

5. Github

```
{
  "message": "Not Found",
  "documentation_url": "https://docs.github.com/rest"
}
```

Referensi: https://api.github.com/repos

```
{
      "message": "Validation Failed",
      "errors": [
            {
                  "resource": "Issue",
                  "field": "title",
                  "code": "missing_field"
            }
      ]
}
```

Referensi: https://docs.github.com/en/rest/overview/resources-in-the-rest-api

Berdasarkan contoh diatas, Anda bebas menggunakan format untuk representasi error, yang penting format pesan error tersebut memudahkan baik user maupun client (apliaksi) menghandle error tersebut.

5.2.3. BLOB

Dalam dunia API maupun REST API, pada umumnya representasi response menggunakan JSON, namun demikian pada kondisi tertentu kita perlu menggunakan format lainnya yang tidak kalah penting yaitu fomat Binary atau disebut juga BLOB (Binary Large Object)

Format BLOB ini digunakan ketika kita menyediakan fitur download/ekspor, seperti api yang memungkinkan client untuk mengakspor data ke dalam format tertentu yang berbentuk file, seperti Csv, Excel, PDF atau format lainnya. Pada kondisi demikian, kita dapat mengirim data dalam format JSON yaitu dengan mengubah data binary tersebut ke format lain misal base64, namun demikian hal ini tidak efisien, karena akan menyebabkan size data menjadi lebih besar, sekitar 30% lebih besar (13Mb menjadi 17,3Mb)

Sebagai alternatif, kita dapat mengirim data tersebut apa adanya (format binary), yang nantinya response binary ini oleh client diolah sedemikian rupa sehingga membentuk file yang bisa didownload.

Berikut contoh resoure binary untuk file PDF.

Pada PHP data binary file ini dapat kita peroleh menggunakan fungsi file_get_contents() sebagai berikut

```php
<?php
$content = file_get_contents('buku.pdf');
echo $cotnent;
```

Note: Ketika mengirim dengan type binary ini, kita harus memastikan bahwa Content-Type headernya harus benar benar sesuai dengan file yang dikirim, misal PDF: application/pdf, Excel (xlsx): application/vnd.openxmlformats-officedocument.spreadsheetml.sheet.

5.3. Status Code

Pada bab Komunikasi Server dan Client kita telah membahas tentang status code, pada bab tersebut kita membahas beberapa status code yang biasanya digunakan untuk mengembangkan aplikasi API, pada bagian ini kita akan membahas lebih lanjut penggunaan status code tersebut.

Sebagaimana telah disebutkan sebelumnya bahwa HTTP status code ini sudah ada di standar "HTTP Semantics" yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc9110.html#name-status-codes, namun demikian dalam implementasinya terdapat berbagai perbedaan iterpretasi

atas penerapan standar tersebut, masing masing miliki argumen sendiri, berikut beberapa perbedaan pendapat yang sering terjadi.

5.3.1. Satu status code untuk semua?

Ada yang berpendapat bahwa menggunakan satu status code untuk semua lebih praktis karena tidak perlu perlu ribet mendefinisikan status code, toh jika kita mendefinisikan status kode dengan 200 OK semua, client akan tahu apakah server memberikan hasil error atau tidak karena kita telah mengirimkan detil dari respon server, misal: ketika client mengirim perintah menambah data dan server mengecek ada inputan yang kurang, server dapat saja mengirim status kodenya 200 OK dan memberikan pesan seperti ini:

```
{
     status : 'error',
     message : 'Field nama tidak boleh kosong'
}
```

sehingga client tahu bahwa request yang dikirim client menghasilakan error.

Model ini terlihat bagus, tetapi sebenarnya memiliki beberapa kelemahan yaitu:

- client tidak dapat langsung tahu hasil dari request yang dikirim dan bisa jadi client mengira bahwa request yang dikirim baik baik saja.
- client harus mengecek dua kali baik status code dan message yang dikirim.
- Tidak ada standar yang baku status respon sehingga tidak dapat diketahui errornya disebabkan karena apa apakah karena user belum login, karena ada masalah pada server, dll, sehingga bisa jadi menyebabkan client salah mengambil tindakan.

dengan demikian untuk mengecek hasil dari request sukses atau error kita harus menggunakan status code sedangkan untuk detail error nya dapat dilihat pada message body.

5.3.2. Status Code Data Tidak Ditemukan

Dalam praktik, kita sering menemui error terkait data tidak ditemukan, misal ketika kita ingin menampilkan data user dengan URI sebagai berikut:

```
http://localhost/api/user/5
```

namun data user tersebut tidak ditemukan di database, atau pada contoh lain ketika kita ingin memfilter data penjualan dengan URI sebagai berikut:

```
http://localhost/api/penjualan?start_date=2023-05-
01&end_date=2023-05-31
```

namun data penjualan dengan kriteria tersebut tidak ditemukan.

Pada kondisi tersebut, status code apa yang seharusnya digunakan?

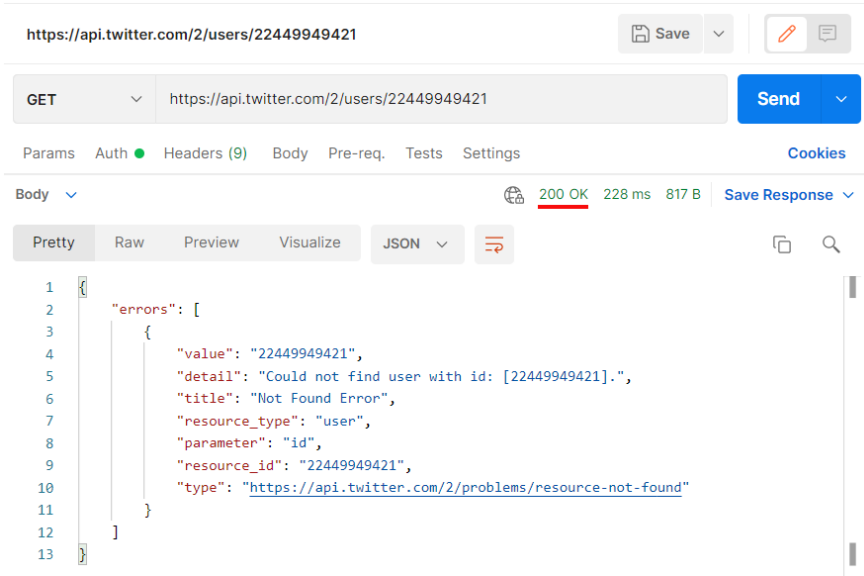
Dalam praktik, setidaknya ada empat pendapat mengenai hal ini, yaitu:

- 204 (No Content) karena request sebenarnya berhasil (tidak terjadi error) namun demikian data tidak ada (karena tidak ditemukan).
- 200 (OK) dengan pesan tertentu karena memang request sukses dieksekusi oleh server, tidak terjadi error.
- 400 Bad Request karena error tersebut disebabkan oleh client, parameter yang dikirim client tidak tepat sehingga perlu diperbaiki.
- 404 (Not Found) karena memang data tidak ditemukan (Not Found).

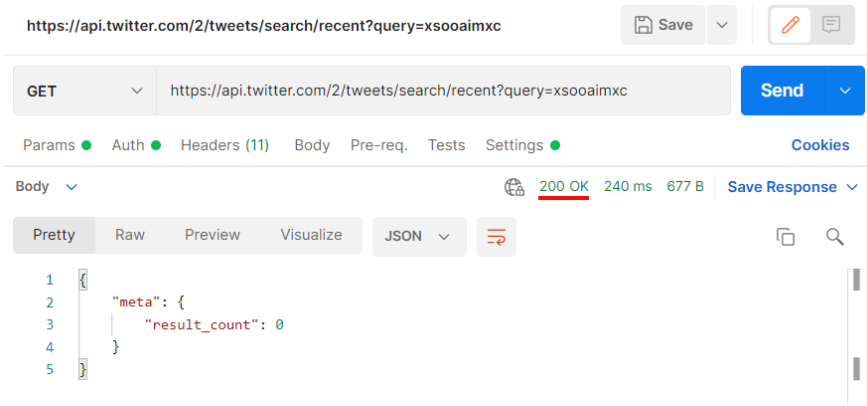
Mari kita lihat bagaimana vendor besar mengimplementasikannya.

1. Twitter

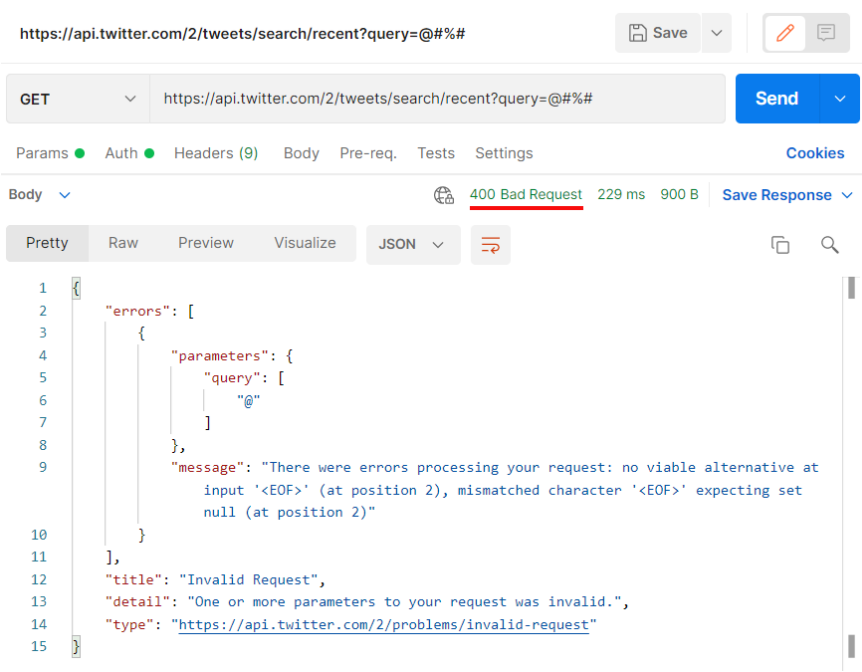
Twitter menggunakan status code baik 200 OK maupun 400 Bad Request untuk data yang tidak ditemukan. Berikut ini contoh response yang diberikan ketika user tidak ditemukan



Pada contoh diatas, jika user tidak ditemukan Twitter memberikan status code 200 OK dengan pesan error tertentu, selanjutnya pada pencarian menggunakan query string, jika data tidak ditemukan, Twitter juga memberikan status code 200 OK, contoh sebagai berikut:



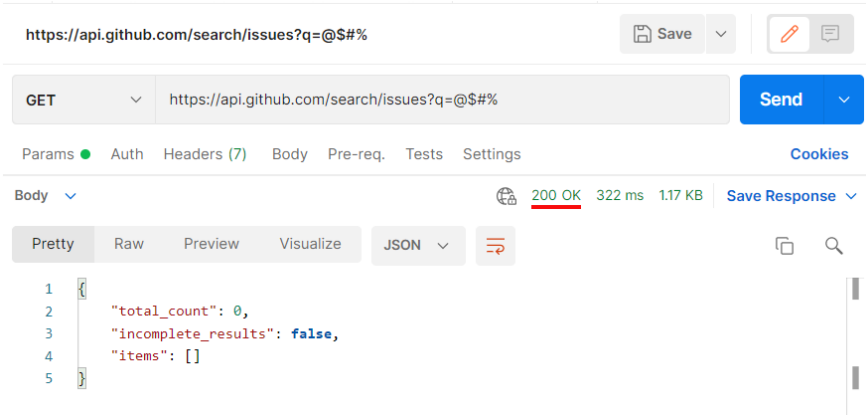
namun demikian jika isian parameter pada query string tidak sesuai yang disyaratkan misal mengandung karakter yang tidak diperkenankan maka Twitter memberikan response code 400 Bad Request



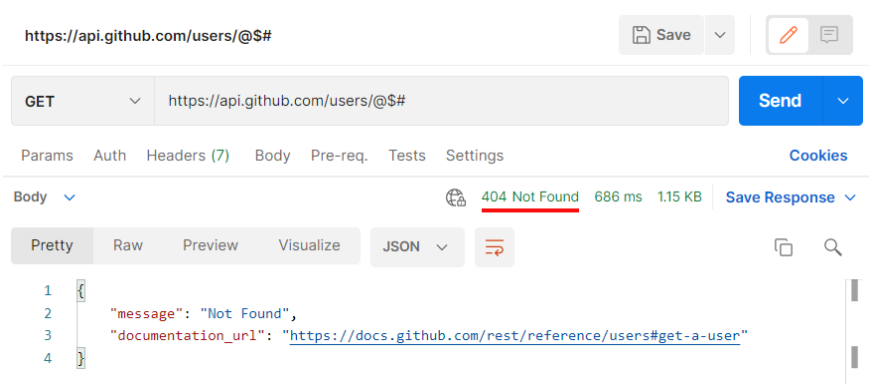
pada contoh diatas error terjadi dikarenakan kita menggunakan spesial karakter pada query string.

2. Github

Pada Github, jika pencarian menggunakan query string tidak memberikan hasil, maka status yang diberikan adalah 200 OK misal sebagai berikut:

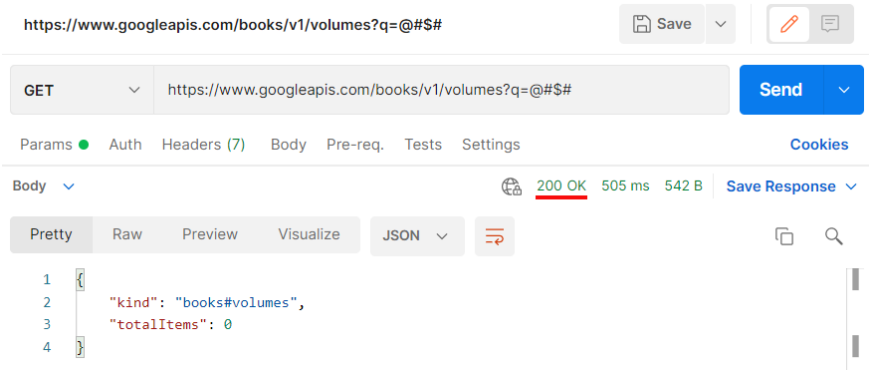


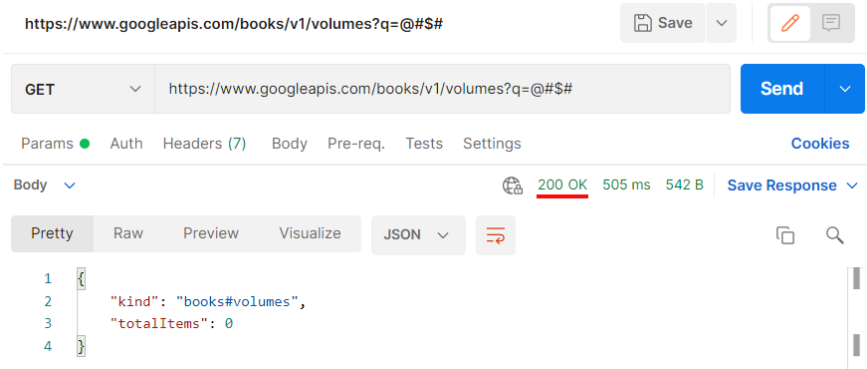
namun demikian, jika pencarian menggunakan path tidak memberikan hasil, maka jika data tidak ditemukan status code yang diberikan adalah 404 Not Found yang berarti resource tidak ditemukan, misal:



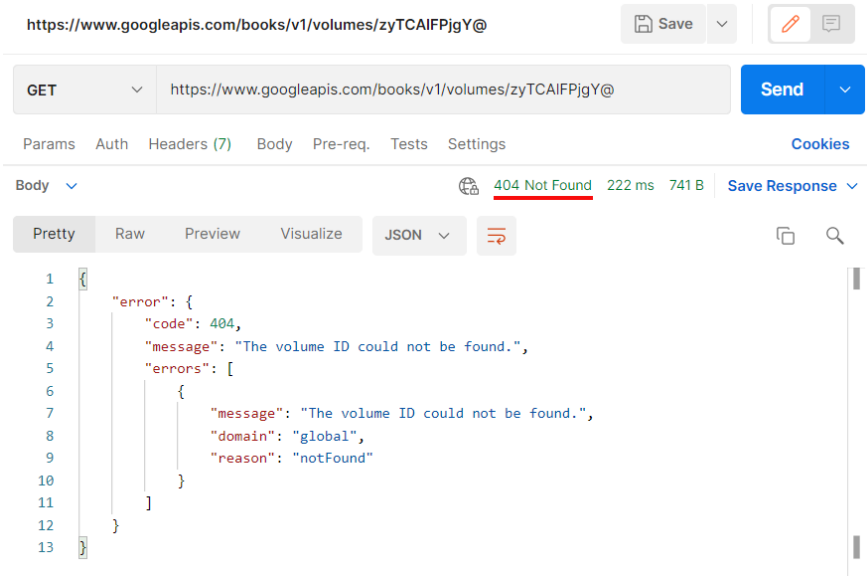
3. Google

Contoh berikutnya adalah Google, pencarian menggunakan query string dengan hasil pencarian tidak ditemukan maka Google akan memberikan status code 200 OK





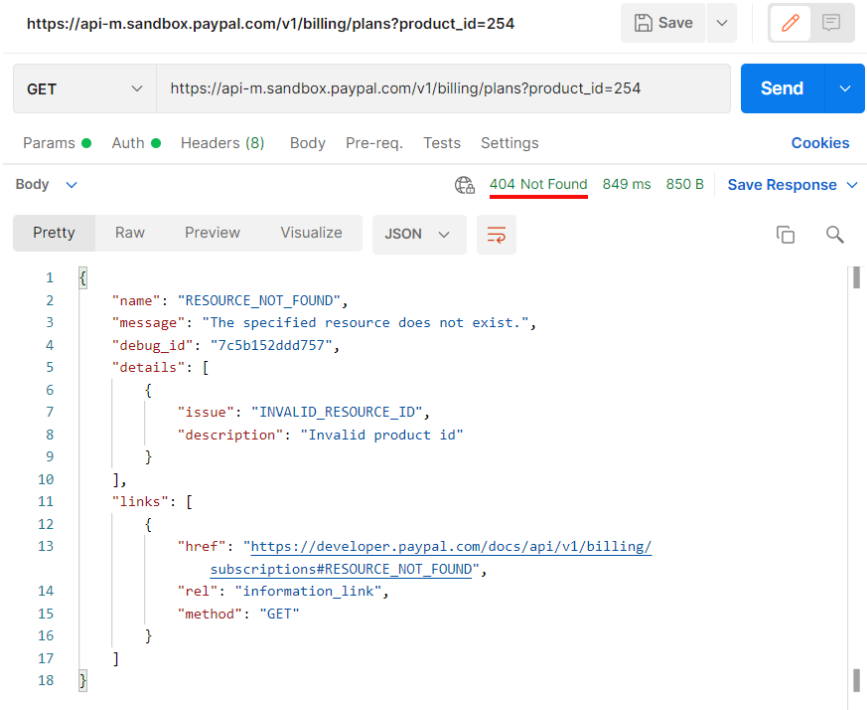
sedangkan jika pencarian menggunakan path tidak membuahkan hasil maka status code yang diberikan adalah 404 Not Found misal sebagai berikut:



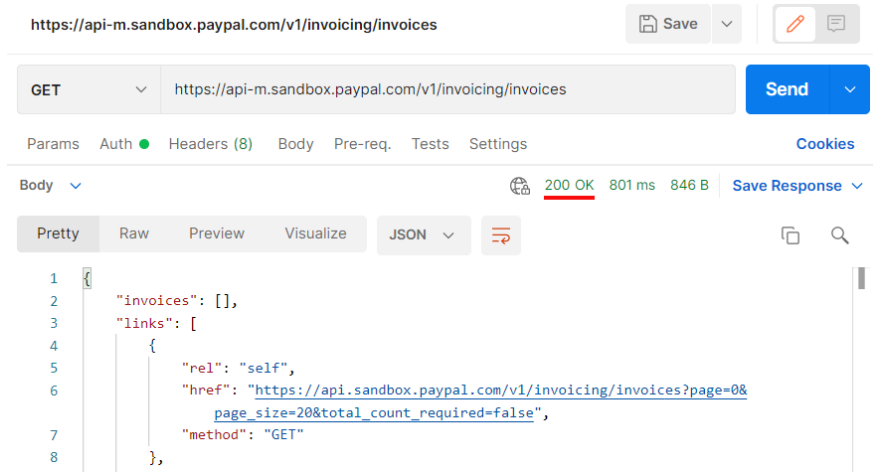
4. Paypall

Contoh berikutnya adalah PayPall, untuk pencarian menggunakan query string, jika data tidak ditemukan maka status code yang diberikan adalah

404 Not Found dengan memberikan informasi error yang terjadi misal sebagai berikut:

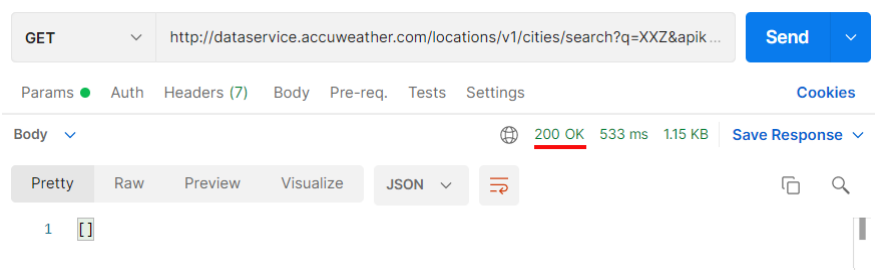


namun untuk pencarian menggunakan path, jika data tidak ditemukan, maka PayPall akan memberikan response code 200 OK sebagai berikut:

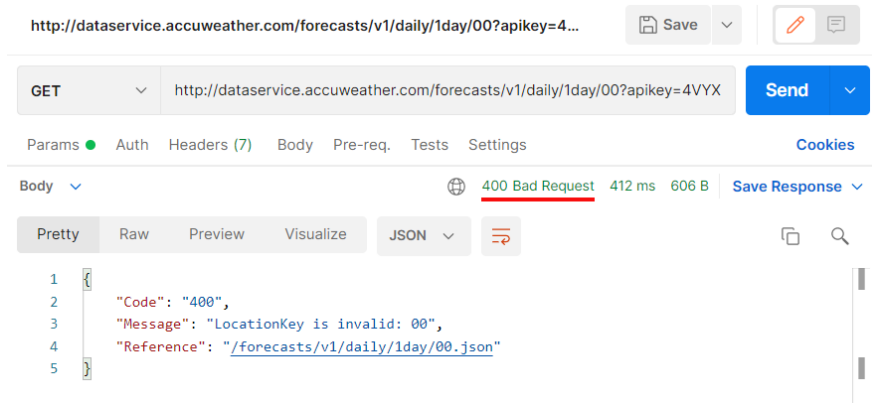


5. AccuWheather

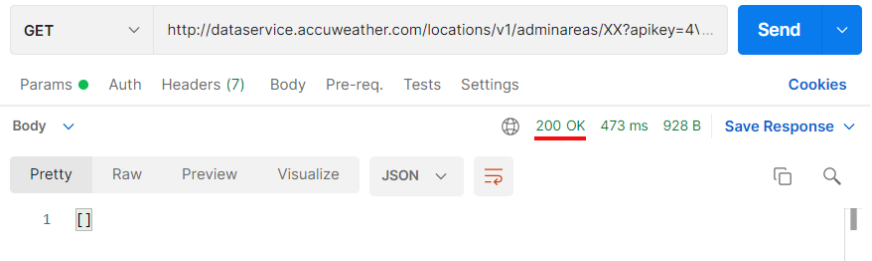
Contoh yang lain adalah AccuWheather, untuk pencarian menggunakan query string, jika tidak membuahkan hasil maka response code yang dihasilkan adalah 200 OK dengan body response berupa array kosong sebagai berikut:



Selanjutnya jika path tidak memberikan hasil maka response yang diberikan adalah juga 400 Bad Request sebagai berikut:

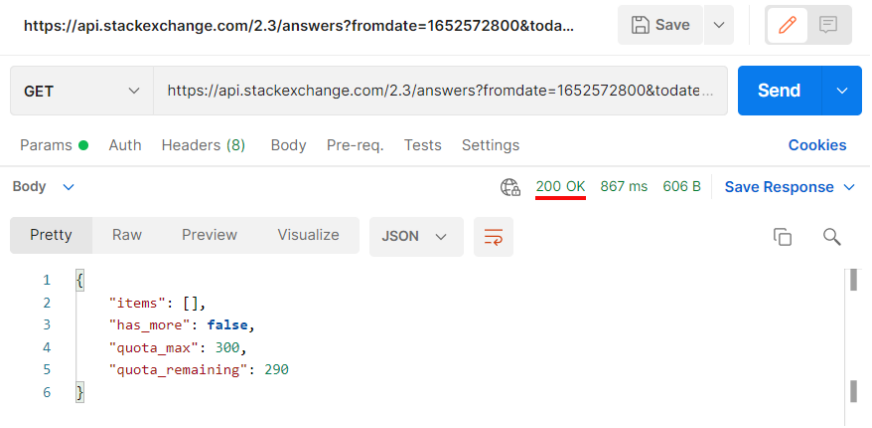


namun kadang juga memberikan response 200 OK sebagai berikut:

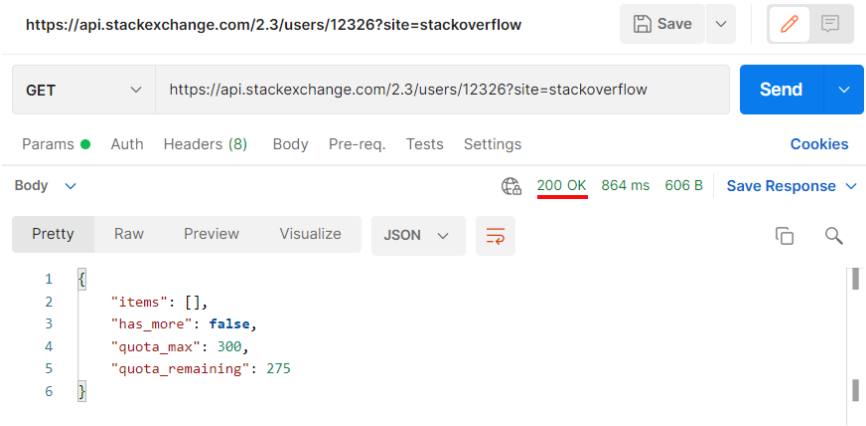


6. Stackoverfow

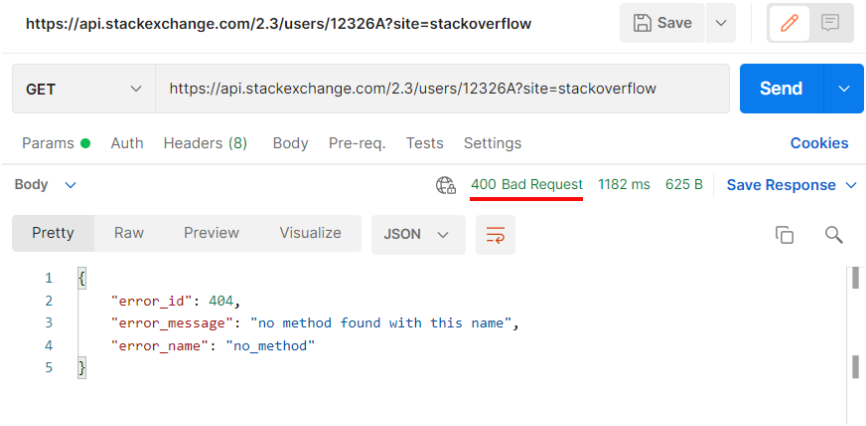
Agar lebih meyakinkan, berikut ini response code dari api Stackexchange (Stackoverflow) komunitas programmer terbesar saat ini, web tersebut memberikan response 200 OK pada query string yang tidak membuahkan hasil:



Demikian juga pencarian menggunakan path, Stackexchange akan memberikan response code 200 OK jika data tidak ditemukan, misal mencari user dengan ID tertentu namun tidak ditemukan, sebagai berikut:



namun demikian jika parameter nya invalid misal bukan bertipe number, maka Stackexchange akan memberikan response Bad Request sebagai berikut:



Resume

Agar lebih mudah dibandingkan berikut ini resume hasil pencarian dalam bentuk tabel

No	Domain	Query String	Path	Parameter Tidak Sesuai
1	Twitter.com	200 OK	200 OK	400 Bad Request
2	GitHub.com	200 OK	404 Not Found	-
3	Google.com	200 OK	404 Not Found	-
4	PayPall.com	404 Not Found	200 OK	-
5	AccuWheather.com	200 OK	400 Bad Request 200 OK	400 Bad Request
6	Stackexchange.com (Stackoverflow.com)	200 OK	200 OK	400 Bad Request

Berdasarkan contoh implementasi diatas diketahui bahwa tidak ada standar baku response code untuk data tidak ditemukan, Anda pun bebas memilih menggunakan response code seperti apa, yang penting penggunaannya konsisten. Penulis sendiri lebih memilih menggunakan satus code 200 OK untuk pencarian menggunakan query string dan 404 untuk path tidak ditemukan.

5.3.3. Konsisten

Contoh diatas merupakan sedikit contoh penerapan status code yang berbeda beda, bahkan oleh aplikasi manstream sekalipun. Memang dalam implementasi, masing masing pengembang memiliki interpretasi sendiri sendiri terhadap status code ini, namun demikian yang terpenting adalah ketika mendefinisikan status code, maka Anda harus **konsisten** disemua tempat, karena status kode tersebut akan digunakan client untuk menterjemahkan response yang diterima dari server.

Halaman ini sengaja dikosongkan
Jagowebdev.com

BAB 6 Versioning

Perubahan adalah keniscayaan, begitu juga dengan API, seiring berjalannya waktu, resource mengalami perkembangan, ada yang ditambahkan, ada yang diubah, ada juga yang dikurangi, perubahan ini hendaknya dikelola sedemikian rupa sehingga dapat meminimalisir dampak yang dialami client.

Pada bab awal telah disebutkan bahwa REST API ini ketika direlease dan digunakan oleh publik, maka sulit untuk diubah, karena perubahan pada API akan berpotensi menyebabkan aplikasi client yang bergantung pada API tersebut akan terganggu, pada bab ini kita akan membahas bagaimana mengelola perubahan tersebut sehingga tidak berdampak fatal bagi client yaitu dengan menerapkan versioning pada API.

6.1. Pentingnya Versioning

Mendefinisikan versi pada API menunjukkan kepada user bahwa ada perubahan pada API kita, perubahan tersebut biasanya diberitahukan melalui dokumentasi atau catatan pada changelog.

Ketika mengetahui ada perubahan pada API yang digunakan, karena alasan tertentu, tidak serta merta user ingin menerapkan perubahan tersebut atau bisa juga ingin menerapkan tapi secara bertahap, pada kondisi inilah mengapa penggunaan versi pada API sangatlah penting, dengan demikian versioning yang baik adalah jika perubahan versi tidak berpengaruh ke fungsionalitas versi sebalumnya dan versi sebelumnya tetap berfungsi dengan baik.

6.2. Metode Penomoran

Ketika membahas versioning/versi pasti yang terbayang oleh kita adalah versi pada software, seperti Microsoft Windows 7, 8, 10, PHP 8.2.5, MySQL 5.7, 8.0.32, Bootstrap 4.2, 5.3, dst. Hal ini wajar karena umumnya

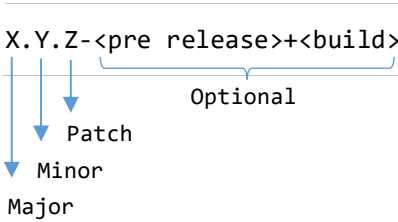
penggunaan versi digunakan oleh software. Pada software setiap perubahan versi ini menunjukkan perubahan pada software tersebut.

Pada API dikenal juga dengan penggunaan versi namun demikian tidak ada standar yang baku bagaimana menerapkan versi pada API, meskipun tidak ada standar yang baku saat ini terdapat model penomoran yang dapat digunakan sebagai panduan penomoran versi, yaitu semantic version dan calendar version.

6.2.1. Semantic Versioning

Pada semantic versioning, penomoran versi menggunakan angka dengan pola tertentu, dokumentasi lengkapnya dapat dibaca dihalaman "Semantic Versioning" yang dapat diakses melalui tautan https://semver.org

Secara ringkas, dapat dikatakan bahwa pada semver, pola penomoran yang digunakan adalah sebagai berikut:



Keterangan:

- Major: Berisi angka yang terus bertambah ketika ada update besar (major update) pada API.
- Minor: Berisi angka yang terus bertambah ketika ada minor update pada API.
- Patch: Berisi angka yang terus bertambah jika ada perbaikan bug pada API.
- Pre Release dan Buid: berisi string yang menunjukan versi pre release dan build misal: alpha, beta, RC1, dll

6.2.1.1. Major

Pada bagian major, nomor akan bertambah ketika ada update pada API yang menyebabkan API yang lama menjadi tidak berfungsi, misal kita memiliki API dengan versi 1.2.1, maka jika kita menerapkan update pada API yang menyebebkan versi 1 (major) tidak dapat digunakan, maka kita perlu menambahkan versi major menjadi versi 2.0.0.

Penambahan nomor pada versi major ini akan mereset nomor pada versi minor dan patch menjadi nol (0) sehingga pada contoh diatas versi api berubah dari 1.2.1 menjadi 2.0.0.

Contoh perubahan major adalah:

- Perubahan pada resource path seperti perubahan URI, penambahan URI, atau ada URI yang dihapus.
- Perubahan pada HTTP Method yang digunakan.
- Perubahan pada parameter yang digunakan untuk mengakses API seperti perubahan tipe data atau perubahan field yang diperlukan, contoh ketika menambahkan (create) data user diperlukan tambahan field NIK.

sebagai gambaran, url berikut menjelaskan perubahan major pada Google Drive API v2 dibanding v3:

https://developers.google.com/drive/api/guides/v2-to-v3-reference#resource_field_differences_between_v2_and_v3

6.2.1.1. Minor

Pembaruan minor terjadi jika terdapat update kecil dari aplikasi dimana pembaruan tersebut tidak mepengaruhi fungsionalitas versi sebelumnya (backward compatibility), sebagai contoh pada versi api 1.0. resource user

dengan URI https://jagowebdev.com/api/v1/user/1 menghasilan response berikut:

```
{
      "username": "budi"
      "nama": "Budi Santoso Wijaya",
      "email": "budi.santosa.wijaya@gmail.com",
}
```

kemudian diperbarui dengan versi v1.1 dengan menambahkan properti lain, sehingga response yang dihasilkan menjadi sebagai berikut:

```
{
      "username": "budi"
      "nama": "Budi Santoso Wijaya",
      "email": "budi.santosa.wijaya@gmail.com",
      "hp": "08719990009",
      "tgl_lahir": "2020-01-07"
}
```

dengan perubahan ini maka client yang sudah terlanjur menggunakan versi 1.0 masih tetap dapat menggunakan api versi tersebut yaitu https://jagowebdev.com/api/v1/user/1 Contoh lain perubahan yang backward complatible dapat dibaca di halaman "Stripe API Documentation" yang dapat diakses melalui tautan https://stripe.com/docs/upgrades#what-changes-does-stripe-consider-to-be-backwards-compatible

Selain penambahan field pada response, pembaruan minor juga dapat terjadi jika terdapat penambahan fitur, method, atau resource pada API, penambahan ini dapat diabaikan oleh pengguna versi sebelumnya jika mereka tidak memerlukannya.

Pembaruan minor ini akan merest nilai versi patch menjadi 0, misal pada versi API 2.1.2 dilakukan pembaruan minor maka versi yang baru berubah menjadi 2.2.0.

6.2.1.2. Patch

Patch merupakan perbaikan error pada API yang sudah ada atau semacam bug fix, sama dengan versi minor, versi patch ini harus backward compatible yang artinya versi yang udah ada harus dapat berjalan dan berfungsi dengan baik.

6.2.1.3. Praktik Dilapangan

Dilapangan penomoran versi URI tidak sedalam seperti yang didefinisikan pada Semantic Versioning, banyak provider API yang hanya menggunakan versi major saja pada URI misal v1, v2, v3 dan beberapa yang menggunakan versi sampai minor saja misal v1.2, v2.3, dll, perubahan versi hanya didokumentasikan saja, tidak diterapkan di URI, hal ini dikarenakan versi minor dan patch bersifat backward compatible.

Dalam parktik meskipun ada perubahan endpoint, banyak provider API yang tidak merelease versi major, mereka mereleasenya dalam waktu yang sangat lama, hal ini dikarenakan perubahan pada versi major memberikan dampak yang besar bagi client maupun server baik dari segi waktu, biaya, dan tenaga, sebagai contoh Twitter merelease API versi 2 pada Agustus 2020, sampai sekarang banyak perubahan pada versi 2 tersebut seperti penghapusan dan penambahan endpoint, namun demikian perubahan tersebut tidak merubah versi hanya didokumentasikan saja, selengkapnya bisa dibaca di halaman Developer Platform Twitter yang dapat diakses melakui tautan https://developer.twitter.com/en/updates/changelog

Contoh lain adalah Stripe, Stripe menggunakan versi API v1, namun demikian padaa versi tersebut banyak sekali perubahan penting yang dilakukan yaitu penambahan dan pengurangan endpoint, Stripe mengelola perubahan tersebut menggunakan calendar, sehingga jika ditanya current version maka akan dijawab 2022-11-15, jika ada ada request dari user maka akan dibaca kapan user tersebut terdaftar selanjutnya api yang digunakan adalah api versi terbaru sesuai bulan dan tahun user tersebut terdaftar, selengkapnya dapat dibaca di halaman APIs as infrastructure yang dapat diakses melalui tautan https://stripe.com/blog/api-versioning

6.2.2. Calendar Versioning

Selain Semantic Versioning, terdapat metode lain yang populer digunakan untuk melakukan versioning yaitu Calendar Versioning, dokumentasi metode ini dapat dibaca di halaman Calendar Versioning yang dapat diakses melalui tautan https://calver.org. Contoh API yang menggunakan calendar versioning adalah Shopify, platform populer yang digunakan untuk membangun toko onlline.

Pada calendar versioning format yang digunakan memuat informasi waktu, bentuk formatnya juga fleksibel, bisa hanya memuat tahun (2023), tahun dan bulan (2023-07), tahun dan urutan release, misal (2023.17), tahun, bulan, dan urutan release (2023-07.1), dll. Dengan model calendar ini maka waktu perubahan API dapat langsung terlihat.

Perusahaan besar yang menggunakan calendar versioning diantaranya Twillo, Shopify, dan Github. Berikut contoh URI API yang digunakan oleh Twillo

```
https://api.twilio.com/2010-04-01/Accounts/CA123abc/Messages.json
```

Pada contoh diatas, Twillo menggunakan YYYY-MM-DD untuk mendefinisikan versi dari API, contoh lainnya dari Shopify sebagai berikut:

```
https://mystore.myshopify.com/admin/api/2022-04/products.json
```

Pada contoh diatas, Shopify menggunakan fomat YYYY-MM untuk mendefinisikan versi API, berikutnya adalah Github, Github menggunakan format sebagai berikut:

```
POST https://api.github.com/repos/OWNER/REPO/branches/BRANCH/rename
Accept: application/vnd.github+json
Authorization: Bearer <YOUR-TOKEN>
X-GitHub-Api-Version: 2022-11-28

{"new_name":"my_renamed_branch"}
```

Pada contoh diatas Github menggunakan format YYYY-MM-DD

6.3. Kapan Tidak Perlu Versioning

Saat ini banyak aplikasi yang dalam pengembangannya memisahkan antara frontend dan backend, seperti misal aplikasi model single page application (SPA). Bagian frontend aplikasi SPA ini biasanya dikembangkan menggunakan Javascript dan bagian backend dikembangkan menggunakan bahasa pemrograman serverside, bisa node.js (javascript), PHP atau yang lainnya, nantinya frontend ini mengakses data pada backend menggunakan API.

Pengembangan aplikasi SPA baik backend maupun frontend ini banyak yang dikembangkan oleh satu atau beberapa programmer, pada kondisi demikian, versioning tidak diperlukan karena tentu fitur API yang digunakan adalah fitur yang terbaru. Dengan tidak menggunakan versioning ini, maka akan membuat pengembangan api menjadi lebih simpel, cepat, dan mudah karena tidak perlu melakukan maintenance API yang sudah tidak digunakan, dengan demikian energi bisa di fokuskan ke pengembangan hal hal yang lebih penting.

6.4. Metode Implementasi

Setidaknya ada empat cara untuk menerapkan versioning pada aplikasi API yaitu:

1. Melalui URI

Cara yang pertama adalah mendefinisikan versi API pada URI, sebagai contoh:

```
https://jagowebdev.com/api/v1/products
```

Pada URI diatas, v1 merupakan varsi major dari API untuk resource produk, selanjutnya ketika versi ini berubah maka angka 1 berubah menjadi 2, misal:

```
https://jagowebdev.com/api/v2/products
```

Model versioning melalui uri ini merupakan model yang paling populer digunakan, perusahaan besar seperti Google, Facebook, Twitter, Stackoverflow, dll menggunakan model ini.

Meskipun banyak digunakan karena banyak memiliki kelebihan, metode ini tidak memenuhi kriteria REST yang dikemukakan Roy Fielding karena satu resource memiliki lebih dari satu endpoint, misal:

```
https://example.com/api/v1/products
https://example.com/api/v2/products
```

Pada contoh diatas, resource products memiliki lebih dari satu endpoint, namun demikian sekali lagi tidak ada standar penerapan untuk REST API, penerapan hendaknya fleksibel sesuai kondisi dilapangan dengan tetap memperhatikan prinsip prinsip REST.

2. Melalui HTTP Header

Pendekatan lain untuk menerapkan Versioning pada API adalah melalui HTTP Header. Dengan pendekatan ini, maka ketika client mengirim request ke server API, maka pada HTTP Header request tersebut perlu menyertakan versi api yang ingin diakses.

Keuntungan dari model ini adalah setiap ada perubahan versi, URI dari API tetap, tidak berubah, satu resource satu URI/pointer/endpoint, sehingga lebih mudah dalam maintenance, selain itu URI juga akan tampak lebih rapi.

Kekurangan dari model ini adalah lebih sulit untuk melakukan testing baik server maupun client, karena perlu menambahkan parameter pada HTTP Header, selain itu lebih rumit ketika melakukan routing pada aplikasi API.

Contoh penggunaan versi pada header adalah sebagai berikut:

```
POST https://api.github.com/repos/OWNER/REPO/branches/BRANCH/rename
Accept: application/vnd.github+json
Authorization: Bearer <YOUR-TOKEN>
X-GitHub-Api-Version: 2022-11-28

{"new_name":"my_renamed_branch"}
```

Pada contoh diatas, Github menggunakan custom HTTP Header untuk mendefinisikan versi yaitu menggunakan `X-Github-Api-Version`

Contoh lain adalah sebagai berikut:

```
curl \
--header "Authorization: Bearer <token>" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
--data '{
  "url":"https://www.forgerock.com/favicon.ico",
  "method":"GET"
}' \
"http://<tenant-env-fqdn>/openidm/external/rest?_action=call"
```

Sumber: https://backstage.forgerock.com/docs/idcloud-idm/latest/rest-api-reference/rest-api-versioning.html

Pada contoh diatas, custom header yang digunakan adalah `Accept-API-Version`

Contoh berikutnya adalah Stripe, perusahaan yang bergerak di sektor keuangan, Stripe menggunakan versioning sebagai berikut:

```
curl https://api.stripe.com/v1/charges \
  -u sk_test_4eC39HqLyjWDarjtT1zdp7dc: \
  -H "Stripe-Version: 2022-11-15"
```

Sumber: https://stripe.com/docs/api/versioning

Pada contoh diatas, custom header yang digunakan Stripe adalah `Stripe-Version`

Note: Pada contoh diatas terlihat bahwa tidak ada standar nama header untuk mendefinisikan versi API, masing masing pengembang mendefinisikan nama sendiri-sendiri.

3. Melalui Query String

Pada pendekatan ini client mendefinisikan versi api melalui parameter query string pada URI, contoh sebagai berikut:

```
https://jagowebdev.com/api/user?version=v1
```

Pada metode ini, versi biasanya ditulis di bagian akhir dari query string, sehingga jika sebelumnnya terdapat parameter lain, penulisan versi menjadi agak berbeda yaitu menggunakan awalah &, misal:

```
https://jagowebdev.com/api/penjuelan?start_date=2023-05-01&version=v1
```

Dari contoh diatas penulisan versi tidak konsisten dan jika query parameternya panjang akan sulit mendeteksi versi apa yang digunakan, selain itu juga rawan untuk lupa menuliskan versi.

Contoh penerapan model ini salah satunya pada Microsoft Azure sebagai berikut:

```
https://{myconfig}.azconfig.io/kv?api-version=1.0
```

Sumber: https://learn.microsoft.com/en-us/azure/azure-app-configuration/rest-api-versioning

4. Melalui Content Negotiation

Metode terakhir yang dapat digunakan untuk melakukan versioning adalah melalui content negotiation, caranya adalah menambahkan versi pada parameter Accept pada HTTP Header sebagai berikut:

```
Accept: application/json;version=1
```

atau

```
Accept: application/v1+json
```

Pada metode ini, yang dilakukan versioning adalah resource nya bukan api secara keseluruhan.

Keuntungan menggunakan metode ini adalah ketika terdapat versi baru maka kita hanya merubah sedikit pada code program yaitu code program yang hanya terkait dengan resource yang diupdate versinya.

Kekurangan dari metode ini adalah sama seperti pada metode versioning menggunakan HTTP Header yaitu lebih sulit untuk melakukan testing baik server maupun client, karena perlu menambah parameter pada HTTP Header, selain itu aplikasi API menjadi lebih kompleks dan manajemen versi menjadi lebih sulit karena untuk melihat versi API, kita harus melihat satu persatu versi pada resourse.

Contoh perusahaan yang menggunakan model versioning seperti ini adalah Adidas, misal sebagai berikut:

```
GET /greeting HTTP/1.1
Accept: application/vnd.example.resource+json; version=2
```

Sumber: https://adidas.gitbook.io/api-guidelines/rest-api-guidelines/message/content-negotiation

URI Versioning VS HTTP Header Versioning

Berdasarkan uraian diatas terlihat berbagai kelebihan dan kekurangan menggunakan versi api pada URI maupun pada HTTP Header, saya sendiri lebih prefer menggunakan URI dengan beberapa pertimbangan sebagai berikut:

- Lebih mudah diakses, dengan versi pada URI, kita dapat menggunakan browser untuk mengakses api.
- Developer server maupun client dapat dengan mudah mengetahui versi api yang digunakan.

- Lebih reliable, aman dari kesalahan penulisan, dengan menerapkan versi pada URI, maka kemungkinan client lupa menuliskan versi akan kecil, hal ini berbeda jika versi diterapkan pada HTTP Header.
- Ketika diimplementasikan pada aplikasi (coding), lebih mudah menerapkan routing nya.

6.5. URI Versioning: Penulisan Versi

Pada bagian sebelumnya telah disebutkan bahwa banyak perusahaan besar yang menggunakan URI sebagai metode versioning API, dalam praktik, setidaknya ada dua model yang dapat diterapkan untuk penulisan versi menggunakan metode ini yaitu dengan tambahan prefix v atau tanpa tambahan prefix v, berikut beberapa contoh penerapan URI versioning pada beberapa vendor besar:

Vendor	URL
Google	https://www.googleapis.com/drive/v3/files/fileId
Twitter	https://api.twitter.com/2/users
PayPall	https://api-m.paypal.com/v2/invoicing/invoices
Tokopedia	https://fs.tokopedia.net/inventory/v1/fs/13004/product/info
AccuWheather	http://dataservice.accuweather.com/locations/v1/cities/search
Stackoverflow	https://api.stackexchange.com/2.3/users?site=stackoverflow
Facebook	https://graph.facebook.com/v4.0/{my-user-id}&access_token={access-token} (https://developers.facebook.com/docs/graph-api/guides/versioning)

pada contoh diatas, terlihat terdapat perbedaan penulisan nomor versi, ada yang menggunakan prefix/awalan v ada juga yang tidak menggunakan awalan v.

Awalan v pada versi ini tidak mencerminkan semantic value melainkan untuk menandai bahwa nomor tersebut adalah nomor versi, dengan menambahkan awalan v maka akan lebih bermakna dan jelas, penulis

pribadi lebih memilih menggunakan awalah v karena dapat langsung diketahui bahwa maksud dari nomor tersebut adalah nomor versi, namun demikian Anda bebas memilih apakah ingin menggunakan awalan v atau tidak.

Pada contoh diatas juga terlihat bahwa ada beberapa vendor yang hanya menuliskan versi major saja, tanpa versi minor maupun patch, hal ini dikarenakan versi minor maupun patch bersifat backward compatible sehingga ketika client merequest dengan versi major tertentu maka akan diberikan response dengan versi minor dan patch terbaru, cara ini akan memudahkan client karena tidak perlu merubah URI ketika terdapat update versi minor maupun patch.

Halaman ini sengaja dikosongkan
Jagowebdev.com

BAB 7 Codeigniter 4

Aplikasi REST API yang kita kembangkan menggunakan framework PHP Codeigniter 4 dengan pertimbangan utama bahwa framework ini cepat dan ringan selain itu framework ini didukung secara resmi dalam jangka waktu yang panjang baik dalam penambahan fitur maupun perbaikan bug, tidak seperti framework lain seperti Laravel dimana dukungan untuk versi major hanya dua tahun. Dalam konteks pengembangan REST API framework Codeigniter 4 ini juga sudah didesain sedemikian rupa sehingga mudah digunakan untuk mengembangkan REST API diantaranya sebagai berikut:

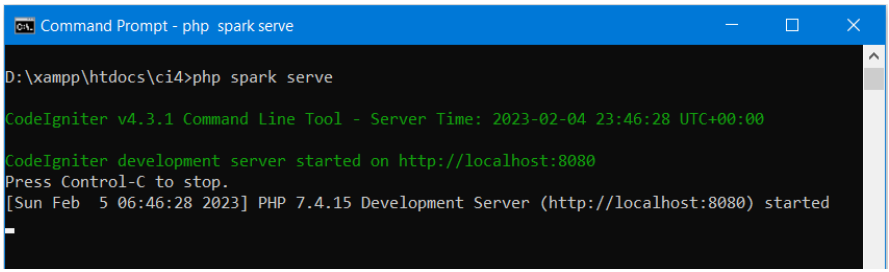
- pendefinisian routing yang fleksibel. Routing telah mendukung berbagai method (get, put, post, delete, dll)
- pengelolaan request yang mudah. Codeigniter 4 telah dilengkapi fitur yang memudahkan pengembang untuk mengelola request yang masuk ke server seperti membaca content-type header, membaca data HTTP Body, dll.
- pengelolaan response yang mudah. Codeigniter 4 juga telah dilengkapi fitur yang memungkinkan pengembang mengelola response yang dikirim ke client dengan mudah, seperti mendefinisikan response code, mendefinisikan format data, mendefinisikan response sukses dan error, dll.
- built-in validasi yang dapat digunakan untuk memastikan data/form input yang dikirim oleh user sesuai dengan kriteria yang telah ditetapkan.

7.1. Instalasi

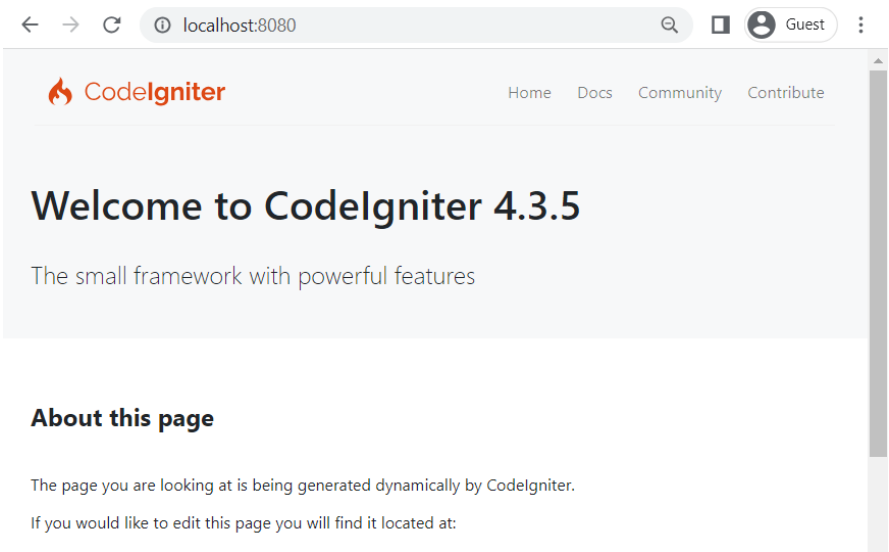
Agar lebih mudah memahami fitur REST API yang ada pada Codeigniter 4, mari kita langsung mencoba praktik, pertama tama download Codeigniter 4 dari website resminya di tautan https://codeigniter.com/download,

kemudian install/ekstrak ke PC Anda,. Pada kesempatan kali ini saya menggunakan Codeigniter versi 4.3.5 yang saya install di D:\xampp\htdocs\ci4\, adapun versi PHP yang saya gunakan adalah versi 8.2.4 (xampp versi 8.2.4).

Setelah proses instalasi selesai selanjutnya jalankan Codeigniter 4 menggunakan local development server yang disertakan pada file download, contoh sebagai berikut:



Selanjutnya buka browser dan buka alamat http://localhost:8080 jika berhasil maka tampilan yang kita peroleh adalah sebagai berikut:



Setelah berhasil menginstall Codeigniter 4, selanjutnya aktifkan konfigurasi autoroute (ubah false menjadi true) yang ada pada file Routes.php (app\Config\Routes.php) sebagai berikut:

Dengan autoroute aktif maka ketika Codeigniter 4 memparsing alamat URI Codeigniter otomatis akan menentukan controller dan method yang akan dieksekusi sehingga kita tidak perlu mendefinisikan routing secara manual, misal pada URI http://localhost:8080/artikel/add maka Codeigniter akan otomatis meload file controller Artikel.php (app\Controllers\Artikel.php) kemudian menginisiasi class Artikel yang ada di file tersebut dan mengekseskusi method add().

Autoroute ini hanya kita gunakan pada percobaan ini saja, nantinya ketika mengembangkan aplikasi REST API kita harus mendefinisikan routing secara manual karena memang pemetaan URI nya tidak bisa dilakukan secara otomatis.

7.2. Alur Codeigniter 4

Pada framework Codeigniter 4 setiap request yang masuk akan diproses berurutan mulai dari:

1. Filters (app\Filters). Pada bagian ini kita bisa mendefinisikan berbagai hal sebelum request diproses di controller, seperti mendefinisikan HTTP Response Header yang akan dikirim ke client, mengecek CSRF token, dll.

2. Router (app\Config\Router.php). Pada bagian ini akan ditentukan controller dan method yang akan dieksekusi berdasarkan URI yang diterima.

3. Controller (app\Controllers). Pada bagian ini file controller akan di load, class diinisiasi dan class method dieksekusi sesuai dengan yang telah ditentukan pada bagian router.

7.3. Request dan Response

Codeigniter 4 menyediakan berbagai fitur (method/fungsi) yang dapat memudahkan kita untuk mengelola request dan response, pada bagian ini kita bahas beberapa fitur yang dapat kita gunakan untuk mengembangkan aplikasi REST API.

7.3.1 Mendapatkan Request Header

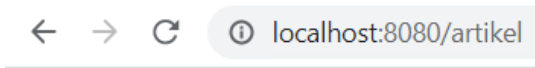
Pada aplikasi REST API yang kita kembangkan, setiap request yang masuk ke server terlebih dahulu kita cek HTTP headernya, hal ini dilakukan untuk berbagai keperluan seperti membaca token pada header, membaca format data pada HTTP Body, dll. Pada Codeigniter 4 pembacaan header ini sudah dihandle oleh sistem Codoeigniter, kita hanya perlu menginisiasinya saja.

Sebagai contoh, pada Codeigniter 4 yang telah kita install sebelumnya kita buat file controller Artikel.php (app\Controllers\Artikel.php), selanjutnya pada file Artikel.php tersebut kita isi dengan script berikut:

```
namespace App\Controllers;

class Artikel extends BaseController
{
    public function index()
    {
      $request = \Config\Services::request();
      echo $request->getMethod() . '<br/>' .
            $request->getHeaderLine('accept-language');
    }
}
```

Note: Pada script diatas, method getMethod() digunakan untuk mendapatkan value/nilai dari HTTP Method dan method getHeaderLine() digunakan untuk mendapatkan value/nilai dari HTTP Header tertentu, yang pada contoh diatas adalah Header accept-language.

Jika script diatas dijalankan pada browser, maka hasil yang kita peroleh adalah sebagai berikut:



get
en-US,en;q=0.9
get
en-US,en;q=0.9

Selain kedua method diatas, Codeigniter 4 juga menyediakan berbagai method lain untuk mengelola request yang masuk ke server, seperti getPost() untuk mendapatkan POST data pada variable $_POST, method getJSON() untuk mendapatkan data JSON pada HTTP Body, dll, selengkapnya dapat dibaca pada halaman dokumentasi "IncomingRequest Class" yang dapat diakses melalui tautan: https://codeigniter.com/user_guide/incoming/incomingrequest.html

7.3.2. Mengirim Response

Codeigniter 4 telah menyediakan berbagai fitur yang dapat digunakan untuk memudahkan kita mengirim HTTP response ke user/client, salah satunya adalah menggunakan class method respond(), dengan class method ini maka secara otomatis Codeigniter akan mendefinisikan HTTP Response sesuai parameter yang diberikan sehingga kita tidak perlu repot mendifinisikannya secara manual.

Class method respond() pada Codeigniter 4 berisi tiga argumen yaitu data, status code, dan message, adapun susunannya adalah sebagai berikut:

```
respond($data[, $statusCode = 200[, $message = '']])
```

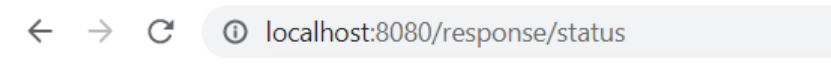
Dari tiga argumen tersebut diatas, argumen status code dan message bersifat opsional, tidak harus diisi.

Agar lebih memahami method respond() ini, mari kita coba menjalankan method tersebut, pada Codeigniter 4 yang telah kita install sebelumnya buat file Response.php pada folder app\Controllers, pada file tersebut kita buat script sebagai berikut:

```
<?php
namespace App\Controllers;
use CodeIgniter\RESTful\ResourceController;

class Response extends ResourceController
{
    public function index() {}

      public function status() {
            $result = ['status' => 'ok'
                        , 'data' => [
                              'nama' => 'Agus Prawoto Hadi'
                              , 'website' =>
'https://jagowebdev.com'
                        ]
                    ];
            return $this->respond($result);
      }
}
```

Note: Untuk dapat menggunakan method respond(), maka class harus kita extend ke `ResourceController` seperti contoh diatas.

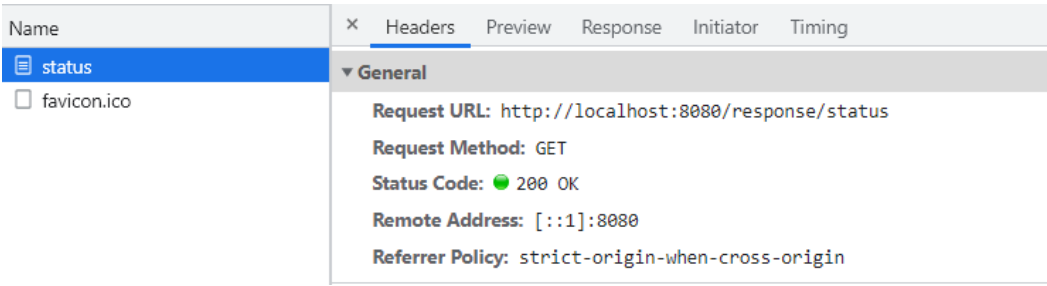
Selanjutnya pada browser jalankan URI http://localhost:8080/response/status, jika berhasil, maka hasil yang kita peroleh adalah sebagai berikut:




```
{
    "status": "ok",
    "data": {
        "nama": "Agus Prawoto Hadi",
        "website": "https://jagowebdev.com"
    }
}
```

Pada contoh diatas, terlihat bahwa response yang dihasilkan method ini berbentuk JSON, karena memang secara default method respond() ini akan

mendefinisikan Header Content-type dengan nilai application/json dan format data pada HTTP Body berbentuk JSON, lebih lanjut mengenai hal ini dibahas pada sub bab berikutnya, selanjutnya jika kita cek pada developer tools browser terlihat bahwa status code yang dihasilkan adalah 200 OK



Selain method respond() Codeigniter 4 juga menyediakan berbagai method lainnya yang dapat digunakan untuk mempermudah mendefinisikan response, yaitu:

```
// Generic failure response
$this->fail($errors, 400);

// Item created response
$this->respondCreated($data);

// Item successfully deleted
$this->respondDeleted($data);

// Command executed by no response required
$this->respondNoContent($message);

// Client isn't authorized
$this->failUnauthorized($description);

// Forbidden action
$this->failForbidden($description);

// Resource Not Found
$this->failNotFound($description);

// Data did not validate
$this->failValidationError($description);

// Resource already exists
```

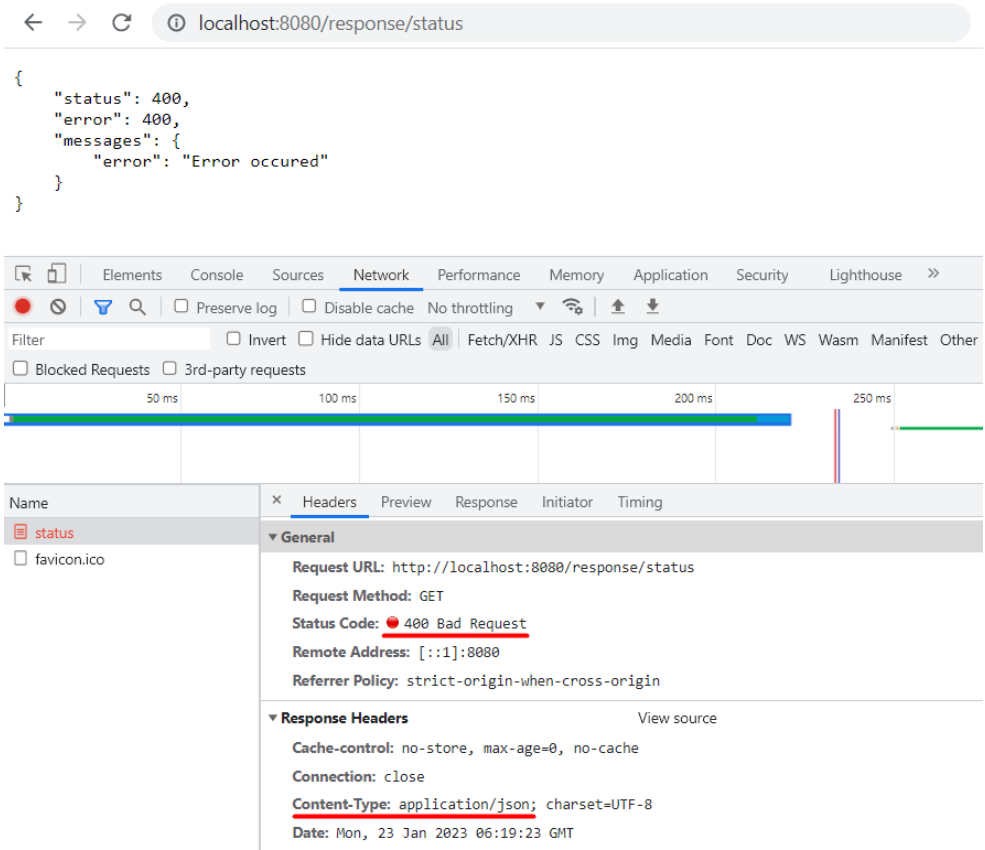
```
$this->failResourceExists($description);

// Resource previously deleted
$this->failResourceGone($description);

// Client made too many requests
$this->failTooManyRequests($description);
```

Lebih lanjut mengenai method diatas dapat dibaca di halaman "API Response Trait" yang dapat diakses pada halaman https://codeigniter4.github.io/userguide/outgoing/api_responses.html

Agar lebih memahami method diatas, mari kita coba salah satu method yaitu method fail(). Buka file app\Controllers\Response.php dan ubah method status menjadi seperti berikut:

```
<?php
namespace App\Controllers;
use CodeIgniter\RESTful\ResourceController;

class Response extends ResourceController
{
    public function index() {}

      public function status() {
            return $this->fail('Error occured');
      }
}
```

Ketika dijalankan pada browser maka hasil yang kita peroleh adalah sebagai berikut:



Pada gambar diatas terlihat bahwa Status Code yang dihasilkan adalah 400, Content-Type nya bernilai application/json, dan data JSON yang dihasilkan adalah:

```
{
    "status": 400,
    "error": 400,
    "messages": {
        "error": "Error occured"
    }
}
```

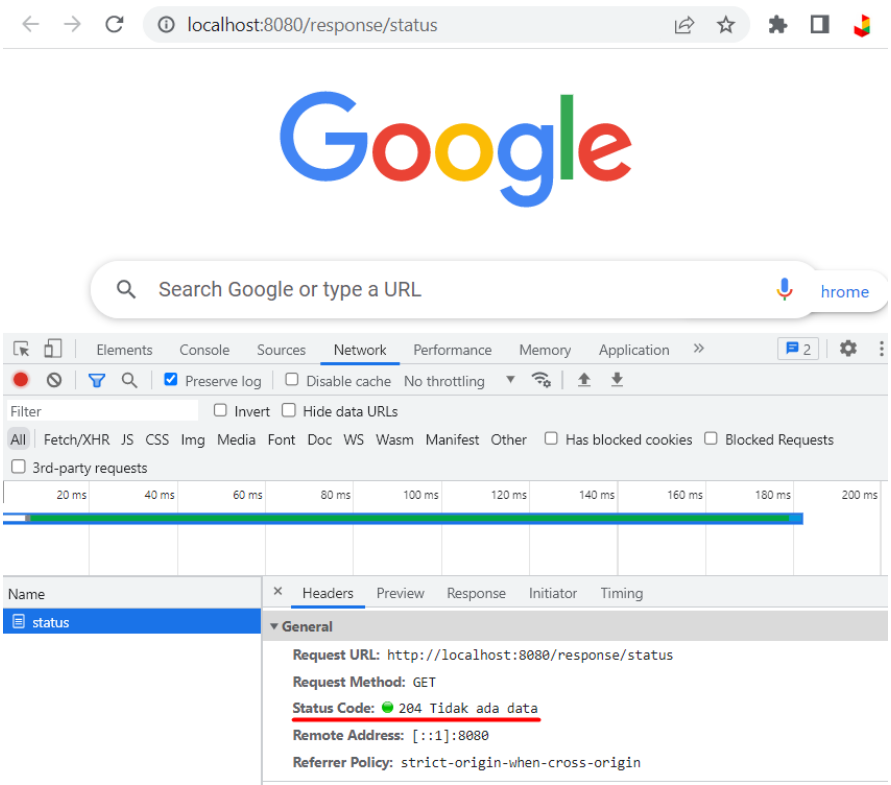
Bentuk data JSON diatas berbeda dengan bentuk data JSON yang dihasilkan oleh response code 200 OK yang ada pada contoh sebelumnya, pada response code 200 terdapat data yang disertakan (seperti data user,

penjualan, dll) sedangkan pada response 400 tidak ada data, hanya message saja.

Lain lagi dengan respondNoContent(). Pada method respondNoContent(), dengan status code 204 benar benar tidak ada data yang dikirim oleh server, mari kita coba dengan merubah method status pada file app\Controllers\Response.php menjadi sebagai berikut:

```
<?php
namespace App\Controllers;
use CodeIgniter\RESTful\ResourceController;

class Response extends ResourceController
{
    public function index() {}

      public function status() {
            return $this->respondNoContent('Tidak ada data');
      }
}
```

Ketika kita jalankan pada browser, hasil yang kita peroleh adalah sebagai berikut:



Ketika kita klik tab Respose pada Developer Tools browser kita tidak akan menemukan apa apa alias kosong, kenapa demikian, karena sesuai standar yang ada (https://www.rfc-editor.org/rfc/rfc7231#page-53) tidak ada response body yang dikirim pada status code 204.

Contoh diatas hanya beberapa method dari sekian banyak method yang disediakan oleh Codeigniter 4, untuk method lainnya dapat Anda coba coba sendiri sehingga Anda menjadi familiar dengan response yang dihasilkan oleh method tersebut sehingga nantinya, dalam praktik, Anda akan langusng tahu method apa yang seharusnya digunakan.

Berbagai respon method yang disediakan Codeigniter (seperti fail(), respondNoContent(), dll) pada dasarnya adalah shorthand method dari method respond(), hal ini dapat dilihat di file system\API\ResponseTrait.php.

Sebagai contoh, script method fail() sebagai berikut:

```
<?php
protected function fail($messages, int $status = 400, ?string
$code = null, string $customMessage = '')
{
      if (! is_array($messages)) {
            $messages = ['error' => $messages];
      }

      $response = [
            'status'   => $status,
            'error'    => $code ?? $status,
            'messages' => $messages,
      ];

      return $this->respond($response, $status, $customMessage);
}
```

Pada script diatas terlihat bahwa method fail memberikan nilai kembalian (return) berupa method respond() dengan argumen tertentu.

Dengan demikian, untuk alasan tertentu, misal keseragaman penulisan dan keseragaman pesan yang dihasilkan, Anda dapat menggunakan satu method saja yaitu method respond().

Note: Pada aplikasi api yang kita bangun, saya terkadang menggunakan shorthand method tapi kebanyakan menggunakan method respond(), sehingga dapat lebih leluasa mendefinisikan pesan dan status code sendiri.

Pada berbagai shorthand method diatas Codeigniter 4 secara otomatis memberikan response code sesuai dengan shorthand method yang digunakan, adapun daftar response code yang digunakan adalah sebagai berikut:

```
protected $codes = [
      'created'                   => 201,
      'deleted'                   => 200,
      'updated'                   => 200,
      'no_content'                => 204,
      'invalid_request'           => 400,
      'unsupported_response_type' => 400,
      'invalid_scope'             => 400,
```

```
        'temporarily_unavailable'   => 400,
        'invalid_grant'             => 400,
        'invalid_credentials'       => 400,
        'invalid_refresh'           => 400,
        'no_data'                   => 400,
        'invalid_data'              => 400,
        'access_denied'             => 401,
        'unauthorized'              => 401,
        'invalid_client'            => 401,
        'forbidden'                 => 403,
        'resource_not_found'        => 404,
        'not_acceptable'            => 406,
        'resource_exists'           => 409,
        'conflict'                  => 409,
        'resource_gone'             => 410,
        'payload_too_large'         => 413,
        'unsupported_media_type'    => 415,
        'too_many_requests'         => 429,
        'server_error'              => 500,
        'unsupported_grant_type'    => 501,
        'not_implemented'           => 501,
];
```

Response code ini dapat dilihat pada file system\API\ResponseTrait.php

7.3.3. Mendefinisikan Content-Type

Pada Codeigniter ketika kita menggunakan method respond() maka otomatis content yang dikirim berbentuk json dan Codeigniter secara otomatis mendefinisikan nilai content-type pada header dengan nilai application/json.

Format data yang yang dikirim (nilai content-type pada header) ini dapat diatur pada property `$supportedResponseFormats` yang ada pada file app\Config\Format.php, secara default nilai property tersebut adalah sebagai berikut:

```
public array $supportedResponseFormats = [
        'application/json',
        'application/xml', // machine-readable XML
        'text/xml', // human-readable XML
];
```

Pada property diatas, Codeigniter 4 akan membaca urut dari atas ke bawah, pertama tama Codeigniter akan mencoba menggunakan format application/json, namun jika tidak tersedia maka format yang digunakan adalah application/xml, jika tidak bisa juga maka format yang digunakan adalah text/html.

Pendefinisian format ini terlihat dari source code method respond() pada file system\API\RespondTrait.php sebagai berikut:

```
protected function respond($data = null, ?int $status = null,
string $message = '')
{
      if ($data === null && $status === null) {
            $status = 404;
            $output = null;
      } elseif ($data === null && is_numeric($status)) {
            $output = null;
      } else {
            $status = empty($status) ? 200 : $status;
            $output = $this->format($data);
      }

      if ($output !== null) {
            if ($this->format === 'json') {
                  return $this->response->setJSON($output)-
>setStatusCode($status, $message);
            }

            if ($this->format === 'xml') {
                  return $this->response->setXML($output)-
>setStatusCode($status, $message);
            }
      }

      return $this->response->setBody($output)-
>setStatusCode($status, $message);
}
```

7.4. Routing

Setelah melakukan pengecekan pada filters, selanjutnya dilakukan proses routing yaitu memparsing URI sehingga diperoleh file controller yang akan di load dan class method yang akan dieksekusi.

Pada pembahasan bab sebelumnya disebutkan bahwa pada REST API satu URI bisa mengandung perintah bermacam macam tergantung method yang digunakan (GRT, POST, PUT, DELETE), sehingga pada bagian routing ini, selain mendefinisikan URI, kita juga harus mendefinisikan method yang digunakan.

Codeigniter 4 memungkinkan kita untuk mendefinisikan method pada routing sehingga memudahkan kita untuk membuat routing untuk REST API, misal sebagai berikut:

```
$routes->put("artikel", "Artikel::edit");
```

Pada routing diatas, jika ada URI artikel dengan request method PUT, maka file controller yang akan diload adalah Artikel.php (app\Controllers\Artikel.php) dan class method yang dieksekusi adalah method edit().

Sebelum membahas lebih lanjut, mungkin Anda bertanya-tanya, pada REST API ini bisakah kita menggunakan autoroute sehingga tidak perlu mendefinisikan routing secara manual?

Autorouting ini dapat digunakan jika request dilakukan hanya menggunakan method get (seperti aplikasi web pada umumnya). sedangkan pada REST API terdapat method lain selain get, satu URI bisa bermacam macam method sehingga mau tidak mau kita harus mendefinisikan routing secara manual.

Jadi kesimpulannya adalah tidak ada cara yang mudah untuk melakukan routing ketika mengembangkan Aplikasi REST API, kita harus mendefinisikan routing secara manual.

Mari kita kembali membahas pendefinisian REST API routing pada Codeigniter 4.

Pada aplikasi Codeigniter 4 yang telah kita install sebelumnya, mari kita coba mendefinisikan routing sebagai berikut:

```
$routes->put("artikel", "Artikel::edit");
```

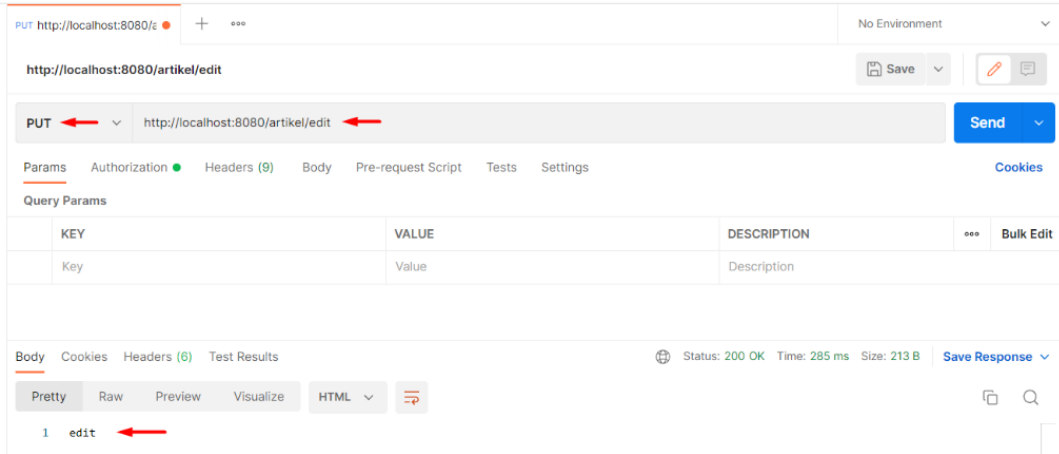
Pada file app\Config\Routes.php script nya menjadi seperti berikut

```
/*
 * --------------------------------------------------------------------
 * Router Setup
 * --------------------------------------------------------------------
 */
$routes->setDefaultNamespace('App\Controllers');
$routes->setDefaultController('Home');
$routes->setDefaultMethod('index');
$routes->setTranslateURIDashes(false);
$routes->set404Override();
// The Auto Routing (Legacy) is very dangerous. It is easy to create vulnerable apps
// where controller filters or CSRF protection are bypassed.
// If you don't want to define all routes, please use the Auto Routing (Improved).
// Set `$autoRoutesImproved` to true in `app/Config/Feature.php` and set the following to true.
$routes->setAutoRoute(true);

/*
 * --------------------------------------------------------------------
 * Route Definitions
 * --------------------------------------------------------------------
 */

// We get a performance increase by specifying the default
// route since we don't have to scan directories.
$routes->get('/', 'Home::index');
$routes->put("artikel", "Artikel::edit");
```

selanjutnya mari kita tes apakah routing tersebut dapat berjalan dengan baik, pada file app\Controllers\Artikel.php kita tambahkan method edit(), sehingga keseluruhan script menjadi sebagai berikut:

```
namespace App\Controllers;

class Artikel extends BaseController
{
    public function index()
    {
      $request = \Config\Services::request();
            echo $request->getMethod() . '<br/>' .
                  $request->getHeaderLine('accept-language')  .
'<br/>';

            echo $this->request->getMethod()  . '<br/>' .
                  $this->request->getHeaderLine('accept-language');

        // return view('welcome_message');
    }

      public function edit() {
            echo 'edit';
      }
}
```

selajutnya mari kita tes menggunakan aplikasi postman dengan mengakses url: http://localhost/ci4/artikel dengan pilihan method PUT



Pada contoh diatas, response (bagian tab Body sebelah bawah) yang dihasilkan adalah teks "edit". Hal menunjukkan bahwa routing yang kita definisikan telah berjalan dengan baik.

Note: Pada saat melakukan pengujian kita tidak dapat menggunakan browser karena browser hanya bisa mengirim data dengan method get (melalui url) dan post (melalui form).

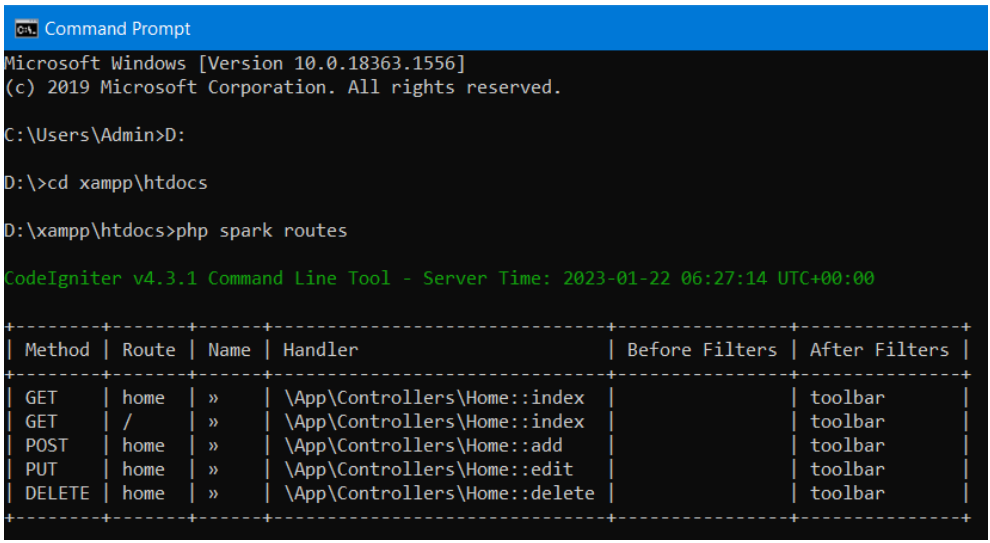
Pada pembahasan sebelumnya telah dibahas bahwa pada REST API kita tidak bisa menggunakan autoroute sehingga kita perlu mendefinisikan secara manual setiap method pada routing, sebenarnya hal ini hanya berlaku untuk method selain get saja, namun demikian agar lebih konsisten, semua method baik get dan post kita definisikan secara manual.

Melanjutkan contoh sebelumnya, kita lanjutkan dengan mendefinisikan method get, post, put, dan delete sebagai berikut:

```
$routes->get("artikel", "Artikel::index");
$routes->put("artikel ", "Artikel::edit");
$routes->post("artikel ", "Artikel::add");
$routes->delete("artikel ", "Artikel::delete");
```

Note: Anda bebas menggunakan nama class method, misal edit bisa diganti update, add bisa diganti create, dll.

Untuk memastikan bahwa routing tersebut telah didefinisikan dengan benar, mari kita cek semua routing yang tersedia, buka command prompt dan ketikkan perintah php sparks routes, jika berhasil maka hasil yang kita peroleh adalah sebagai berikut:



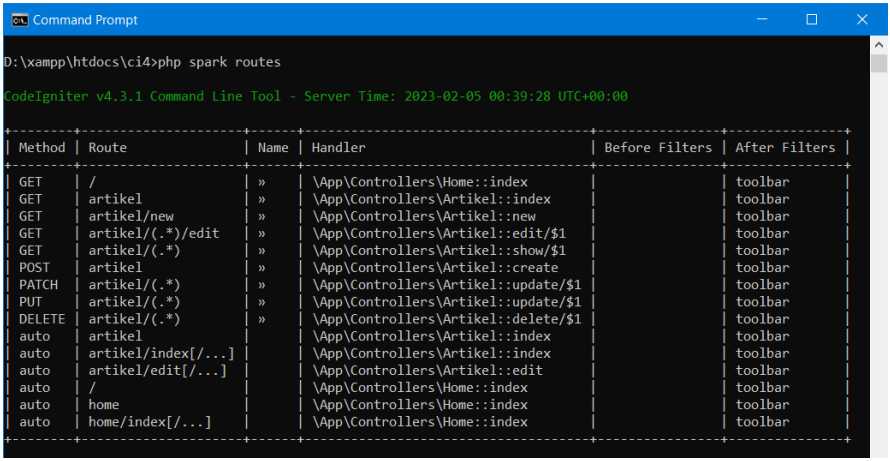
pada gambar diatas terlihat bahwa routing telah sesuai dengan telah yang kita definisikan.

Pendefinisian routing diatas dapat diringkas dengan shorthand routing yang telah disediakan oleh Codeigniter 4, dengan shorthand ini kita cukup mendefinisikan sekali maka routing untuk semua method otomatis terdefinisikan, misal sebagai berikut:

```
$routes->resource('artikel');
```

```php
/*
 * --------------------------------------------------------------------
 * Router Setup
 * --------------------------------------------------------------------
 */
$routes->setDefaultNamespace('App\Controllers');
$routes->setDefaultController('Home');
$routes->setDefaultMethod('index');
$routes->setTranslateURIDashes(false);
$routes->set404Override();
// The Auto Routing (Legacy) is very dangerous. It is easy to create vulnerable apps
// where controller filters or CSRF protection are bypassed.
// If you don't want to define all routes, please use the Auto Routing (Improved).
// Set `$autoRoutesImproved` to true in `app/Config/Feature.php` and set the following to true.
$routes->setAutoRoute(true);

/*
 * --------------------------------------------------------------------
 * Route Definitions
 * --------------------------------------------------------------------
 */

// We get a performance increase by specifying the default
// route since we don't have to scan directories.
$routes->get('/', 'Home::index');
$routes->resource('artikel');
```

Maka secara otomatis routing yang didefinisikan adalah sebagai berikut:



Method	Route	Name	Handler	Before Filters	After Filters
GET	/	»	\App\Controllers\Home::index		toolbar
GET	artikel	»	\App\Controllers\Artikel::index		toolbar
GET	artikel/new	»	\App\Controllers\Artikel::new		toolbar
GET	artikel/(.*)/edit	»	\App\Controllers\Artikel::edit/$1		toolbar
GET	artikel/(.*)	»	\App\Controllers\Artikel::show/$1		toolbar
POST	artikel	»	\App\Controllers\Artikel::create		toolbar
PATCH	artikel/(.*)	»	\App\Controllers\Artikel::update/$1		toolbar

```
| PUT    | artikel/(.*)        | »    | \App\Controllers\Artikel::update/$1 |               | toolbar      |
| DELETE | artikel/(.*)        | »    | \App\Controllers\Artikel::delete/$1 |               | toolbar      |
| auto   | artikel             |      | \App\Controllers\Artikel::index     |               | toolbar      |
| auto   | artikel/index[/...] |      | \App\Controllers\Artikel::index     |               | toolbar      |
| auto   | artikel/edit[/...]  |      | \App\Controllers\Artikel::edit      |               | toolbar      |
| auto   | /                   |      | \App\Controllers\Home::index        |               | toolbar      |
| auto   | home                |      | \App\Controllers\Home::index        |               | toolbar      |
| auto   | home/index[/...]    |      | \App\Controllers\Home::index        |               | toolbar      |
+--------+---------------------+------+-------------------------------------
--+----------------+---------------+
```

Note: Pendefinisian routing diatas mensyaratkan nama Controller sama dengan alamat URI yaitu sama sama artikel, alamat URI: http://localhost:8080/artikel dan file controller Artikel.php (app\Controllers\Artikel.php), jika nama Controller berbeda dengan alamat URI maka kita harus mendefinisikan nama controllernya secara manual, misal jika pada URI artikel/kategori (http://localhost:8080/artikel/kategori) controller yang digunakan adalah Artikel_kategori (app\Controllers\Artikel_kategori.php) maka pendefinisian routing nya adalah

```
$routes->resource('artikel/kategori', ['controller'
=>'Artikel_kategori']);
```

7.4.1. Default method

Ketika kita menggunakan shorthand routing seperti contoh sebelumnya maka Codeigniter otomatis akan menggunakan class method sesuai dengan yang telah ditentukan oleh Codeigniter sendiri, misal

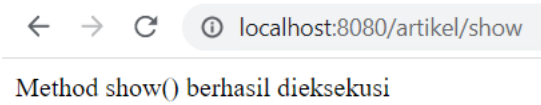
```
+--------+--------------------+------+-------------------------------------+
| Method | Route              | Name | Handler                             |
+--------+--------------------+------+-------------------------------------+
| GET    | /                  | »    | \App\Controllers\Home::index        |
| GET    | artikel            | »    | \App\Controllers\Artikel::index     |
| GET    | artikel/new        | »    | \App\Controllers\Artikel::new       |
| GET    | artikel/(.*)/edit  | »    | \App\Controllers\Artikel::edit/$1   |
| GET    | artikel/(.*)       | »    | \App\Controllers\Artikel::show/$1   |
| POST   | artikel            | »    | \App\Controllers\Artikel::create    |
| PATCH  | artikel/(.*)       | »    | \App\Controllers\Artikel::update/$1 |
| PUT    | artikel/(.*)       | »    | \App\Controllers\Artikel::update/$1 |
| DELETE | artikel/(.*)       | »    | \App\Controllers\Artikel::delete/$1 |
| auto   | artikel            |      | \App\Controllers\Artikel::index     |
| auto   | artikel/index[/...] |      | \App\Controllers\Artikel::index     |
| auto   | artikel/edit[/...] |      | \App\Controllers\Artikel::edit      |
```

```
| auto   | /                  |      | \App\Controllers\Home::index        |
| auto   | home               |      | \App\Controllers\Home::index        |
| auto   | home/index[/...]   |      | \App\Controllers\Home::index        |
+--------+--------------------+------+-------------------------------
-------+
```

Pada routing diatas, pada URI artikel/(.*) dengan method GET, Class dan method yang digunakan adalah Artikel::show/$1 sehingga kita perlu mendefinisikan method show pada class Artikel pada file controller Artikel.php (app\Controllers\Artikel.php), misal:

```php
namespace App\Controllers;

class Artikel extends BaseController
{
      public function index()
      {
            $request = \Config\Services::request();
            echo $request->getMethod() . '<br/>' .
                  $request->getHeaderLine('accept-language')  . '<br/>';

            echo $this->request->getMethod()  . '<br/>' .
                  $this->request->getHeaderLine('accept-language');

        // return view('welcome_message');
      }

      public function edit() {
            echo 'edit';
      }

      public function show() {
            echo 'Method show() berhasil dieksekusi';
      }
}
```

Jika kita jalankan pada browser, hasil yang kita peroleh adalah sebagai berikut:



jika class Artikel ini mengextend class ResourceController, dan nantinya untuk mengembangkan REST API setiap class pada controller harus mengextend class ResourceController, maka argumen pada method ini harus sama dengan argumen method show() yang ada di class ResourceController yaitu `show($id = null)`, hal ini juga berlaku untuk method lainnya seperti edit, update, create, delete, dll

```
+--------+--------------------+------+-----------------------------------+
| Method | Route              | Name | Handler                           |
+--------+--------------------+------+-----------------------------------+
| GET    | /                  | »    | \App\Controllers\Home::index      |
| GET    | artikel            | »    | \App\Controllers\Artikel::index   |
| GET    | artikel/new        | »    | \App\Controllers\Artikel::new     |
| GET    | artikel/(.*)/edit  | »    | \App\Controllers\Artikel::edit/$1 |
| GET    | artikel/(.*)       | »    | \App\Controllers\Artikel::show/$1 |
| POST   | artikel            | »    | \App\Controllers\Artikel::create  |
| PATCH  | artikel/(.*)       | »    | \App\Controllers\Artikel::update/$1 |
| PUT    | artikel/(.*)       | »    | \App\Controllers\Artikel::update/$1 |
| DELETE | artikel/(.*)       | »    | \App\Controllers\Artikel::delete/$1 |
| auto   | artikel            |      | \App\Controllers\Artikel::index   |
| auto   | artikel/index[/...] |      | \App\Controllers\Artikel::index   |
| auto   | artikel/edit[/...] |      | \App\Controllers\Artikel::edit    |
| auto   | /                  |      | \App\Controllers\Home::index      |
| auto   | home               |      | \App\Controllers\Home::index      |
| auto   | home/index[/...]   |      | \App\Controllers\Home::index      |
+--------+--------------------+------+-----------------------------------
-------+
```

Class ResourceController ini sendiri ada di folder system\RESTful\ResourceController.php.

Sebagai contoh Class Artikel pada script sebelumnya kita extend ke Class ResourceController sebagai berikut:

```
namespace App\Controllers;
use CodeIgniter\RESTful\ResourceController;

class Artikel extends ResourceController
{
      public function index()
```

```
    {
        $request = \Config\Services::request();
        echo $request->getMethod() . '<br/>' .
            $request->getHeaderLine('accept-language')  . '<br/>';

        echo $this->request->getMethod()  . '<br/>' .
            $this->request->getHeaderLine('accept-language');

      // return view('welcome_message');
    }

    public function edit() {
        echo 'edit';
    }

    public function show() {
        echo 'Method show() berhasil dieksekusi';
    }
}
```

selanjutnya jika jalankan url http://localhost:8080/artikel/show pada browser maka hasil yang kita peroleh adalah pesan error sebagai berikut:

Pesan error ini disebabkan karena argumen pada method show() yang ada di Class Artikel tidak sama dengan argumen pada method show() yang ada di Class ResourceController (system\RestFul\ResourceController.php), adapun method show() yang ada di Class ResourceController memiliki argumen $id dengan nilai default null sebagai berikut:

```
class ResourceController extends BaseResource
{
    // Script lainnya
    public function show($id = null)
    {
```

```
        return $this->fail(lang('RESTful.notImplemented',
['show']), 501);
    }
      // Script lainnya
}
```

Untuk mengatai error tersebut maka method show harus kita ubah menjadi `show($id = null)` demikian juga dengan method edit(), sehingga method show() dan edit() pada Class Artikel menjadi seperti berikut:

```
namespace App\Controllers;
use CodeIgniter\RESTful\ResourceController;

class Artikel extends ResourceController
{
    public function index()
    {
            $request = \Config\Services::request();
            echo $request->getMethod() . '<br/>' .
                  $request->getHeaderLine('accept-language')  .
'<br/>';

            echo $this->request->getMethod()  . '<br/>' .
                  $this->request->getHeaderLine('accept-language');

        // return view('welcome_message');
    }

      public function edit($id = null) {
            echo 'edit';
      }

      public function show($id = null) {
            echo 'Method show() berhasil dieksekusi';
      }
}
```

Jika berhasil, maka ketika kita tes pada browser seharusnya kita sudah tidak mendapati pesan error lagi.

Pada aplikasi REST API yang kita bangun, setiap controller mengextend class BaseController (app\Controllers\BaseController), misal pada controller Dashboard sebagai berikut:

```
namespace App\Controllers;

class Dashboard extends BaseController
{
	// Script lainnya
}
```

Sedangkan Class BaseController sendiri mengextend Class ResourceController sebagai berikut

```
use CodeIgniter\RESTful\ResourceController;
class BaseController extends ResourceController
{
	// Script lainnya...
}
```

sehingga dapat dikatakan bahwa setiap class pada controller (app\Controllers) secara tidak langsung mengextend/inherit dengan Class ResourceController sehingga penulisan method show(), creste(), update(), dll harus memenuhi ketentuan pendefinisian method pada ResourceController (system\RestFul\ResourceController.php)

7.4.2. Pola URL

Pada Aplikasi REST API, ketika kita menambahkan resource maka kita perlu mendefinisikan URI untuk mengakses resource tersebut. Pendefinisian URI ini telah kita bahas pada Bab 5.1 Mendesain URI, diantaranya penggunaan resource dan sub resource, misal untuk resource berupa artikel, maka alamat URI nya adalah http://domain/artikel, selanjutnya lebih spesifik lagi untuk artikel tertentu (subresource) maka polanya http://domain/artikel/{idartikel} misal http://domain/artikel/1 selanjutnya jika ada subresource yang menjadi bagian dari artikel.

Note: Untuk readablitity kedalaman resiurce dan subresource ini disarankan maksimal 3 kedalaman.

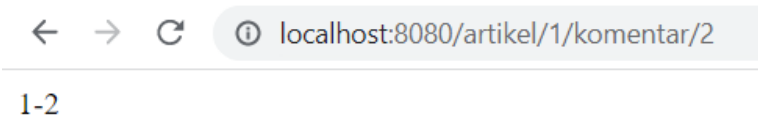
Mari kita coba terapkan pola URI diatas pada routing Codeigniter 4 misal untuk url http://domain/artikel/1/komentar/1 routing dapat kita buat sebagai berikut:

```
$routes->get("artikel/(.*)/komentar/(.*)",
"Artikel::getKomentarByIdArtikelAndIdKomentar/$1/$2");
```

Selanjutnya mari kita coba routing tersebut. Pada file app\Controllers\Artikel.php tambahkan method `getKomentarByIdArtikelAndIdKomentar()` sebagai berikut:

```
namespace App\Controllers;
class Artikel extends BaseController
{
       // Code lainnya
      public function getKomentarByIdArtikelAndIdKomentar
($id_kategori, $id_komentar) {
            echo $kategori;
      }
}
```

Jika kita jalankan pada browser maka hasil yang kita peroleh adalah



Note: Pada aplikasi api yang kita kembangkan, kita tidak menerapkan sepenuhnya pola pendefinisian URI yang telah kita bahas pada BAB 5.1, hal ini dilakukan karena alasan fleksibilitas, sebagai contoh pada aplikasi api kita terdapat URI: http://localhost/dashboard/penjualan/terbaru ketika membaca URI diatas, maka kita akan langsung mengecek controller Dashboard (app\Controllers\Dashboard.php) sehingga mudah melakukan maintenance karena memang URI tersebut diperuntukkan untuk dashboard. Jika kita buat identifiernya http://localhost/penjualan-terbaru, maka kita akan mengira bahwa Controller yang digunakan adalah Penjualan_terbaru (app\Controllers\Penjualan_terbaru.php).

7.4.3. Urutan Routing

Ketika mendefinisikan routing, maka penting diperhatikan urutan pendefinisiannya, terutama ketika kita menggunakan shorthand routing,

kesalahan pendefinisian ini akan mengakibatkan kesalahan pemangggilan controller sehingga membuat aplikasi REST API kita tidak berjalan sebagaimana mestinya.

Ketika membaca routing, Codeigniter akan membacanya urut dari atas kebawah, ketika menemukan routing yang sesuai maka Codeigniter tidak akan membaca baris berikutnya dan langsung mengeksekusi routing tersebut, sebagai contoh terdapat routing sebagai berikut:

```
$routes->resource('artikel');
$routes->get("artikel/kategori/(.*)", "Artikel::kategori/$1");
```

sehingga keseluruhan routing pada file app\Config\Routes.php adalah sebagai berikut:

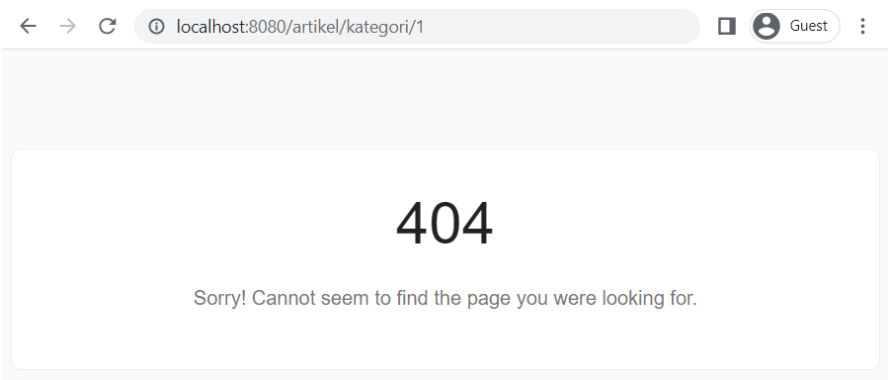
```
/*
 * --------------------------------------------------------------------
 * Route Definitions
 * --------------------------------------------------------------------
 */

// We get a performance increase by specifying the default
// route since we don't have to scan directories.
$routes->get('/', 'Home::index');
$routes->resource('artikel');
$routes->get("artikel/kategori/(.*)", "Artikel::kategori/$1");
```

Selanjutnya pada file Controller Artikel.php yang telah kita buat sebelumnya (app\Controllers\Artikel.php) kita hapus method show() dan kita tambahkan method kategori() sebagai berikut:

```
namespace App\Controllers;

class Artikel extends BaseController
{
      public function index()
      {
            $request = \Config\Services::request();
            echo $request->getMethod() . '<br/>' .
                  $request->getHeaderLine('accept-language')  .
'<br/>';

            echo $this->request->getMethod()  . '<br/>' .
```

```
                $this->request->getHeaderLine('accept-
language');

            // return view('welcome_message');
      }

      public function edit($id = null) {
            echo 'edit';
      }

      public function kategori() {
            echo 'kategori';
      }
}
```

Setelah selesai, jalankan URI http://localhost:8080/ci4/artikel/kategori/1 pada browser, hasil yang kita peroleh adalah pesan error bahwa halaman tidak ditemukan sebagai berikut:

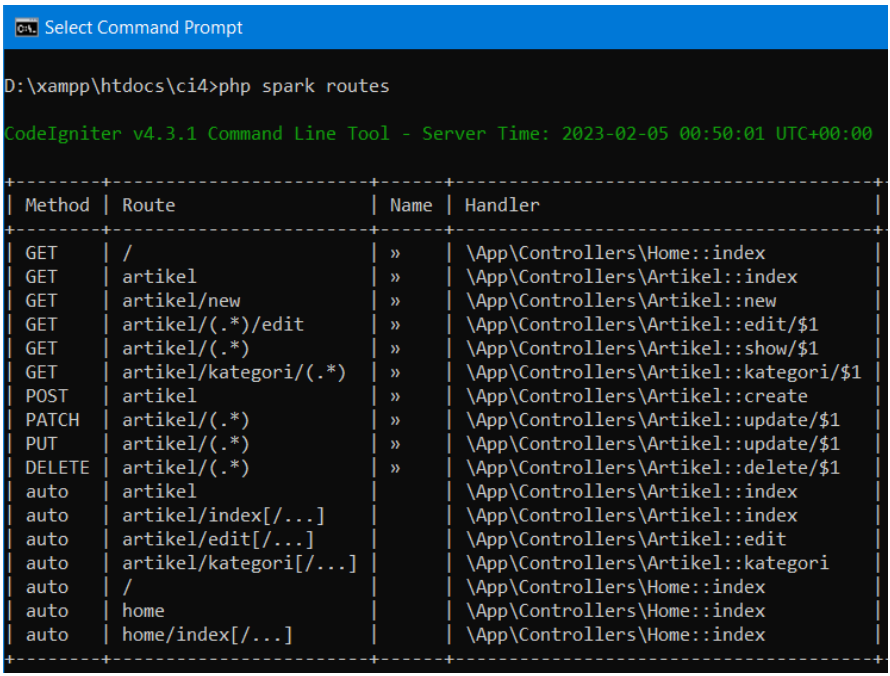


Kenapa demikian?

Seperti yang telah kita bahas sebelumnya bahwa sebenarnya pada shorthand routing `$routes->resource('artikel');` secara internal Codeigniter mendefinisikan beberapa routing, mari kita cek routing apa yang didefinisikan Codeigniter pada roting berikut ini:

```
$routes->resource('artikel');
$routes->get("artikel/kategori/(.*)", "Artikel::kategori/$1");
```

Routing yang didefinisikan Codeigniter adalah sebagai berikut:

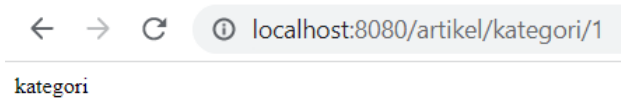


Ketika kita mengakses url http://localhost:8080/artikel/kategori/1 maka Codeigniter mengecek satu persatu routing diatas (dari atas ke bawah) dan ketemu routing artikel/(.*) pada method GET (nomor 5 dari atas) sehingga Codeigniter akan mengeksekusi routing tersebut dan akan memanggil method show() pada Controller Artikel. Karena method show tidak kita definisikan (sudah kita hapus) maka muncul pesan error bahwa halaman tidak ditemukan.

Untuk mengatasi masalah diatas, maka **selalu** tempatkan shorthand routing di paling bawah dari routing yang lain, sehingga bentuknya menjadi seperti berikut ini:

```
$routes->get("artikel/kategori/(.*)", "Artikel::kategori/$1");
$routes->resource('artikel');
```

Jika kita buka kembali url http://localhost:8080/ci4/artikel/kategori/1 maka hasil yang kita peroleh adalah sebagai berikut:



7.4.4. Keterbatasan PHP

Pada bab sebelumnya, disebutkan bahwa kita menggunakan method put untuk melakukan update data. Pada aplikasi client berbasis web, update data ini umumnya dilakukan melalui form HTML, bentuk data form tersebut bisa bermacam macam, bisa berupa data teks, data binary (file upload, seperti file image, file dokumen, dll)

Pada tipe data binary atau yang sering dikenal sebagai blob (binary large object) PHP memiliki builtin function untuk memparsing data tersebut yang hasilnya disimpan pada variabel $_FILES, namun sayangnya, variabel $_FILES ini hanya tersedia pada request method POST, ya karena browser hanya mengenal method POST untuk mengirim data yaitu melalui form (<form method="post">), sehingga ketika melakukan update data yang berisi data binary, kita tidak bisa menggunakan method put, sehingga pada aplikasi REST API kita nantinya ada variasi penggunaan method POST, yaitu bisa saat membuat data maupun update data (menggantikan put).

Note: keterbatasan ini ada pada PHP bukan pada Codeigniternya, sehingga framework apapun yang berbasis PHP akan mengalami hal yang sama.

Jika Anda tetap ingin menggunakan put, maka Anda harus memparsing manual data yang diterima server yang tentunya sangat sulit dilakukan karena saat ini belum ada library PHP yang dapat digunakan untuk memparsing data binary yang ada di HTTP Body.

7.4.5. Method Spoofing

Codeigniter 4 sendiri menyadari hal ini, mereka menyediakan semacam hack sehingga meskipun request method yang digunakan adalah POST, Codeigniter tetap membacanya sebagai put, cara ini disebut dengan

method spoofing, lebih lanjut mengenai method spoofing ini bisa dibaca pada halaman "HTTP Method Spoofing" yang dapat diakses melalui tautan: https://codeigniter4.github.io/userguide/incoming/methodspoofing.html.

Untuk menggunakan method spoofing, caranya adalah dengan mengirimkan data dengan name (nama) `_method` dan value PUT, jika menggunakan script HTML, maka bentuk form nya adalah sebagai berikut:

```
<input type="hidden" name="_method" value="PUT" />
```

Dengan menggunakan method spoofing ini, maka pada routing berikut:

```
+--------+--------------+------+---------------------------------------+----------------+---------------+
| Method | Route        | Name | Handler                               | Before Filters | After Filters |
+--------+--------------+------+---------------------------------------+----------------+---------------+
| GET    | /            | »    | \App\Controllers\Home::index          |                | toolbar       |
| POST   | artikel      | »    | \App\Controllers\Artikel::create      |                | toolbar       |
| PATCH  | artikel/(.*) | »    | \App\Controllers\Artikel::update/$1   |                | toolbar       |
| PUT    | artikel/(.*) | »    | \App\Controllers\Artikel::update/$1   |                | toolbar       |
| DELETE | artikel/(.*) | »    | \App\Controllers\Artikel::delete/$1   |                | toolbar       |
+--------+--------------+------+---------------------------------------+----------------+---------------+
```

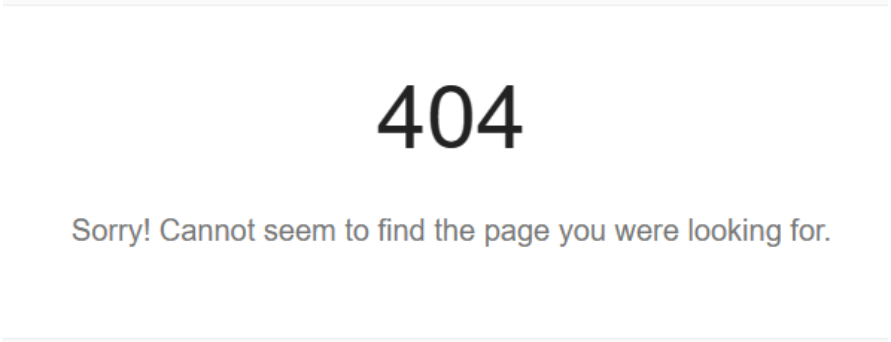
request dengan method POST yang didalamnya terdapat data _method dengan value PUT, routing yang dieksekusi adalah routing PUT (\App\Controllers\Artikel::update/$1)

7.5. Controllers

Setelah melakukan pengecekan pada bagian routing maka selanjutnya Codeigniter akan memanggil file controller sesuai yang didefinisikan pada routing, file controller ini ada di folder app\Controllers. Penting diperhatikan bahwa dalam memanggil file ini, Codeigniter akan menggunakan awalan huruf besar pada file controller yang dipanggil dan sisanya menggunakan huruf kecil, sehingga jika yang dipanggil adalah controller artikel_kategori maka file controller yang diload adalah file Artikel_kategori.php. dengan demikian penting diperhatikan penamaan file controller ini, yaitu selalu gunakan huruf besar di awal nama controller dan sisanya menggunakan huruf kecil, jika tidak maka pada Operating System yang membedakan huruf besar dan huruf kecil (case sensitive)

seperti Linux (Hosting Online) akan mengakibatkan error halaman tidak ditemukan.

Berikut contoh pesan error ketika mengakses alamat URI http://localhost:8080/artikel-kategori dengan nama file controllernya Artikel_Kategori.php.

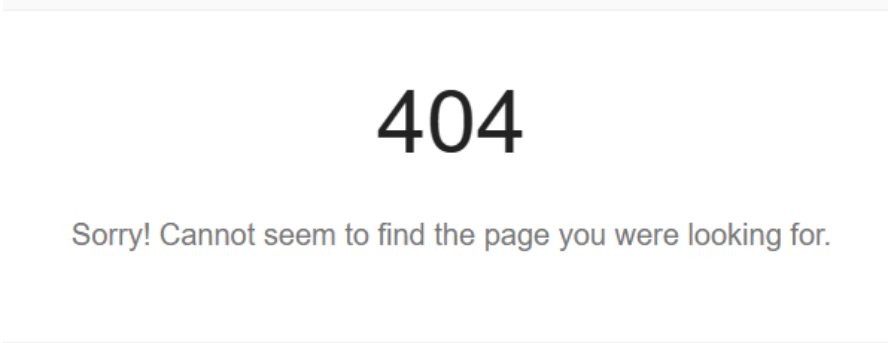


Error tersebut diatas disebabkan karena penulisan penamaan file controller tidak tepat karena sekarusnya huruf besar hanya di awal nama file saja sehingga seharusnya huruf K pada kategori menggunakan huruf kecil, yang benar adalah Artikel_kategori.php

Setelah memanggil file controller, selanjutnya Codeigniter akan menginisiasi class dengan nama yang sama dengan file controller, sehingga perlu diperhatikan bahwa nama class yang ada di file controller harus sama dengan nama file controller, sebagai contoh pada controller artikel_kategori, maka nama class yang ada di file Artikel_kategori.php seharusnya adalah Artikel_kategori sebagai berikut:

```
namespace App\Controllers;
class Artikel_kategori extends BaseController
{
    public function index()
    {
        // Code lainnya
    }
}
```

Penamaan class ini bersifat case insensitive yang artinya Anda bebas menggunakan huruf besar atau huruf kecil, misal Artikel_kategori, Artikel_Kategori, Artikel_Kategori, atau artikel_Kategori, namun agar lebih seragam, sebaiknya penggunaan huruf besar dan kecil disamakan dengan huruf besar dan kecil pada nama file, selanjutnya jika nama class tidak sama dengan nama file controller, misal nama file controllernya Artikel_kategori.php namun nama classnya ArtikelKategori, maka akan muncul pesan error sebagai berikut:



Pemisah Kata

Pada contoh diatas, kita menggunakan underscore (_) sebagai pemisah kata pada nama file controller, hal ini disebabkan karena pada URI pemisah kata yang kita gunakan adalah dash, normalnya pada mode autoroute, Codeigniter akan meload nama file sesuai nama pada URI, namun demikian Codeigniter menyediakan pengaturan yang memungkinkan Codeigniter menterjemahkan dash menjadi underscode yaitu dengan mengganti argumen method setTranslateURIDash() pada file app\Config\Routes.php dari false menjadi true

```
$routes->setTranslateURIDashes(true);
```

Penggunaan dash sebagai pemisah kata menurut penulis lebih url friendly dibanding tanpa menggunakan pemisah. Menurut Standar PSR-1

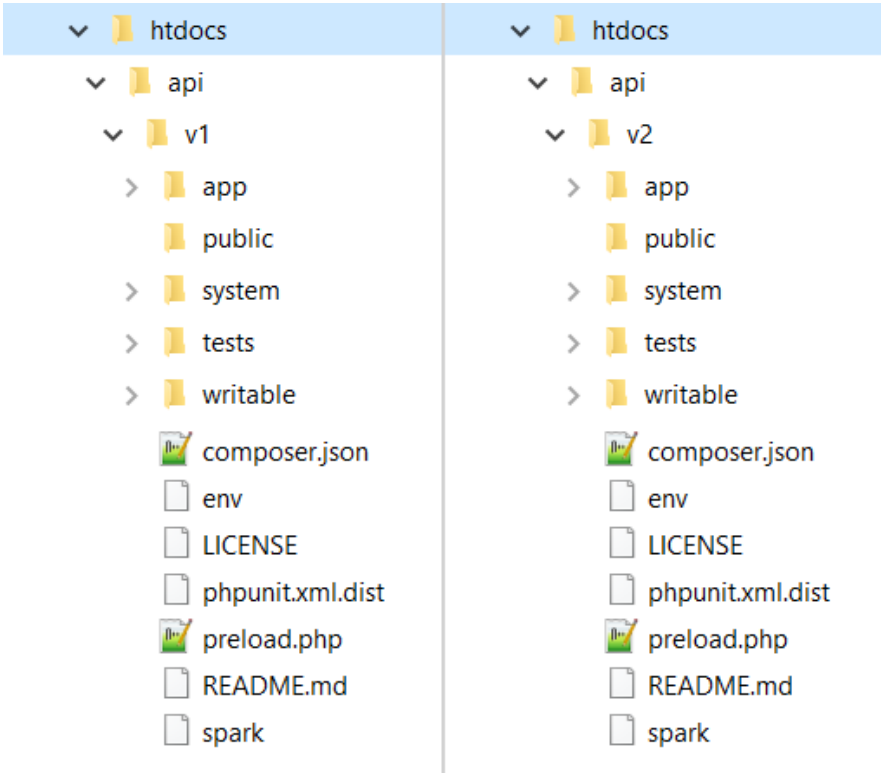
(https://www.php-fig.org/psr/psr-1/) penamaan class seharusnya menggunakan (StudlyCaps / UpperCamelCase), dimana setiap kata diawali dengan huruf besar seperti ArtikelKategori, BaseController, dll, namun demikian jika hal tersebut diterapkan untuk routing, menuruf penulis kurang pas karena URI akan menjadi susah dibaca, misal ketika ingin menggunakan nama class ArtikelKategori maka nama file controller harus ArtikelKategori.php sehingga alamat URI nya menjadi http://localhost:8080/artikelkategori, dengan mempertimbangkan hal tersebut maka pada Aplikasi REST API yang kita kembangkan kita menggunakan dash sebagai pemisah antar kata pada URI.

Setelah menginisiasi class pada controller, selanjutnya Codeigniter akan mengeksekusi method yang ada pada class tersebut sesuai dengan yang telah didefinisikan pada routing. Pada method ini, banyak hal bisa dilakukan seperti meload class model untuk mengambil data database, meload file view, dll.

7.6. Menerapkan versioning

Pada Bab 6 Versioning telah dijelaskan bagaimana menerapkan versioning pada aplikasi REST API, dari berbagai metode versioning yang ada, menurut penulis yang paling pas digunakan adalalah metode URI. Pada Codeigniter 4 kita dapat menerapkan metode ini dengan berbagai cara.

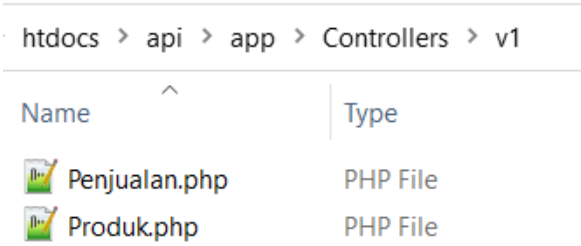
Cara yang pertama adalah menempatkan versi pada folder tersendiri, jadi masing-masing versi kita install Codeigniter, sebagai contoh kita buat folder v1 dan v2 pada folder api, pada masing-masing folder v1 dan v2 tersebut kita install Codeigiter, sehingga bentuk struktur foldernya adalah sebagai berikut:



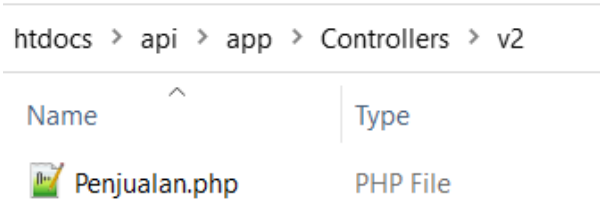
Dengan model seperti ini maka aplikasi REST API v1 dan v2 dikembangkan secara terpisah, sendiri sendiri, sehingga jika ada script yang sama pada v1 dan v2 maka jika ada perbaikan script pada v1 perbaikan tersebut juga harus dilakukan pada v2. Sebagai contoh, pada v1 terdapat resource Produk dan Penjualan kemudian pada v2 hanya resource Penjualan saja yang mengalami perubahan, resource produk tetap, maka jika ada perbaikan resource produk pada v1, resource produk pada v2 juga harus diperbaiki.

Cara berikutnya adalah menggunakan satu aplikasi Codeigniter kemudian pada folder app\Controllers kita buat folder v1 dan v2 selanjutnya dengan router kita arahkan setiap request yang datang ke controller sesuai dengan versi yang ada pada request tersebut.

Dengan model ini, maka penulisan code dapat menjadi lebih efisien, sebagai contoh terdapat resource produk dan penjualan pada folder v1 kemudian resource penjualan kita update menjadi v2 sementara resource penjualan tetap, maka pada folder controller v1 file yang kita buat adalah sebagai berikut:



Sedangkan pada folder v2 cukup kita buat file Penjualan.php tanpa file Produk.php karena resource produk v2 sama dengan v1.



Nantinya dengan pengaturan pada router, jika client mengakses resource produk v2, maka kita arahkan ke resource produk v1. Adapun routingnya adalah sebagai berikut (file app\Config\Routes.php)

```
// Versi 1
$routes->group("v1/produk", function ($routes) {
      $routes->post("kategori", "v1\Produk::kategori");
      $routes->put("kategori/(.*)", "v1\Produk::kategori/$1");
});
$routes->resource('v1/produk', ['controller' =>'v1\Produk']);
```

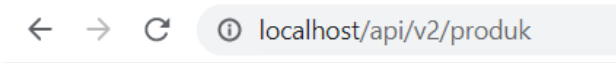
```
$routes->group("v1/penjualan", function ($routes) {
      $routes->get("(.*)/item", "v1\Penjualan::item/$1");
      $routes->post("item", "v1\Penjualan::item");
      $routes->put("item/(.*)", "v1\Penjualan::item/$1");
});
$routes->resource('v1/penjualan', ['controller' =>'v1\Penjualan']);

// Versi 2
$routes->group("v2/produk", function ($routes) {
      $routes->post("kategori", "v1\Produk::kategori");
      $routes->put("kategori/(.*)", "v1\Produk::kategori/$1");
});
$routes->resource('v2/produk', ['controller' =>'v1\Produk']);

$routes->group("v2/penjualan", function ($routes) {
      $routes->get("(.*)/item", "v2\Penjualan::item/$1");
      $routes->post("item", "v2\Penjualan::item");
      $routes->put("item/(.*)", "v2\Penjualan::item/$1");
});
$routes->resource('v2/penjualan', ['controller' =>'v2\Penjualan']);
```

Resource Produk

Agar lebih memiliki gambaran yang utuh, misal pada file app\Controllers\v1\Produk.php kita isi dengan script sebagai berikut:

```
namespace App\Controllers\v1;
use App\Controllers\BaseController;

class Produk extends BaseController
{
    public function index()
    {
            $data = 'Method index Produk v1';
            $response = ['status' => 'ok', 'data' => $data];
            return $this->respond($response);
    }

      public function show($id = null)
      {
            $data = 'Method show Produk v1';
            $response = ['status' => 'ok', 'data' => $data];
            return $this->respond($response);
      }
}
```

Selanjutnya jika kita akses resource produk v1 maupun v2 maka hasil yang kita peroleh sama yaitu sebagai berikut:




```
{
    "status": "ok",
    "data": "Method index Produk v1"
}
```


Resource Penjualan

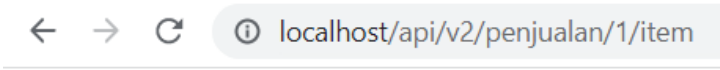
Misal script pada file app\Controllers\v2\Penjualan.php adalah sebagai berikut:

```php
namespace App\Controllers\v2;
use App\Controllers\BaseController;

class Penjualan extends BaseController
{
    public function index()
    {
            $data = 'Method index Penjualan v2';
            $response = ['status' => 'ok', 'data' => $data];
            return $this->respond($response);
    }

      public function show($id = null)
      {
            $data = 'Method show Penjualan v2';
            $response = ['status' => 'ok', 'data' => $data];
            return $this->respond($response);
      }

      public function item($id = null)
      {
            $data = 'Method item Penjualan v2';
            $response = ['status' => 'ok', 'data' => $data];
            return $this->respond($response);
      }
}
```

sedangkan pada app\Controllers\v1\Penjualan.php adalah sebagai berikut:

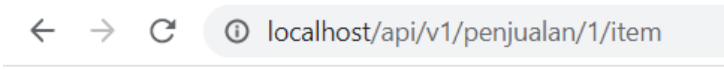
```php
namespace App\Controllers\v1;
use App\Controllers\BaseController;

class Penjualan extends BaseController
{
	public function index()
	{
		$data = [
			[
				'id' => 1,
				'no_invoice' => '1/123/INV/2023'
			],
			[
				'id' => 2,
				'no_invoice' => '1/456/INV/2023'
			],
		];

		$response = ['status' => 'ok', 'data' => $data];
		return $this->respond($response);
	}

	public function show($id = null)
	{
		$data = [
			'id' => $id,
			'no_invoice' => '1/123/INV/2023'
		];
		$response = ['status' => 'ok', 'data' => $data];
		return $this->respond($response);
	}

	public function item($id = null)
	{
		$data = [
			[
				'id' => 1,
				'nama_barang' => 'Penjualan v1: CPU'
			],
			[
				'id' => 2,
				'nama_barang' => 'Penjualan v1: Monitor'
			],
		];
		$response = ['status' => 'ok', 'data' => $data];
		return $this->respond($response);
	}
}
```

Jika kita akses resource Penjualan v2 maka hasil yang kita peroleh sebagai berikut:




```
{
    "status": "ok",
    "data": "Method item Penjualan v2"
}
```

Sedangkan Penjualan v1 sebagai berikut:

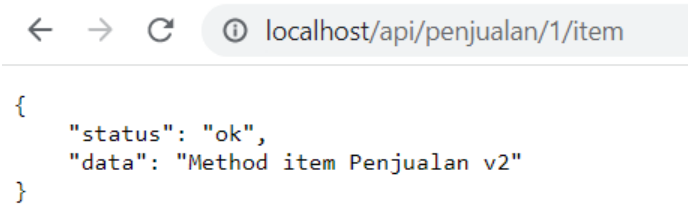



```
{
    "status": "ok",
    "data": "Method item Penjualan v1"
}
```

Dengan model seperti ini maka kitapun dapat mengatur URI API tanpa menyertakan versi, routing nya adalah sebagai berikut:

```php
// Tanpa versi
$routes->group("produk", function ($routes) {
      $routes->post("kategori", "v1\Produk::kategori");
      $routes->put("kategori/(.*)", "v1\Produk::kategori/$1");
});
$routes->resource('produk', ['controller' =>'v1\Produk']);

$routes->group("penjualan", function ($routes) {
      $routes->get("(.*)/item", "v2\Penjualan::item/$1");
      $routes->post("item", "v2\Penjualan::item");
      $routes->put("item/(.*)", "v2\Penjualan::item/$1");
});
$routes->resource('penjualan', ['controller' =>'v2\Penjualan']);
```

Hasilnya adalah:



Note: Agar lebih simpel, pada aplikasi REST API yang kita kembangkan kita tidak menggunakan versioning, sehingga URI tidak menyertakan versi dan juga tidak menggunakan folder versi (v1 dan v2) pada folder Controllers.

Halaman ini sengaja dikosongkan
Jagowebdev.com

BAB 8 Mengembangkan Aplikasi REST API

Setelah kita membahas berbagai teori tentang REST API tiba saatnya kita pada bagian inti dari buku ini yaitu mengembangkan aplikasi REST API.

8.1. Beberapa Standar

Pada bagian awal buku ini telah kita bahas berbagai standar yang dapat digunakan pada pengembangan REST API seperti HTTP Method, Content-type, dan response code, pada bagian ini kita akan membahas beberapa diantaranya yang akan kita gunakan dalam mengembangkan aplikasi REST API.

8.1.1. HTTP Request Method

Pada BAB 3 kita telah membahas tentang HTTP Method, pada bab tersebut telah dijelaskan fungsi dari masing masing method mulai dari GET, POST, PUST, dst, pada aplikasi yang kita kembangkan nanti, kita hanya menggunakan empat method, yaitu GET, POST, DELETE, dan PUT, berikut fungsi masing-masing method tersebut pada aplikasi yang kita bangun:

Method	Perintah
GET	Mengambil data
POST	Menambah data
DELETE	Menghapus data
PUT	Mengubah data

Dengan menggunakan method ini maka nantinya satu URI dapat mengandung perintah yang bermacam-macam, misal:

URI	Method	Perintah
https://jagowebdev.com/api/artikel	GET	Mengambil data
https://jagowebdev.com/api/artikel	POST	Menambah data
https://jagowebdev.com/api/artikel	DELETE	Menghapus data
https://jagowebdev.com/api/artikel	PUT	Mengubah data

Mungkin Anda bertanya kenapa perlu menggunakan HTTP Method?

Dalam arsitektur REST, komunikasi terjadi antar aplikasi (client dan server) dan masing masing aplikasi dikembangkan oleh pengembang yang berbeda beda, sehingga agar antar aplikasi dapat saling memahami perintah masing masing maka digunakanlah HTTP Method.

Mungkin Anda juga bertanya tanya, membaca HTTP Method pada header terkadang tidak mudah, tidak adakah cara lain yang lebih mudah?

Pada aplikasi umum (non API) method ini biasanya ditulis pada bagian URI misal https://jagowebdev.com/artikel/add atau https://jagowebdev.com/artikel/edit, atau semacamnya. Model ini mudah diterapkan, namun demikian dengan menggunakan model seperti ini akan sulit menetapkan standar sehingga akan membingungkan pengguna aplikasi, terutama bagi yang tidak menggunakan bahasa Inggris.

Jadi tanamkan dalam diri kita bahwa HTTP Header (khususnya HTTP Method) merupakan hal yang sangat penting untuk dipahami ketika mengembangkan aplikasi REST API.

Selanjutnya mungkin Anda juga bertanya tanya bukankah semua perintah dapat dilakukan menggunakan method POST? sehingga lebih simpel dan mudah?

Jawabnya: YA, benar, semua dapat dilakukan dengan method POST, seperti yang biasa kita lakukan ketika mengembangkan aplikasi web (misal dengan pemrograman PHP) pada aplikasi web tersebut, menambah, mengubah, maupun menghapus data dapat dilakukan menggunakan method POST, dalam konteks REST API model seperti ini seharusnya tidak digunakan karena seperti yang telah disebutkan sebelumnya, pada aplikasi

REST API yang berkomunikasi adalah aplikasi dengan aplikasi sehingga perlu standard yang dapat dimengerti oleh aplikasi.

8.1.2. Content-Type dan Format Data

Pada BAB 3 telah kita bahas mengenai header content-type, dimana header tersebut digunakan untuk memberitahu aplikasi api maupun aplikasi client format data yang ada di HTTP Body, pada aplikasi yang kita bangun, baik server maupun client, kita akan menggunakan header content-type ini untuk membaca format data yang ada di HTTP Body.

Pada BAB 3 juga disebutkan bahwa format data yang populer digunakan saat ini adalah XML dan JSON, pada aplikasi yang kita bangun, kita akan menggunakan format data JSON karena lebih ringkas, lebih ringan, dan lebih mudah pengelolaan datanya karena PHP telah mensupport penuh format data JSON ini.

8.1.3. Response Code

Pada BAB 3 telah kita bahas bahwa setiap HTTP Response selalu menyertakan response code seperti 200, 301, 404, dll, response code ini nantinya digunakan oleh client untuk menentukan tindakan yang tepat, dalam praktik penggunaan status code ini bervariasi karena banyaknya perbedaaan interpretasi, untuk itu pada aplikasi yang kita kembangkan response code yang kita gunakan sebisa mungkin sesuai dengan standar yang ada, meskipun terkadang tidak selalu sama karena interpretasi yang berbeda.

8.2. Filters CORS

Pada BAB 7 Codeigniter 4 telah disebutkan bahwa pada Codeigniter 4 request yang masuk pertama tama akan di proses di bagian Filters. Pada aplikasi REST API yang kita kembangkan, pada bagian filters ini kita mendefinisikan HTTP Response Header agar response dari aplikasi kita dapat diterima dengan baik oleh client.

Filter yang kita definisikan ini kita beri nama filter cors, nama cors sendiri merupakan kependekan dari Cross-Origin Resource Sharing, adapun tujuan filter cors ini adalah untuk mengatasi batasan cors yang diterapkan oleh browser.

8.2.1. Kenapa perlu filter cors?

Untuk aplikasi client yang menggunakan javascript (misal javascript ajax - jQuery Ajax atau aplikasi SPA seperti React, Vue, Angular) maka ketika akan melakukan request ke server, pertama tama browser akan mengirim terlebih dahulu request ke server untuk memeriksa response dari server (prefilght request: https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request), apakah response berasal dari origin yang sama (same origin) atau tidak, jika ya maka lanjutkan request client tersebut jika tidak maka tolak request tersebut.

Note: Preflight request ini sebgaian telah dibahas pada sub bab 3.8.9. OPTIONS

1. Same origin.

   Same origin artinya lokasi client dan lokasi server sama persis termasuk port yang digunakan, misal jika lokasi apliaksi client ada di https://jagowebdev.com dan lokasi server api ada di https://jagowebdev.com maka termasuk kriteria same origin, pada kondisi demikian maka browser akan meneruskan request dari aplikasi client ke server.

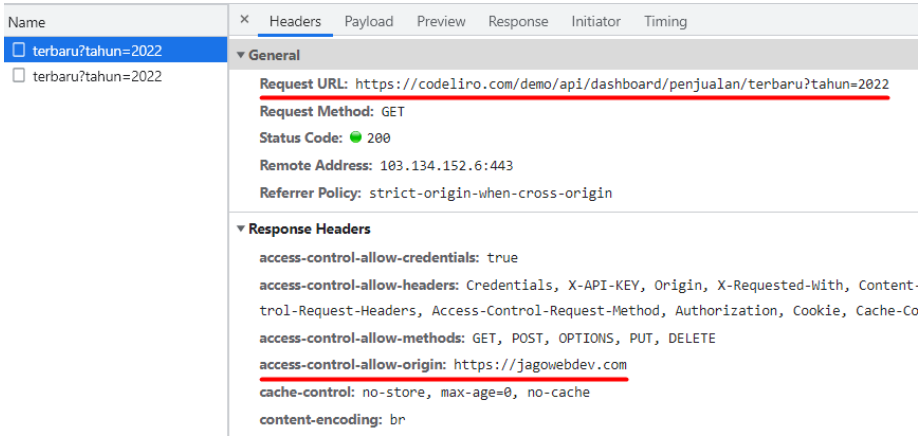
2. Tidak same origin.

   Jika client dan server tidak same origin, misal berbeda domain atau berbeda port seperti: http://localhost dengan http://localhost:8080 atau https://jagowebdev.com dengan https://codeliro.com maka pertama tama browser akan mengecek header pada response apakah nilai header Access-Control-Allow-Origin pada response memperbolehkan origin client untuk mengakses data pada server.

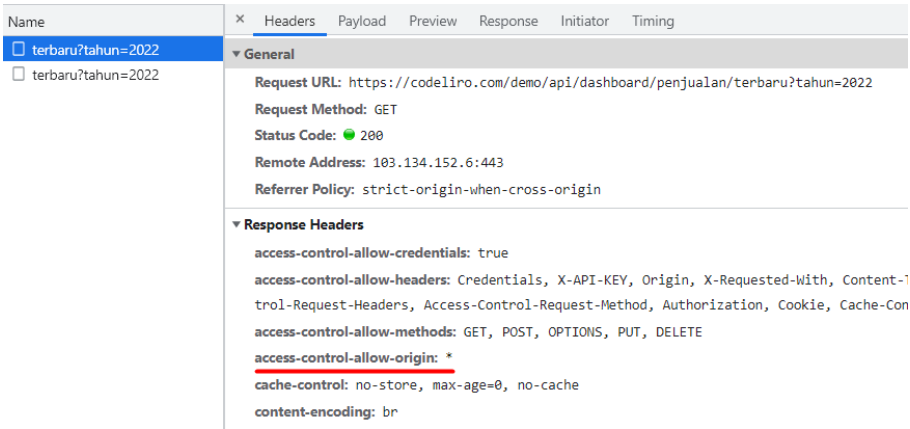
Agar client dapat mengakses response dari server, maka nilai header Access-Control-Allow-Origin pada HTTP Response Header yang dikirim server:

a. harus sama dengan origin aplikasi client atau;
b. memperbolehkan semua origin (tanda asterik *).

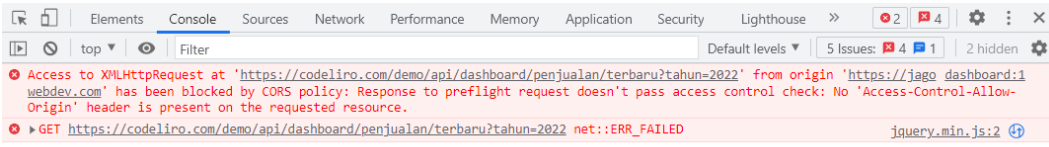
Misal alamat lokasi aplikasi client ada di https://jagowebdev.com/demo/api-client/ dan alamat aplikasi api nya ada di https://codeliro.com/demo/api/, maka jika HTTP Response Header yang dikirim oleh aplikasi api nilai Access-Control-Allow-Origin nya bernilai https://jagowebdev.com seperti berikut:



atau nilai Access-Control-Allow-Origin nya bernilai asterik (*) seperti berikut ini:



maka browser akan memperbolehkan aplikasi client untuk mengirim request ke server, namun demikian jika nilai access-control-allow-origin tidak sesuai atau tidak ada parameter access-control-allow-origin maka akan muncul pesan error sebagai berikut:



Kenapa demikian?

Pembatasan ini sengaja diterapkan untuk alasan keamanan. Batasan ini disebut CORS (Cross-Origin Resource Sharing) sebuah standar Internasional yang ditetapkan untuk meningkatkan keamanan pengguna browser. lebih lanjut tentang CORS ini dapat dibaca di https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS atau https://fetch.spec.whatwg.org/

Untuk mengatasi batasan tersebut, maka pada aplikasi api yang kita bangun, kita harus mendefinisikan header Access-Control-Allow-Origin.

8.2.2. Membuat Filter Cors

Untuk membuat filter cors, pertama tama kita buat file dengan nama Cors.php pada folder Filters (app\Filters), adapun script yang ada pada file Cors,php tersebut adalah sebagai berikut:

```
namespace App\Filters;

use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use CodeIgniter\Filters\FilterInterface;

Class Cors implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
      if (!empty($_SERVER['HTTP_ORIGIN'])) {
            header("Access-Control-Allow-Origin: " . $_SERVER['HTTP_ORIGIN']);

      } else {
            header("Access-Control-Allow-Origin: *");
      }

        header("Access-Control-Allow-Headers: Credentials, X-API-KEY, Origin,
X-Requested-With, Content-Type, Accept, Access-Control-Request-Headers,
Access-Control-Request-Method, Authorization, Cookie, Cache-Control");
        header("Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT,
DELETE");
      header("Access-Control-Allow-Credentials: true");

        $method = $_SERVER['REQUEST_METHOD'];
        if ($method == "OPTIONS") {
                die();
        }
    }

    public function after(RequestInterface $request, ResponseInterface
$response, $arguments = null)
    {
      // kosong
    }
}
```

Note: Pada script diatas terdapat method before() dan after(). Method before() akan dieksekusi sebelum controller dieksekusi dan method after() akan dieksekusi setelah controller dieksekusi, karena kita ingin mendefinisikan HTTP Header maka script kita buat pada method before(),

meskipun method after() tidak kita gunakan, kita tetap harus mendefinisikannya.

Pada script diatas terdapat script:

```
if (!empty($_SERVER['HTTP_ORIGIN'])) {
      header("Access-Control-Allow-Origin: " . $_SERVER['HTTP_ORIGIN']);

} else {
      header("Access-Control-Allow-Origin: *");
}
```

Script diatas digunakan untuk mendefinisikan header Access-Control-Allow-Origin pada response header. Umumnya untuk mengatasi batasan CORS pendefinisian menggunakan asterik (*) sudah cukup:

```
header("Access-Control-Allow-Origin: *");
```

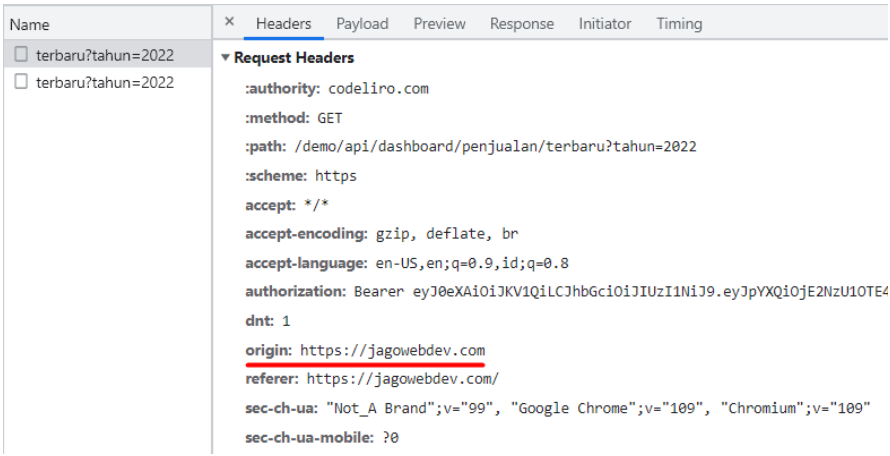
namun demikian tidak jarang browser tetap menolak untuk meneruskan request jika nilai header access-control-allow-origin seperti ini, maka untuk mengatasinya, kita definisikan header Access-Control-Allow-Origin dengan nilai sama dengan origin client, caranya pertama kita cek apakah header pada client terdapat parameter origin, jika ya, beri nilai Access-Control-Allow-Origin sesuai nilai pada header origin tersebut, contoh sebagai berikut:

```
if (!empty($_SERVER['HTTP_ORIGIN'])) {
      header("Access-Control-Allow-Origin: " . $_SERVER['HTTP_ORIGIN']);
}
```

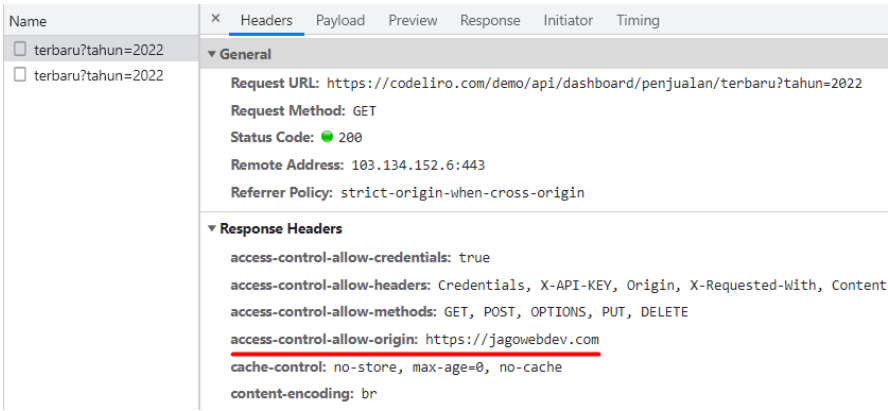
Jika tidak ada maka beri nilai header Access-Control-Allow-Origin dengan tanda asterik (*)

```
else {
      header("Access-Control-Allow-Origin: *");
}
```

Setiap request ajax yang dilakukan browser selalu menyertakan header origin berupa alamat aplikasi client yang akan melakukan request ke server, misal terdapat request ajax dari aplikasi api client yang ada di https://jagowebdev.com ke server api yang ada di https://codeliro.com, pada bagian header request tersebut akan muncul parameter origin dengan nilai https://jagowebdev.com sebagai berikut:



Dengan script cors yang kita buat tadi maka ketika aplikasi api server menerima request tersebut, nilai Access-Control-Allow-Origin pada response header yang dikirim akan bernilai https://jagowebdev.com sebagai berikut:



Dengan respose header seperti diatas maka respon dari server api dapat dipastikan aman dan pasti akan dieksekusi oleh browser.

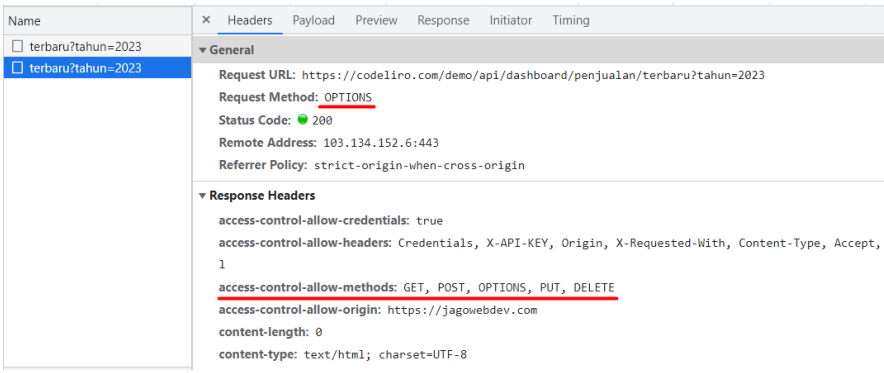
Selanjutnya pada script Cors.php diatas juga terdapat script sebagai berikut:

```
$method = $_SERVER['REQUEST_METHOD'];
if ($method == "OPTIONS") {
      die();
}
```

Script tersebut digunakan untuk menghentikan eksekusi script ketika terdapat request dengan method OPTIONS sehingga response yang dikirim ke client hanya header nya saja.

Sebagaimana telah dibahas sebelumnya, ketika aplikasi client mengirim request via ajax, maka pertama tama browser akan mengirim request ke server untuk memeriksa response dari server (prefilght request), request yang dilakukan browser ini menggunakan method OPTIONS, request ini dilakukan browser selain untuk mengecek header access-control-allow-origin juga untuk mengecek apakah method yang nantinya dikirim oleh aplikasi client diperbolehkan oleh server api.

Sebagai contoh kita akan melakukan update data menggunakan javascript ajax. Pada request yang kita kirim, parameter method pada bagian header kita definisikan dengan nilai PUT, maka sebelum mengirim request tersebut, browser akan mengirim preflight request (dengan url yang sama) menggunakan method OPTIOS sebagai berikut:

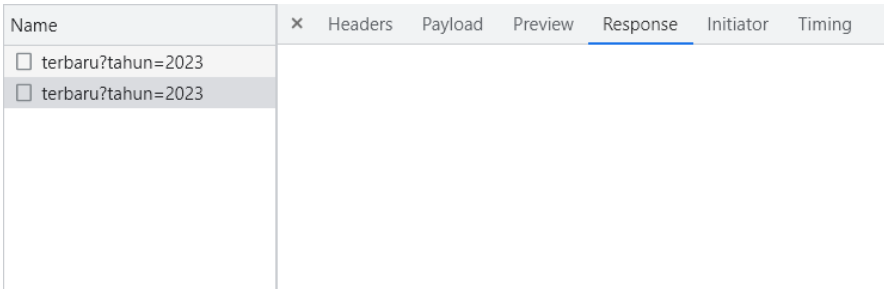


Pada contoh diatas server memberikan response dengan parameter header access-control-allow-methods: GET, POST, OPTIOS, PUT, DELETE karena method PUT ada pada parameter access-control-allow-methods tersebut maka browser akan mengeksekusi url yang mengandung method PUT tadi.

Kenapa nilai access-control-allow-methods yang dikirim server bisa bernilai GET, POST, OPTIOS, PUT, DELETE ? hal ini karena pada file Cors.php kita telah mendefinisikan nilai header Access-Control-Allow-Methods sebagai berikut;

```
header("Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT,
DELETE");
```

selanjutnya pada preflight request tadi, jika kita cek bagian response body (tab Response) maka akan tampak kosong, hal ini terjadi karena kita hentikan eksekusi script menggunakan perintah die(), sehingga tidak ada response body yang dikirim ke client.



8.2.3. Mengaktifkan Filter Cors

Filter cors yang telah kita buat pada bagian sebelumnya tidak otomatis aktif, untuk mengaktifkannya kita perlu mendaftarkannya di module filters (app\Config\Filters), caranya buka file app\Config\Filters.php kemudian pada property $aliases tambahkan nilai `'cors' => \App\Filters\Cors::class` sebagai berikut:

```
public $aliases = [
      'csrf'     => CSRF::class,
      'toolbar'  => DebugToolbar::class,
      'honeypot' => Honeypot::class,
      'cors'     => \App\Filters\Cors::class,
];
```

Agar filter cors tersebut dieksekusi sebelum controller dieksekusi, maka kita perlu menambahkan filter cors tersebut pada property $globals bagian key before sebagai berikut:

```
public $globals = [
      'before' => [
            'cors'
      ],
      'after'  => [
```

```
            'toolbar',
            // 'honeypot',
        ],
];
```

8.3. Routing

Setelah request diproses pada bagian Filter Cors, proses berikutnya adalah dilakukan routing yaitu menentukan controller yang akan dipanggil berdasarkan alamat URI yang ada pada request.

Sebagaimana telah kita bahas pada BAB 7 Codeigniter 4, pada aplikasi REST API yang kita kembangkan, kita tidak bisa menggunakan autoroute karena request method yang digunakan bermacam macam tidak hanya get dan post sehingga semua endpoint harus kita definisikan routingnya secara manual.

Selain penggunaan method yang bermacam macam, autoroute juga tidak bisa digunakan karena, (1) penamaan URI tidak mengandung nama method sedangkan nama class method (untuk memudahkan identifikasi) menggunakan nama method, seperti getArtikel(), getPenjualan(), dll, (2) konsep resource dan sub resource pada URI REST API yang terkadang tidak bisa otomatis dipetakan menjadi controller dan class method, berikut contoh routing pada aplikasi REST API yang disertakan pada buku ini:

```
$routes->group("dashboard", function ($routes) {
      $routes->get("penjualan", "Dashboard::ajaxGetPenjualan");
      $routes->get("penjualan/item", "Dashboard::ajaxGetItemTerjual");
      $routes->get("penjualan/kategori", "Dashboard::ajaxGetKategoriTerjual");
      $routes->get("penjualan/terbaru", "Dashboard::ajaxGetPenjualanTerbaru");
      $routes->get("pelanggan/terbesar", "Dashboard::ajaxGetPelangganTerbesar");
      $routes->post("penjualan/terbesar/datatables",
"Dashboard::getDataDTPenjualanTerbesar");
});
$routes->resource('dashboard', ['controller' =>'Dashboard']);
```

Note: pada contoh diatas, penulis menggunakan group routing untuk mengelompokkan URI sehingga lebih muda dimaintenance, lebih lanjut mengenai group routing dapat dibaca di halaman user manual "URI

Routing" yang bisa diakses melalui tautan https://codeigniter.com/user_guide/incoming/routing.html#grouping-routes

8.4. Memproses Request

Setelah pada bagian routing dipetakan controller dan method yang akan dieksekusi, maka selanjutnya Codeigniter akan memanggil file controller, untuk kemudian menginisiasi Class, dan mengeksekusi method yang ada pada file controller tersebut.

8.4.1. Inisiasi Awal

Pada apikasi REST API yang kita kembangkan, setiap class pada controller meng-*extend* Class BaseController yang ada di file app\Controllers\BaseController.php. Dengan meng-extend class BaseController maka pada class controller, kita dapat menggunakan property maupun method yang ada pada Class BaseController.

Pada PHP, ketika kita menginisiasi class maka jika didalam class tersebut terdapat method __construct(), method tersebut akan otomatis dieksekusi, pada aplikasi REST API yang kita kembangkan setiap class yang ada pada app\Controllers memiliki method __construct(), didalam method tersebut terdapat perintah parent::construct() yang artinya ketika class tersebut dieksesuki maka method __construct() yang ada di parent (BaseController) akan dieksekusi.

Sebagai contoh pada aplikasi REST API yang disertakan pada buku ini, Class Artikel yang ada di file app\Controllers\Artikel.php memiiki method __construct() dan di dalam method tersebut terdapat perintah parent::__construct().

```
class Artikel extends BaseController
{
      namespace App\Controllers;
      use App\Models\ArtikelModel;
      use CodeIgniter\HTTP\ResponseInterface;
```

```
	public function __construct() {

		parent::__construct();
		$this->model = new ArtikelModel;
	}

	// Script lainnya
}
```

ketika Class Artikel ini diinisiasi misal ketika user mengakses URI http://localhost/api/artikel maka method __construct() pada BaseController akan otomatis dieksekusi.

Perintah parent::__construct() ini sangat penting untuk dijalankan karena pada method __construct() yang ada di Class BaseController semua bagian penting dari aplikasi api diinisiasi, seperti menu, user, role user, permission user, dll, sehingga penting diperhatikan ketika membuat controller maka selalu jalankan method parent_::construct().

Secara garis besar, isi dari method __constract() pada Class BaseController adalah sebagai berikut:

```
class BaseController extends ResourceController
{
	// Code lainnya

	public function __construct()
	{
		date_default_timezone_set('Asia/Jakarta');

		$this->model = new BaseModel;
		$this->config = new \Config\App;
		$this->request = \Config\Services::request();
		$this->newToken = '';

		$this->settingToken = $this->getSettingToken();

		$this->getCurrentModule();
		$this->setUser();

		$this->getUserPermission();
		$this->getMenu();
		$this->getSettingLayout();
		$this->initAkses();
```

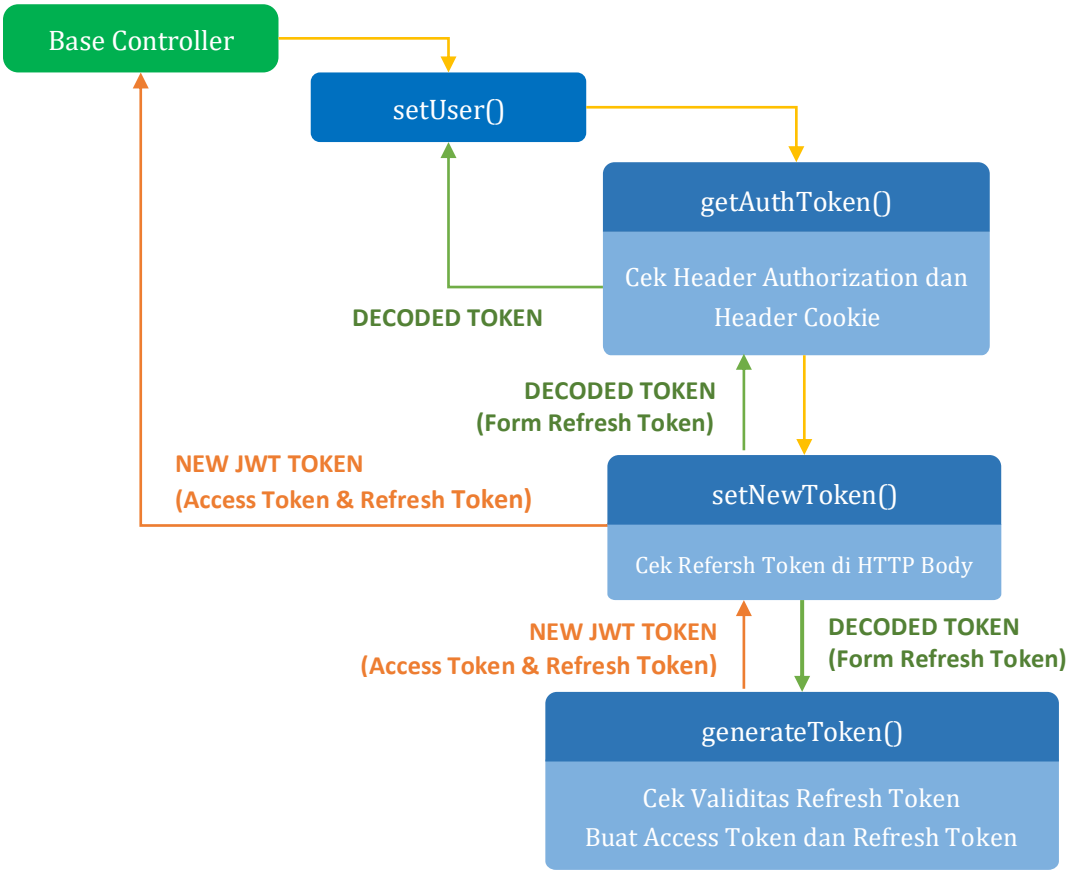
```
            // Code lainnya
        }
}
```

Pada method __construct() diatas, terdapat banyak method yang dieksekusi seperti setUser() untuk mengecek apakah user didefinisikan (termasuk pengecekan token dan inisiasi user), method getUserPermission() untuk mendapatkan permission yang dimiliki user, dan method initAkses() untuk menginisiasi property initAkses yang nantinya dikirim ke client.

Property initAkses berisi berbagai informasi penting yang dapat digunakan untuk menjalankan aplikasi client, seperti data user, data menu, data permission, pesan error, dll, pada aplikasi REST API yang kita bangun, property initAkses ini dapat diibaratkan ruhnya aplikasi karena dapat berjalan atau tidaknya aplikasi client bergantung pada property ini.

8.4.2. Pengecekan Token

Ketika aplikasi client mengirim request ke server maka pertama tama kita cek apakah request tersebut membawa token, jika ya maka dengan token tersebut kita lakukan identifikasi user, alur pengecekan token seperti tampak pada gambar berikut:



Penjelasan:

1. Pada BaseController, method __construct() memanggil method setUser();
2. Method setUser() memanggil method getAuthToken();
3. Pada method getAuthToken() dilakukan pengecekan apakah request header mengandung header Authorization atau header Cookie, jika ya, maka kita cek apakah token pada header tersebut valid, jika valid maka decode token dan kirim kembali hasilnya ke method setUser(), jika tidak maka panggil method setNewToken() untuk mengecek apakah memungkinkan untuk membuat token baru.

4. Pada method setNewToken() jika terdapat refresh token pada HTTP Body maka decode refresh token tersebut, kemudian hasil decode tersebut: (1) dikirim kembali ke method getAuthToken() untuk selanjutnya dikirim ke method setUser(), (2) dengan method generateToken() buat Access Token dan Refresh Token baru untuk kemudian disimpan di property newToken yang ada di BaseController. Property newToken ini nantinya disimpan pada property initAkses yang selanjutnya akan dikirim ke client.

Catatan: Pada rekomendasi yang ada saat ini, refresh token seharusnya diterbitkan pada request kedua setelah request pertama memberikan response 401 Unauthorized. Pada aplikasi REST API yang kita kembangkan, hal ini tidak bisa kita terapkan karena kita menggunakan library datatables dimana request hanya bisa dilakukan sekali, sehingga ketika menggunakan datatables, maka pada data yang kita kirim ke server kita selalu menyertakan data refresh token pada HTTP Body.

Lebih lanjut mengenai proses pengecekan token dapat disimak pada penjelasan berikut ini:

Proses 1: Memanggil Method getAuthToken()

Pertama tama pada method setUser() kita panggil method getAuthToken(), hasilnya kita simpan pada variabel $token_decoded sebagai berikut:

```
private function setUser()
{
      $token_decoded = $this->getAuthToken();
      // Script lainnya
}
```

selanjutnya pada method getAuthToken() kita lakukan pengecekan token pada header Authorization dan header Cookie, adapun script method getAuthToken() adalah sebagai berikut:

```
rotected function getAuthToken()
{
      // Authorizarion Header
      $token = '';
```

```
        $auth_header = $this->request->getServer('HTTP_AUTHORIZATION');

        if ($auth_header) {
                //JWT is sent from client in the format Bearer XXXXXXXXX
                $token_item = explode(' ', $auth_header);
                if ($token_item[0] != 'Bearer') {
                        return false;
                }

                if (count($token_item )> 1) {
                        $token = $token_item[1];
                }
        } else {

                if (key_exists('HTTP_COOKIE', $_SERVER)) {

                        $exp = explode(';', $_SERVER['HTTP_COOKIE']);
                        foreach ($exp as $val) {
                                $split = explode('=', $val);
                                $cookie_name = trim($split[0]);
                                unset($split[0]);
                                $cookie_value = join($split);
                                $cookie_header[$cookie_name] = trim($cookie_value);
                        }

                        if (key_exists($this->settingToken['token_name'],
$cookie_header)){
                                $token = $cookie_header[$this-
>settingToken['token_name']];
                        }
                }
        }

        $result = '';

        if ($token) {
                try {
                        $config = new \Config\App;
                        $key = $config->jwtKeyAccessToken;
                        $decoded_token = JWT::decode(trim($token), trim($key),
['HS256']);
                        $result = (array) $decoded_token;
                }
                catch (Exception $e) {
                        if (strtolower($e->getMessage()) == 'expired token') {
                                $result = $this->setNewToken();
                        }
                }
        } else {
```

```
            $result = $this->setNewToken();
        }

        return $result;
}
```

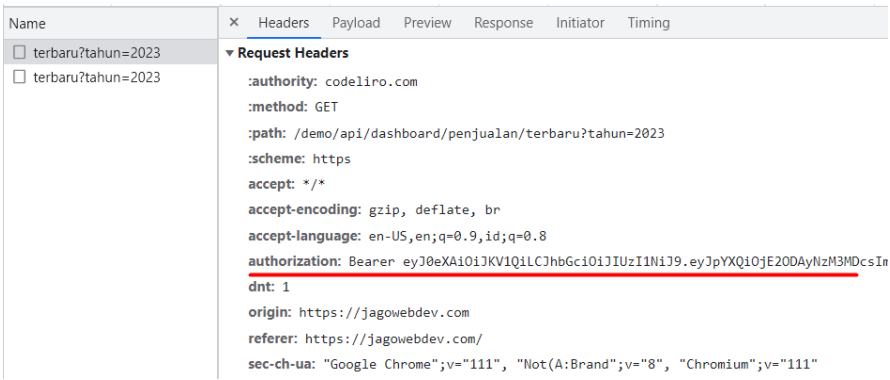
Pada script diatas, pertama tama kita cek header Authorization pada request yang masuk, jika terdapat data/tidak kosong maka kita cek apakah data tersebut mengandung awalan Bearer, jika ya maka kita ambil data setelah kata Bearer tersebut dan kita simpan hasilnya pada variabel $token.

```
protected function getAuthToken()
{
        // Authorizarion Header
        $token = '';
        $auth_header = $this->request->getServer('HTTP_AUTHORIZATION');

        if ($auth_header) {
                //JWT is sent from client in the format Bearer XXXXXXXXX
                $token_item = explode(' ', $auth_header);
                if ($token_item[0] != 'Bearer') {
                        return false;
                }

                $token = $token_item[1];
        }
        // Code lainnya
}
```

Note: pengecekan kata Bearer pada awal token ini penting untuk dilakukan, hal ini bertujuan untuk memastikan bahwa metode authentikasi yang digunakan adalah Bearer.

Seperti yang telah kita bahasa pada BAB 4, bentuk token pada skema Bearer adalah sebagai berikut:

```
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2Nzk0N
Dg1ODgsImV4cCI6MTY3OTYyMTM4OCwidHlwZSI6InByaXZhdGUiLCJpZF91c2VyIjoiMSJ9.W
PEeF3kv2RSA83Wyq5VbiCCcHyCGgt_zcQ-TH-HtVAU
```



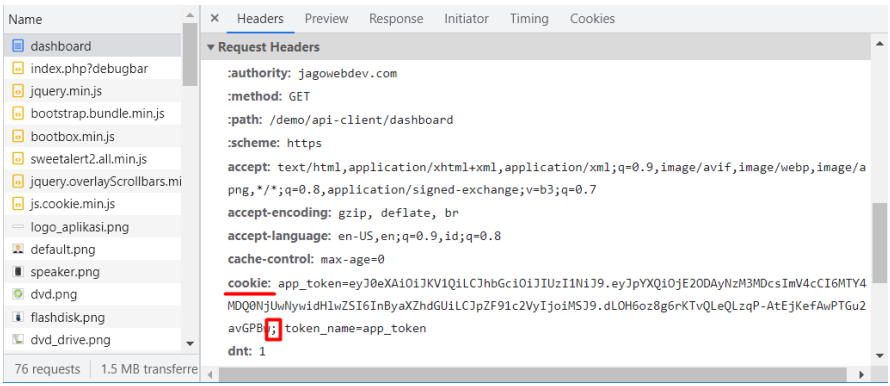
sehingga untuk mengambil data token, kita menggunakan fungsi `explode(' ', $auth_header)[1]`

Selanjutnya jika header Authorization tidak ditemukan, maka kita cek header Cookie. Jika terdapat header Cookie maka kita parsing nilai pada header Cookie tersebut kemudian kita cari nama app_token (nama default token `$this->settingToken['token_name']`) jika ditemukan maka kita simpan nilainya di variabel $token.

```
if (key_exists('HTTP_COOKIE', $_SERVER)) {

      $exp = explode(';', $_SERVER['HTTP_COOKIE']);
      foreach ($exp as $val) {
            $split = explode('=', $val);
            $cookie_name = trim($split[0]);
            unset($split[0]);
            $cookie_value = join($split);
            $cookie_header[$cookie_name] = trim($cookie_value);
      }

      if (key_exists($this->settingToken['token_name'], $cookie_header)){
            $token = $cookie_header[$this->settingToken['token_name']];
      }
}
```

Proses parsing data cookie

Nilai Cookei pada header dipisahkan dengan titik koma (;) sehingga pada script diatas kita split nilai cookie header menggunakan titik koma `$exp = explode(';', $_SERVER['HTTP_COOKIE']);` menjadi bentuk array.



Selanjutnya array tersebut kita lakukan looping dan hasilnya (jika ada) kita simpan pada variabel array $cookie_header, berikutnya kita cek apakah variabel $cookie_header mengandung key app_token, jika ya, simpan nilai tersebut pada variabel $token

```
if (key_exists($this->settingToken['token_name'], $cookie_header)){
      $token = $cookie_header[$this->settingToken['token_name']];
}
```

Cek variabel $token

Berikutnya kita cek apakah variabel $token ada isinya, jika ya (ada token di header Authorization atau header Cookie) maka kita decode token tersebut dengan script sebagai berikut:

```
$result = '';

if ($token) {
      try {
            $config = new \Config\App;
```

```
            $key = $config->jwtKeyAccessToken;
            $decoded_token = JWT::decode(trim($token), trim($key), ['HS256']);
            $result = (array) $decoded_token;
      }
      catch (Exception $e) {
            if (strtolower($e->getMessage()) == 'expired token') {
                  $result = $this->setNewToken();
            }
      }
} else {
      $result = $this->setNewToken();
}

return $result;
```

Pada script diatas jika variabel $token ada nilainya maka kita mencoba melakukan decode pada token, jika proses decode tersebut berhasil hasilnya kita simpan pada variabel $result `$result = (array) $decoded_token;` jika gagal maka library JWT (Firebase/php-jwt) akan melempar (throw) exception, selanjutnya exception tersebut kita tangkap menggunakan fungsi catch().

Jika pesan error ketika mendecode token JWT bernilai expired token `if (strtolower($e->getMessage()) == 'expired token')` maka kita coba (dengan refresh token) untuk membuat token baru, yaitu dengan memanggil method `setNewToken()`, demikian juga jika variabel $token tidak ada nilainya yang artinya tidak ada token pada header Authorization maupun header Cookie, maka kita juga coba untuk membuat token baru dengan mengeksekusi method `setNewToken()`, hasilnya kita simpan pada variabel $result.

Terakhir kirim variabel $result ke method setUser() untuk disimpan ke variabel $token_decoded, adapun contoh hasil decode token adalah sebagai berikut:

```
Array
(
    [iat] => 1680311989
    [exp] => 1680312289
    [id_user] => 1
)
```

Proses 2: Memanggil Method setNewToken()

Method setNewToken() digunakan untuk mengecek apakah request yang dikirim client menggunakan method post dan bagian body dari request tersebut terdapat refresh token (token yang digunakan untuk merefresh akses token), jika ya maka kita generate ulang akses token yang baru, adapun script method setNewToken() adalah sebagai berikut:

```
private function setNewToken() {

      $refresh_token = $this->request->getPost('refresh_token');

      if (!$refresh_token)
            return;

      $new_token = '';
      try {
            $config = new \Config\App;
            $key = $config->jwtKeyRefreshToken;
            $decode = \Firebase\JWT\JWT::decode(trim($refresh_token),
trim($key), ['HS256']);
            $decoded_token = (array) $decode;

            if ($decoded_token) {
                  if ($decoded_token['exp'] > time())
                  {
                        $data = $decoded_token;
                        $data['type'] = 'refresh_token';
                        $data['status'] = 'expired';
                        $token_expired = $this->model-
>checkRefreshToken($data);
                        if ($token_expired) {
                              return false;
                        }

                        $data['date_expired'] = date('Y-m-d H:i:s',
$data['exp']);
                        $save = $this->model->saveJwt($data);

                        if ($save) {
                              $token = $this->generateToken($decoded_token);
                              $new_token = ['new_token' => $token,
'decoded_token' => $decoded_token];
                        }
                        return false;
                  } else {
                        return false;
```

```
                }
            }
        } catch (Exception $e) {
            if ($e->getMessage() == 'expired token') {
                // Catch error
            }
        }

        $result = '';
        if ($new_token) {
            $result = $new_token['decoded_token'];
            $this->newToken = $new_token['new_token'];
        }
        return $result;
}
```

Pada script diatas, jika HTTP Request Body menyertakan refresh token, maka kita coba untuk mendecode refresh token, hasinya kita simpan pada variabel $decoded_token, jika proses decode berhasil maka:

1. kita cek apakah refresh token belum expired `if ($result['exp'] > time());`
2. selanjutnya kita cek di database apakah refresh token tersebut belum expired (refresh token belum pernah digunakan untuk membuat token baru);
3. jika belum expired maka kita simpan data refresh token tersebut ke database (untuk ditandai bahwa refreah token sudah digunakan);
4. selanjutnya dengan method generateToken() kita buat token baru (access token dan refresh token) berdasarkan data refresh token tersebut, hasilnya kita simpan pada variabel $token `$token = $this->generateToken($result);`
5. Token baru tersebut kita simpan ke property newToken (`$this->newToken`) untuk nantinya dikirim ke client melalui property initAkses dan hasil decode dari refresh token kita kirim kembali ke method getAuthToken() (`return $result`) untuk selanjutnya dikirim ke method setUser().
6.

Method generateToken()

Method generateToken() berada di BaseController. Method ini merupakan satu-satu nya method yang digunakan untuk membuat token (access token dan refresh token), method ini digunakan ketika menggenerate token baru, seperti pada method setNewToken() diatas dan pada saat client berhasil melakukan login (app\Controllers\Login.php)

Adapun script pada method generateToken() adalah sebagai berikut:

```
protected function generateToken($user)
{
	// Access Token
	$time = time();
	if (!$this->settingToken) {
		$this->settingToken = $this->getSettingToken();
	}
	$access_token_expire = $time + $this-
>settingToken['access_token_age'];

	$access_token_payload = [
		'iat' => $time,
		'exp' => $access_token_expire
	];

	if ($user['id_user']) {
		$access_token_payload['id_user'] = $user['id_user'];
	}

	$config = new \Config\App;
	$access_token = $this->generateJWT($access_token_payload, $config-
>jwtKeyAccessToken);

	// Refresh Token
	$refresh_token_expire = '';
	$refresh_token = '';

	if ($user['id_user']) {
		if ($this->settingToken['using_refresh_token'] == 'Y') {

			if ($this->settingToken['refresh_token_age_limited'] ==
'Y')
			{
				$this->settingToken['refresh_token_age_unit'];
				$list_unit = ['tahun' => 'year', 'bulan' =>
'month', 'hari' => 'day'];
```

```
                        $priode = '+' . $this-
>settingToken['refresh_token_age'] . ' ' . $list_unit[ $this-
>settingToken['refresh_token_age_unit'] ];
                        $refresh_token_expire = strtotime($priode, $time);
                  }

                  $uuid = Uuid::uuid4();
                  $jti = $uuid->toString();
                  $refresh_token_payload = [
                        'iat' => $time,
                        'jti' => $jti,
                        'id_user' => $user['id_user']
                  ];

                  if ($refresh_token_expire) {
                        $refresh_token_payload['exp'] =
$refresh_token_expire;
                  }

                  $refresh_token = $this-
>generateJWT($refresh_token_payload, $config->jwtKeyRefreshToken);
            }
      }

      $response = [
                        'status' => 'ok',
                        'access_token' => $access_token,
                        'refresh_token' => $refresh_token,
                        'access_token_expire' => $access_token_expire,
                        'refresh_token_expire' => $refresh_token_expire,
                        'token_name' => $this->settingToken['token_name'],
                        'user' => $this->model-
>getUserById($user['id_user']),
                        'message' => 'User authenticated successfully'
                  ];
      return $response;

}
```

Pada script diatas pertama tama, kita buat akses token (access token), akses token ini berisi claims iat dan exp, jika argumen $user mengandung id_user, maka kita tambahkan claims id_user.

Selanjutnya jika pada konfigurasi token aplikasi menggunakan refresh token maka kita generate refresh token, refresh token ini berisi iat, jti, dan id_user, jika refresh token ini diatur memiliki masa expired maka kita

tambahkan claims exp, claims jti merupakan ID token yang nantinya digunakan untuk melakukan rotasi token.

8.4.3. Menggunakan Exception

Pada script method `getNewToken()` yang kita bahas diatas terdapat block catch() yang berisi class Exception yaitu:

```
catch (Exception $e) {
      if (strtolower($e->getMessage()) == 'expired token') {
            $result = $this->setNewToken();
      }
}
```

Exception sendiri merupakan class bawaan PHP untuk menghandle error, class ini memiliki beberapa method, diantaranya method getMassage() seperti contoh diatas, lebih jauh mengenai class Exception ini dapat dibaca di halaman "PHP: Exception" yang dapat diakses melalui tautan https://www.php.net/manual/en/class.exception.php

Untuk dapat menggunakan class Exception, kita harus mendefinisikan perintah use Exception di bagian paling atas sebelum nama class, misal sebagai berikut:

```
namespace App\Controllers;
use CodeIgniter\HTTP\ResponseInterface;
use App\Models\UserModel;
use App\Models\BaseModel;
use Exception;

class Login extends \App\Controllers\BaseController
{
      // Script lainnya
}
```

kenapa demikian? Karena kita menggunakan namespace (pada contoh diatas `namespace App\Controllers`) Jika tidak menggunakan keyword use maka nama class akan relatif terhadap namespace nya, misal jika tidak menggunakan use Exception maka class exception yang dipanggil adalah App\Controllers\Exception yang tentu saja tidak eksis.

Jika memang tidak ingin menggunakan keyword use, Anda dapat menggunakan tanda back slash (\) sehingga class yang dipanggil absolute, tidak relatif terhadap namesapce, misal:

```
catch (\Exception $e) {
      if (strtolower($e->getMessage()) == 'expired token') {
            $result = $this->setNewToken();
      }
}
```

8.4.4. Inisiasi User

Pada bagian sebelumnya telah dibahas bahwa pada method setUser() kita menjalankan method getAuthToken() untuk mendapatkan data token, jika method getAuthToken() memberikan hasil/tidak kosong, maka variabel $token_decoded pada method setUser() akan berisi data token, data token tersebut salah satunya berisi data id user, misal sebagai berikut:

```
Array
(
    [iat] => 1680311989
    [exp] => 1680312289
    [id_user] => 1
)
```

selanjutnya berdasarkan data id_user tersebut kita ambil data detail user

Selain data user, kita juga ambil data data terkait user tersebut, seperti role dan permission yang dimiliki user, nantinya data-data ini akan disertakan pada properti initAkses yang akan dikirim ke client.

Jika metod getAuthToken() memberikan kembalian kosong (tidak ada nilainya) maka tidak ada data user yang bisa diambil, sehingga nantinya data user pada property initAkses akan bernilai kosong.

Setelah setUser() selesai dieksekusi, maka selanjutnya method __construct() akan memanggil method initAkses(). Method ini berisi data data yang diperlukan client untuk menampilkan aplikasi. Adapun script dari method initAkses() adalah sebagai berikut:

```
protected function initAkses()
{
      $result['sidebar_menu'] = $this->menu;
      $result['setting_layout'] = $this->settingLayout;
      $result['setting_aplikasi'] = $this->model->getSettingAplikasi();
      $result['setting_token'] = $this->settingToken;

      $setting_registrasi = $this->model->getSettingRegistrasi();
      $result['setting_registrasi'] = $setting_registrasi;
      $result['module_akses'] = $this->currentModule;
      $result['permission'] = $this->userPermission;
      $result['user'] = $this->user;
      $result['status'] = '';
      $result['message'] = '';
      $result['token'] = $this->newToken;
      $result['error_type'] = '';
      $result['status_code'] = '';

      $this->initAkses = $result;
}
```

Pada script diatas terlihat bahwa property initAkses berisi berbagai macam data seperti data menu sidebar, data setting, data module akses, data permission, data user, data token (Access Token dan Refresh Token yang dihasilkan dari Refresh Token), dll.

8.4.5. Pengecekan Lanjutan

Setelah method __construct() selesai mengeksekusi method initAkses(), selanjutnya method __construct() akan melakukan pengecekan apakah module yang sedang diakses ($this->currentModule) ada di database:

```
if (!$this->currentModule || ($this->currentModule &&
!key_exists('id_module', $this->currentModule)) ) {

      $nama_module = $this->getNamaModule();
      $error = 'Module ' . $nama_module . ' tidak ditemukan di database';
      $this->exitError($error, 404);
}
```

setelah itu akan dilakukan pengecekan apakah (1) module yang sedang diakses memerlukan login, (2) apakah request method bernilai GET dan user memiliki permission dengan awalan read pada module yang sedang

diakses, dan (3) apakah status module yang sedang diakses bernilai selain aktif (tidak bernilai 1)

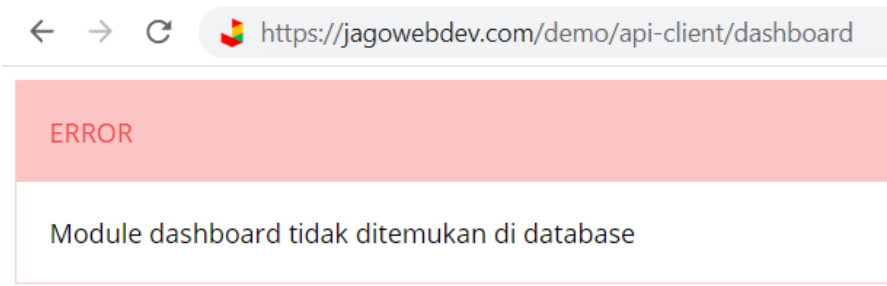
```
if ($this->currentModule) {

      if ($this->currentModule['login'] == 'Y') {

            if (!$this->user) {
                  $error = 'Akses ditolak (token tidak ditemukan)';
                  $this->exitError($error, Response::HTTP_UNAUTHORIZED,
'need_login');
            }

            if ($this->request->is('get') && !$this-
>hasPermissionPrefix('read')) {
                  $this->exitError('Anda tidak diperkenankan mengakses
module ' . $this->currentModule['nama_module'], 401);
            }
      }

      if ($this->currentModule['id_module_status'] != 1) {
            $error = 'Module ' . $this->currentModule['judul_module'] . ' '
. $this->currentModule['nama_status'];
            $this->exitError($error, 404);
      }
}
```

Jika terjadi error, maka aplikasi akan mengeksekusi method exitError(), adapun script method exitError() adalah sebagai berikut:

```
public function exitError($message = '', $code = 400, $error_type = '') {

      $this->initAkses['status'] = 'error';
      $this->initAkses['message'] = $message;
      $this->initAkses['error_type'] = $error_type;

      $response = service('response');
      $response->setStatusCode($code);
      $response->setJSON($this->initAkses);
      $response->setHeader('Content-type', 'application/json');
      $response->noCache();
      $response->send();
      exit;
}
```

Pada script diatas, kita tambahakan data error tersebut pada property initAkses, setelah itu kita kita kirim respon ke client (`$response->send()`) berupa data property initAkses dan kita hentikan eksekusi script (exit)

Dari sisi client nantinya pesan error ini ditangkap dan di tampilkan ke user, misal sebagai berikut:



8.4.6. Membaca Request Body

Setelah method __construct() pada BaseController dieksekusi, maka selanjutnya Codeigniter akan mengeksekusi method dari controller sesuai dengan hasil parsing URI pada routing, misal pada routing http://localhost/api/artikel dengan request method PUT maka akan dieksekusi method update pada controller Artikel.

Ketika memproses request dari client, maka bisa jadi request tersebut memiliki body (Request Body) misal ketika request tersebut bertujun melakukan update data, maka request tersebut perlu menyertakan data yang ingin diupdate, Data tersebut disertakan pada Request Body.

Pada model konvensional, request body ini oleh PHP otomatis disimpan pada variabel $_POST, kenapa? Ya karena browser hanya menggunakan method POST untuk mengirim data.

Sebagaimana dosebutkan pada bab sebelumnya, pada REST API, method bisa bermacam macam bisa GET, POST, PUT, DELETE, dll, nah untuk dapat menangkap data tersebut maka kita perlu melakukan beberapa hal diantaranya melakukan pengecekan header content type, agar penulisan code untuk pengecekan header content-type tidak berulang, maka kita

buat method getRequestInput() pada BaseController (app\Controllers\BaseController.php), contoh penggunaan method ini ada di file controller Artikel.php sebagai berikut:

```
class Artikel extends BaseController
{
	public function __construct() {
		parent::__construct();
		$this->model = new ArtikelModel;
	}

	public function update($id = null)
	{
		$input = $this->getRequestInput($this->request);
		// Code lainnya
	}
}
```

Adapun script method getRequestInput() adalah sebagai berikut:

```
public function getRequestInput(IncomingRequest $request)
{
	$content_type_header = $request->getHeader('Content-Type');
	$input = '';

	if ($content_type_header) {
		$content_type = $content_type_header->getValue();
		if ($content_type == 'application/json') {
			$input = $request->getJSON(true);
		}
	}

	if (!$input) {
		$input = $request->getPost();
	}

	if (!$input) {
		$input = $request->getGet();
	}

	if (empty($input)) {
		$this->exitError('No input Recieved ($_POST, $_GET
kosong)');
	}

	return $input;
```

```
}
```

Pada script diatas, pertama tama kita cek header content-type (`$request->getHeader('Content-Type');`), jika bernilai application/json, maka kita jalankan method getJSON: `$input = $request->getJSON(true);` method ini merupakan method bawaan Codeigniter yang digunakan untuk mengubah data JSON mejadi array (jika argumen true dihilangkan maka yang dihasilkan adalah object bukan array), selanjutnya kita simpan hasilnya hasilnya pada variabel $input.

Selanjutnya jika variabel $input bernilai kosong yang artinya header content-type tidak bernilai application/json misal `multipart/form-data`, maka kita jalankan perintah `$request->getPost();` untuk mendapatkan data pada variabel $_POST data jika variabel $input masih juga kosong kita jalankan perintah `$request->getGet();` untuk mendapatkan data pada variabel $_GET.

Hasil akhir dari method `getRequestInput()` ini adalah data dengan format array.

8.5. Token Rotation

Pada pembahasan sub bab 8.4.2. Pengecekan Token, ketika kita menggenerate token baru menggunakan method setNewToken() maka selain menggenerate access token baru kita juga menggenerate refresh token baru dan refresh token yang lama ditarik (revoke), dalam dunia REST API cara ini disebut juga refresh token rotation, yaitu membuat refresh token baru secara berkala.

Jika melihat pada standar yang ada, penerbitan refresh token baru telah sesuai dengan standar pada statement RFC 6749 https://www.rfc-editor.org/rfc/rfc6749#section-6 sebagai berikut:

> The authorization server **MAY issue a new refresh token**, in which case the client MUST discard the old refresh token and replace it with the new refresh token. **The authorization server MAY revoke the old refresh**

token after issuing a new refresh token to the client. If a new refresh token is issued, the refresh token scope MUST be identical to that of the refresh token included by the client in the request.

Selanjutnya mungkin Anda bertanya tanya kenapa perlu melakukan token rotation?

Token rotation ini dilakukan untuk alasan keamanan karena refresh token memiliki usia yang panjang, misal ketika refresh token diketahui oleh pihak yang tidak bertanggung jawab namun user telah menggunakan refresh token tersebut untuk membuat refresh token yang baru, maka mereka tidak dapat menggunakan refresh token ini untuk merequest token baru.

Pada implementasi dilapangan untuk menandai bahwa refresh token telah digunakan, ada beberapa cara yang dapat dilakukan salah satunya adalah melakukan pengecekan di database apakah refresh token tersebut pernah digunakan, implementasi metode ini dapat dilihat kembali pada method setNewToken(). Meskipun hal ini menambah ekstra query ke database, namun ekstra query ini jarang dilakukan misal setiap 15 atau 30 menit sekali karena refresh token dilakukan setelah usia access token habis.

Selanjutnya agar kita dapat melakukan pengecekan pada database maka setiap menerbitkan token baru maka kita simpan informasi token tersebut pada database sehingga nantinya jika ada permintaan token yang baru kita dapat melakukan pengecekan token yang lama, implementasinya dapat dilihat pada method setNewToken() script bagian `$save = $this->model->saveJwt($data);`

Untuk menyimpan data refresh token di database, kita tidak perlu menyimpan token secara utuh, terlalu panjang, membutuhkan space yang lebih dan query yang lebih lama, melainkan hanya ID dari token tersebut yang kita simpan pada database, ID dari refresh token ini kita simpan di claims jti, claims jti ini kita buat menggunakan UUID Versi 4 dengan library Ramsey\Uuid, script untuk menggenerate UUUiD ini ada di method generateToken() pada file BaseController.php

```
$uuid = Uuid::uuid4();
$jti = $uuid->toString();
$refresh_token_payload = [
      'iat' => $time,
      'jti' => $jti,
      'id_user' => $user['id_user']
];
```

Selanjutnya ketika kita menerbitkan refresh token baru maka data jti ini kita simpan didatabase sebagai berikut (method setNewToken()):

```
$save = $this->model->saveJwt($data);
```

Data jti ini kita simpan di database tabel token dengan skema sebagai berikut:

jti	exp	date_expired	id_user	type	status
69bf55a1-3ce9-4c50-9e77-23ea4af51986	1688508938	2023-07-04 17:15:38	1	refresh_token	expired
3e96dded-2a85-4287-aa15-b47ee43305f1	1688509356	2023-07-05 05:22:36	1	refresh_token	expired
ad4bb19a-c32f-432d-a457-ef20a2f4636f	1688549483	2023-07-05 16:31:23	1	refresh_token	expired
95dbe5c9-5b17-4ac1-be91-4d6d5879578a	1688550256	2023-07-05 16:44:16	1	refresh_token	expired

Pada data diatas kolom exp berisi nilai claims exp (berbentuk timestamp) yang ada pada token jwt dan kolom date_expired berisi nilai exp dalam bentuk tanggal.

Kenapa kita perlu menyimpan data date_expired token ini?

Hal ini kita lakukan agar nantinya kita dapat membersihkan data token yang sudah expired, sehingga dapat mengurangi penggunaan space hardisik dan query database dapat dilakukan dengan lebih cepat. Pengapusan ini dapat dilakukan melalui schedule event maupun cron job.

Selanjutnya pada method getNewToken() sebelum menerbitkan token baru kita melakukan pengecekan apakah refresh token yang digunakan sudah expired atau belum, contoh pada method setNewToken():

```
$token_expired = $this->model->checkRefreshToken($data);
if ($token_expired) {
      return false;
}
```

8.6. Mendapatkan Token (Login)

Aplikasi REST API yang kita bangun menggunakan token untuk melakukan validasi setiap request yang dikirim client (Token-based Authentication), token yang kita gunakan adalah JWT sehingga skema authentication yang kita gunakan adalah skema Bearer.

Seperti yang telah kita bahas pada BAB 4, alur skema Bearer ini adalah pertama tama user login ke sistem, setelah berhasil maka user akan diberikan token oleh server, selanjutnya dengan token tersebut user dapat mengakses resource pada server.

Pada aplikasi REST API yang kita kembangkan pengecekan user login dilakukan pada controller Login (app\Controllers\Login),

```
public function login()
{

      $rules = [
            'username' => 'required|min_length[3]|max_length[50]',
            'password' => 'required|max_length[255]'
      ];

      $input = $this->getRequestInput($this->request);
      if (!$this->validateRequest($input, $rules)) {
            return $this
                  ->respond(
                        [
                              'status' => 'error',
                              'message' => join($this->validator-
>getErrors())
                        ],
                        400
            );
      }

      return $this->loginUser($input);
}
```

Pada script diatas, pertama tama dengan method validateRequest() kita cek apakah data username dan password telah memenuhi kriteria yang

telah ditetapkan, jika tidak memenuhi (terjadi error) maka kirim respon dengan code 400, jika memenuhi maka eksekusi method loginUser().

Script pada method loginUser() sendiri adalah sebagai berikut:

```
private function loginUser($input) {
      try {

            $model = new UserModel();
            $user = $model->getUserByUsername($input);
            $error = false;

            if ($user) {
                  if ($user['verified'] == 0) {
                        $message = 'User belum aktif';
                        $error = true;
                  }

                  if (!password_verify($input['password'],
$user['password'])) {
                        $message = 'Username dan/atau Password
tidak cocok';
                        $error = true;
                  }

            } else {
                  $message = 'User tidak ditemukan';
                  $error = true;
            }

            if ($error) {
                  $response = [
                                    'status' => 'error',
                                    'message' => $message
                              ];
                  return $this->respond($response, 401);

            } else {

                  $response = $this->generateToken($user);
                  return $this->respond($response);
            }
      } catch (Exception $e) {
            return $this->fail($e->getMessage());
      }
}
```

Pada script diatas, kita melakukan serangkaian pengecekan terhadap data user seperti apakah user eksis (ada di databasel), user telah aktif (nilai verified 1), dan apakah password yang dikirim cocok dengan password yang ada di database, jika pengecekan menghasilkan kesalahan/error maka kita kirim respon dengan code 401, jika tidak maka dengan method generateToken() buat token untuk user dan dengan method respond() kirim token tersebut ke client.

```
$response = $this->generateToken($user);
return $this->respond($response);
```

Selanjutnya pada request berikutnya client akan menggunakan token tersebut untuk mengakses resource pada server api.

Catatan:

Mungkin Anda bertanya tanya kenapa pengecekan diawal diberikan status kode 400 dan yang berikutnya diberikan status code 401?

Jawabnya. Untuk error terkait kesalahan yang dilakukan user seperti kesalahan pengisian form (misal format alamat email tidak benar, username masih kosong, dll) maka kode yang kita gunakan adalah 400, sedangkan untuk error terkait authentikasi user (seperti username tidak ada di database, password tidak cocok dengan yang ada di database) maka kode yang kita gunakan adalah 401, untuk lebih jelasnya bisa dibaca kembali BAB 3.

8.7. Invalidate Token

Pada BAB 4 telah dibahas bahwa token baik access token maupun refresh token memiliki sifat statis, yang artinya sekali diterbitkan maka tidak bisa diubah, dengan sifat seperti ini maka jika sewaktu waktu terjadi kondisi yang mengharuskan sistem menolak token (meskipun token belum expired), misal user logout, user mengganti password, dll, maka kita harus dapat membuat sistem untuk menghandle kondisi tersebut.

8.7.1. Logout

Ketika user logout dari sistem, maka kita harus bisa menolak request yang di lakukan oleh user, meskipun token yang digunakan masih valid/belum expired, lantas bagaimana caranya?

Dalam praktik, ada beberapa alternatif cara yang dapat dilakukan untuk menolak akses user yang sudah logout, diantaranya adalah menandai token yang sudah terbit bahwa token tersebut sudah tidak valid, sehingga jika ada request yang menyertakan token dan token tersebut termasuk token yang ditandai, maka akses akan ditolak.

Bagaimana cara menandai token tersebut? Caranya yaitu menyimpan semua token yang terbit ke database, selanjutnya jika ada user yang logout maka token yang ada pada database tersebut kita tandai bahwa token sudah invalid, selanjutnya jika ada request yang menggunakan token tersebut, maka tolak request tersebut. Cara ini terlihat solutif, namun demikian jika kita perhatikan lagi, cara ini akan mengakibatkan (1) terjadi ekstra query database disetiap request yang masuk, setiap kali ada request yang masuk kita harus cek ke database apakah token yang ada pada request masih valid, (2) terdapat tambahkan space database untuk menyimpan token, dan (3) tambahan ekstra script dan tenaga untuk mengelola access token yang sudah expired. Dengan berbagai dampak tersebut, menurut penulis cara ini tidak efisien.

Selain menyimpan token didatabase, ada cara lain yang menurut penulis lebih efektif, yaitu mencatat kapan user terakhir kali logout, selanjutnya setiap kali ada request masuk, maka kita cek nilai claims iat (Issued At - kapan token diterbitkan) token, jika nilai iat < dari tanggal logout, yang artinya token diterbitkan sebelum user logout, maka tolak token tersebut.

Mungkin Anda bertanya bukankan ini sama saja terdapat ekstra query database untuk melakukan pengecekan data logout pada user?

Pada database, data kapan terakhir user logout ini kita simpan di tabel user kolom last_logout, sehingga pengambilan data ini dilakukan bersamaan dengan pengambilan data detail user karena toh setiap kali menerima

request kita harus mengambil data user untuk berbagai keperluan seperti menampilkna profil, mengetahui role dan permission user, dll, sehingga pada pengecekan ini tidak ada tambahan ekstra query database, selain itu juga tidak ada tambahan extra space yang signifikan.

Pada Aplikasi REST API yang kita kembangkan, pengecekan ini kita lakukan pada method `setUser()` sebagai berikut:

```
private function setUser()
{
      $token_decoded = $this->getAuthToken();
      if ($token_decoded) {
            $user = $this->model->getUserById($token_decoded['id_user']);
            if ($user) {

                  if ( $user['last_logout'] &&  $user['last_logout'] !=
'0000-00-00 00:00:00' ) {
                        if ($token_decoded['iat'] <
strtotime($user['last_logout'])) {
                              return;
                        }
                  }

                  // Script lainnya
            }
      }
}
```

Pada script diatas, jika iat token < data last_logout (`if ($token_decoded['iat'] < strtotime($user['last_logout'])) )` maka berikan nilai kembalian (`return`) kosong.

Pengecekan user yang logout juga dilakukan ketika melakukan pengecekan refresh token, hal ini bisa dilihat pada method getNewToken().

Pengecekan ini juga di lakukan ketika user melakukan request token baru menggunakan url login/refresh-token yaitu method refreshToken() pada controller Login (app\Controllers\Login.php).

8.7.2. Ganti Password

Kondisi berikutnya yang mengharuskan kita menolak token meski belum expired adalah kondisi ketkia user mengganti password. Ketika usar mengganti password, maka seharusnya token yang diterbaitkan menjadi tidak valid.

Kenapa demikian?

Misal terdapat kondisi sebagai berikut:

User berhasil login ke sistem dan sudah diterbitkan token, baik access token maupun refresh token. Beberapa saat kemudian user menyadari bahwa password nya telah diketahui orang lain dan digunakan untuk login ke sistem, setelah menyadari hal tersebut, user langsung mengganti password. User yang mengganti password ini tentu ingin agar user lain yang menggunakan passwordnya tidak dapat mengakses sistem (mengakses data miliknya), bagaimana caranya? caranya adalah mengivalidate/menonaktifkan semua token yang diterbitkan untuk user tersebut.

Bagaimana cara mengimplementasikannya?

Untuk mengimplementasikan metode tersebut diatas setidaknya ada beberapa cara yang dapat digunakan, diantaranya adalah menggunakan password user sebagai JWT key, sehingga setiap user memiliki JWT key sendiri-sendiri, dengan demikian ketika user mengganti password maka key JWT berubah sehingga token menjadi tidak valid.

Lagi lagi, cara ini terlihat solutif, tetapi didalamnya terdapat permasalahan mendasar, yaitu ketika terdapat request masuk, kita tidak tahu password user yang mana yang akan kita gunakan untuk mendecode token karena pada request tersebut kita tidak bisa mengetahui data user jika tidak mendecode token tersebut.

Mungkin Anda berfikiran, bukankan payload hanya di encode dengan base64 dan dapat dengan mudah di decode dan diketahui data id_user nya sehingga bisa kita dapatkan password user di database berdasarkan id_user tersebut?

Memang bisa demikian, namun bagaimana jika orang yang tidak berhak mengetahui password dari user tersebut yang artinya mereka tahu key JWT nya? Mereka dapat dengan mudah membuat token yang valid dan mengirim request ke sistem.

Baiklah mungkin sebagian Anda ada yang berpendapat, bagaimana jika keynya password user dengan tambahan salt (angka acak) dari kita? Sehingga meskipun ada yang tahu password user mereka tidak dapat membuat token karena key nya tidak hanya passeord tapi juga angka acak yang kita buat.

Baiklah mungkin ini terlihat solutif, namun demikian menurut penulis dengan cara seperti ini, maka kita perlu tambahan ekstra query database ketika memvalidasi token, karena kita perlu melakukan query database untuk mendapatkan password user sebagai key JWT nya, sehingga menurut penulis cara ini kurang efektif.

Cara yang lebih efektif menurut penulis adalah mirip dengan cara ketika menangani user yang logout yaitu dengan memberikan tanggal kapan terakhir user ganti password, selanjutnya ketika ada token yang masuk dan claims iat dari token tersebut < tanggal kapan terakhir ganti password yang berarti bahwa token tersebut diterbitkan sebelum user ganti password maka tolak token tersebut.

Dengan menggunakan cara seperti ini maka ketika user ganti password, seketika user akan diminta untuk login kembali.

Tanggal terakhir ganti password ini kita simpan di tabel user kolom last_change_password sehingga tidak ada tambahan query database ketika melakukan pengecekan token.

Sama seperti pegecekan user yang logout, pada aplikasi api yang kita kembangkan pengecekan user yang ganti password dilakukan di method `setUser()` yang ada pada controller BaseController app\Controllers\BaseController.php sebagai berikut:

```
private function setUser()
{
```

```
        $token_decoded = $this->getAuthToken();
        if ($token_decoded) {
            $user = $this->model->getUserById($token_decoded['id_user']);

            if ($user) {

                // Script lainnya

                if ( $user['last_change_password'] &&
$user['last_change_password'] != '0000-00-00 00:00:00' ) {
                    if ($token_decoded['iat'] <
strtotime($user['last_change_password'])) {
                        return;
                    }
                }

                // Script lainnya
            }
        }
}
```

Pada script diatas, jika iat token < data last_logout (`if ($token_decoded['iat'] < strtotime($user['last_change_password'])) `) maka berikan nilai kembalian (`return`) return.

Sama seperti pengecekan logout, pengecekan user yang ganti password juga dilakukan ketika melakukan pengecekan refresh token, hal ini bisa dilihat pada method getNewToken().

Pengecekan ini juga di lakukan ketika user melakukan request token baru menggunakan url login/refresh-token yaitu method refreshToken() pada controller Login (app\Controllers\Login.php).

8.7.3. Menarik (Revoke) Semua Token Yang Terbit

Pada kondisi tertentu mungkin kita ingin membuat semua token yang sudah terbit menjadi invalid dan meminta user untuk login dan merequest token yang baru.

Bagaimana cara melakukan hal tersebut?

Seperti kita ketahui bahwa token bersifat statis yang artinya jika sudah terbit maka tidak bisa dirubah. Langkah yang bisa kita lakukan hanya melakukan validasi dari sisi server. Dari desain sistem REST API yang sejauh ini telah kita kembangkan, maka kita dapat membuat semua token menjadi invalid dengan cara memperbarui nilai kolom last_logout tabel user. Dengan cara ini akan memaksa user untuk melakukan login ulang ke dalam sistem.

Cara lain yang mungkin bisa dilakukan adalah membuat data setting dengan nama misal invalidate_all_token yang berisi tanggal kita melakukan invalidate semua token, selanjutnya setiap token yang dikirim kita lakukan pengecekan, jika token tersebut dibuat sebelum tanggal pada nilai invalidate_all_token maka arahkan user ke halaman login.

8.8. Tentang Implementasi Token

Pada bagian sebelum kita telah membahas banyak tentang token JWT, berikut ini hal hal yang perlu Anda ketahui terkait implementasi JWT ini

8.8.1. Data Sensitif

Pada BAB 4 telah kita bahas bahwa JWT ini tidak terenkripsi hanya tersigned (tersignature) sehingga bagian payload dapat dibaca dengan mudah, untuk itu sebaiknya tidak mengekpose data senditif pada payload ini dan hanya menyertakan data seperlunya, pada aplikasi yang kita kembangkan kita hanya mengekpose data seperlunya yaitu hanya id_user, detail dati data user nantinya kita ambil di database.

8.8.2. Library JWT

Untuk mempermudah pengelolaan token JWT, kita dapat menggunakan library JWT yang dikembangkan untuk PHP. Terdapat beberapa pilihan library PHP, daftar library php yang dapat digunakan dapat dilihat di https://jwt.io/libraries?language=PHP, pada aplikasi REST API yang kita kembangkan, kita menggunakan library php-JWT dari Firebase https://github.com/firebase/php-jwt, alasan menggunakan library ini karena library ini aktif dikembangkan dan di maintain oleh Google, karena

yang mengembangkan Google, maka menurut penulis library ini dapat diandalkan. Pada aplikasi REST API yang kita kembangkan, library ini kita simpan di folder app\ThirdParty\Firebase.

Pada library php-JWT ini, ketika melakukan decode token jangan lupa untuk menggunakan block try {} catch {}, sehingga jika terjadi error (misal token expired) kita dapat menangkap pesan error apa yang dihasilkan, sebagaimana contoh pada method getNewToken() sebagai berikut:

```
try {
      $config = new \Config\App;
      $key = $config->jwtKeyAccessToken;
      $decoded_token = JWT::decode(trim($token), trim($key),
['HS256']);
      $result = (array) $decoded_token;
}
catch (Exception $e) {
      if (strtolower($e->getMessage()) == 'expired token') {
            $result = $this->setNewToken();
      }
}
```

8.8.3. Mengatur Usia Token

Pada BAB 4 telah kita bahas bahwa untuk alasan keamanan, Access Token dan Refersh Token memiliki usia tertentu, access token memilii usia yang pendek sedangkan refreesh token memiliki usia yang lebih panjang.

Pada aplikasi yang kita bangun, agar pengaturan batas waktu ini lebih fleksibel, batas waktu token ini kita simpan di datatabse sehingga nantinya dapat diatur dengan mudah melalui aplikasi client (menu Setting > Setting Token).

8.9. Module

Pada aplikasi REST API yang kita kembangkan, kita menggunakan sistem module untuk mengelola resource, misal module user untuk mengelola resource user, module produk untuk mengelola resource produk, dll, pada implementasinya, module ini sama dengan controller, sehingga jika kita mengakses module produk berarti sama juga dengan kita mengakses

controller produk, selanjutnya karena module sama dengan controller maka pemetaan module ini sama dengan pemetaan controller yaitu pada bagian routing.

Pada routing dengan autoroute aktif, maka controller otomatis dapat langsung dipetakan sehingga module pun juga dapat langsung dipetakan, misal untuk URI http://localhost/produk/add berarti controller nya adalah produk sehingga nama modulenya adalah produk, pada routing manual sebagaimana yang kita gunakan, controller dipetakan secara manual sehingga bisa jadi alamat URI tidak mencerminkan module, namun demikian pada aplikasi yang kita kembangkan agar mudah melakukan pengembangan dan maintenance aplikasi, alamat URI resource mencerminkan module yang diakses.

Penggunaan sistem module ini merupakan gagasan penulis sendiri yang awalnya penulis gunakan untuk mengembangkan aplikasi Admin Template, dengan menggunakan sistem module ini kita akan lebih leluasa mengontrol resource, misal mengaktifkan dan menonaktifkan resource, memberi permission pada resource, dll.

8.9.1. Membuat Module

Sebagaimana disebutkan sebelumnya bahwa module ini sama dengan controller, sehingga untuk membuat module caranya sama dengan membuat controller yaitu dengan membuat file ontroller pada folder app\Controllers. Module yang kita buat tersebut tidak serta merta dapat diakses, agar dapat diakses maka ada beberapa hal yang perlu dilakukan:

1. Mendaftarkan module pada database. Agar dapat melakukan pengaturan pada module (seperti mengaktifkan dan menonaktifkan module), maka pertama tama kita harus mendaftarkan module pada database. Pada database, data module ini disimpan pada tabel module, contoh datanya adalah sebagai berikut:

id_module	nama_module	judul_module	id_module_status	login	deskripsi
1	builtin/menu	Menu Manager	1	Y	Administrasi Menu
2	builtin/module	Module Manager	1	Y	Pengaturan Module
3	builtin/role	Role Manager	1	Y	Pengaturan Role
4	builtin/user	User Manager	1	Y	Pengaturan user
5	login	Login	1	R	Login ke akun
6	builtin/user-role	Assign Role ke User	1	Y	Assign role ke user
7	builtin/menu-role	Menu - Role	1	Y	Assign role ke menu

Karena module merupakan controller, maka penamaan module ini relatif terhadap path app\Controllers, sebagai contoh nama_module builtin/user-role maka nama file controller nya adalah User_role.php yang disimpan di folder (app\Controllers\Builtin), nantinya alamat URI untuk mengakses module tersebut adalah http://localhost/api/builtin/user-role.

2. Mendefiisikan permission pada module. Setelah didaftarkan module pada database selanjutnya kita harus mendefinisikan permission pada module, pendefinisian permission ini dilakukan pada controller Module_permission (app\Controllers\Builtin\Module_permission.php). Permission ini digunakan untuk mendefinisikan aksi apa yang dapat dilakukan pada module. Data permission ini disimpan pada database tabel module_permission, contoh datanya adalah sebagai berikut:

id_module_permission	id_module	nama_permission	judul_permission	keterangan
1	1	create	Create Data	Hak akses untuk menambah data
2	1	read_all	Read All Data	Hak akses untuk membaca data chartjs
3	1	update_all	Update All Data	Hak akses untuk mengupdate semua data
4	1	delete_all	Delete All Data	Hak akses untuk menghapus semua data
5	2	read_all	Read All Data	Hak akses untuk membaca data chartjs
6	2	update_all	Update All Data	Hak akses untuk mengupdate semua data
7	2	delete_all	Delete All Data	Hak akses untuk menghapus semua data

3. Melakukan assign permission pada role. Assign permission pada role bertujuan agar user yang memiliki role tertentu dapat melakukan tidakan pada module sesuai dengan permission yang di assign ke role tersebut, assign permission ini dilakukan pada controller

Role_permission (app\Controllers\Builtin\Role_permission.php), sedangkan data role permission ini disimpan pada database tabel role_module_permission, contoh datanya adalah sebagai berikut:

id_role	id_module_permission
1	1
1	2
1	3
1	4
1	5
1	6
1	7

Semua tahapan tersebut diatas terlihat panjang dan komplek, namun demikian pada aplikasi client yang kita kembangkan semua pengaturan tersebut dapat dilakukan dengan mudah di satu halaman (dibahas pada BAB 9)

8.9.2. Membaca Module

Ketika client mengakses resource pada aplikasi REST API maka aplikasi akan menentukan module apa yang akan digunakan untuk memproses request tersebut, karena module sama dengan controller maka module yang akan digunakan adalah controller yang telah didefinisikan pada bagian routing.

Pembacaan nama module yang digunakan dilakukan oleh method getCurrentModule() yang ada di BaseController, method ini akan memberi nilai pada property currentModule berupa data module beserta atribut module sesuai yang ada di database, contoh nilai dari property currentModule adalah sebagai berikut:

```
Array
(
    [id_module_status] => 1
    [id_module] => 8
    [nama_module] => builtin/menu-role
    [judul_module] => Menu - Role
    [login] => Y
    [deskripsi] => Assign role ke menu
    [nama_status] => Aktif
```

```
    [keterangan] =>
)
```

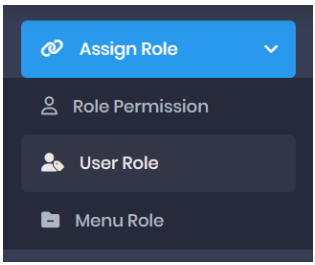
Setelah diketahui module yang digunakan unruk memproses request selanjutnya melalui method setUser() yang ada pada baseController aplikasi REST API akan menambahkan data permission yang ada pada module tersebut ke data user sesuai dengan role yang dimiliki oleh user.

8.9.3. Membuat Menu

Setelah berhasil membuat module, maka agar module tersebut dapat diakses dengan mudah maka kita perlu membuat menu untuk module tersebut, pembuatan menu ini dilakukan di controller Menu yang ada di folder (app\Controllers\Builtin) sedangkan data menu ini disimpan pada database tabel menu, adapun contoh datanya adalah sebagai berikut:

id_menu	nama_menu	id_menu_ka...	class	url	id_module
1	User	1	fas fa-users	builtin/user	5
2	Assign Role	1	fas fa-link	#	(NULL)
3	User Role	1	fas fa-user-tag	builtin/user-role	7
4	Module	1	fas fa-network-wired	builtin/module	2
6	Menu	1	fas fa-clone	builtin/menu	1
7	Menu Role	1	fas fa-folder-minus	builtin/menu-role	8

Note: pada contoh data diatas, data id_module digunakan untuk melakukan hightlight menu yang ada di parent, contoh sebagai berikut:



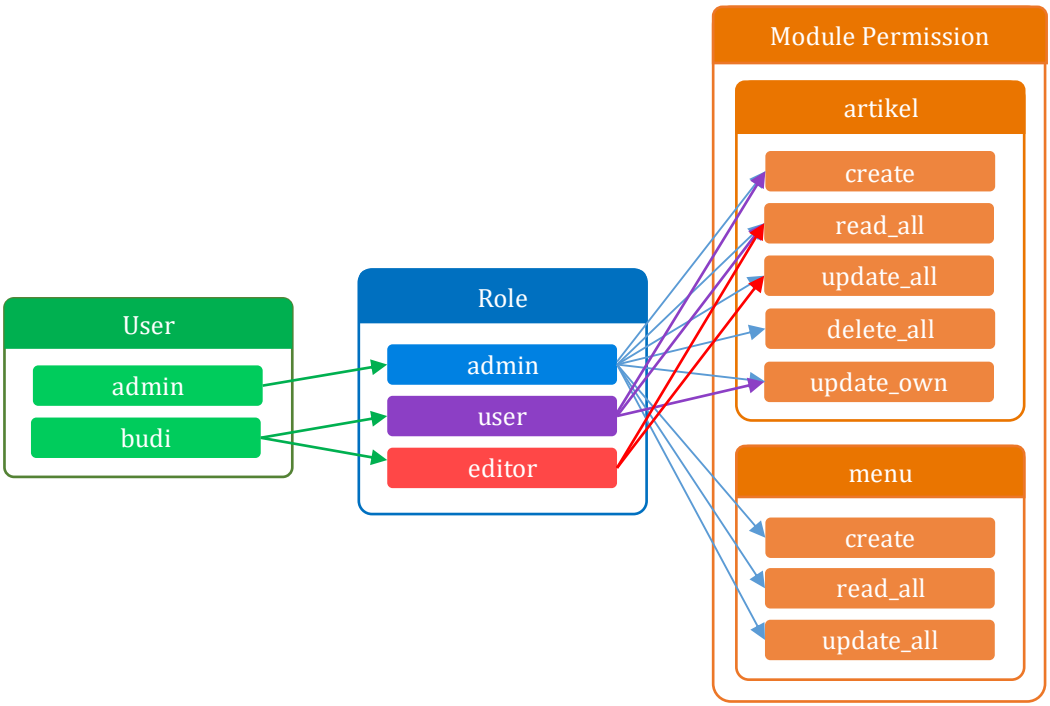
Pada contoh gambar diatas, menu Assign Role ikut terhighlight ketika user mengakses menu User Role.

Setelah menu dibuat, menu ini tidak serta merta muncul pada halaman user, agar muncul di halaman user maka kita perlu mengassign menu tersebut ke role yang dimiliki user, proses assign ini dilakukan pada controller Menu_role (app\Controllers\Builtin\Menu_role.php) sedangkan data assign ini disimpan pada database tabel menu_role, adapun contoh datanya adalah sebagai berikut:

id_menu	id_role
2	1
3	1
4	1
6	1
1	1

8.10. Role dan Permission

Sebagaimana telah dibahas pada sub bab 8.9. Module, dapat disimpulkan bahwa agar user dapat mengakses resoure/module pada sistem maka user tersebut harus memiliki role dan role tersebut harus memiliki permission pada module, jika digambarkan dalam bentuk diagram akan tampak seperti gambar berikut:



Pada gambar diatas:

- user admin memiliki role admin sehingga dapat melakukan semua aksi pada module.
- user budi memiliki role:
    - **user**, dengan role user maka pada module artikel budi dapat membuat (create) artikel, membaca semua artikel (read_all), dan mengubah/edit artikel yang hanya dia buat sendiri (update_own).
    - **editor**, dengan role editor maka pada module artikel budi dapat membaca semua artikel dan mengubah semua artikel (update all).

  meski memiliki role user dan editor, namun budi tidak memiliki role admin sehingga budi tidak dapat menghapus data artikel, selain itu budi juga tidak dapat mengakses module menu.

Pada penjelasan diatas dapat disimpulkan bahwa role adalah atribut yang dimiliki user yang berisi sekumpulan permission yang dapat digunakan untuk melakukan aksi pada module.

Catatan:
Mungkin anda bertanya tanya, kenapa permission pada module tidak langsung diassign saja ke user melainkan di assign ke role?

Hal ini bisa saja dilakukan tetapi kita akan kesulitan untuk mengelola permission yang ada pada user, misal terdapat 10 user, semua user tersebut diberi akses ke semua permission pada module, beberapa waktu kemudian kita ingin membatasi akses beberapa user terhadap module tertentu, maka kita harus satu per satu mengubah permission yang dimiliki oleh user, dengan mengassign permission pada role, maka dengan mudah kita dapat membuat role baru dan mengasign user yang akan dibatasi hak aksesnya ke role yang baru tersebut.

Permission pada module hanya sekedar atribut, bukan suatu sistem yang berjalan otomatis, sehingga permission yang telah kita definisikan ini tidak serta merta langsung berfungsi, untuk membuatnya berfungsi kita harus mendefinisikan sendiri fungsi atribut tersebut, berikut ini merupakan contoh bagaimana kita mendefinisikan fungsi pada permission update_all yang ada pada module produk: misal ketika client mengakses URI http://localhost/api/produk dengan metjhod PUT, maka berdasarkan routing yang telah didefinisikan yaitu:

```
$routes->resource('produk', ['controller' =>'Produk']);
```

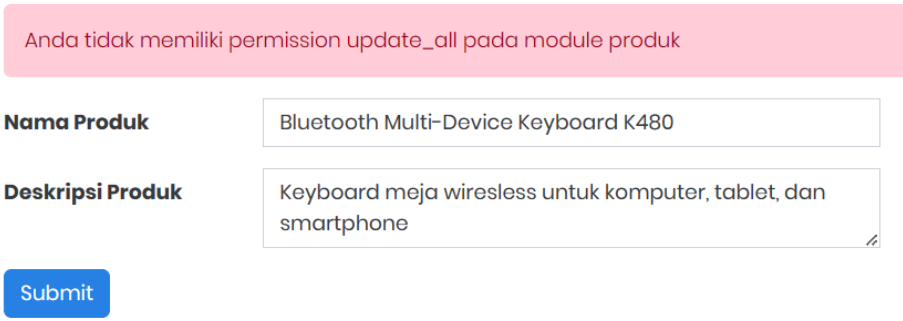
---

module yang diakses adalah module produk/controller Produk (file app\Controllers\Produk.php) dan method yang dieksekusi adalah method update(), adapaun script pada method update() pada controller Produk adalah sebagai berikut:

```
public function update($id = null)
{
```

```
        $this->hasPermission('update_all', true);

        $produk = $this->model->getProdukById($id);
        if (!$produk) {
            $response = ['status' => 'error', 'message' => 'Data
tidak ditemukan', 'error_type' => 'not_found'];
            return $this->fail($response);
        }

        return $this->saveData();
}
```

Pada script diatas, dengan method hasPermission('update_all', true) yang ada di BaseController kita cek apakah user memiliki permission update_all pada module produk, jika tidak maka maka akan muncul pesan error yang menyebutkan bahwa user tidak memiliki permission update_all pada module produk



Adapun script pada method hasPermission() adalah sebagai berikut:

```
protected function hasPermission($action, $exit = false)
{
        if (!in_array($action, $this->userPermission))
        {
            if ($exit) {
                $this->exitError('Anda tidak memiliki
permission ' . $action . ' pada module ' . $this-
>currentModule['nama_module'], 401);
            }
        }
        return in_array($action, $this->userPermission);
```

```
}
```

Pada script diatas method hasPermission() akan melakukan pengecekan apakah pada property userPermission terdapat key update_all, jika tidak (`if (!in_array($action, $this->userPermission))`) dan argumen exit pada method hasPermission() bernilai true (seperti contoh pada method update diatas), maka dengan method exitError() kirim pesan error ke user dan hentikan eksekusi script.

Property userPermission sendiri berupa data permission dalam bentuk array yang dimiliki oleh user pada module yang sedang diakses, data pada property ini diinisiasi oleh method getUserPermission() yang ada pada BaseController, adapun bentuk data dari property userPermission adalah sebagai berikut:

```
(
      [create] => create
      [delete_all] => delete_all
      [read_all] => read_all
      [update_all] => update_all
)
```

Pada data diatas dapat dikatakan bahwa pada module yang sekarang diakses, yaitu module produk, user memiliki permission create, delete_all, read_all, update_all.

Pada method hasPermission() untuk mengirim respon ke client dan menghentikan eksekusi script kita gunakan method exitError() yang ada di BaseController, adapun script pada method exitError() adalah sebagai berikut:

```
public function exitError($message = '', $code = 400,
$error_type = '') {
      $this->initAkses['status'] = 'error';
      $this->initAkses['message'] = $message;
      $this->initAkses['error_type'] = $error_type;
      $this->initAkses['status_code'] = $code;

      $response = service('response');
      $response->setStatusCode($code);
```

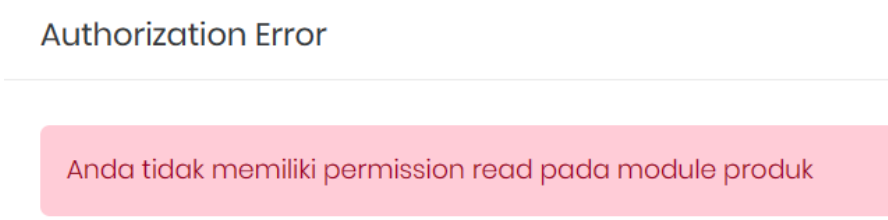
```
        $response->setJSON($this->initAkses);
        $response->setHeader('Content-type', 'application/json');
        $response->noCache();
        $response->send();
        exit;
}
```

Note: Pada method exitError() diatas terdapat argumen error_type, argumen ini terutama ditujukan untuk error 401 Unauthorized, untuk error 401 yang memerlukan login maka argumen error_type ini berisi string need_login, client yang menerima pesan error ini akan melakukan redirect ke halaman login. Error 401 tidak selalu mengharuskan user kembali login, sebagai contoh ketika user mengupdate resource produk namun tidak memiliki permission update atau ketika user mengakses module produk namun sama sekali tidak memiliki permission pada module produk, pada kondisi ini maka argumen error_type bisa diisi string selain need_login seperti missing_permission, dll.

Note: method hasPermission() dieksekusi di level controller, jika user sama sekali tidak memiliki permission atau user tidak memiliki permission read, maka pada method __construct() yang ada di BaseController kita kirim pesan error kepada user dan hentikan eksekusi script sebagai berikut:

```
If ($this->request->is('get') && !$this-
>hasPermissionPrefix('read')) {
        $this->exitError('Anda tidak memiliki permission read pada
module ' . $this->currentModule['nama_module'], 401,
'missing_permission_read');
}
```

Pada script diatas jika request method berupa GET (`is('get')`) dan user tidak memiliki permission read pada module yang sedang diakses maka jalankan method exitError(), misal jika user mengakses module produk namun tidak memiliki permission read maka client akan menampilkan pesan error yang menyebutkan bahwa user tidak memiliki permission read pada module produk, contoh seperti tampak pada pada gambar berikut:



Note: Agar module dapat diakses maka module tersebut harus memiliki permission read baik read_all maupun read_own, hal ini dikarenakan untuk menambah data, mengubah data, maupun menghapus data, maka user harus membaca data tersebut terlebih dahulu, jika tidak memiliki permission read maka akan muncul error seperti contoh sebelumnya "Anda tidak memiliki permission read pada module..."

Method userCan() dan hasPermissionPrefix()

Selain method hasPermission() method lain yang dapat digunakan untuk melakukan pengecekan permission adalah method userCan() yang ada di BaseController, method ini digunakan untuk pengecekan nama permission yang mengandung pemisah kata underscore (_), seperti update_all, read_own, dll, dengan method ini kita dapat mengetahui level permission user, seperti membaca semua data (all) atau membaca datanya sendiri (own).

Method userCan() ini akan menghasilkan kata setelah karakter underscore misal jika user memiliki permission update_all maka ketika method userCan('update') dieksekusi, hasil yang diperoleh adalah string all, contoh

penggunaan method ini dapat dilihat pada controller Setting_layout (app\Controllers\Builtin\Setting_layout.php) method update() sebagai berikut:

```
public function update($id = null)
{
      $input = $this->getRequestInput($this->request);
      $error = false;
      if ($this->userCan('update')) {
            // Script lainnya
      }
}
```

Pada contoh diatas, jika user memiliki permission dengan awalan update dengan akhiran apapun, all, own, atau yang lainnya (`if ($this->userCan('update'))`) maka lanjutkan eksekusi script.

Method lain yang dapat digunakan untuk melakukan pengecekan permission adalah method hasPermissionPrefix() yang ada di BaseController(), seperti namanya, method ini digunakan untuk melakukan pengecekan awalan permission, misal jika user memiliki permission update_all, maka jika dilakukan pengecekan menggunakan method hasPermissionPrefix(update_all) akan diperoleh nilai true, contoh penggunaan method ini ada pada BaseController method __construct() sebagai berikut:

```
if ($this->request->is('get') && !$this-
>hasPermissionPrefix('read')) {
      $this->exitError('Anda tidak diperkenankan mengakses module
' . $this->currentModule['nama_module'], 401, 'exit');
}
```

Pada script diatas, jika user tidak memiliki permission dengan awalan read maka dengan method exitError() hentikan eksekusi script dan kirim pesan error ke user.

Method hasPermissionPrefix() dan method userCan()terlihat mirip, namun demikian keduanya menghasilkan output yang berbeda, method hasPermissionPrefix() hanya menghasilkan nilai true atau false, sedangkan methid userCan() akan menghasilkan string tertentu seperti all, own, dll.

Custom Permission

Pada uraian diatas telah dijelaskan bahwa permission hanya atribut pada module, karena hanya berupa atribut maka kita bebas mendefinisikannya misal send_email, download_pdf, dll setelah mendefinisikannya kita harus membuat script (fungsi atau method) agar atribut tersebut dapat berfungsi, misal ketika kita membuat permission send_email pada module produk, maka agar permission tersebut dapat berfungsi kita harus membuat script sedemikian rupa sehingga hanya user yang memiliki permission send_email yang dapat mengirim email data produk.

Halaman ini sengaja dikosongkan
Jagowebdev.com

BAB 9 Mengembangkan Aplkasi Client

Setelah kita membahas bagaimana mengembangkan aplikasi REST API, pada bab ini kita akan membahas bagaimana mengembangkan aplikasi client yang digunakan untuk memanggil resource yang ada di aplikasi REST API.

Aplikasi client yang kita bahas kali ini dibangun menggunakan framework Codeigniter 4, aplikasi jadinya bisa dilihat pada file api-client.zip, sebagaimana telah dibahas pada BAB 1, aplikasi client ini penulis diinstall di folder xampp/htdocs/api-client/, sehingga nantinya untuk mengakses aplikasi ini, alamat URL yang digunakan adalah http://localhost/api-client/.

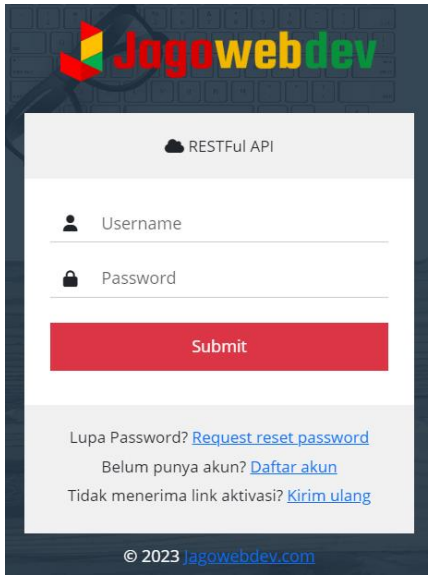
9.1. Authentication

Dalam pembahasan kita kali ini, hal pertama yang akan kita bahas adalah Autentikasi karena sebagaimana disebutkan pada BAB 8 untuk dapat mengkases resource pada aplikasi REST API kita harus memiliki token, sedangkan token ini diperoleh setelah proses authentikasi berhasil.

9.1.1. Form Login

Sebagaimana telah dibahas pada BAB 8 bahwa pada sistem REST API yang kita bangun skema authentikasi yang digunakan adalah skema bearer dimana untuk mendapatkan token maka client harus login kedalam sistem menggunakan username dan password.

Pada aplikasi client yang kita bangun, kita menggunakan form HTML sebagai sarana bagi user untuk login ke dalam sistem REST API, adapun tampilan form HTML nya seperti tampak pada gambar berikut:



Form login ini akan muncul setiap kali server mengirim response dengan kode 401 (Unauthorized), proses ini dilakukan pada method sendRequest() yang ada di file BaseController sebagai berikut:

```
protected function sendRequest($url, $method = 'get', $post_fields =
[], $headers = []) {

      // Script lainnya
      else if ($response_code == 401) {
            if ($this->currentModule['nama_module'] != 'login') {

                  if ($result['error_type'] == 'need_login') {
                        header('location: ' . base_url() . '/login');
                        exit;
                  }
            }
      }
      // Script lainnya
}
```

Pada script diatas terlihat bahwa ketika response code yang dikirim server 401 dan nilai dari key error_type adalah need_login, maka redirect ke halaman login.

9.1.2. Mendapatkan Token (Login)

Pada form HTML tersebut diatas, ketika user memasukkan username dan password dan mengklik tombol submit maka aplikasi client akan mengirim username dan password tersebut ke aplikasi REST API, proses ini dilakukan pada controller Login (app\Controllers\Login.php) method login(), method login() ini sendiri diakses oleh method index() yaitu ketika pada variabel $_POST terdapat data password, adapun script pada method index() adalah sebagai berikut:

```
public function index()
{
	$this->mustNotLoggedIn();

	$this->data['status'] = '';
	if ($this->request->getPost('password')) {

		$login = $this->login();
		if ($login) {
			return redirect()->to($this->config->baseURL);
		}
	}

	$this->data['style'] = ' style="max-width:375px"';
	return view('themes/modern/builtin/login', $this->data);
}
```

Pada script diatas, terlihat bahwa jika terdapat data password pada variabel $_POST ($this->request->getPost('password')) maka aplikasi akan mengakses method login(), hasil eksekusi method tersebut disimpan pada variabel $login, adapun script pada method login() adalah sebagai berikut:.

```
private function login()
{
	$url = $this->config->apiURL . 'login/login';
	$result = $this->sendRequest($url, 'post', ['username' => $this-
>request->getPost('username')
                                                                    ,
'password' => $this->request->getPost('password')
                                                                    ]
                                                );
```

```
        if ($result['status'] == 'error')
        {
                $this->data['message'] = $result['message'];
                return false;
        }

        if ($result['user']['status'] != 'active') {
                $this->data['message'] = 'Status akun Anda ' .
ucfirst($result['user']['status']);
                return false;
        }

        $this->setCookieToken($result);
        return $result;
}
```

Pada method login() diatas, melalui method sendRequest(), aplikasi mengirim username dan password ke server REST API, pada aplikasi REST API username dan password tersebut digunakan untuk proses login, jika proses login berhasil maka aplikasi REST API akan mengirim token (akses token dan refresh token) ke aplikasi client, selanjutnya oleh aplikasi client token tersebut disimpan di cookie browser, proses penyimpanan cookie ini dilakukan pada method setCookieToken() yang ada di BaseController. Pada method setCookieToken() tersebut kita beri flag Secure dan HttpOnly pada Cookie, hal ini dilakukan agar Cookie aman.

Note: selain dipanggil oleh method login(), method setCookieToken() ini juga dipanggil oleh method setToken() yaitu ketika user merequest token baru ke server melalui URI /refresh-token.

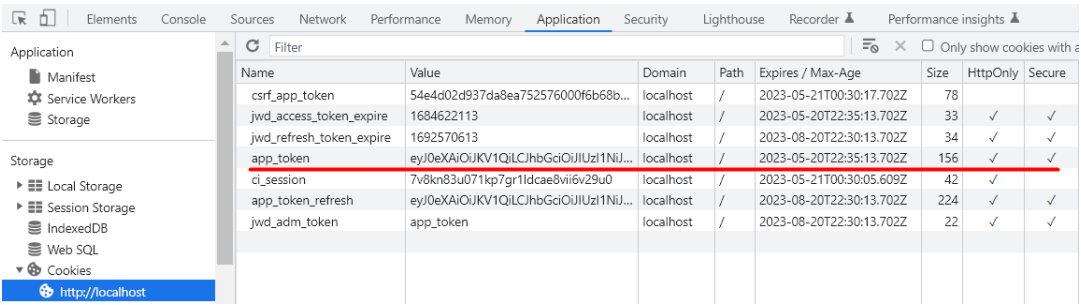
Setelah penyimpanan token kedalam cookie berhasil maka variabel $login pada method index() akan bernilai true, karena bernilai true maka aplikasi client akan melakukan redirect ke halaman depan (`return redirect()->to($this->config->baseURL);`)

9.1.3. Menyimpan Token

Pada pembahasan sebelumnya telah disebutkan bahwa aplikasi client menggunakan method setCookieToken() untuk menyimpan data token ke Cookie browser, adapun script setCookieToken() adalah sebagai berikut:

```
protected function setCookieToken($data)
{
	setcookie($this->config->serverTokenNameIdentifier, $data['token_name'], (
$data['refresh_token_expire'] ?: $data['access_token_expire'] ), '/', '', true,
true);
	setcookie($data['token_name'], $data['access_token'],
$data['access_token_expire'], '/', '', true, true);
	setcookie('jwd_access_token_expire', $data['access_token_expire'],
$data['access_token_expire'], '/', '', true, true);

	$_COOKIE[$this->config->serverTokenNameIdentifier] = $data['token_name'];
	$_COOKIE[$data['token_name']] = $data['access_token'];
	$_COOKIE['jwd_access_token_expire'] = $data['access_token_expire'];

	if ($data['refresh_token']) {
		setcookie($data['token_name'] . '_refresh', $data['refresh_token'],
$data['refresh_token_expire'], '/', '', true, true);
		setcookie('jwd_refresh_token_expire', $data['refresh_token_expire'],
$data['refresh_token_expire'], '/', '', true, true);
		$_COOKIE[$data['token_name'] . '_refresh'] = $data['refresh_token'];
		$_COOKIE['jwd_refresh_token_expire'] = $data['refresh_token_expire'];
	}
}
```

Pada script diatas, fungsi PHP yang digunakan untuk menyimpan cookie adalah setcookie(), pada fungsi tersebut, kita memberikan argumen true pada argumen ke 7 dan 8 yang artinya cookie yang disimpan akan diberi flag Secure dan HttpOnly. Jika kita cek pada Developer Tools Browser maka akan terlihat bahwa flag Secure dan HttpOnly pada cookie token akan tercentang.



Note: untuk melihat data cookie pada Developer Tools Browser. klik tab Application, kemudian pada panel sebelah kiri klik bagian Cookie. Data cookie diatas akan muncul setelah kita berhasil login ke sistem.

Flag Secure sendiri artinya bahwa cookie tidak bisa diakses oleh Javascript dan flag HttpOnly dimaksudkan agar cookie dikirim hanya melalui

protokol HTTP sehingga dengan memberi flag Secure dan HttpOnly ini data cookie akan menjadi lebih aman.

Mungkin Anda bertanya tanya kenapa menyimpan data token pada Cookie? tidak ditempat lain?

Pada Browser, data dapat disimpan di cookie, local storage, dan session storage, namun demikian kita lebih memiliih menggunakan cookie dengan pertimbangan sebagai berikut:

1. **Paling aman**. Pada browser, cookie merupakan tempat penyimpanan yang paling aman (meskipun masih memungkinkan adanya celah csrf), yaitu dengan memberikan flag Secure dan HttpOnly pada cookie.
2. **Memudahkan pengiriman token.** Dengan menyimpan token pada cookie akan memudahkan kita mengirim token ke server karena setiap kita mengakses url menggunakan browser, maka browser juga akan mengirim data cookie ke server melalui HTTP Header Cookie.
3. **Memudahkan debugging aplikasi**. Dengan cookie ikut dikirim oleh browser maka kita dapat langsung mengakses resource REST API melalui browser (dengan catatan alamat server API dan client API sama, misal sama sama localhost), hal ini memungkinkan karena pada aplikasi REST API yang kita bangun sebelumnya memungkinkan client untuk mengirim Authentikasi token melalui HTTP Header Cookie.
4. **Mengatur waktu expired**. Dengan menyimpan data pada cookie, maka kita dapat mendefinisikan waktu expired cookie (waktu expired cookie kita samakan dengan waktu expired token), sehingga setelah cookie expired, kita dapat langsung merequest token yang baru, tanpa harus melakukan testing request ke server.

Penamaan Cookie

Pada browser, data cookie disimpan dalam bentuk pasangan name dan value, pada gambar sebelumnya terlihat bahwa terdapat nama cookie app_token, cookie app_token ini digunakan untuk menyimpan data akses token, selain app_token juga terdapat cookie dengan nama

app_token_refresh, cookie app_token_refresh ini digunakan untuk menyimpan data refresh token.

Nama app_token merupakan nama default untuk cookie akses token, pengaturan nama ini dapat dilakukan pada aplikasi client menu Setting > Setting Token. Pada database, data nama ini ada di tabel setting dengan kolom type bernilai token dan kolom param bernilai token_name, sebagai berikut:

type	param	value
token	refresh_token_age	3
token	refresh_token_age_lim...	Y
token	refresh_token_age_unit	bulan
token	token_age	2
token	token_age_unit	hari
token	token_name	app_token
token	using_refresh_token	N

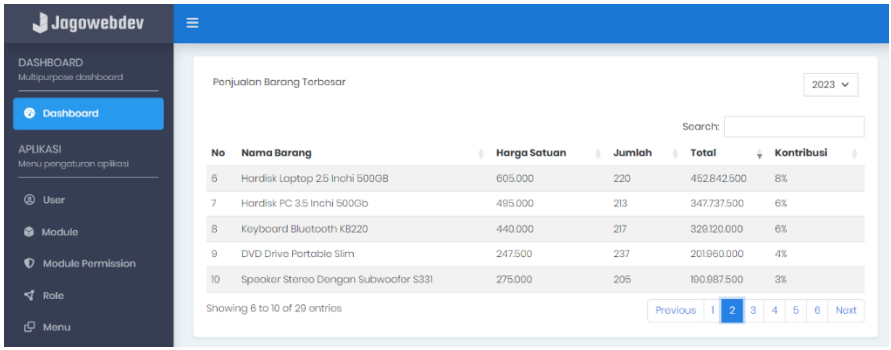
sedangkan nama app_token_refresh merupakan nama app_token ditambah akhiran _refresh.

Menyisipkan Token Pada Dokumen HTML

Selain menyimpan token pada cookie, pada aplikasi api client yang kita kembangkan, kita juga menyisipkan token pada dokumen HTML, tepatnya pada tag <meta> bagian <head>.

Kenapa demikian?

Pada aplikasi client yang kita kembangkan, selain menggunakan PHP untuk melakukan request ke aplikasi REST API, kita juga menggunakan Javascript Ajax, contoh pada data penjualan terbesar yang ada pada halaman dashboard



Note: data penjualan terbesar diatas ditampilkan menggunakan library datatables ajax sehingga request data ke server dilakukan menggunakan javascript ajax.

Setiap request yang dikirim ke server REST API (termasuk request menggunakan Javascript Ajax) harus menyertakan token, sedangkan token yang kita simpan pada Cookie kita beri flag Secure sehingga tidak bisa diakses melalui javascript, lantas bagaimana caranya agar token tersebut dapat dikirim?

Untuk mengatasi permasalahan tersebut diatas salah satu cara yang dapat kita lakukan adalah dengan menyisipkan data access token dan refresh token pada meta html header, contoh implementasinya dapat dilihat pada file header.php yang ada di folder app\Views\themes\modern sebagai berikut:

```
$cookie_token_name = $cookie_access_token_name = @$_COOKIE[$config-
>serverTokenNameIdentifier];
$cookie_refresh_token_name = $cookie_token_name . '_refresh';
$access_token_expire = @$_COOKIE['jwd_access_token_expire'] ?: '';
$refresh_token_expire = @$_COOKIE['jwd_refresh_token_expire'] ?: '';

echo'<meta name="' . $cookie_access_token_name . '" content="' .
@$_COOKIE[$cookie_access_token_name] . '">';
echo "\r\n" . '<meta name="' . $cookie_refresh_token_name . '"
content="' . @$_COOKIE[$cookie_refresh_token_name] . '">';
```

dengan script diatas maka setiap kita membuka halaman aplikasi client (setelah login) pada tag <meta> akan terlihat data akses token dan data refresh token.

```
<title>Dashboard | RESTFul API Codeigniter 4</title>
<meta name="descrition" content="Dashboard"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="shortcut icon" href="https://codeliro.com/demo/api/public/images/favicon.png?r=1680410028" />

<meta name="app_token" content="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2ODA0MTAwMTYsImV4cCI6MTY4
<meta name="app_token_refresh" content="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2ODA0MTAwMTYsImlk
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/fontaw
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/bootst
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/bootst
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/sweeta
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/overla
```

Catatan:

Selanjutnya muncul pertanyaan, bukankah hal ini sama dengan mengekpose data token, sehingga berpotensi membuka celah keamanan?

Hal ini menurut penulis masih aman karena data token tersebut muncul hanya jika user berhasil login ke server.

Agar pengambilan data token dengan Javascript dapat dilakukan dengan mudah maka kita perlu mendefinisikan beberapa parameter (variabel) javascript, variabel ini nantinya digunakan untuk berbagai keperluan seperti pengecekan waktu expired token, pengecekan nama token pada tag <meta> pada HTML header, dll, nilai variabel ini berubah ubah sesuai data yang ada pada database sehingga kita tidak bisa menuliskannya secara langsung pada script (hard coding), untuk itu kita perlu menggunakan PHP, Script yang digunakan untuk mendefinisikan variabel ini ada di file header.php (file app\Views\theme\modern\header.php) bagian tag <head> sebagai berikut:

```
<script type="text/javascript">
      let user_permission = <?=json_encode($user_permission)?>;

      cookie_access_token_name = "<?=$cookie_access_token_name?>";
      cookie_refresh_token_name = "<?=$cookie_refresh_token_name?>";
      access_token_expire = "<?=$access_token_expire?>";
      refresh_token_expire = "<?=$refresh_token_expire?>";
</script>
```

Variabel ini kita tempatkan di paling atas agar nantinya dapat diakses secara global.

Jika aplikasi dijalankan, tampilan source (View Source) variabel javascript tersebut menjadi sebagai berikut:

```
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/material-icons/css.cs
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/vendors/datatables/extensions/
<link rel="stylesheet" type="text/css" href="https://jagowebdev.com/demo/api-client/public/themes/modern/css/dashboard.cs
<script type="text/javascript">
    let base_url = "https://jagowebdev.com/demo/api-client/";
    let module_url = "https://jagowebdev.com/demo/api-client/dashboard";
    let current_url = "https://jagowebdev.com/demo/api-client/index.php/dashboard";
    let theme_url = "https://jagowebdev.com/demo/api-client//public/themes/modern/builtin/";
    let api_url = "https://codeliro.com/demo/api/";
    let filepicker_server_url = "https://codeliro.com/demo/api/filepicker/";
    let filepicker_icon_url = "https://codeliro.com/demo/api/public/images/filepicker_images/";
    let current_bootswatch_theme = "cosmo";
    let user_permission = {"create":"create","delete_all":"delete_all","read_all":"read_all","update_all":"update_all"};

    cookie_access_token_name = "app_token";
    cookie_refresh_token_name = "app_token_refresh";
    access_token_expire = "1680410671";
    refresh_token_expire = "1688272771";
</script>
```

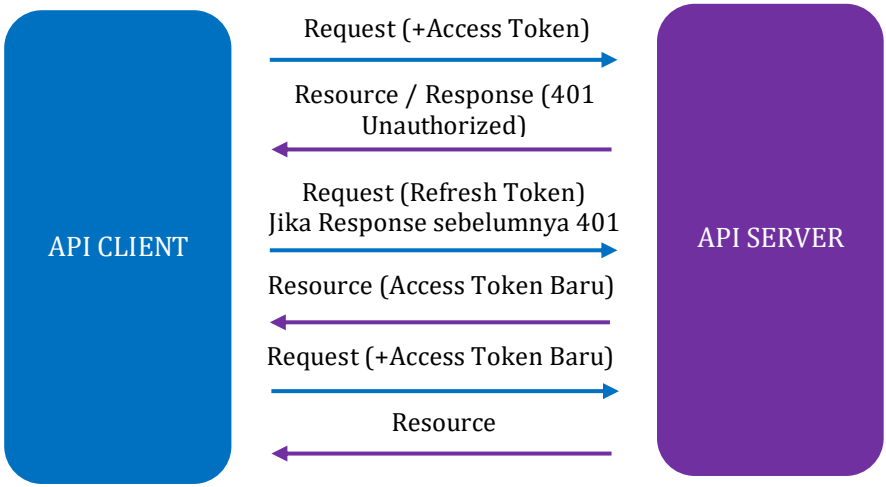
9.2. Request dan Response

Pada bagian ini kita akan membahas bagaimana aplikasi client yang kita bangun mengelola request yang dikirim ke server dan response yang di terima dari server.

9.2.1. Skema Request

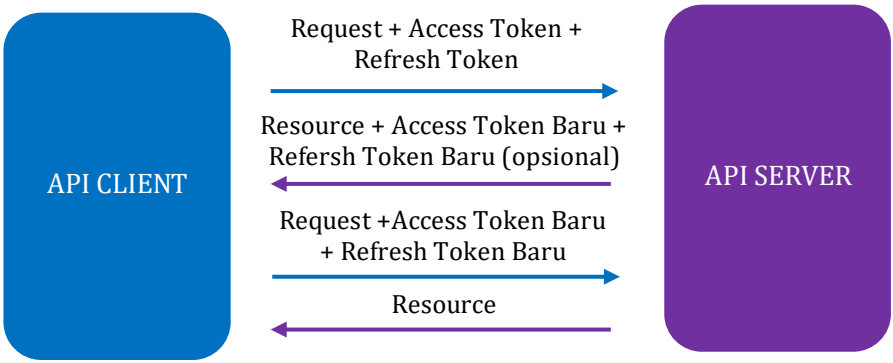
Sebelum membahas pengelolaan request dan response pada aplikasi client, kita perlu memahami alur komunikasi (baik request maupun response) antara aplikasi client dan aplikasi REST API.

Komunikasi antara aplikasi client dan server terjadi dimulai dengan client mengirim request ke aplikasi server, jika request tersebut berhasil di authentikasi oleh server maka server akan mengirim resource ke client, namun demikian jika proses authentikasi gagal dan server memberikan response 401 (Unauthorized) maka client mengirim refresh token untuk mendapatkan akses token yang baru. Setelah mendapat akses token yang baru, client kembali mengulang request yang pertama menggunakan akses token yang baru tersebut. Jika digambarkan keedalam bentuk diagram maka akan tampak seperti gambar berikut:



Alur komunikasi diatas merupakan alur komunikasi yang umum terjadi antara aplikasi client dan aplikasi api, pada aplikasi client yang kita kembangkan, tidak semua proses diatas kita lakukan, ketika aplikasi client tahu bahwa Access Token sudah expired (ketika membaca cookie pada aplikasi client) maka aplikasi client akan langsung melakukan request access token yang baru kemudian dengan access token yang baru tersebut aplikasi client melakukan request resource ke aplikasi api.

Pada kondisi lain yaitu ketika menggunakan datatables ajax, selain mengirim akses token, aplikasi client juga mengirim refresh token sehingga jika nantinya akses token expired, maka dapat langsung digenerate akses token yang baru, alurnya seperti tampak pada gambar berikut:



Pada ganbar diatas terlihat bahwa pada request pertama selain akses token, client juga mengirim refresh token, jika akses token belum expired maka server akan mengirim resource saja, namun jika akses token expired maka selain resource server juga mengandung akses token yang baru.

9.2.2. Mengirim request dan mengelola Response

Pada aplikasi client yang kita bangun, pengiriman request dan pengelolaan response dilakukan pada method sendRequest() yang ada di BaseController (app\Controllers\BaseController.php), dalam praktik, method ini banyak digunakan di file controller (app\Controllers). Adapun script pada method sendRequest() adalah sebagai berikut:

```
protected function sendRequest($url, $method = 'get', $post_fields = [],
$headers = []) {

      if ($method == 'post' && empty($post_fields)) {
            return ['status' => 'error', 'message' => 'POST data masih
kosong'];
      }

      $method = strtolower($method);
      $headers['Origin'] = $_SERVER['REQUEST_SCHEME'] . '://' .
$_SERVER['HTTP_HOST'];

      $cookie_access_token_name = $cookie_token_name = @$_COOKIE[$this->config-
>serverTokenNameIdentifier];
```

```
        if ($cookie_access_token_name) {
                if (@key_exists($cookie_access_token_name, $_COOKIE))
                {
                        $headers['Authorization'] = 'Bearer ' .
$_COOKIE[$cookie_access_token_name];
                } else {

                        // Request refresh token
                }
        }

        $ch = curl_init();
        if ($headers) {
                $str_header = [];
                foreach ($headers as $key => $val) {
                        $str_header[] = $key . ':' . $val;
                }
                curl_setopt($ch, CURLOPT_HTTPHEADER, $str_header);
        }

        curl_setopt($ch, CURLOPT_URL, $url);

        if ($headers) {
                curl_setopt($ch, CURLOPT_HTTPHEADER, $str_header);
        }

        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, strtoupper($method));
        curl_setopt($ch, CURLOPT_HEADERFUNCTION, [$this, 'parseResponseHeader']);
        curl_setopt($ch, CURLOPT_TIMEOUT, 120);
        curl_setopt($ch, CURLOPT_VERBOSE, 1);
        curl_setopt($ch, CURLINFO_HEADER_OUT, true);

        if ($post_fields && is_array($post_fields)) {
                $post_fields = $this->flattenArray($post_fields);
        }

        if ( $method == 'post' || $method == 'put') {
                curl_setopt($ch, CURLOPT_POST, true);
                curl_setopt($ch, CURLOPT_POSTFIELDS, $post_fields);
        }

        $result = curl_exec($ch);
        $response_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
        $header_out = curl_getinfo($ch, CURLINFO_HEADER_OUT);

        curl_close($ch);
```

```
        if ($response_code) {
            if ($this->responseContentType == 'application/json') {
                $result = json_decode($result, true);
            }

            $response_exit = [404, 500];

            if (in_array($response_code, $response_exit))
            {
                $this->data['status'] = 'error';
                $this->data['title'] = 'Server Error ' . $response_code;

                $this->data['content'] = '';
                if (is_array($result)) {
                    if (@$result['status'] == 'error') {
                        $this->data['content'] = $result['message'];
                    }
                }

                if (!$this->data['content']) {
                    $this->data['content'] = $result;
                }

                $this->data['title'] = 'ERROR ' . $response_code;
                $this->exitError($this->data);

            } else if ($response_code == 401) {
                if ($this->currentModule['nama_module'] != 'login') {

                    if ($result['error_type'] == 'need_login') {
                        header('location: ' . base_url() . '/login');
                        exit;
                    }
                }
            }
            else if ($response_code == 204) {
                $result = ['status' => 'error', 'message' => 'Data tidak
ditemukan'];
            } else {
                if (is_array($result)) {
                    if ( key_exists('status', $result) &&
is_numeric($result['status']) ) {
                        if (key_exists('messages', $result)) {
                            $result = $result['messages'];
                        }
                    }
                }
                return $result;
```

```
            }

      } else {
            $result = [];
            $result['status'] = 'error';
            $result['error_type'] = 'failed_to_connect';
            $result['message'] = 'Gagal terhubung ke Server API: ' . $this-
>config->apiURL;
      }
      return $result;
}
```

Pada script diatas kita menggunakan library PHP CURL untuk mengambil data via URL, PHP sendiri menyediakan berbagai cara untuk mengambil data via URL, diantaranya menggunakan menggunakan library CURL (yang kita gunakan), menggunakan fungi file_get_contents(), dan fungsi fopen(). Kita memilih menggunakan library CURL karena dengan CURL ini kita lebih leluasa mengatur request yang dikirim, seperti mendefinisikan sendiri HTTP Request Header.

Mengirim Request

Pada script sendRequest() diatas, setiap kali kita mengirim request ke server api, maka pertama tama kita cek apakah pada variabel $_COOKIE terdapat key nama token (app_token), jika ya maka kita tambahkan header Authorization pada request yang kita kirim sebagai berikut:

```
if (@key_exists($cookie_access_token_name, $_COOKIE))  {
      $headers['Authorization'] = 'Bearer ' .
$_COOKIE[$cookie_access_token_name];
}
```

Note: Pada bab 9.1.1. telah dibahas bahwa token yang diperoleh dari server kita simpan pada cookie sehingga ketika browser mengirim request ke server (server aplikasi client) maka data cookie akan ikut terkirim sehingga dengan script diatas kita cek apakah terdapat nama cookie akses token.

Jika cookie app_token tidak ditemukan yang berarti akses token sudah expired dan dihapus oleh browser, maka kita cek apakah terdapat cookie

refresh token app_token_refresh, jika ya maka kita request token baru (access token dan refresh token) ke server melalui URI http://localhost/api/login/refresh-token, adapun scriptnya adalah sebagai berikut:

```
if (@key_exists($cookie_refresh_token_name, $_COOKIE)) {

      $ch = curl_init();
      $url_refresh = $this->config->apiURL . 'login/refresh-token';
      curl_setopt($ch, CURLOPT_URL, $url_refresh);

      if ($headers) {
            curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
      }

      curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
      curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
      curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
      curl_setopt($ch, CURLOPT_HEADERFUNCTION, [$this,
'parseResponseHeader']);
      curl_setopt($ch, CURLOPT_TIMEOUT, 120);
      curl_setopt($ch, CURLOPT_VERBOSE, 1);
      curl_setopt($ch, CURLINFO_HEADER_OUT, true);
      curl_setopt($ch, CURLOPT_POST, true);
      curl_setopt($ch, CURLOPT_POSTFIELDS, ['grant_type' => 'refresh_token',
'refresh_token' => $_COOKIE[$cookie_refresh_token_name] ]);
      $result = curl_exec($ch);
      if ($result) {
            if ($this->responseContentType == 'application/json')          {
                  $result = json_decode($result, true);
                  if (@$result['code'] == 500) {
                        $this->exitError(['status' => 'error', 'title' =>
'Server Error 500', 'content' => $result]);
                  }
            }

            $headers['Authorization'] = 'Bearer ' . $result['access_token'];
            $this->setCookieToken($result);
      }
}
```

Pada script diatas, refresh token dikirim melalui HTTP Request Body dengan method post, hal ini sesuai dengan standar yang telah ditetapkan bahwa refresh token dikirim pada request body menggunakan method post, lebih lanjut dapat dibaca di halaman "RFC 6749: The OAuth 2.0

Authorization Framework" yang dapat diakses melalui tautan https://www.rfc-editor.org/rfc/rfc6749#section-6

Selanjutnya jika server memberikan response token baru maka kita perbaruai data token pada cookie dan data token pada variabel $_COOKIE melalui method setCookieToken().

Flatten Array

Selanjutnya pada script sendRequest() diatas terdapat pemanggilan method falttenArray() sebagai berikut:

```
if ($post_fields && is_array($post_fields)) {
      $post_fields = $this->flattenArray($post_fields);
}
```

method ini digunakan untuk mengubah multi dimensional array menjadi single dimensional array, misal multi dimensional array berikut:

```
Array
(
    [role] => Array
        (
            [artikel] => Array
                (
                    [0] => penulis
                    [1] => editor
                )

            [sistem] => Array
                (
                    [0] => admin
                    [1] => user
                )

        )

    [submit] =>
)
```

akan diubah menjadi:

```
Array
```

```
(
    [role[artikel][0]] => penulis
    [role[artikel][1]] => editor
    [role[sistem][0]] => admin
    [role[sistem][1]] => user
    [submit] =>
)
```

Kenapa hal ini perlu kita lakukan?

Hal ini diperlukan karena ketika mengirim data menggunakan library CURL, maka bentuk data yang dikirim harus sama dengan bentuk data yang dikirim menggunakan form HTML, misal terdapat form sebagai berikut:

```
<?php
      if (isset($_POST['submit'])) {
            echo '<pre>'; print_r($_POST); echo '<pre>';
      }
?>

<form method="post">
      <label>
            <input type="checkbox" name="role[artikel][]"
value="penulis" checked> Penulis
      </label>
      <label>
            <input type="checkbox" name="role[artikel][]"
value="editor" checked> Editor
      </label>
      <label>
            <input type="checkbox" name="role[sistem][]"
value="admin" checked> Admin
      </label>
      <label>
            <input type="checkbox" name="role[sistem][]"
value="admin" checked> User
      </label>
      <button name="submit" type="submit">Submit</button>
</form>
```

maka jika form html tersebut dikirim menggunakan CURL maka harus dibuat menjadi bentuk sebagai berikut:

```
Array
```

```
(
    [role[artikel][0]] => penulis
    [role[artikel][1]] => editor
    [role[sistem][0]] => admin
    [role[sistem][1]] => user
    [submit] =>
)
```

Selanjutnya ketika server api menerima data tersebut, data tersebut akan disimpan pada variabel $_POST dengan bentuk multidimensional array seperti layaknya form disubmit sebagai berikut:

```
Array
(
    [role] => Array
        (
            [artikel] => Array
                (
                    [0] => penulis
                    [1] => editor
                )

            [sistem] => Array
                (
                    [0] => admin
                    [1] => user
                )

        )

    [submit] =>
)
```

9.2.3. Mengelola Response

Ketika server memberikan response, maka pertama tama kita cek apakah server memberikan response code, jika ya maka kita cek apakah Content-type pada header response tersebut bernilai application/json, jika ya maka decode data json yang ada pada HTTP Body, **hasilnya simpan pada variabel $result**.

```
if ($response_code) {
      if ($this->responseContentType == 'application/json') {
```

```
            $result = json_decode($result, true);
      }

      // Code lainnya
}
```

Jika response code nya adalah 404 atau 500 maka keluar dari aplikasi dan tampilkan pesan error.

```
$response_exit = [404,500];
if (in_array($response_code, $response_exit))
{
      $this->data['status'] = 'error';
      $this->data['title'] = 'ERROR ' . $response_code;
      $this->data['content'] = $result['message'];
      $this->exitError($this->data);
}
```

Note: utuk mempermudah proses keluar halaman dan menampilkan pesan error, kita buat method exitError(). Method ini berada satu file dengan method sendRequest() yaitu di app\Controllers\BaseController.php

Selanjutnya jika response code nya adalah 401 dan module yang sedang diakses bukan merupakan module login dan nilai error_type adalah need_login maka redirect ke halaman login.

```
else if ($response_code == 401) {
      if ($this->currentModule['nama_module'] != 'login') {
            if ($result['error_type'] == 'need_login') {
                  header('location: ' . base_url() . '/login');
                  exit;
            }
      }
}
```

Berikutnya Jika response code nya adalah 204 (tidak ada data yang dikirim) kita buat pesan error bahwa data tidak ditemukan dan update nilai variabel $result dengan pesan error tersebut.

```
else if ($response_code == 204) {
```

```
      $result = ['status' => 'error', 'message' => 'Data tidak
ditemukan'];
}
```

Jika response code bernilai selain 401,404, 500, dan 204, maka kita cek apakah nilai key status yang ada pada response body berupa angka numeric seperti 400, jika ya maka kita cek, apakah response body memiliki key message, jika ya, update varibel $result dengan message pada response body tersebut.

```
if ( key_exists('status', $result) &&
is_numeric($result['status']) ) {
      if (key_exists('messages', $result)) {
            $result = $result['messages'];
      }
}
```

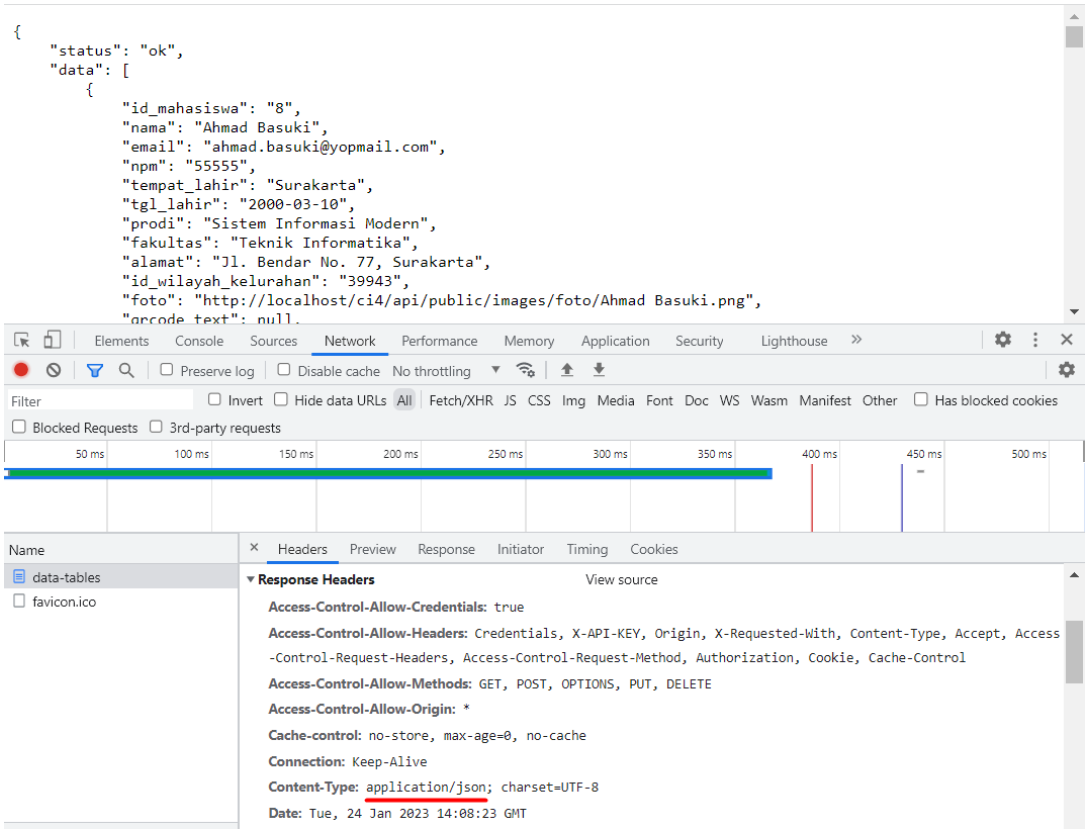
Note: response code 400 kita gunakan salah satunya jika parameter yang dikirim client tidak lengkap, misal ketika melakukan update data dan ada parameter yang tidak dikirim.

Terakhir beri nilai kembalian method `sendRequest()` dengan variabel $result.

Contoh penggunaan method sendRequest() adalah untuk melakukan pengambilan resource pada aplikasi rest api, misal pada module Datatables app\Controllers\Data_tables.php terdapat script sebagai berikut:

```
$data['result'] = $this->sendRequest($this->config->apiURL .
'data-tables');
```

Pada request diatas client mengirim permintaan ke server api menggunakan method GET, selanjutnya server akan memberikan respon sebagai berikut:



Contoh berikutnya pada saat menambahkan data, pada controller Produk (app\Controllers\Produk.php) method add() kita mengirim permintaan untuk menambahkan data sebagai berikut

```
$this->sendRequest($this->config->apiURL . 'produk', 'POST',
$_POST);
```

Pada method diatas request method yang kita gunakan adalah post dan kita tidak mendefinisikan header Content-Type (karena data yang kita kirim bukan data JSON), karena header tidak kita definisikan, maka otomatis PHP akan mendefinisikan header Content-Type sesuai dengan data yang dikirim. Pada contoh diatas, nilai header Content-type yang ditambahkan PHP adalah:

```
Content-Type: multipart/form-data;
```

Contoh berikutnya merubah data, pada controller Produk (app\Controllers\Produk.php) method edit() kita mengirim permintaan untuk update data sebagai berikut:

```
$this->sendRequest($this->config->apiURL . 'produk/' .
$_GET['id'], 'PUT', json_encode($_POST), ['Content-Type' =>
'application/json']);
```

Pada contoh diatas, request yang kita kirim menggunakan method PUT, data yang kita kirim berbentuk json `json_encode($_POST)`, dan header Content-Type kita beri nilai application/json.

Kenapa kita perlu mengirim data berbentuk JSON seperti diatas? karena seperti telah kita bahas sebelumnya, pada PHP data yang bisa ditangkap hanya data yang dikirim menggunakan method get dan post dimana masing-masing akan disimpan pada variabel `$_GET` dan `$_POST`, sedangkan data yang kita kirim sekarang menggunakan method PUT, sehingga data tersebut tidak dapat disimpan pada variabel $_GET dan $_POST, untuk itu agar data pada body dapat dengan mudah diparsing maka kita gunakan data dengan format JSON, nantinya oleh aplikasi api kita data JSON tersebut akan diubah menjadi data array (menggunakan method `getRequestInput()`).

Pada pembahasan sebelumnya telah dibahas bahwa pada update data yang menyertakan file, method yang digunakan adalah POST kenapa? Karena pada PHP variabel `$_FILES` (yang digunakan untuk menghandle file yang diupload) hanya bisa digunakan pada method POST. Sebagai contoh pada controller Data_tables (app\Controllers\Data_tables.php) method sendRequest() yang digunakan untuk menambahkan dan mengupdate data sama yaitu:

```
$this->sendRequest($url, 'POST', $post_fields);
```

Kenapa demikian? Karena data yang dikirim ke server didalam nya terdapat file image.

Selanjutnya seperti yang telah kita bahas pada method spoofing, agar Codeigniter membaca seolah olah request method yang diterima adalah put (meskipun pada header nilainya post), maka ketika melakukan update data kita perlu menambahkan data _method dengan value PUT pada HTTP Body yang dikirim sebagai berikut:

```
if ($_POST['id']) {
      $post_fields['_method'] = 'PUT';
      $url = $this->config->apiURL . 'data-tables/' . $_POST['id'];
}
```

Pada script diatas, jika ada data yang diedit (ada key id pada variabel $_POST) maka pada data yang dikirim kita tambahkan `$post_fields['_method'] = 'PUT';` sehingga pada aplikasi api yang dieksekui adalah method untuk update data (put), selain itu url nya juga kita buat menjadi `$this->config->apiURL . 'data-tables/' . $_POST['id'];` karena di konfigurasi api kita, untuk update data routingnya mengikuti shorthand routing bawaan Codeigniter:

```
$routes->resource('data-tables', ['controller' =>'Data_tables']);
```

```
D:\xampp\htdocs\api>php spark routes

CodeIgniter v4.1.9 Command Line Tool - Server Time: 2023-01-24 16:56:48 UTC-06:00

+--------+---------------------------+-----------------------------------------------+
| Method | Route                     | Handler                                       |
+--------+---------------------------+-----------------------------------------------+
| GET    | data-tables               | \App\Controllers\Data_tables::index           |
| GET    | data-tables/new           | \App\Controllers\Data_tables::new             |
| GET    | data-tables/(.*)/edit     | \App\Controllers\Data_tables::edit/$1         |
| GET    | data-tables/(.*)          | \App\Controllers\Data_tables::show/$1         |
| POST   | data-tables               | \App\Controllers\Data_tables::create          |
| PATCH  | data-tables/(.*)          | \App\Controllers\Data_tables::update/$1       |
| PUT    | data-tables/(.*)          | \App\Controllers\Data_tables::update/$1       |
| DELETE | data-tables/(.*)          | \App\Controllers\Data_tables::delete/$1       |
| CLI    | migrations/([^/]+)/([^/]+) | \CodeIgniter\Commands\MigrationsCommand::$1/$2 |
| CLI    | migrations/([^/]+)        | \CodeIgniter\Commands\MigrationsCommand::$1   |
| CLI    | migrations                | \CodeIgniter\Commands\MigrationsCommand::index |
| CLI    | ci(.*)                    | \CodeIgniter\CLI\CommandRunner::index/$1      |
+--------+---------------------------+-----------------------------------------------+
```

Contoh berikutnya adalah method `sendRequest()` untuk menghapus data. Pada controllers Data_tables kita mengirim permintaan menghapus data dengan menjalankan perintah berikut:

```
$this->sendRequest($this->config->apiURL . 'data-tables/' .
$_POST['id'], 'DELETE');
```

Note: pada contoh diatas, request dikirim tanpa data (tanpa HTTP Body).

9.2.4. Mengirim Request Dengan Javascript

Pada method sendRequest() yang kita bahas sebelumnya, jika cookie access token tidak ditemukan, maka aplikasi client akan mengirim permintaan access token yang baru, selanjutnya dengan access token yang baru tersebut aplikasi client akan mengirim request yang sebenarnya, proses tersebut dapat dilakukan secara berurutan (Syncronous).

Javascript, by nature memiliki sifat Asyncronous, dimana eksekusi script tidak menunggu script yang lain selesai dieksekusi, sehingga jika ada request yang di kirim ke server, maka untuk mengeksekusi script berikutnya javascript tidak meunggu request sebelumnya selesai. Untuk bahasa front end (User Interface) seperti javascript memang seharnya didesain seperti ini karena akan meningkatkan user experience, user dapat langsung melihat tampilan awal web secara keseluruhan sambil menunggu request ke server selesai.

Dalam konteks aplikasi api client yang kita kembangkan, dengan behavior javascript yang seperti itu, maka ketika membuat fungsi untuk mengirim request, logic yang digunakan agak berbeda dengan method sendRequest yang kita buat pada PHP.

Pada aplikasi api client yang kita kembangkan kita menggunakan fungsi sendRequest() untuk mengelola request yang dikirim melalui Javascript. Fungsi sendRequest() ini ada di file functions.js yang ada di folder public\themes\modern\builtin\js sebagai berikut:

```
function sendRequest( param = {}) {
```

```
	if ( !('url' in param) ) {
		return bootbox.alert('URL harus diisi');
	}

	paramDefault = {
			method: 'GET',
			data : null,
			success : null,
			error: null,
			headers: {}
		}
	$.extend(paramDefault, param);

	accessToken = '';
	date_now = Date.now();
	date_token_expire = access_token_expire * 1000;
	accessToken = $('meta[name="' + cookie_access_token_name +
'"]').attr('content');

	if ( date_token_expire < date_now || !accessToken) {
		date_token_expire = refresh_token_expire * 1000;
		refresh_token = $('meta[name="' + cookie_refresh_token_name +
'"]').attr('content');
		if ( refresh_token ) {
			if (date_token_expire > date_now) {
				// Request new token
			} else {
				if (module_url.substr(-5) != 'login') {
					window.location = base_url + 'login';
				}
			}
		} else {
			if (module_url.substr(-5) != 'login') {
				window.location = base_url + 'login';
			}
		}
	} else {
		return executeRequest(param);
	}
}
```

Pada script diatas, pertama tama kita cek apakah pada bagian header terdapat access token atau apakah tanggal expired token < tanggal sekarang, Jika ya, maka request access token yang baru ke server, jika tidak maka jalankan request menggunakan fungsi `executeRequest()`

```
if ( date_token_expire < date_now || !accessToken) {
      // Script lainnya
} else {
      return executeRequest(param);
}
```

Note: Variabel access_token_expire, refresh_token_expire, dan data terkait token lainnya kita definisikan pada file header.php (app\Views\themes\modern\header.php) atau bisa di view source halaman aplikasi client.

Fungsi `executeRequest()` merupakan fungsi untuk mengirim request ke server, fungsi ini ada di file functions.js, satu file dengan fungsi sendRequest(), adapun isi dari fungsi ini adalah sebagai berikut:

```
function executeRequest(param) {
      accessToken = $('meta[name="' + cookie_access_token_name +
'"]').attr('content');
      headers = $.extend(param.headers, {
                        'Authorization' : 'Bearer ' + accessToken
                  });
      $.ajax({
            url: param.url,
            type: param.method,
            data: param.data,
            headers : headers,
            success: function(data) {
                  if (param.success) {
                        param.success(data);
                  }
                  return data;
            },
            error: function(xhr) {
                  if (param.error) {
                        param.error(xhr);
                  } else {
                        bootbox.alert('Ajax Error: Cek Console Browser');
                  }
                  return {'status' : 'ajax_error', 'xhr' : xhr}
            }
      })
}
```

Note: Pada script diatas kita menambahkan header Authorization pada request yang dikirim.

Selanjutnya, untuk melakukan request access token yang baru, pertama tama kita cek apakah ada data refresh token pada tag <meta>

```
refresh_token = $('meta[name="' + cookie_refresh_token_name +
'"]').attr('content');
```

```
<title>Dashboard | RESTFul API Codeigniter 4</title>
<meta name="descrition" content="Dashboard"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="shortcut icon" href="https://codeliro.com/demo/api/public/images/

<meta name="app_token" content="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQ
<meta name="app_token_refresh" content="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

ika ya maka kita cek apakah nilai variabel date_token_expire > tanggal sekarang, jika ya kirim request access token ke server menggunakan refresh token yang ada.

```
if ( refresh_token ) {
      if (date_token_expire > date_now) {
            executeRequest({
                  url: base_url + 'login/refreshToken',
                  method: 'POST',
                  data: 'refresh_token=' + refresh_token,
                  success: function(data) {
                        data = JSON.parse(data);
                        $('meta[name="' + data.token_name +
'"]').attr('content', data.access_token);
                        $('meta[name="' + data.token_name +
'_refresh"]').attr('content', data.refresh_token);
                        cookie_access_token_name = data.token_name;
                        cookie_refresh_token_name = data.token_name +
'_refresh';
                        access_token_expire = data.eccess_token_expire;
                        refresh_token_expire = data.refresh_token_expire;
                        return executeRequest(param);
                  }
            });
      }
}
```

Pada script diatas, untuk merequest access token yang baru kita akses url aplikasi client dengan alamat login/refreshToken, misal http://localhost/api-client/login/refreshToken, ketika kita mengakses url tersebut diatas, maka aplikasi client akan mengeksekusi method refreshToken() yang ada pada controller Login, berikut bentuk script method refreshToken():

```
public function refreshToken()
{
      $url = $this->config->apiURL . 'login/refresh-token';
      $result = $this->sendRequest($url, 'post',
                  ['grant_type' -> 'refresh_token', 'refresh_token' =>
$this->request->getPost('refresh_token')]
            );
      $this->setToken($result);
      echo json_encode($result);
}
```

Pada method refreshToken() diatas, aplikasi client akan melakukan request ke aplikasi api untuk mendapatkan data access token yang baru.

Catatan:

Mungkin Anda bertanya tanya kenapa kita tidak langsung mengakses server api dengan Javascript?

Seperti telah dibahas sebelumnya bahwa cookie yang digunakan adalah HttpOnly sehingga Javascript tidak bisa menuli maupun membaca cookie tersebut untuk itu kita melakukannya melalui PHP.

Jika proses berhasil, maka kita ganti token yang ada di meta tag dan kita perbarui variabel javascript terkait token, selanjutnya setelah kita dapatkan access token yang beru kita eksekusi request awal `return executeRequest(param);`

```
success: function(data) {
      data = JSON.parse(data);
```

```
        $('meta[name="' + data.token_name + '"]').attr('content',
    data.access_token);
        $('meta[name="' + data.token_name + '_refresh"]').attr('content',
    data.refresh_token);
        cookie_access_token_name = data.token_name;
        cookie_refresh_token_name = data.token_name + '_refresh';
        access_token_expire = data.eccess_token_expire;
        refresh_token_expire = data.refresh_token_expire;
        return executeRequest(param);
    }
```

Contoh penggunaan fungsi sendRequest() ini ada di file javascript yang ada di folder public\themes\modern\js seperti dashboard.js, gallery.js, dll.

9.2.5. Data Tables Server Side

Pada bagian sebelumnya telah dibahas bahwa untuk mengirim request menggunakan Javascript kita menggunakan fungsi sendRequest(). Pada kondisi tertentu, kita tidak dapat menggunakan fungsi tersebut salah satunya ketika menggunakan data tables server side.

Pada data tables server side, url server harus langsung kita definisikan, sehingga tidak memungkinkan untuk menggunakan fungsi sendRequest(), contoh sebagai berikut:

```
$('#data-tables').DataTable({
      processing: true,
      serverSide: true,
      ajax: {
            url: api_url + 'dashboard/penjualan/terbaru?tahun=' + tahun,
            headers: {
                  'Authorization' : 'Bearer ' + $('meta[name="' +
cookie_access_token_name + '"]').attr('content'),
            }
      }
});
```

Pada script diatas, kita sudah dapat mengakses resource pada aplikasi api, namun demikian, script diatas tidak dapat kita gunakan karena belum bisa mengakomodir refresh token.

Untuk mengatasi permasalahan diatas setidaknya terdapat dua cara yang dapat kita lakukan yaitu (1) request datatables dilakukan melalui aplikasi client, (2) request datatables langsung diarahkan ke alamat server API.

Pada cara pertama, url kita arahkan ke aplikasi client, selanjutnya aplikasi client meneruskan permintaan tersebut ke aplikasi api, response yang dikirim dari aplikasi api ke aplikasi client akan diteruskan ke datatables, ilustrasinya adalah sebagai berikut:

Dengan model diatas maka akan mempermudah penulisan script data tables karena semua keperluan pengiriman request ke aplikasi api, seperti pengaturan method, header, data, dll di handle oleh aplikasi client.

Sebagai contoh module From_Ajax (app\Controllers\Form_Ajax.php), module tersebut menggunakan file javascript form-ajax.js (public\themes\modern\js\form-ajax.js), pada file javascript tersebut url untuk data tables ajax mengarah ke file server client, adapun scriptnya adalah sebagai berikut:

```
const url = base_url + 'form-ajax/getDataDT?ajax=true';
const settings = {
      "processing": true,
      "serverSide": true,
      "scrollX": true,
      "ajax": {
            "url": url,
            // Script lainnya
      }
}
```

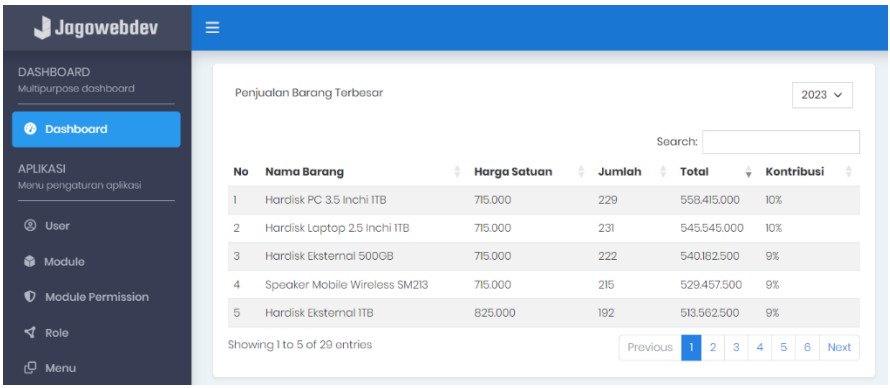
Pada script diatas, url akan mengakses controller Form_Ajax method `getDataDT()`, method tersebut nantinya akan mengambil data ke aplikasi api, adapun script pada method `getDataDT()` adalah sebagai berikut:

```
public function getDataDT() {
      echo  $this->sendRequest( $this->config->apiURL . 'form-
ajax/datatables', 'post', $_POST);
}
```

Pada script diatas, aplikasi client akan mengakses controller Form_Ajax yang ada pada aplikasi api, selanjutnya respon yang dikirim oleh aplikasi api akan digunakan oleh data tables untuk memproses data.

Untuk cara kedua, yaitu url kita arahkan langsung ke server api kita perlu menambahkan refresh token pada HTTP Body sehingga pada data tables ajax ini, setiap request yang kita kirim selalu menyertakan access token dan refresh token, access token pada HTTP Header dan refresh token pada HTTP Body.

Sebagai contoh module penjualan barang terbesar pada dashboard seperti contoh pada gambar berikut:



Pada module penjualan terbesar tersebut data diatampilkan menggunakan data tables ajax, script Javascriptnya ada di file file dashboard.js yang ada di folder (public\themes\modern\js), bentuk scriptnya adalah sebagai berikut:

```
const settings = {
      processing: true,
      serverSide: true,
      ajax: {
            url: api_url +
'dashboard/penjualan/terbesar/datatables?tahun=2023',
```

```
            beforeSend: function(xhr, settings) {
                  settings.data = settings.data + '&refresh_token=' +
$('meta[name="' + cookie_refresh_token_name + '"]').attr('content');
                  xhr.setRequestHeader('Authorization', 'Bearer ' +
$('meta[name="' + cookie_access_token_name + '"]').attr('content'));
            },
            dataSrc: function (json) {
                  setNewToken(json);
                  return json.data;
            },
            error: function (xhr, error, code) {
                  if (xhr.status == 401) {
                        window.location = base_url + 'login';
                  } else {
                        bootbox.alert('Ajax Error, cek console browser');
                        console.log(xhr);
                  }
            }
      },
      "columns": column
}
```

Pada script diatas, kita tambahkan parameter beforeSend pada ajax. Parameter beforeSend ini akan **selalu dieksekusi** tepat sebelum request ajax dikirim. Pada contoh diatas, kita tambahkan data refresh_token pada HTTP body dan Authorization pada HTTP header, penambahan data pada beforeSend ini dilakukan realtime sebelum ajax dikirim. Hal ini berbeda jika penambahan HTTP Header dilakukan melalui paramter header seperti contoh sebelumnya:

```
$('#data-tables').DataTable({
      processing: true,
      serverSide: true,
      ajax: {
            url: api_url + 'dashboard/penjualan/terbaru?tahun=' + tahun,
            headers: {
                  'Authorization' : 'Bearer ' + $('meta[name="' +
cookie_access_token_name + '"]').attr('content'),
            }
      }
});
```

Pada script diatas data token diambil hanya sekali yaitu ketika script di load (halaman direfresh) sehingga nantinya jika access token berubah via

ajax (misal ketika ada request access token yang baru via ajax), maka nilai token tersebut tidak berubah.

Selanjutnya jika respon dari server menyertakan data token, dengan fungsi setNewToken() kita update data token yang ada di cookie dan yang ada di meta tag tanpa merefresh halaman, namun jika terjadi error, unauthorized, maka kita redirect halaman ke halaman login.

```
dataSrc: function (json) {
      setNewToken(json);
      return json.data;
},
error: function (xhr, error, code) {
      if (xhr.status == 401) {
            window.location = base_url + 'login';
      } else {
            bootbox.alert('Ajax Error, cek console browser');
            console.log(xhr);
      }
}}
```

Berikut script fungsi `setNewToken()`

```
function setNewToken(json)
{
      if (json.token) {
            $.ajax({
                  url: base_url + 'login/setToken',
                  method: 'post',
                  data: 'token=' + JSON.stringify(json.token),
                  success: function(data) {
                        data = json.token;
                        $('meta[name="' + data.token_name +
'"]').attr('content', data.access_token);
                        $('meta[name="' + data.token_name +
'_refresh"]').attr('content', data.refresh_token);
                        cookie_access_token_name = data.token_name;
                        cookie_refresh_token_name = data.token_name +
'_refresh';
                        access_token_expire = data.eccess_token_expire;
                        refresh_token_expire =
data.refresh_token_expire;
                  },
                  error: function(xhr) {
```

```
                    bootbox.alert('Ajax Error: Cek Console
Browser');
                    console.log(xhr);
                }
            })
        }
}
```

Pada script diatas, lagi lagi kita menggunakan aplikasi client untuk mengupdate data cookie (`url: base_url + 'login/setToken'`) kenapa? Karena seperti telah disebutkan sebelumnya, Javascript tidak dapat membaca maupun menulis cookie dengan flag Secure dan HttpOnly sehingga kita perlu bantuan PHP untuk melakukannya.

Selanjutnya berikut ini script data tables ajax penjualan terbesar yang ada pada aplikasi api:

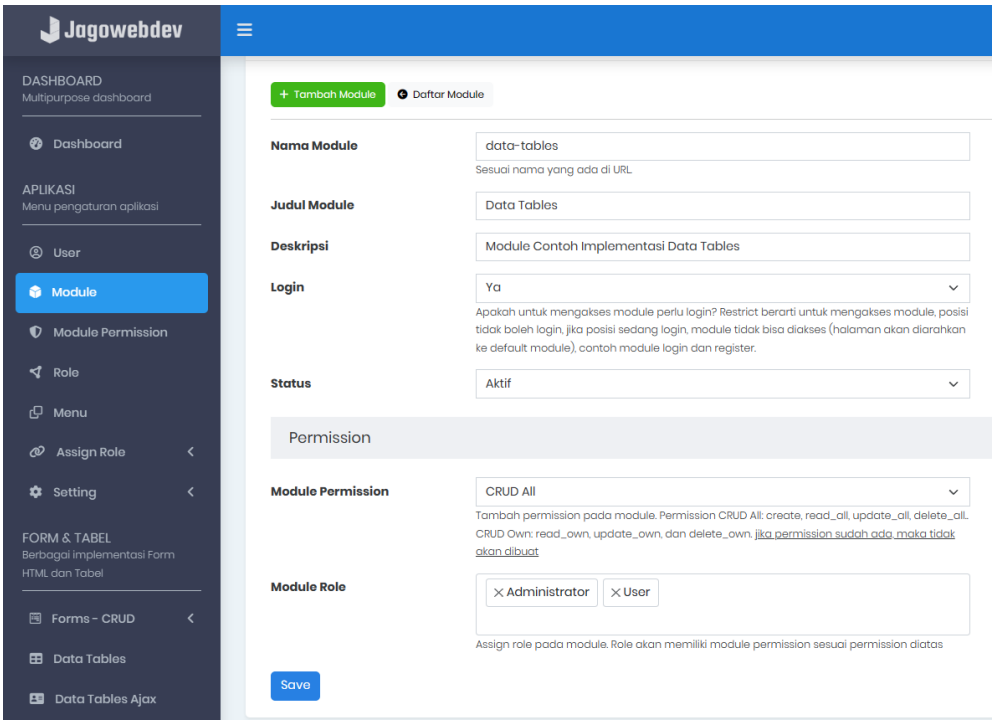
```
public function getDataDTPenjualanTerbesar() {

    // Script lainnya
    $result['token'] = $this->initAkses['token'];
    $result['data'] = $query['data'];
    echo json_encode($result); exit();
}
```

Pada script diatas, data response dari data tables kita tambahkan data token yang diambil dari property initAkses.

9.3. Module

Pada sub bab 8.9 telah dibahas bahwa pada Aplikasi REST API yang kita bangun kita menggunakan sistem module, pada bab tersebut juga disebutkan bahwa untuk membuat module terdapat beberapa hal yang harus kita lakukan yaitu mendaftarkan module pada database, mendefinisikan permission pada module, mengassign permission module pada role, dan terakhir membuat file controller.

Pada aplikasi client yang kita bangun, pendaftaran module pada databse, pendefinisian permission pada module, dan assign permission module pada role dilakukan sekaligus pada halaman module, contoh tampilannya adalah sebagai berikut:



Pada contoh gambar diatas, nama module adalah data-tabeles, sehingga nantinya module tersebut pada aplikasi client diakses menggunakan URI http://localhost/api-client/data-tables, sedangkan pada aplikasi server diakses menggunakan URI http://localhost/api/data-tables.

Selanjnya pada bagian permission kita mendefinisikan permission pada module data-tables. Pada bagian ini terdapat beberapa pilihan opsi yaitu CRUD All, CRUD Own, , Crud All + Crud Own, dan Manual, penjelasannya adalah sebagai berikut:

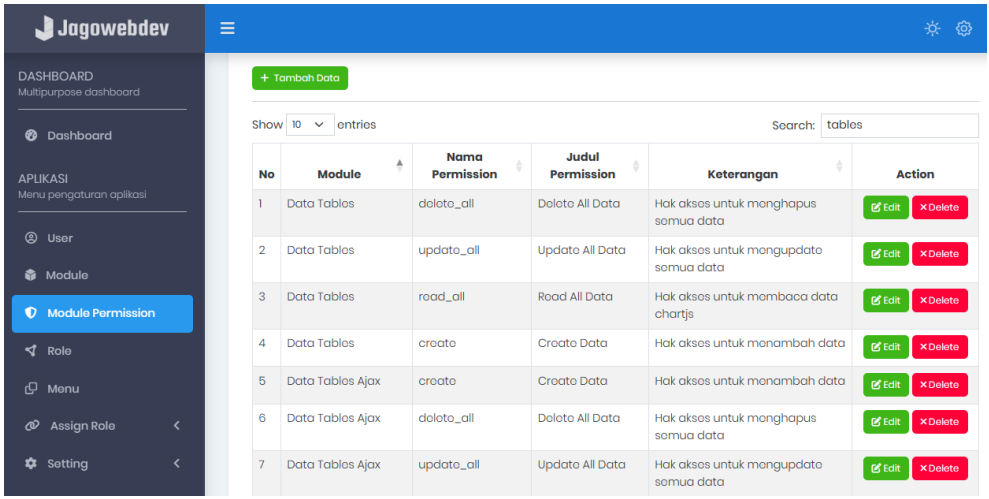
- Crud ALL akan menambah permission create, read_all, update, all, dan delete_all pada module.

- Crud Own akan menambah permission create, read_own, update_own, dan delete_own pada module.
- Manual berarti kita mendefinisikan permission secara manual misal kirim_email, download_pdf, dll

pada contoh diatas kita pilih opsi Crud All sehingga nantinya module data-tables akan memiliki permission create, read_all, update_all, dan delete_all.

Pilihan berikutnya adalah mengassign permission tersebut ke role, pada contoh diatas, kita assign permission ke role administrator sehingga nantinya role administrator akan memiliki permission create, read_all, update_all, dan delete_all pada module data-tables.

Setelah disimpan, maka permission yang telah kita buat tadi akan muncul di menu Module Permission, contoh seperti pada gambar berikut:



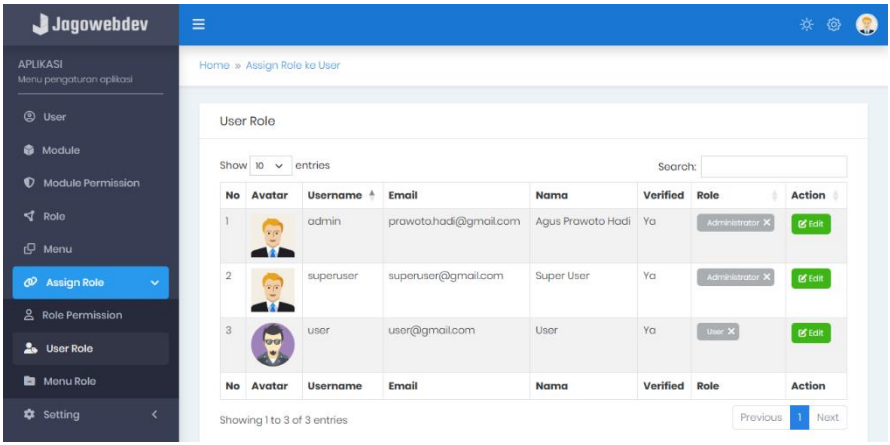
Selanjutnya hasil dari assign permission pada role (pada bagian isian Module Permission halaman module) akan muncul pada menu Assign Role > Role Permission.

Setelah mendefinisikan module pada sistem, selanjutnya kita harus membuat file controller sesuai dengan nama module. Sebagaimana dijelaskan sebelumnya, nama controller harus sesuai dengan nama

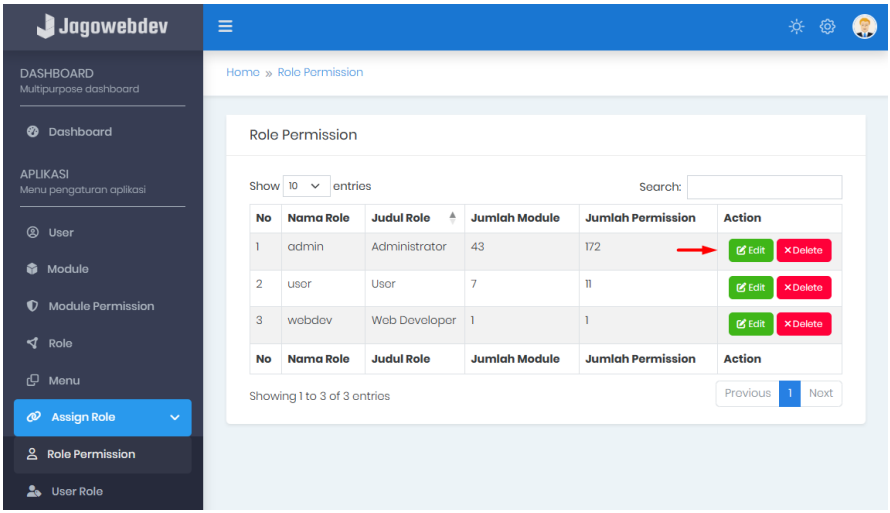
module, jika kita membuat module data-tables seperti pada contoh sebelumnya maka file controller kita buat adalah Data_tables.php. File controller tersebut kita simpan pada folder App\Controllers.

9.4. Role dan Permission

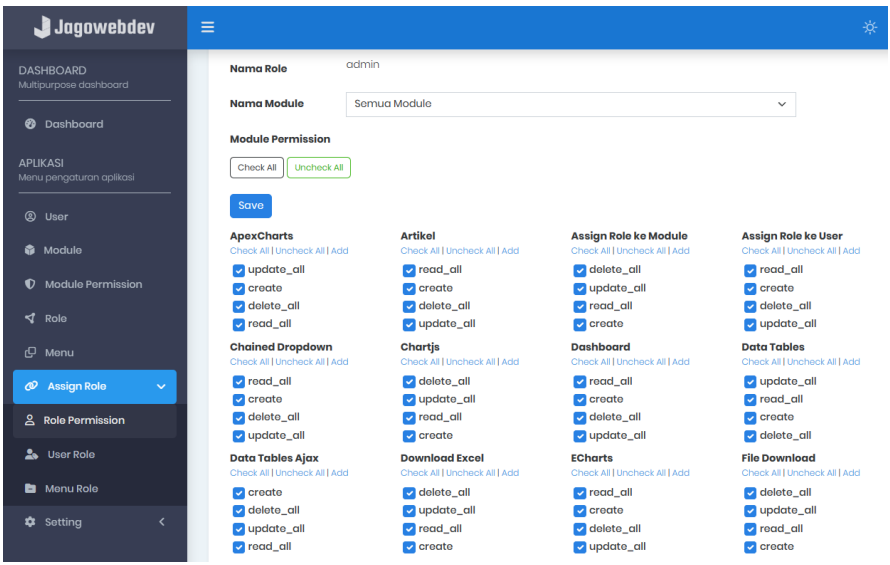
Sebagaimana telah kita bahas pada BAB 8 bahwa pada aplikasi api yang kita kembangkan, pengelolaan hak akses terhadap suatu module menggunakan sistem role – permission dimana untuk dapat mengakses suatu module maka user harus memiliki role dan role tersebut harus memiliki permission pada module, pada aplikasi client, pengaturan assign role pada user dapat dilakukan pada menu Assign Role > User Role sebagai berikut:



Selanjutnya pengaturan assign permission pada role dapat dilakukan pada menu Assign Role > Role Permission sebagai berikut:



Pada contoh diatas, jika kita ingin meng *assign* permission pada role Administrator klik tombol edit pada role Administrator, selanjutnya pada halaman Module Permission, centang permission yang ingin diassign



9.4.1. Loading Permission

Pada aplikasi api, informasi role dan permission yang dimiliki user dapat diakses melalui module util method init_akses() dengan URI http://localhost/api/util/init-akses. Pada aplikasi client, informasi ini kita dapatkan melalui method initAkses() yang ada pada baseController.php adapun script method initAkes() adalah sebagai berikut:

```
public function initAkses() {

      $url = $this->config->apiURL . 'builtin/util/init-
akses?param_nama_module=' . $this->currentModule['nama_module'];
      $this->initAkses = $this->sendRequest($url);

      if (is_string($this->initAkses)) {
            $this->exitError(['status' => 'error', 'content' => $this-
>initAkses]);
            exit;
      }

      if (@$this->initAkses['status'] == 'error') {
            if (@$this->initAkses['error_type'] == 'failed_to_connect') {
                  $this->exitError(['status' => 'error', 'content' => $this-
>initAkses['message']]);
                  exit;
            }
      }

      if (@$this->initAkses['code'] == 404) {
            $this->exitError(['status' => 'error', 'content' => $this-
>initAkses['message']]);
            exit;
      }

      if (@$this->initAkses['error_type'] == 'need_login') {
            header('location: '. base_url() . '/login');
      }

      $this->data['setting_registrasi'] = $this-
>initAkses['setting_registrasi'];
}
```

pada method initAkses() diatas kita mengirim request ke aplikasi api dengan URI http://localhost/api/util/init-akses hasil dati request tersebut kita simpan pada roperty initAkses (`$this->initAkses = $this-`

>sendRequest($url)) data property ini nantinya digunakan untuk berbagai keperluan seperti melakukan pengecekan user, status module, permission dll, kita dapat melihat hasil dari property initAkses() ini dengan mencetak datanya misal dengan menambahkan script:

```
echo '<pre>'; print_r($this->initAkses); die;
```

sehingga script pada method initAkses() menjadi:

```
public function initAkses() {

      $url = $this->config->apiURL . 'builtin/util/init-
akses?param_nama_module=' . $this->currentModule['nama_module'];
      $this->initAkses = $this->sendRequest($url);

      echo '<pre>'; print_r($this->initAkses); die;

      // Script lainnya

}
```

Sebagai contoh, jika kita akses module data-tables, misal: http://localhost/api-client/data-tables, maka proprty initAkses akan berisi data array yang memuat barbagai informasi diantaranya data permission yang dimiliki user pada module data-tables sebagai berikut:

```
[permission] => Array
        (
            [create] => create
            [delete_all] => delete_all
            [read_all] => read_all
            [update_all] => update_all
        )
```

pada data diatas terlihat bahwa user memiliki permission create, delete_all, read_all, dan update_all pada module data-table. Selanjutnya permission ini disimpan pada key user_permission pada properti data:

```
$this->data['user_permission'] = $this->initAkses['permission'];
```

Sehingga nantinya permission ini dapat diakses di bagian view (app\Views) melalui variabel $user_permission. Selain disimpan pada properti data, permission ini juga disimpan pada properties userPermission:

```
$this->userPermission = $this->initAkses['permission'];
```

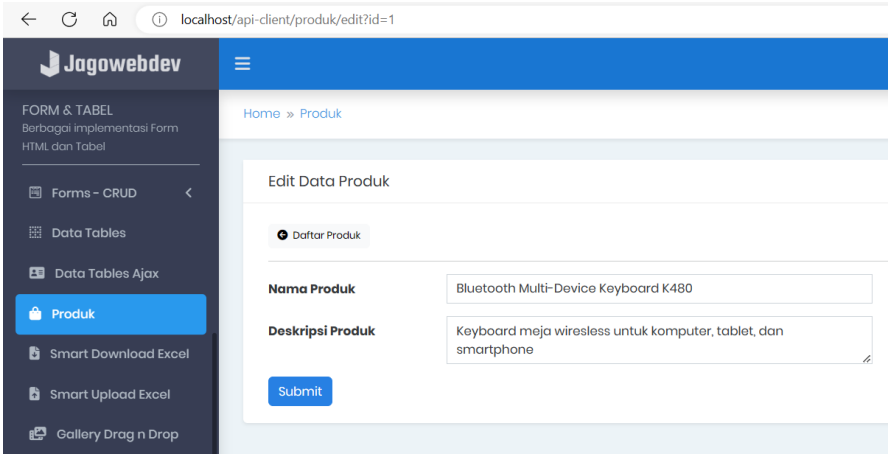
sehingga data ini bisa diakses di seluruh bagian controller baik BaseController maupun controller pada module.

Selain permission pada module yang sedang diakses, properti initAkses juga memuat informasi semua permission yang dimiliki oleh user pada semua module, informasi ini berupa data array yang dikelompokkan berdasarkan module, informasi ini ada di key [user][all_permission]

9.4.2. Menerapkan Permission

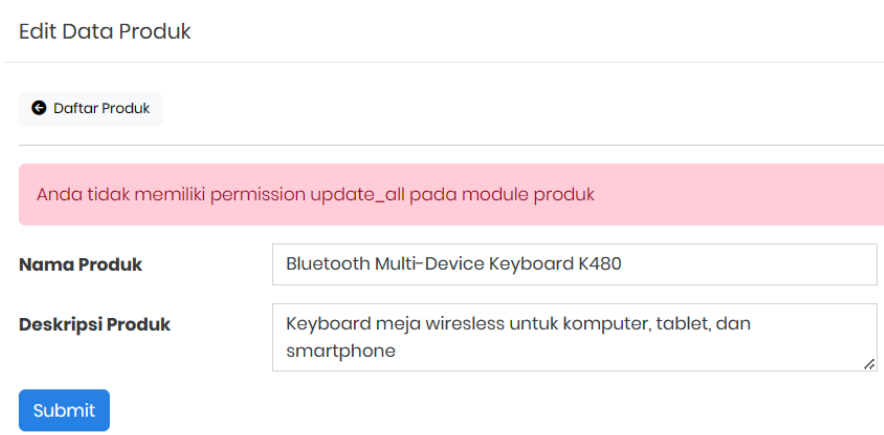
Pada BAB 8 telah disebutkan bahwa permission yang telah kita definisikan tidak bisa serts merta berfungsi, untuk menerapkan permission tersebut kita harus mendefinisikan fungsi permission melalui coding.

Sebagai contoh pada aplikasi api yang kita kembangkan kita membuat permission Crud All (create, read_all, update_all, delete_all) pada module produk, selanjutnya semua permission tersebut kita assign pada role Administrator kecuali permission update_all, sehingga role Administrator hanya memiliki permission create, real_all, dan delete_all pada module produk, selanjutnya pada aplikasi client, jika kita login sebagai user dengan role Administrator dan membuka halaman edit pada module produk, maka halaman edit tersebut akan ditampilkan beserta data yang akan diedit, misal sebagai berikut:



Kenapa user tersebut dapat membuka halaman edit padahal tidak memiliki role update?

Ketika membuka halaman edit, user masih dalam posisi membaca (read) data yang akan diedit dan belum melakukan perubahan data pada database, sehingga user masih bisa membuka halaman edit, namun ketika user mengklik submit, maka akan muncul pesan error yang menyebutkan bahwa user tidak memiliki permission update_all pada module produk seperti contoh pada gambar berikut:

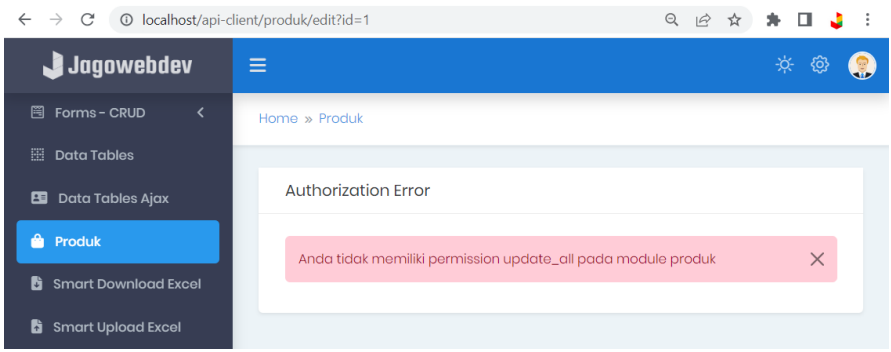


Error tersebut terjadi karena ketika dilakukan pengecekan pada aplikasi api, user tidak memiliki permission permission update_all.

Hal ini tentu tidak memberikan pengalaman yang baik bagi user/pengguna karena user mengira bahwa dia dapat melakukan update data karena bisa membuka halaman edit. Untuk mengatasi permasalahan tersebut, maka pada module produk yang ada di aplikasi client kita juga harus menerapkan sistem permission, caranya buka file Produk.php yang ada pada folder (app\Controllers), kemudian pada method edit() tambahkan script if (!$this->hasPermission('update_all')) return; sehingga script pada method edit() menjadi:

```
public function edit()
{
      if (!$this->hasPermission('update_all')) return;

      if (empty($_GET['id'])) {
            $this->dataNotfound(); return;
      }

      // Code lainnya
}
```

Dengan demikian ketika user membuka halaman edit, maka akan muncul pesan error sebagai berikut:



Selanjutnya metode ini dapat Anda terapkan pada halaman depan module produk dimana jika user tidak memiliki permission update maka jangan tampilkan tombol edit.

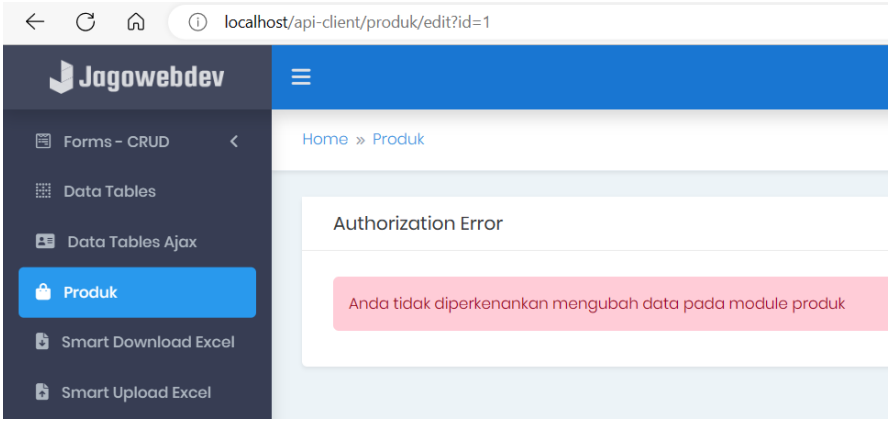
Method UserCan

Pada aplikasi client, untuk melakukan pengecekan permission yang dimiliki user, selain menggunakan method hasPermission(), kita juga dapat menggunakan method userCan(), fungsi method ini sama sperti method userCan() yang ada pada aplikasi api yaitu untuk mengecek permission yang mengandung pemisah kata underscore (_), seperti update_all, read_own, dll.

Method userCan() digunakan untuk mengetahui level permission user, misal jika user memiliki permission update_all maka ketika method userCan('update') dijalankan akan menghasilkan string all yang artinya user dapat mengupdate semua data, contoh penggunaan method ini misal pada method edit() pada contoh sebelumnya kita ubah `if (!$this->hasPermission('update_all')) return;` menjadi `if (!$this->userCan('update')) return;` sebagai berikut:

```
public function edit()
{
      if (!$this->userCan('update')) return;

      if (empty($_GET['id'])) {
            $this->dataNotfound(); return;
      }

      // Code lainnya
}
```

Selanjutnya jika user tidak memiliki permission dengan awalan update seperti update_all atau update_own maka akan muncul pesan error sebagai berikut:



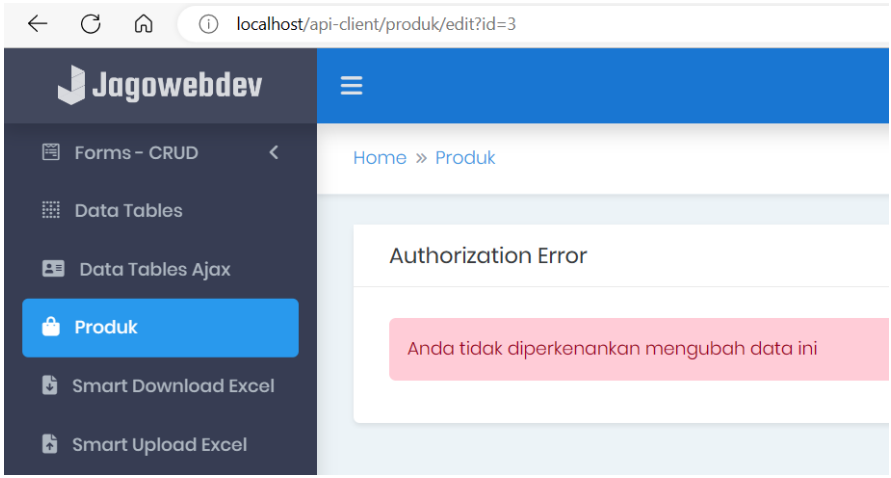
Method userCan() selain digunakan untuk melakukan pengecekan level permission, juga dapat digunakan untuk membatasi akses tertentu pada data misal membatasi hanya user yang membuat data dan user dengan permission update_all yang dapat mengupdate data, sebagai contoh melanjutkan contoh sebelumnya pada method `edit()` module produk kita batasi hanya user dengan permission update_all dan user yang membuat data tersebut yang bisa melakukan update data, kita ubah script `if (!$this->userCan('update')) return;` menjadi sebagai berikut:

```
public function edit()
{
      $akses = $this->userCan('update', ['scope' => 'item', 'item'
=> ['field' => 'id_user_input', 'source' => 'produk:id_produk:' .
$_GET['id']]]);
      if (!$akses) return;

      if (empty($_GET['id'])) {
            $this->dataNotfound(); return;
      }

      // Script lainnya
}
```

Pada script diatas, method userCan() akan melakukan pengecekan jika user memiliki permission update_own maka method userCan() akan mengirim request ke server untuk mendapatkan resource produk dengan id_produk sama dengan nilai pada variabel `$_GET['id'].` selanjutnya

akan dicocokkan apakah id_user yang login sama dengan nilai id_user_input yang ada pada resource produk, jika tidak maka akan muncul pesan error.

Misal jika kita ingin mengupdate data produk dengan id_produk 3 dan permission yang kita miliki pada module produk adalah update_own maka akan muncul pesan error sebagai berikut:



Error ini disebabkan karena id user yang kita gunakan untuk login adalah 1 sedangkan nilai id_user_input produk dengan id_produk 3 adalah 2 sehingga kita bukan pemilik dari data tersebut.

Jika script method userCan() diatas terlalu panjang, Anda dapat menggantinya dengan script sebagai berikut:

```
public function edit()
{
      $produk = $this->sendRequest($this->config->apiURL .
'produk/' . $_GET['id']);
      if (!$this->userCan('update', ['user' =>
$produk['data']['id_user_input']]) ) return;

      if (empty($_GET['id'])) {
            $this->dataNotfound(); return;
      }
```

```
      // Script lainnya
}
```

Contoh lain penggunaan method userCan() ini dapat dilihat pada module User pada file App\Controllers\Builtin\User.php sebagai berikut:

```
if ( !$this->userCan('update', ['user' => $this->request-
>getGet('id')]) )
          return;
```

Pada contoh diatas, method userCan() akan mencocokkan apakah user memiliki permmission update_own dan apakah data id user yang akan diedit (`$this->request->getGet('id')`) cocok dengan id user yang login (`$this->user['id_user']`) script diatas juga dapat dibuat seperti berikut (App\Controllers\Builtin\User.php):

```
if ( $this->userCan('update') ) {
      if ($this->user['id_user'] != $this->request->getGet('id')) {
            $this->showMessage(['status' => 'error', 'title' =>
'Authorization Error', 'message' => 'Anda tidak diperkenankan mengakses
data ini']);
            return;
      }
} else {
      return;
}
```

Method `userCan()` ini juga dapat digunakan seperti berikut ini:

```
if (!$this->userCan('read', 'all')) return;
if (!$this->userCan('update', 'all')) return;
if (!$this->userCan('delete', 'all')) return;
```

Property userPermission

Cara lain untuk melakukan pengecekan permission yang dimiliki user adalah melakukan pengecekan permission pada property userPermission, poperty userPermission ini sendiri berbentuk array yang memuat permission yang dimiliki user pada module yang sedang diakses, adapun bentuknya adalah sebagai berikut:

```
[create] => create
[delete_all] => delete_all
[read_all] => read_all
[update_all] => update_all
```

Contoh pengecekan menggunakan metode ini adalah sebagai berikut:

```
if (!key_exists('create', $this->userPermission)) {
      echo 'Anda tidak memiliki hak untuk menambah data';
      exit;
}
```

9.5. Handling Error Response

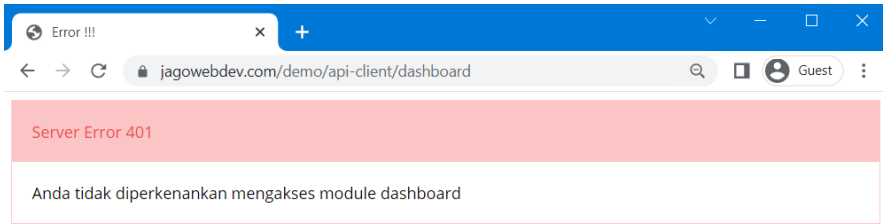
Ketika melakukan request ke aplikasi api. terkadang aplikasi api mengirim response dengan response code yang menunjukkan bahwa telah terjadi error dalam memproses request, misal ketika ada form yang tidak diisi dengan lengkap maka aplikasi api mengirim response code 400 atau jika user tidak diperbolehkan mengakses data tertentu maka akan mengirim response code 401, pada aplikasi client respon error ini kita tampilkan kepada user sedemikian rupa sehingga dapat menunjukkan error yang dimaksud.

Pada aplikasi client yang kita kembangkan, penanganan response error dapat dilakukan setidaknya dengan dua cara yaitu (1) dengan memunculkan pesan error dan keluar dari aplikasi atau (2) memunculkan pesan error tanpa keluar aplikasi. Jika Anda menginginkan opsi yang pertama, maka Anda dapat menambahkan response code pada method sendRequest() (file BaseControllers.php) bagian variabel $response_exit sebagai berikut:

```
protected function sendRequest($url, $method = 'get', $post_fields = [],
$headers = []) {

      // Script lainnya
      if ($response_code) {
            if ($this->responseContentType == 'application/json') {
                  $result = json_decode($result, true);
            }
```

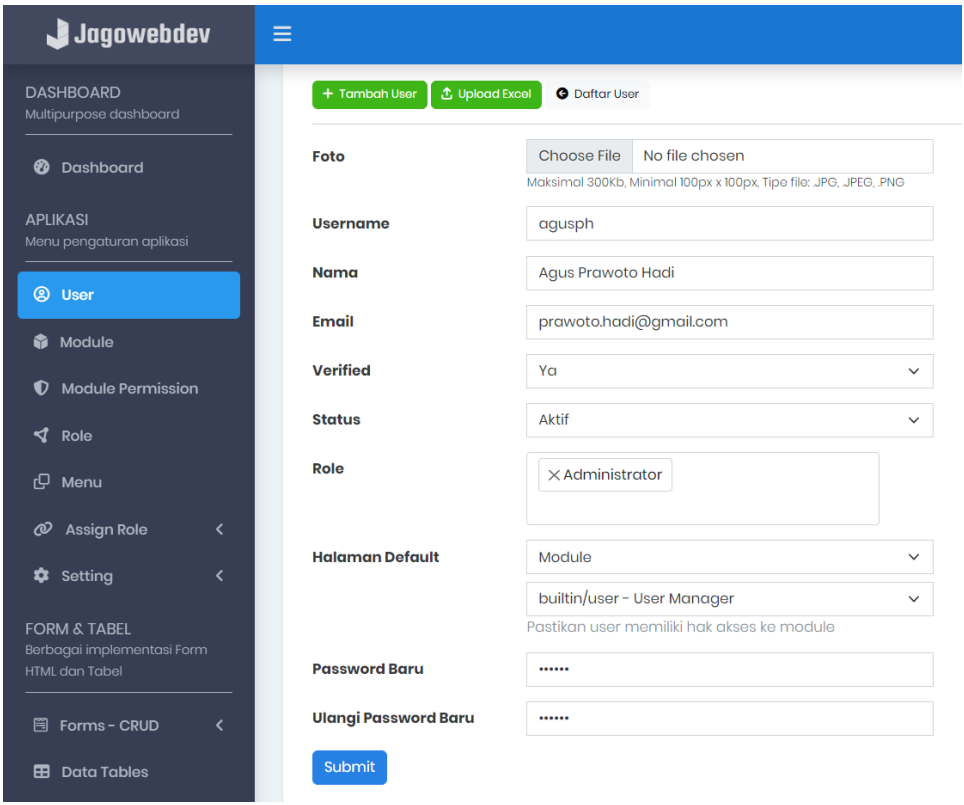
```
            $response_exit = [400, 401, 500];
            if (in_array($response_code, $response_exit))
            {
                  $this->data['status'] = 'error';
                  $this->data['title'] = 'Server Error ' . $response_code;
                  $this->data['content'] = '';
                  if (is_array($result)) {
                        if (@$result['status'] == 'error') {
                              $this->data['content'] = $result['message'];
                        }
                  }
                  if (!$this->data['content']) {
                        $this->data['content'] = $result;
                  }
                  $this->exitError($this->data);
            }
      }
}
```

Pada contoh diatas, maka jika server mengirim response dengan code 400, 401, dan 500 maka aplikasi client akan memunculkan pesan error dan keluar dari aplikasi. Sebagai contoh login dengan akun user, kemudian akses halaman dashboard, maka akan muncul error sebagai berikut:



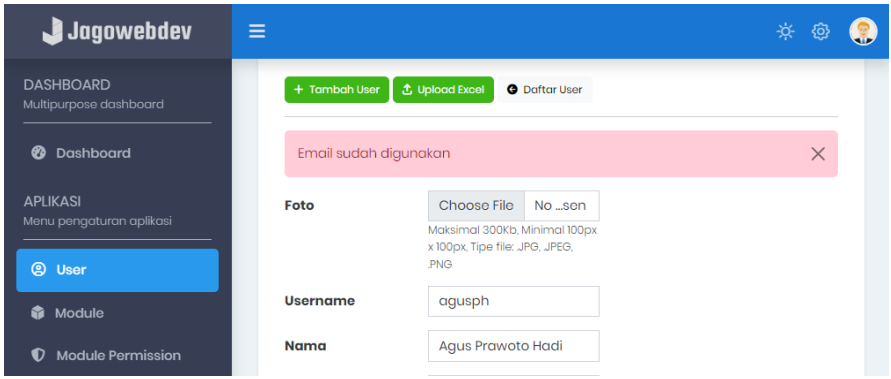
Hal ini terjadi karena user tidak memiliki permission read pada module dashboard.

Dengan model ini kita tidak perlu menuliskan script pengecekan permission pada setiap module, namun demikian jika ada kode error dimana halaman perlu ditampilkan akan menjadi repot, misal pada halaman user, kita tambahkan user baru dengan alamat email yang sudah terdaftar disistem misal: prawoto.hadi@gmail.com sebagai berikut:



Ketika kita sumbit maka kita akan memperoleh pesan error sebagai berikut:

Dengan model seperti ini, user akan kesulitan memperbaiki isian, untuk itu kode 400 perlu kita keluarkan dari variabel $response_exit sehingga jika kita submit ulang form, hasil yang kita peroleh adalah sebagai berikut:



Pada contoh diatas, pengecekan error dilakukan di controller User (App\Controllers\Builtin\User.php) sebagai berikut:

```
public function edit()
{
      // ...
      if ($result['status'] == 'error') {
            $this->showMessage($result);
            return;
      } else {
            $this->data = array_merge($this->data, ['user_edit' =>
$result['user']]);
      }
      // ...
}
```

Bab 10 Debugging

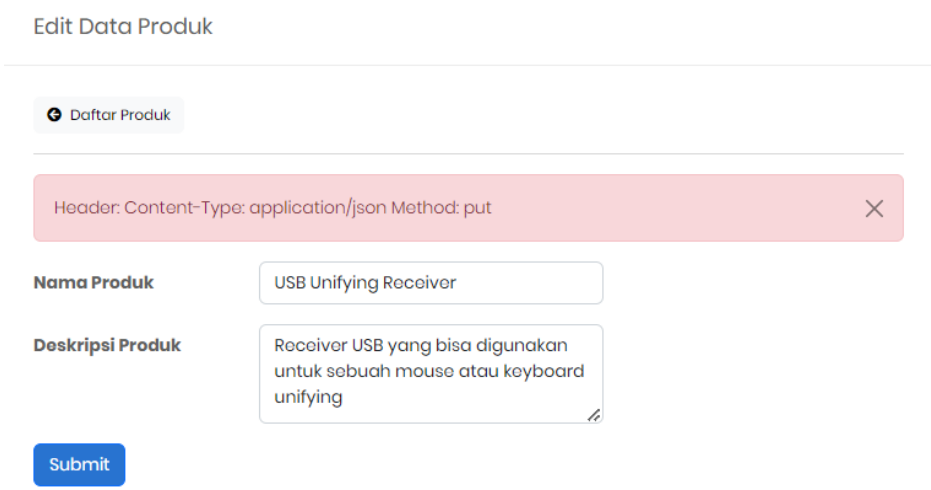
Ketika mengembangkan aplikasi tentu kita tidak akan terlepas dari yang namanya error, demikian juga pada aplikasi api yang kita kembangkan, pada aplikasi berbasis php, pada umumnya error ini dapat langsung kelihatan karena hasil eksekusi script langsung ditampilkan pada browser, namun demikian pada aplikasi api yang kita kembangkan error tersebut tidak langsung kelihatan karena output dari eksekusi script tidak langsung ditampilkan pada browser, melainkan dikirim ke kembali ke aplikasi client, agar kita dapat menampilkan pesan error (de-bug-ing) pada aplikasi api yang kita kembangkan, setidaknya ada tiga kita dapat kita lakukan yaitu menggunakan method `exitError()` pada aplikasi client, menampilkan error melalui aplikasi client, dan menggunakan apliaksi postman.

10.1 Method exitError()

Method ini digunakan aplikasi api untuk menghentikan eksekusi script dan mengirim pesan error kepada client. Pada aplikasi client method ini ada di file app\Controllers\BaseController.php. Sebagai contoh ketika mengedit data produk kita ingin melihat nilai Header Content-type dan nilai HTTP Method yang dikirim ke server, untuk keperluan tersebut kita dapat menambahkan method `exitError()` pada method `getRequestInput()` yang ada di base controller sebagai berikut:

```
public function getRequestInput(IncomingRequest $request)
{
      $content_type_header = $request->getHeader('Content-Type');
      $input = '';
      $this->exitError('Header: ' . $content_type_header . '
Method: ' . $this->request->getMethod());
      // Code lainnya
}
```

Selanjutnya pada aplikasi client, buka halaman produk klik tombol edit pada salah satu produk kemudian klik submit, hasil yang kita peroleh adalah sebagai berikut:

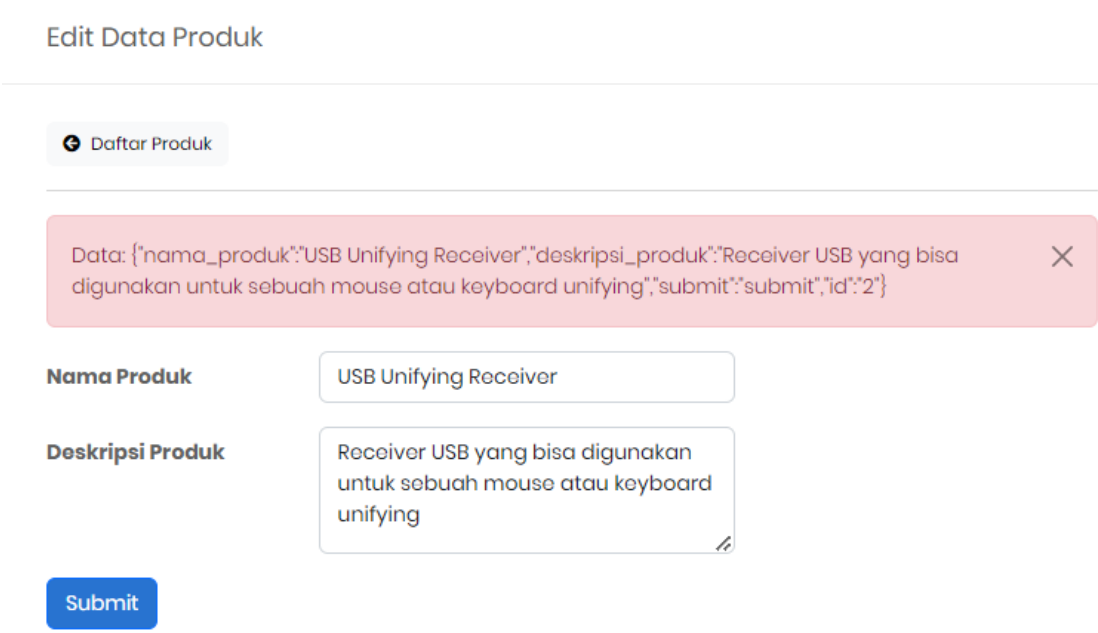


Pada gambar diatas terlihat bahwa Header Content-type yang dikirim ke server adalah application/json dan HTTP Method yang digunakan adalah put.

Pada method `exitError()` argumen yang digunakan adalah string, jika selain string misal array, maka array tersebut perlu diubah menjadi string, misal menjadi JSON. Sebagai contoh kita akan update data produk dan melihat data apa saja yang dikirim ke aplikasi api, caranya tambahkan `json_encode($request->getJSON(true))` pada argumen method `exitError()` sebagai berikut:

```
$this->exitError('Data: ' . json_encode($request->getJSON(true)));
```

Ketika kita update data produk dengan mengklik tombol submit maka hasil yang kita peroleh adalah sebagai berikut:



Anda juga dapat melakukan hal yang sama pada method lain, misal method post (tambah data), caranya ubah argumen pada method `exitError()` menjadi seperti berikut:

```
$this->exitError('Data: ' . json_encode($_POST));
```

10.2. Testing di domain yang sama

Cara kedua untuk melakukan debugging adalah dengan menginstall aplikasi api dan aplikasi client pada domain yang sama misal sama sama localhost (dengan port yang sama) seperti yang kita lakukan, yaitu aplikasi api kita install di http://localhost/api dan aplikasi client di http://localhost/api-client

Dengan cara seperti ini, maka setelah login menggunakan aplikasi client, kita dapat langsung mengakses aplikasi api tanpa melalui aplikasi client, hal ini dapat kita lakukan karena ketika mengakses aplikasi api cookie app_token yang telah terbentuk (setelah login) akan ikut terkirim ke aplikasi api sehingga oleh aplikasi api akan terbaca bahwa kita sudah login.

Sebagai contoh pertama tama kita login pada aplikasi client melalui url http://localhost/api-client/, selanjutnya buka halaman http://localhost/api/produk/1 maka hasil yang diperoleh adalah sebagai berikut:

Anda juga dapat mencoba coba output lain, misal pada aplikasi api, pada method show() yang ada pada controller Produk (app\Controllers\Produk.php) kita tambahkan script berikut:

```
echo '<pre>'; print_r($produk); die;
```

Sehingga method show() menjadi seperti berikut:

```
public function show($id = null)
{
      $produk = $this->model->getProdukById($id);
      echo '<pre>'; print_r($produk); die;
      if (!$produk) {
            $response = ['status' => 'error', 'message' => 'Data
tidak ditemukan', 'error_type' => 'not_found'];
            return $this->fail($response);
      }

      $response = ['status' => 'ok', 'data' => $produk];
      return $this->respond($response);
}
```

Selanjutnya jika kita buka halaman http://localhost/api/produk/1 maka hasil yang kita peroleh adalah sebagai berikut:

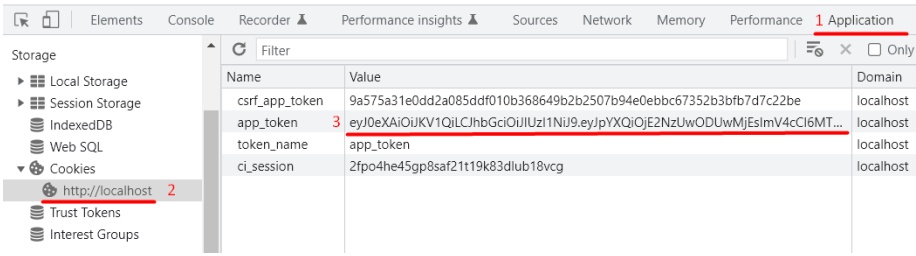
10.3. Menggunakan Postman

Cara terakhir yang dapat kita gunakan untuk melakukan debugging adalah menggunakan aplikasi postman, cara ini digunakan jika domain aplikasi api dan domain aplikasi client berbeda atau domain sama tetapi port nya berbeda seperti localhost dan localhost:8080.

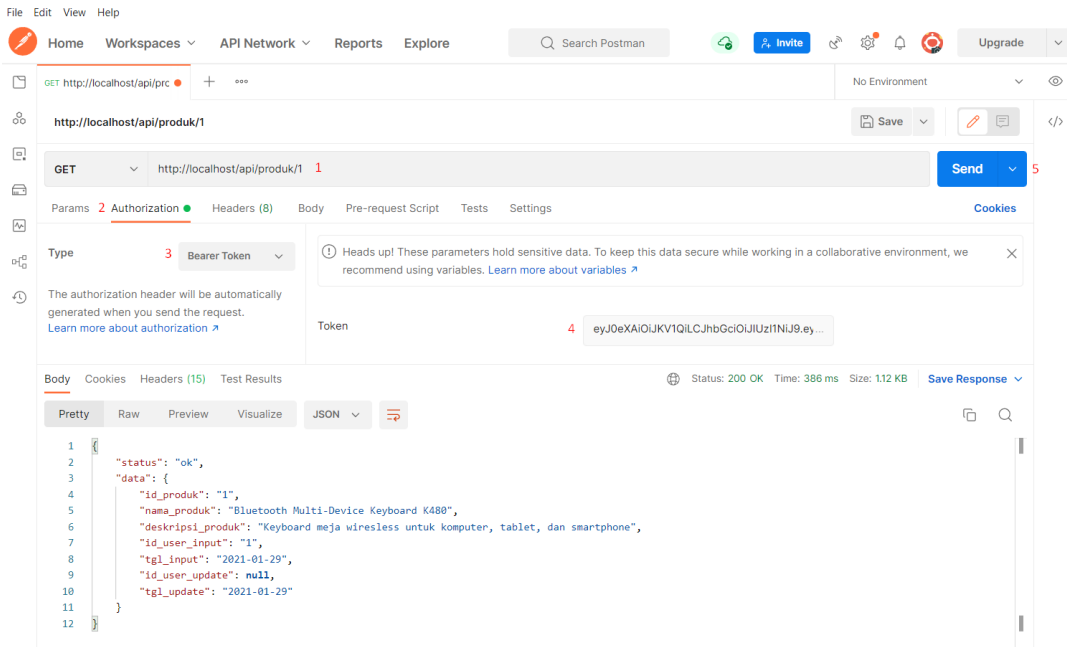
Agar aplikasi postman dapat mengakses aplikasi api maka kita perlu mendapatkan token. Token ini nantinya disertakan pada setiap request yang kita kirim ke aplikasi api sehingga aplikasi api kita mengenali bahwa kita telah login.

Token ini kita dapatkan setelah kita berhasil logi pada aplikasi api. Untuk mendapatkan token ini, pertama tama kita login ke aplikasi api menggunakan aplikasi api client, misal http://localhost/api-client/ setelah berhasil login, buka developer tools pada browser dan pilih tab Application, selanjutnya pada panel sebelah kiri klik Cookies kemudian klik domain dimana aplikasi client diinstall yang pada contoh kali ini adalah localhost

Selanjutnya cari cookie dengan nama app_token, value dari cookie tersebut merupakan token yang kita gunakan untuk mengakses aplikasi api, misal sebagai berikut:



Setelah mendapatkan token tersebut selanjutnya buka aplikasi postman. Pada bagian url, isikan alamat http://localhost/api/produk/1 selanjutnya klik tab Authorization kemudian pada bagian Type pilih Bearer Token dan pada isian Token isikan token yang telah kita dapatkan tadi. Terakhir klik Send. Jika berhasil maka kita akan mendapatkan produk sengan id 1 sebagai berikut:



DAFTAR PUSTAKA

Amundsen, Mike. 2011. *Building Hypermedia APIs with HTML5 and Node*. USA: O'Reilly Media, Inc.

Amundsen, Mike. *Hypermedia in 15 Minutes*. O'Reilly Media, Inc. 2017.

De, Brajesh. 2017. *API Management An Architect's Guide to Developing and Managing APIs for Your Organization*. USA: Apress.

Doglio, Fernando. 2018. *REST API Development with Node.js Manage and Understand the Full Capabilities of Successful REST Development*. USA: Apress.

Lauret, Arnaud. 2019. *The Design of Web APIs*. USA: Manning Publication.

Masse, Mark. 2011. *REST API Design Rulebook*. USA: O'Reilly Media, Inc.

Patni, Sanjay. 2017. *Pro RESTful APIs Design, Build and Integrate with REST, JSON, XML and JAX-RS*. USA: Apress.

Preibisch, Sascha. 2018. *API Development: A Practical Guide for Business Implementation Success*. USA: Apress.

Richardson, Leonard, Mike Amundsen, dan Sam Ruby. 2013. *RESTful Web APIs*. USA: O'Reilly Media, Inc.

Siriwardena, Prabath. 2020. *Advanced API Security OAuth 2.0 and Beyond*. USA: Apress.

Subramanian J, Harihara dan Pethuru Raj. 2019. *Hands-On RESTful API Design Patterns and Best Practices*. UK: Packt Publishing.

9 HTTP methods and how to use them. testfully.io. 01 Desember 2021. https://testfully.io/blog/http-methods/, [diakses Juni 2023].

Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST).ics.uci.edu.https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2_1_1, [diakses Juni2023].

Intro to JWT (JSON Web Token). howtodoinjava.com. https://howtodoinjava.com/java/java-security/jwt-json-web-token/,[diakses Juni 2023].

RESTful API Design: 13 Best Practices to Make Your Users Happy. florimond.dev .26 Agustus 2018.

https://florimond.dev/en/posts/2018/08/restful-api-design-13-best-practices-to-make-your-users-happy/, [diakses Juni 2023].

REST API Design Best Practices for Sub and Nested Resources.moesif.com.12 Maret 2022.https://www.moesif.com/blog/technical/api-design/REST-API-Design-Best-Practices-for-Sub-and-Nested-Resources/, [diakses Juni2023].

REST - Stop calling your HTTP APIs as RESTful APIs.linkedin.com.31 Mei 2019. https://www.linkedin.com/pulse/rest-stop-calling-your-http-apis-restful-arpit-jain/, [diakses Juni 2023].

Roy Fielding on Versioning, Hypermedia, and REST.infoq.com.17 Desember 2014. https://www.infoq.com/articles/roy-fielding-on-versioning/,[diakses Juni 2023].

That's not Hypermedia!.amundsen.com.19 Januari 2014. http://amundsen.com/blog/archives/1149,[diakses Juni 2023].