

Ahmad Rosid [Follow](#)

<https://ahmadrosid.com> The more I learn, the more I realize how much I don't know.  
Sep 19, 2016 · 9 min read

## Membangun Web API dengan Lumen 5.3 part 1



# Lumen.

The stunningly fast micro-framework by Laravel.



```
1 <?php
2 /**
3  * Reimagine what you expect...
4  */
5 $app->get('/', function() {
6     return ['version' => '5.3'];
7 });
8 /**
9  * From your micro-framework...
10 */
11 $app->post('framework/{id}', function($framework) {
12     $this->dispatch(new Energy($framework));
13 });
14
15
16
17
18
```

<https://lumen.laravel.com/>

## Apa itu Lumen?

Untuk tau apa itu lumen yang pertama kita lakukan adalah berselancar di official web dari lumen ini. Silahkan Buka <http://lumen.laravel.com>

*The stunningly fast micro-framework by Laravel.*

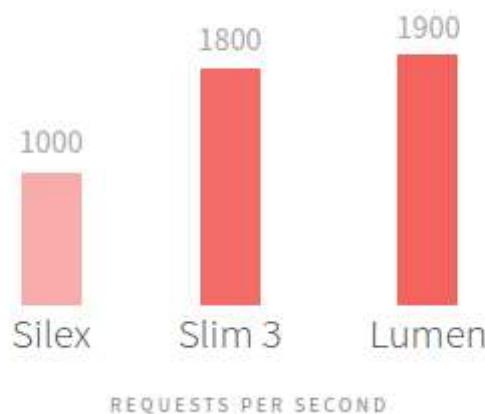
Diatas adalah penjelasan singkat dari lumen, Emm... menarik bukan? Ya, Lumen ini adalah sebuah micro-framework dari laravel yang lebih di khusukan untuk membuat web API, dimana komponen didalam nya sama seperti laravel namun bayak komponen dari laravel yang di hilangkan.

### **Kenapa sih laravel ini membuat micro-framework Lumen ini?**

Menurut saya begini, jadi laravel itu buanyak sekali fitur—fitur nya dan dari banyaknya fitur yang ada itu justru membuat laravel itu menjadi lambat, nah lumen ini di buat dengan mengutamakan speed dan simplify dari fitur—fitur yang ada di laravel. Nah seberapa cepat sih lumen ini kita akan bahas lebih dalam lagi.

## **Kenapa harus membuat web API dengan lumen?**

Pertama kali saya melihat lumen itulah pertanyaan pertama yang timbul di benak saya. Yah karena saya terbiasa pake full framework laravel di tambah lagi fitur route baru laravel yang support untuk api. Tapi saya penasaran denga lumen ini, untuk memenuhi hasrat penasaran saya akhir nya saya coba buka official web lumen ini. Setelah beberapa kali scroll saya menemukan satu alasan yang cukup menarik.



Yah disamping adalah grafik Benchmark dari lumen dengan beberapa api framework PHP yang lain. Cukup menarik bukan, lumen bisa

menangani 1900 request perdetik. Kalau mau coba benchmarking sendiri bisa baca caranya [disini](#).

## Persiapan membuat web API dengan lumen

Ok untuk menggunakan lumen kita perlu menginstall beberapa hal package php berikut ini :

- PHP >= 5.6.4
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension

Sama seperti laravel untuk requirement nya jika temen—temen sudah pernah install laravel pasti sudah familiar dengan ini.

Kali ini kita akan coba membuat web api ini dengan fitur auth sederhana dimana nanti user akan login dan mendapatkan api\_token dimana nanti akan digunakan untuk mendapatkan akses ke api yang kita buat. Kenapa disini saya bahas fitur auth? Yah menurut saya ini cukup penting dalam sebuah web API karna untuk di web API sendiri itu adalah jembatan untuk masuk kedalam sistem dari aplikasi yang kita buat, untuk itu kita perlu melakukan proteksi terhadap akses API yang kita buat.

## Membuat project web API Laravel Lumen 5.3

Ok sekarang kita akan membuat project disini kita pakai cara paling mudah menggunakan composer. Silahkan install lumen dengan memjalankan perintah berikut ini :

```
composer create-project --prefer-dist laravel/lumen lumen-web-api
```

## Konfigurasi Lumen

Setelah selesai install sekarang kita melakukan konfigurasi untuk koneksi database seperti biasa kita lakukan jika menggunakan laravel yaitu di file .env seperti ini :

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=base64:JHdpwwjAk2lQxJtLfvoLxy8D2vQZW8ats0GEYF9GuLaCY
=
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=lumen-web-api
DB_USERNAME=root
DB_PASSWORD=alphaomega*9

CACHE_DRIVER=memcached
QUEUE_DRIVER=sync
```

Disini kita menggunakan database mysql dan dengan nama databasenya lumen-web-api.

Selanjutnya kita masuk ke file **bootstrap/app.php**. Disini kita akan mengaktifkan auth dan eloquent. Untuk itu kita perlu menghilangkan komen di beberapa bagian seperti berikut ini :

```
$app->withFacades();

$app->withEloquent();

$app->routeMiddleware([
    'auth' => App\Http\Middleware\Authenticate::class,
]);

$app->register(App\Providers\AuthServiceProvider::class);
```

Sekarang buka console masuk ke direktori public dan kita jalankan dengan perintah berikut ini :

```
php -S localhost:1000
```

Buka di web browser dan buka `http://localhost:1000/` jika sudah tampil **Lumen (5.3.0) (Laravel Components 5.3.\*)** maka kita sudah berhasil melakukan installasi lumen.

## Database migration Lumen

Database migration adalah salah satu fitur kesukaan saya dari framework laravel dimana dengan menggunakan database laravel ini kita bisa membuat schema database tanpa kita membuat query mysql yang cukup panjang. OK pada kasus ini kita akan membuat schema untuk user. Di lumen ini juga ada php artisan juga salah satu fitur kesukaan saya dimana artisan kita bisa membuat kelas dengan menggunakan console. Ok langsung kita buat database migrationnya dengan menjalankan perintah seperti berikut ini :

```
php artisan make:migration create_users_table --create=users
```

Dan kemudian masuk `/lumen-web-api/database/migrations/2016_09_17_192026_create_users_table.php` dan buat schema seperti berikut ini

```
Schema::create('users', function (Blueprint $table) {
    $table->increments('id');
    $table->string('username');
    $table->string('email')->unique();
    $table->string('password');
    $table->string('api_token');
    $table->rememberToken();
    $table->timestamps();
    $table->softDeletes();
});
```

Schema ini standar yang biasa kita gunakan ketika membuat table user. Migration laravel sudah support banyak fitur dalam membuat schema database untuk mengetahui lebih lanjut bisa baca selengkapnya disini.

Jika sudah selesai dengan membuat schema sekarang kita lakukan migrationnya dengan menjalankan perintah berikut ini

```
php artisan migrate
```

Jika berhasil maka kita akan mendapat pesan **Migration table created successfully**. Jika masih belum mendapatkan pesan itu periksa lagi codingan teman—teman pasti masih ada yang salah.

## Konfigurasi Model

Sudah selesai dengan migration sekarang kita akan melakukan konfigurasi model. Dengan model ini nanti akan mempermudah kita untuk melakaukan query ke database. Silahkan buka file **/lumen-web-api/app/User.php**. Class ini sudah ada setelah kita menginstall lumen. Untuk memahami lebih banyak tentang model ini bisa baca [disini](#). Untuk konfigurasinya seperti berikut ini :

```
/*
 * The attributes that are mass assignable.
 *
 * @var array
 */
protected $fillable = [
    'username', 'email', 'password', 'api_token'
];

/*
 * The attributes excluded from the model's JSON form.
 *
 * @var array
 */
protected $hidden = [
    'password', 'api_token'
];
```

Dari code di atas yang perlu di perhatikan adalah variable **\$fillable** dimana ini berfungsi memberikan izin colom mana saja dari database users ini yang bisa kita pakai. Dan untuk variable **\$hidden** ini nanti berfungsi agar tidak ditampilkan ketika kita melakukan query untuk mendapatkan semua data dari colom yang ada di database users.

## Routing pada Lumen

Sekarang kita bahas sedikit tentang routing pada lumen. Pada dasarnya konsep routing di lumen ini sama dengan laravel. Berikut ini ada sedikit perbedaan routing lumen 5.3 dengan laravel 5.3

- Lokasi file route **app/Http/routes.php**
- Cara mendaftarkan route dengan variable **\$app->getroute()** (**get, post, put, patch, delete, options**). Berikut ini contohnya :

```
Ex. $app->get('/', function () { return 'Hello World';});
```

Selebihnya sama saja temen—temen bisa dilanjutkan untuk belajar tentang routing ini lebih lanjut disini :

### HTTP Routing - Lumen - PHP Micro-Framework

By Laravel

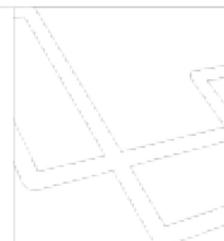
Lumen - The Stunningly Fast PHP Micro-Framework By Laravel

[lumen.laravel.com](https://lumen.laravel.com)

### Routing - Laravel - The PHP Framework For Web Artisans

Laravel - The PHP framework for web artisans.

[laravel.com](https://laravel.com)



Ok sekarang kita akan coba membuat route sesuai kebutuhan dari API yang akan kita buat. Berikut ini route dari web API yang akan kita buat :

```
$app->get('/', function () use ($app) {
    $res['success'] = true;
    $res['result'] = "Hello there welcome to web api using
```

```
lumen tutorial!";
    return response($res);
});

$app->post('/login', 'LoginController@index');
$app->post('/register', 'UserController@register');
$app->get('/user/{id}', ['middleware' => 'auth', 'uses' =>
'UserController@get_user']);
```

Pada line terakhir diatas route nya berbeda, karna disini kita akan mengguanakan middleware dari lumen ini untuk melakukan verifikasi akses token dari user yang akan menggunakan api yang kita buat ini. Unutk lebih lanjut penjelasan tentang middleware ini bisa baca disini :

[HTTP Middleware - Lumen - PHP Micro-Framework By Laravel](#)

[Lumen - The Stunningly Fast PHP Micro-Framework By Laravel](#)

[lumen.laravel.com](http://lumen.laravel.com)

## Controller pada Lumen

Controller pada lumen ini sendiri sama dengan controller yang ada di Laravel dimana ini berfungsi untuk kita gunakan sebagai business logic dari aplikasi yang kita buat.

Untuk membuat controller pada lumen ini sedikit berbeda dengan laravel karna pada lumen ini tidak ada fitur membuat controller dengan artisan. Jadi disini kita membuat file controller secara manual.

Ok yang pertama kita akan membuat controller UserController. Sama seperti di laravel kita membuat file controller di folder **app/Http/Controllers/**. Berikut ini file **UserController** nya :

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\User;
7
8  class UserController extends Controller
9  {
10     /**
11      * Register new user
12      *
13      * @param Request $request
14     */
15    public function register(Request $request)
16    {
17        $hasher = app()->make('hash');
18
19        $username = $request->input('username');
20        $email = $request->input('email');
21        $password = $hasher->make($request->input('password'));
22
23        $register = User::create([
24            'username' => $username,
25            'email' => $email,
26            'password' => $password,
27        ]);
28
29        if ($register) {
30            $res['success'] = true;
31            $res['message'] = 'Success register!';
32
33            return response($res);
34        } else {
35            $res['success'] = false;
36            $res['message'] = 'Failed to register!';
37
38            return response($res);
39        }
40    }
}
```

Setelah membuat fungsi untuk daftar kita akan buat class **LoginController** dimana dengan controller ini kita akan memberikan token kepada user yang akan di gunakan sebagai permission untuk meng akses seluruh aktifitas di dalam API ini. Berikut ini untuk class **LoginController** nya :

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\User;
7
8 class LoginController extends Controller
9 {
10     /**
11      * Index login controller
12      *
13      * When user success login will retrive callback as api
14      */
15     public function index(Request $request)
16     {
17         $hasher = app()->make('hash');
18
19         $email = $request->input('email');
20         $password = $request->input('password');
21
22         $login = User::where('email', $email)->first();
23         if (!$login) {
24             $res['success'] = false;
25             $res['message'] = 'Your email or password incor
26
27             return response($res);
28         }else{
29             if ($hasher->check($password, $login->password))
30                 $api_token = sha1(time());
31                 $create_token = User::where('id', $login->i
```

## Validasi akses token dengan Middleware

Setelah kita membuat fungsi untuk login, register dan get user sekarang kita akan buat validasi akses token nya menggunakan middle ware. Ok sebelum lanjut kita bahas sedikit apakah itu middleware ini. Teman—teman bisa baca dokumentasi lengkapnya [disini](#) mengenai middleware pada lumen ini.

*HTTP middleware provide a convenient mechanism for filtering HTTP requests entering your application*

Jadi middleware ini adalah fungsi yang dijalankan sebelum request ini sampai ke dalam controller. Nah dengan middleware ini kita akan mengecek apakah user ini mempunyai izin untuk menakses dari API ini.

Nah untuk mengaktifkan middleware seperti ini :

```
$app->withFacades();  
  
$app->withEloquent();  
  
$app->routeMiddleware([  
    'auth' => App\Http\Middleware\Authenticate::class,  
]);  
  
$app->register(App\Providers\AuthServiceProvider::class);
```

Sebenarnya ini sudah ada di dalam file **bootstrap/app.php**, namun secara default code—code diatas masih di comment jadi kita tidak perlu mengetik ulang untuk mengaktifkan middleware ini cukup uncomment code—code diatas.

Nah pada kasus ini kita akan membuat validasi bagaimana kita mengetahui apakah user ini mendapatkan ijin untuk mengakses api yang kita buat ini. Jadi begini logikanya ketika user melakukan request kita akan cek apakah dia mengirimkan token yang dia punya dan jika iya apakah tokenya itu ada di database. Disini nanti setiap user mempunyai akses tokenya masing—masing jadi nanti berbeda—beda token dari setiap usernya.

Untuk itu kita perlu menambahkan code berikut ini pada file **app/Http/Middleware/Authenticate.php** berikut ini codenya

```

1  public function handle($request, Closure $next, $guard = nu
2      {
3          if ($this->auth->guard($guard)->guest()) {
4              if ($request->has('api_token')) {
5                  $token = $request->input('api_token');
6                  $check_token = User::where('api_token', $to
7                  if ($check_token == null) {
8                      $res['success'] = false;
9                      $res['message'] = 'Permission not allow
10
11                     return response($res);
12                 }
13             }else{
14                 $res['success'] = false;

```

## Providers

Apa itu Providers? Nah providers ini adalah class yang dijalankan setelah user berhasil di verifikasi. Jadi jika user telah melakukan login dan mengakses api dengan token yang mereka punya kita bisa mendapatkan informasi dari user yang memiliki token tersebut nah dalam hal ini Providers lah yang berperan bagaimana kita mendapatkan data user berdasarkan token ini.

Nah untuk case study kita kali ini akan menggunakan di class **app/Providers/AuthServiceProvider.php** untuk mendapatkan data user berdasarkan token yang telah di verifikasi. Disini secara default dalam study kita tidak perlu menulis lagi codenya karna disini kita menggunakan metode secara default bawaan lumen ini. Disini saya akan jelaskan sedikit bagaimana itu bekerja.

```

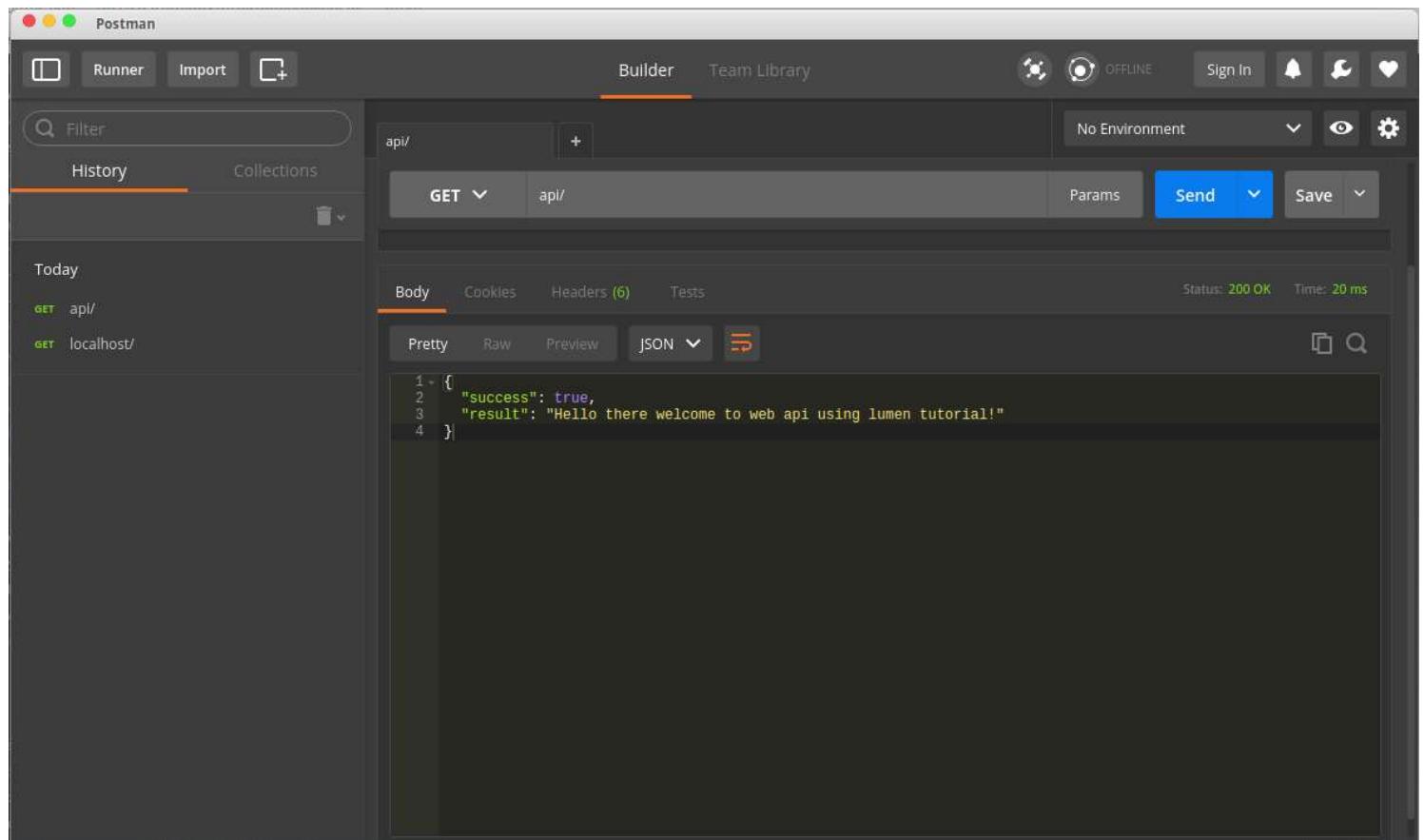
$this->app['auth']->viaRequest('api', function ($request) {
    if ($request->input('api_token')) {
        return User::where('api_token',
                           $request->input('api_token'))-
                           >first();
    }
});
```

Nah code di atas adalah code yang berada didalam method boo()  
dimana boot ini sendiri adalah method yang digunakan untuk  
mendapatkan data user berdasarkan token yang telah di verifikasi oleh  
middleware.

Pada kasus ini kita melakukan query kedalam table **users** berdasarkan  
api\_token yang telah kita terima dari middleware.

## Testing API Service

Ok sekarang kita melakukan testing untuk mengetahui apakah API  
yang kita buat telah berfungsi atau belum. Disini kita akan  
menggunakan postman untuk melakukan testing nya. Silah kan untuk  
teman—teman yang belum ada postman bisa download [disini](#).

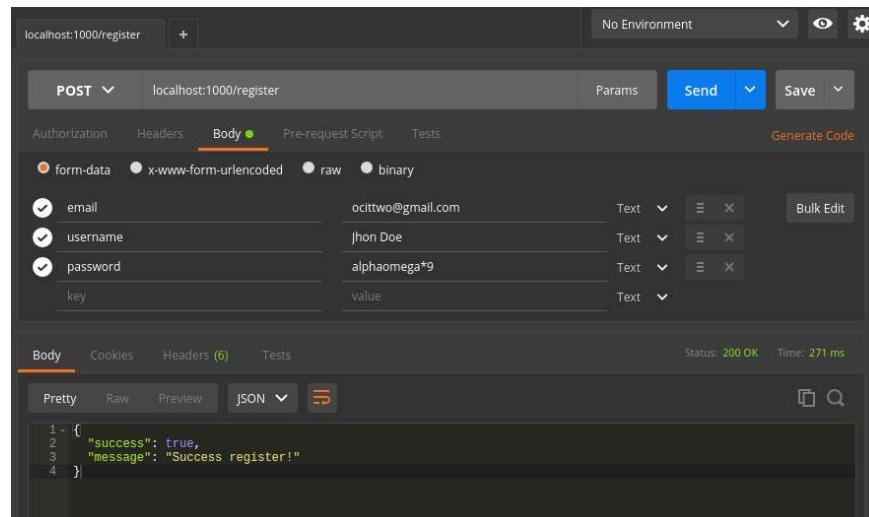


Di sini saya menggunakan server nginx dan dengan settingan  
servername nya api, jadi saya bisa mengaksesnya melalui <http://api/>  
silahkan sesuaikan dengan settingan server teman—teman untuk

melakukan akses url nya. Kita juga bisa menggunakan server default nya php dengan menjalankan perintah berikut ini :

```
php -S localhost:1000
```

Dengan begitu kita akan bisa mengakses ke <http://localhost:1000/>. Ok untuk test case yang pertama kita akan menguji fungsi untuk register seperti berikut ini :



The screenshot shows a Postman interface with a POST request to `localhost:1000/register`. The `Body` tab is selected, showing form-data fields: `email` (value: `ocittwo@gmail.com`), `username` (value: `Jhon Doe`), and `password` (value: `alphaomega*9`). The response tab shows a status of `200 OK` with a response body containing the JSON object: 

```
1: {  
2:   "success": true,  
3:   "message": "Success register!"  
4: }
```

Sudah selesai register sekarang kita coba untuk login seperti berikut ini, disini kita coba melakukan input login yang salah :

The screenshot shows a Postman interface with the following details:

- Request URL:** localhost:1000/login
- Method:** POST
- Body Tab:** The "Body" tab is selected, showing the following form-data fields:
  - username: ocittwo@gmail.com
  - password: alphaomega\*9
- Response Status:** 200 OK
- Response Body (Pretty JSON):**

```
1- {  
2-   "success": false,  
3-   "message": "Your email or password incorrect!"  
4- }
```

Ok sudah benar dan sekarang case yang benar kita coba.

The screenshot shows the Postman interface with a successful login request. The URL is `localhost:1000/login`. The method is `POST`. The `Body` tab is selected, showing form-data fields: `email` (value: `ocittwo@gmail.com`) and `password` (value: `alphaomega*9`). The response status is `200 OK` with a response time of `152 ms`. The response body is a JSON object:

```

1+ {
2   "success": true,
3   "api_token": "f75cc8c0ef661443fe73fb6665ab3bba8f0a747",
4   "message": {
5     "id": 2,
6     "username": "Jhon Doe",
7     "email": "ocittwo@gmail.com",
8     "remember_token": null,
9     "created_at": "2016-09-18 17:21:23",
10    "updated_at": "2016-09-18 17:25:29",
11    "deleted_at": null
12  }
13 }

```

Ok sekarang kita sudah berhasil login dan mendapatkan api\_token dan sekarang kita coba test case untuk menguji apakah akses token sudah berjalan seperti berikut ini :

The screenshot shows a test case for retrieving a user profile. The URL is `localhost:1000/user/1`. The method is `GET`. The `Authorization` tab is selected, showing `Type: No Auth`. The response status is `200 OK` with a response time of `162 ms`. The response body is a JSON object:

```

1+ {
2   "success": false,
3   "message": "Login please!"
4 }

```

Yang pertama disini kita membuat case yang salah dan ternyata sudah berjalan proses pengecekannya, sekarang lakukan lagi test case yang benar :

The screenshot shows a Postman interface with the following details:

- URL:** localhost:1000/user/1?api\_token=6297fdb767771ca1619e1f2800a7675b345f280
- Type:** No Auth
- Body:** JSON (Pretty)
- Response:**

```

1 - {
2   "success": true,
3   "get_user": [
4     {
5       "id": 1,
6       "username": "Ahmad Rosids",
7       "email": "rosid@gmail.com",
8       "remember_token": null,
9       "created_at": "2016-09-17 20:39:30",
10      "updated_at": "2016-09-18 13:57:41",
11      "deleted_at": null
12    }
13  ]
14 }
```
- Status:** 200 OK
- Time:** 43 ms

Silahkan lakukan test—test yang lain dan lakukan variasi test—test karna pasti selalu ada permasalahan dalam kondisi yang berbeda dan untuk itu kita perlu melakukan test ya temen—temen. Saran saya untuk melakukan test sebaiknya kita buat input error nya, yah karena test itu sendiri fungsinya untuk mencari error. Dan jangan takut kalau terjadi error karna semakin sering kita menemui error semakin banyak pelajaran yang kita dapatkan dari situ.

Nah cukup disini dulu next artikel kita akan belajar bagaimana membuat operasi **CRUD (Create, Read, Update, Delete)**, yang mana ini adalah hal umum dalam proses API itu bekerja, dan cukup penting saya rasa, so tunggu next artikelnya ya :)

OK sampai disini dulu kalau ada pertanyaan silahkan hubungi saya via facebook atau social media yang lainnya.

Untuk project latihan bisa akses source codenya disini

[ar-android/Lumen-Web-API](#)

Lumen-Web-API - Simple web API using Lumen 5.3



with Simple Auth System  
[github.com](https://github.com)



