

# Table of contents

[Kata Pengantar](#_bookmark0) 4

[Persiapan Lingkungan Kerja](#_bookmark1) 5

[Tools yang Dibutuhkan](#_bookmark2) 5

[Web Browser](#_bookmark3) 5

[Web Server](#_bookmark4) 6

[PHP](#_bookmark5) 7

[Composer](#_bookmark6) 7

[Visual Studio Code](#_bookmark7) 9

[Git](#_bookmark8) 10

[Postman](#_bookmark9) 11

[Docker](#_bookmark10) 11

[Kesimpulan](#_bookmark11) 11

[Pengenalan Microframework Laravel](#_bookmark12) 12

[Pengertian MVC](#_bookmark13) 12

[Sejarah MVC](#_bookmark14) 12

[Arsitektur MVC](#_bookmark15) 13

[Contoh MVC](#_bookmark16) 13

[Fungsi Lumen](#_bookmark17) 14

[Bedanya dengan Framework sejenis?](#_bookmark18) 14

[Fitur yang ada di Lumen](#_bookmark19) 15

[Route](#_bookmark20) 15

[Metode yang tersedia](#_bookmark21) 16

[Parameter route](#_bookmark22) 16

[Penggunaan Regular Expression](#_bookmark23) 17

[Parameter Opsional](#_bookmark24) 17

[Alias](#_bookmark25) 17

[Group](#_bookmark26) 18

[Namespace](#_bookmark27) 18

[Prefix](#_bookmark28) 18

[Middleware](#_bookmark29) 19

[Mendefinisikan Middleware](#_bookmark30) 19

[Before/After Middleware](#_bookmark31) 20

[Registrasi Middlware Secara Global](#_bookmark32) 21

[Registrasi Middlware pada Route](#_bookmark33) 21

[Parameter pada Middleware](#_bookmark34) 22

[Penggunaan method terminate()](#_bookmark35) 23

[Controller](#_bookmark36) 23

[Request](#_bookmark37) 24

[Response](#_bookmark38) 25

[String & Array](#_bookmark39) 25

[JSON](#_bookmark40) 26

[Eloquent](#_bookmark41) 26

[Pengantar](#_bookmark42) 26

[Eloquent : Relationship](#_bookmark43) 31

[One to One](#_bookmark44) 31

[One to Many](#_bookmark45) 32

[Many to Many](#_bookmark46) 32

[Kesimpulan](#_bookmark47) 33

[Instalasi](#_bookmark48) 34

[Kebutuhan Server](#_bookmark49) 34

[Intalasi Lumen](#_bookmark50) 34

[Struktur pada Lumen](#_bookmark51) 35

[Menjalankan Lumen](#_bookmark52) 36

[Kesimpulan](#_bookmark53) 37

[Konfigurasi](#_bookmark54) 38

[Konfigurasi Database](#_bookmark55) 38

[Registrasi Facedes, Eloquent & Authentication](#_bookmark56) 39

[*Generate*](#_bookmark57)

[APP\_KEY](#_bookmark57)

[Kesimpulan](#_bookmark58) . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

#### [Studi Kasus - TodoManager](#_bookmark59) 41

[Membuat Model dan *Migration*](#_bookmark60)41

[Membuat Model dan *Migration Todo*](#_bookmark61)41

[Membuat *URL* atau *API Endpoint*](#_bookmark62)42

[Membuat *Controller*](#_bookmark63)43

[Membuat Controller TodoManager](#_bookmark64) 43

[Menguji dengan *Postman*](#_bookmark65)46

[Kesimpulan](#_bookmark66) 54

# Kata Pengantar

#### Bismillah

Segala puji syukur penulis panjatkan kehadirat Allah Subhanahu Wa Ta’ala karena berkat Rahmat dan Karunianya penulis dapat menyelesaikan penyusunan Buku **Lumen RESTful API** ini.

Shalawat beserta salam semoga senantiasa terlimpah curahkan kepada Nabi Muhammad ﷺ kepada keluarganya, para sahabatnya, hingga umatnya hingga akhir zaman. Aamiin.

Tidak lupa penulis ucapkan terima kasih kepada berbagai pihak baik yang telah membantu kami secara langsung atau tidak langsung dalam proses penyusunan buku ini.

# Persiapan Lingkungan Kerja

Pada tahap ini kita akan mempersiapan dan instalasi kebutuhan yang akan kita gunakan pada tahap-tahap selanjutnya, kita pastikan semua yang dibutuhkan sudah terinstal.

***Tools* yang Dibutuhkan**

*Tools-tools* yang dibutuhkan untuk mengikuti panduan pada buku ini yaitu:

1. Web Browser
2. Web Server
3. PHP
4. Composer
5. Visual Studio Code
6. Git
7. Postman
8. Docker
9. Niat, Usaha dan Kerja Keras.

##### Web Browser

Karena merupakan aplikasi berbasis *web*, maka kita perlu *web browser* untuk mengakses aplikasi kita. Ada tiga *web browser* yang penulis rekomendasikan yaitu **Google Chrome, Microsoft Edge** dan **Mozilla Firefox**, silakan pilih salah satu saja dan pastikan kita menggunakan versi yang terbaru.

Penulis pribadi menggunakan *browser microsoft edge*, namun jika sudah terbiasa dengan *google chrome* atau

*mozilla firefox* atau bahkan ada *browser* selain ketiga di atas silahkan menggunakan *browser* tersebut. Silahkan kunjungi situs resmi untuk proses instalasi.

* [Google Chrome - Download the Fast, Secure Browser from Google](https://www.google.com/chrome/)
* [Unduh Browser Microsoft Edge Baru | Microsoft](https://www.microsoft.com/id-id/edge)
* [Download Firefox Browser — Fast, Private & Free — from Mozilla](https://www.mozilla.org/en-US/firefox/new/)



**Figure 1:** Chrome, Firefox, Edge

##### Web Server

Merupakan *tools* yang terletak di sisi *server* untuk menangani permintaan dari *client*. Ada dua *web server* yang umum digunakan yaitu **Apache** dan **NginX** (dibaca: *engine* X). Meskipun pada saat pengembangan kita sebenarnya tidak harus menggunakan *web server*, karena PHP telah memiliki *built-in web server* sendiri.

Silahkan kunjungi situs resmi untuk proses instalasi.

* [Welcome to The Apache Software Foundation!](https://apache.org/)
* [Download XAMPP (apachefriends.org)](https://www.apachefriends.org/download.html)
* [NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy](https://www.nginx.com/)



**Figure 2:** apache dan nginx

##### PHP

Merupakan *tools* untuk menjalankan bahasa pemrograman **PHP**. Adapun versi **PHP** yang digunakan adalah versi 7.3 keatas, hal ini berkaitan dengan Lumen yang membutuhkan minimal PHP versi 7.3.

Silahkan kunjungi situs resmi untuk proses instalasi. <https://www.php.net/manual/en/install.php>

##### Composer

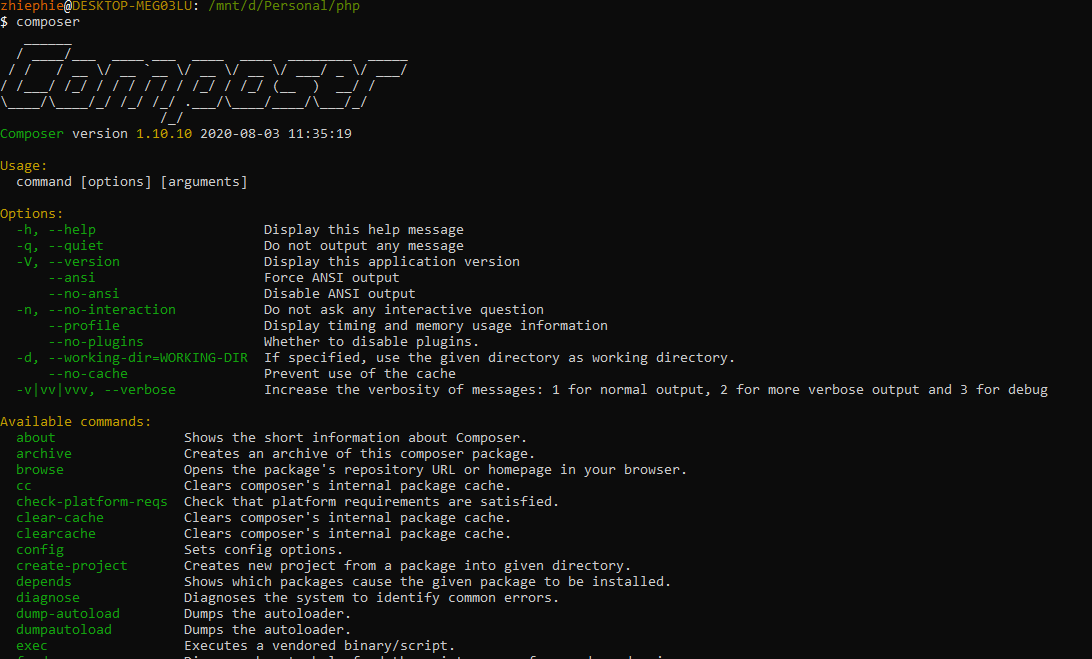
**Composer** adalah *dependency manager* untuk **PHP**, yaitu sebuah *tools* yang digunakan untuk memudahkan kita dalam mengelola *library* **PHP** yang digunakan pada aplikasi kita beserta dependensinya.

Silahkan kunjungi situs resmi untuk proses instalasi. [Composer (getcomposer.org)](https://getcomposer.org/)



**Figure 3:** Install Composer

Untuk mengetahui apakah **Composer** sudah berhasil terinstal di dalam komputer, kita bisa mengecek dengan menggunakan perintah berikut ini composer di terminal/cmd :



**Figure 4:** Validasi Composer

Jika muncul seperti gambar di atas, maka sekarang sudah bisa membuat proyek baru menggunakan **Composer**.

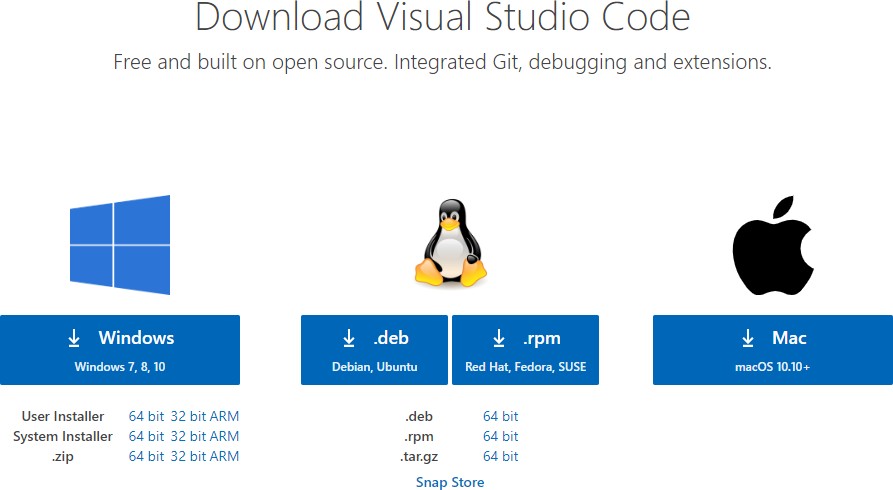
Catatan : Composer versi 2 sudah tersedia, penulis merekomendasikan jika ada yang masih memakai

composer versi 1 silahkan *update* menggunakan perintah berikut : composer self-update --2 jika perintah tersebut eror gunakan sudo .

##### Visual Studio Code

**Visual Studio Code** atau banyak dikenal dengan **VSCode** ini adalah sebuah teks editor yang dikembangkan oleh **Microsoft**.

Silahkan kunjungi situs resmi untuk proses instalasi. [Visual Studio Code - Code Editing. Redefined](https://code.visualstudio.com/)



**Figure 5:** Visual Studio Code

##### Git

**Git** merupakan *tools* yang berfungsi untuk mengontrol revisi kode secara terdistribusi dan mengelola kode sumber (*distributed revision control and source code management*/SCM). Dengan menggunakan **Git**, kita akan lebih mudah dalam mengontrol revisi kode sumber aplikasi kita.

Silahkan kunjungi situs resmi untuk proses instalasi. <https://git-scm.com/downloads>

**Figure 6:** git download

##### Postman

**Postman** adalah *tool* untuk pengujian apakah *web service* berjalan atau tidak sesuai dengan yang diharapkan maka sebenarnya kita bisa gunakan *tools* HTTP *client* yaitu **Postman**.

Silahkan kunjungi situs resmi untuk proses instalasi. <https://www.postman.com/downloads/>

**Figure 7:** Install Postman

##### Docker

**Docker** adalah *platform* perangkat lunak yang memungkinkan kita membuat, menguji, dan menerapkan aplikasi dengan cepat. **Docker** mengemas perangkat lunak ke dalam unit standar yang disebut Kontainer yang memiliki semua yang diperlukan perangkat lunak agar dapat berfungsi termasuk pustaka, alat sistem, kode, dan waktu proses. Dengan menggunakan **Docker**, Kita dapat dengan cepat menerapkan dan menskalakan aplikasi ke lingkungan apa pun dan yakin bahwa kode Kita berjalan.

**Docker** ini bersifat opsional, karena kita juga bisa bekerja tanpa menggunakan *docker*, namun penulis hanya menyarankan untuk instal, karena pada materi akan dibahas juga penggunaan *docker* pada proyek kita.

Silahkan kunjungi situs resmi untuk proses instalasi. [Empowering App Development for Developers | Docker](https://www.docker.com/)

## Kesimpulan

Pada tahap ini kita sudah mempersiapkan kebutuhan dan keperluan proses pada tahap selanjutnya, maka dari itu kita sudah siap untuk masuk ke tahap pengenalan dan pembahasan dasar-dasar fitur yang ada pada Lumen.

# Pengenalan Microframework Laravel



**Figure 8:** Lumen Framework Home Page

Lumen adalah *micro-framework* yang diciptakan pengembang Laravel untuk mengakomodasi kebutuhan *devel- oper* yang ingin membuat aplikasi dalam skala lebih kecil dengan ekosistem Laravel. Secara konsep, Lumen menerapkan konsep MVC (*Model, View, Controller*).

## Pengertian MVC

**MVC** adalah singkatan dari “*Model View Controller*” merupakan suatu konsep yang sangat populer dalam pembangunan sebuah aplikasi. **MVC** memisahkan pengembangan aplikasi berdasarkan 3 jenis komponen utama yaitu manipulasi *data*, *user interface*, dan bagian yang menjadi kontrol aplikasi. Komponen-komponen utama tersebut membangun suatu *MVC pattern* atau bagian yang diberi nama *Model, View* dan *Controller*.

#### Sejarah MVC

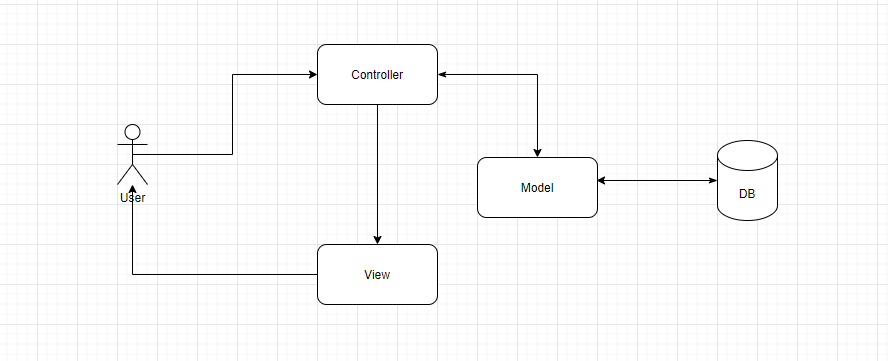
* Arsitektur **MVC** pertama kali dibahas pada tahun 1979 oleh Trygve Reenskaug.
* Model **MVC** pertama kali diperkenalkan pada tahun 1987 dalam bahasa pemrograman *Smalltalk*.
* **MVC** pertama kali diterima sebagai konsep umum, dalam artikel 1988.
* Saat ini, pola MVC banyak digunakan dalam aplikasi *web modern*.

Dengan menggunakan konsep **MVC**, suatu aplikasi dapat dikembangkan sesuai dengan kemampuan *PIC*-nya, *Developer* yang menangani bagian *Model* dan *Controller*, sedangkan *Web Designer* yang menangani bagian *View*, sehingga penggunaan arsitektur **MVC** dapat meningkatkan *maintainability* dan pengorganisasian kode.

Meskipun begitu, namun tetap dibutuhkan komunikasi yang baik *Developer* dan *Web Designer* dalam menangani variabel dan parameter *data* yang ada.

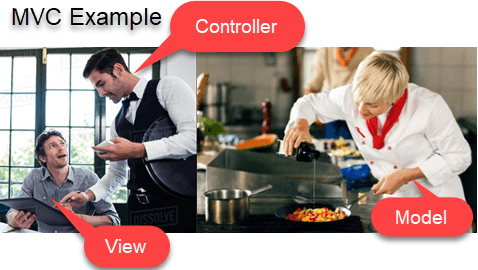
* **Model** adalah kode-kode untuk *model* bisnis dan data. biasanya berhubungan langsung dengan *database* untuk memanipulasi data (*create, read, update, delete*), menangani validasi dari bagian *controller*, namun tidak dapat berhubungan langsung dengan bagian *view*.
* **View** merupakan bagian yang menangani *presentation logic*. berisi kode-kode untuk ditampilan.
* **Controller** merupakan bagian yang mengatur hubungan antara bagian *model* dan bagian *view*, *controller* berfungsi untuk menerima *request* dan data dari user kemudian menentukan apa yang akan diproses oleh aplikasi.

#### Arsitektur MVC



**Figure 9:** Konsep MVC

#### Contoh MVC



**Figure 10:** Contoh MVC

# Fungsi Lumen

Secara umum Lumen digunakan untuk membuat *web service* **RESTful API** dengan serialisasinya menggunakan

*data format JSON*.

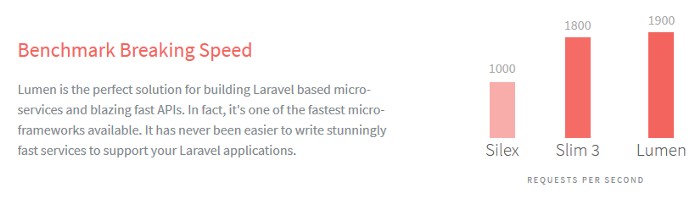
Pembuatan *web service* di lumen sangatlah mudah dan praktis, karena Lumen sendiri dari awal sudah didesain untuk membuat **RESTful API**.

Namun jangan berkecil hati, Lumen juga dapat digunakan untuk membuat *website* pada umumnya, misalnya membuat portal sistem informasi. Kendalanya kembali lagi pada tujuan lumen dibuat. Ketika lumen untuk membuat aplikasi *website* maka *library*nya kurang begitu lengkap seperti halnya Laravel.

## Bedanya dengan Framework sejenis?

Ada beberapa *Micro Framework* lain yang dikhususkan untuk membuat **RESTful API** pada PHP seperti **Slim Framework, Silex** dan lain-lain, tentu dengan keistimewaannya masing-masing, namun yang menarik adalah Lu- men dalam lamannya mampu menghandel hingga 1900 *request per second*. Lebih unggul dari **Slim Framework** yang hanya dapat menghandel 1800 *request per second*.

Perbedaan lain yaitu Lumen menggunakan PHP versi 7 sebagai syarat penggunaannya, sedangkan Slim masih bisa digunakan PHP versi 5 ke atas.



**Figure 11:** Benchmark 1

# Fitur yang ada di Lumen

Lumen sudah dibundel dengan **Eloquent ORM** agar proses *query* lebih mudah dan tidak memakan waktu karena kita tidak perlu lagi menulis *query* yang panjang.

Lumen sendiri sangat mirip sekali dengan Laravel, dimana lumen sudah ada beberapa fitur seperti:

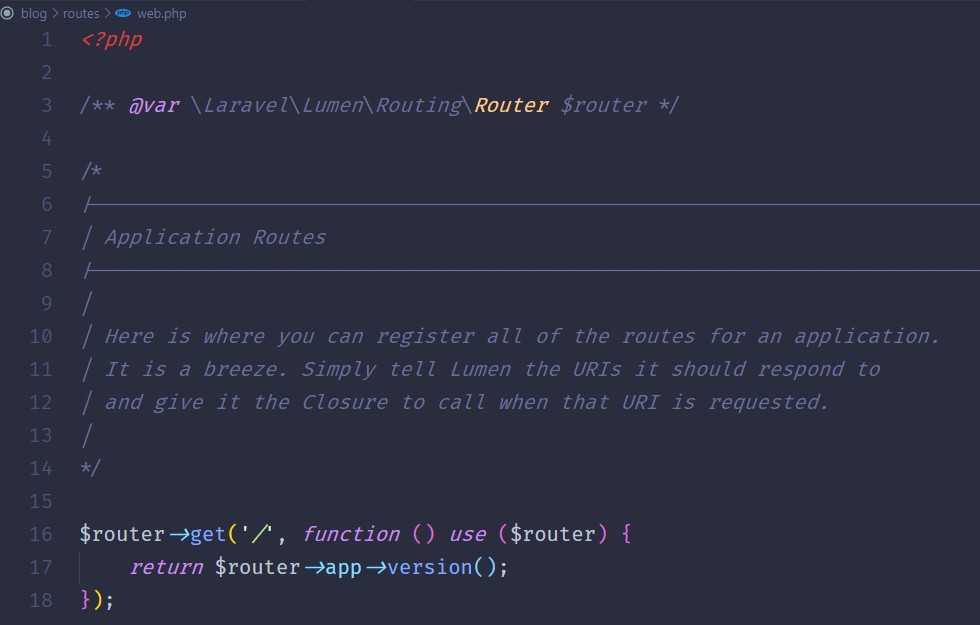
* Routing
* Middleware
* Controller
* Requests
* Responses
* Dataabase
* Queues
* Event
* Unit Test
* dan Lainnya.

Mari kita belajar lebih dalam lagi mengenai fitur-fitur yang ada pada lumen.

### Route

*Route* berfungsi untuk mengatur lalu lintas berdasarkan *request* dari pengguna. *File Route* terletak di dalam

*folder* **/routes/web.php**.



**Figure 12:** route default lumen

#### Metode yang tersedia

Berikut adalah metode *HTTP* yang ada pada Lumen, sama persis dengan Laravel. Karena ini metode standar pada *HTTP*.

**$router**->get(**$uri**, **$callback**);

**$router**->post(**$uri**, **$callback**);

**$router**->put(**$uri**, **$callback**);

**$router**->patch(**$uri**, **$callback**);

**$router**->delete(**$uri**, **$callback**);

**$router**->options(**$uri**, **$callback**);

**Parameter *route***

Pastinya kita perlu untuk menangkap segmen **URI** dalam *route*. Misalnya, Kita perlu mengambil id pengguna dari **URL** itu sendiri. Kita dapat melakukannya dengan menentukan parameter *route* sebagai berikut:

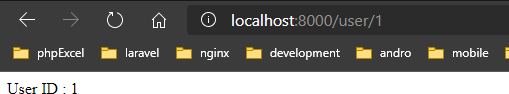
**$router**->get('user/{id}', **function** (**$id**) {

**return** 'User ID : '.**$id**;

});

Pada contoh kode di atas ketika kita jalankan dan akses *url* berikut http://localhost:8000/user/1 maka akan otomatis tercetak pada *web browser* User ID : 1 .

Seperti pada gambar dibawah ini:



**Figure 13:** route parameter

Kita juga bisa memberikan parameter lebih dari satu, karena alasan kompleksitas aplikasi kita membutuhkan parameter lebih dari satu.

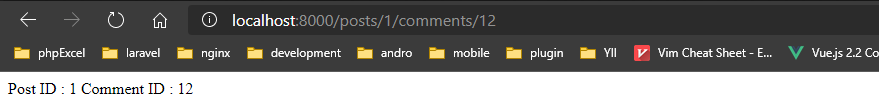
Seperti baris kode berikut ini:

**$router**->get('posts/{postId}/comments/{commentId}', **function** (**$postId**, **$commentId**) {

**return** 'Post ID : '.**$postId**. ' Comment ID : '.**$commentId**;

});

Pada contoh kode di atas akan menghasilkan respon seperti gambar di bawah ini:



**Figure 14:** Route

**Penggunaan *Regular Expression***

Kita juga dapat membatasi format parameter pada *route* dengan menggunakan *Regular Expression* dalam definisi

*route* seperti baris kode berikut:

**$router**->get('user/{name:[A-Za-z]+}', **function** (**$name**) {

//

});

#### Parameter Opsional

Kita dapat menentukan parameter *route* secara opsional yang artinya parameter tersebut tidak mandatori dengan menyertakan bagian dari definisi *URI*. Menempatkan parameter opsional di tengah definisi *route*:

**$router**->get('user[/{name}]', **function** (**$name** = **null**) {

**return $name**;

});

#### Alias

Kita juga dapat menetapkan alias untuk *route* menggunakan *key* as pada saat menentukan *route*:

**$router**->get('profile', ['as' => 'profile', **function** () {

//

}]);

Secara spesifikasi kita juga bisa menentukan *controller* tertentu, seperti baris kode berikut ini:

**$router**->get('profile', [

'as' => 'profile', 'uses' => 'UserController@showProfile'

]);

##### Group

Seperti halnya pada laravel, pada lumen sendiri juga mendukung penamaan secara pengelompokan, kita bisa menggunakan *method* $route->group

Kegunaan $route->group untuk pengelompokan *route* dalam grup tertentu, kita juga dapat menggunakan

middleware dalam atribut grup tersebut. Seperti baris kode berikut ini:

**$router**->group(['middleware' => 'auth'], **function** () **use** (**$router**) {

**$router**->get('/', **function** () {

// Uses Auth Middleware

});

**$router**->get('user/profile', **function** () {

// Uses Auth Middleware

});

});

##### Namespace

Kasus penggunaan umum lainnya untuk grup *route* adalah menggunakan *namespace* **PHP** yang sama ke grup pengontrol.

Kita dapat menggunakan parameter *namespace* di atribut grup:

**$router**->group(['namespace' => 'Admin'], **function**() **use** (**$router**)

{

// Using The "App\Http\Controllers\Admin" Namespace...

**$router**->group(['namespace' => 'User'], **function**() **use** (**$router**) {

// Using The "App\Http\Controllers\Admin\User" Namespace...

});

});

##### Prefix

Atribut grup *prefix* dapat digunakan untuk mengawali setiap *route* dalam grup dengan **URI** tertentu. Misalnya, kita mungkin ingin memberi *prefix* semua **URI** route dalam grup dengan awalan admin :

**$router**->group(['prefix' => 'admin'], **function** () **use** (**$router**) {

**$router**->get('users', **function** () {

// Matches The "/admin/users" URL

});

});

Contoh kode diatas akan menghasilkan *url* seperti berikut ini http://localhost:8000/admin/users

Atau penggunaan secara spesifikasi seperti baris kode berikut ini:

**$router**->group(['prefix' => 'accounts/{accountId}'], **function** () **use** (**$router**) {

**$router**->get('detail', **function** (**$accountId**) {

// Matches The "/accounts/{accountId}/detail" URL

});

});

Contoh kode diatas akan menghasilkan *url* seperti berikut ini http://localhost:8000/accounts/12/detail

## Middleware

Istilah *middleware* biasa digunakan untuk menyebut sebuah perangkat lunak yang berperan sebagai “penen- gah” antara sebuah aplikasi dengan aplikasi lain untuk mempermudah proses integrasi antara aplikasi-aplikasi tersebut.

Dalam konteks **Lumen**, *Middleware* merupakan sebuah *Class* khusus yang berperan sebagai “penengah” antara *request* yang masuk ke *Controller* yang dituju. Secara umum, prinsip kerja *Middleware* adalah menjembatani *request* yang masuk untuk kemudian diproses terlebih dahulu sebelum diberikan kepada *Controller* yang dituju atau diarahkan ke *Controller* yang lain. Dengan menggunakan fitur ini, kita dapat membuat komponen yang *reusable* untuk melakukan pekerjaan-pekerjaan tersebut.

**Mendefinisikan *Middleware***

Secara *default* Lumen telah menyediakan satu buah *Middleware* dengan nama Authenticate.php yang terletak pada direktori app/Http/Middleware . Untuk dapat menambahkan *Middleware* baru kita cukup membuat kelas dengan format seperti berikut ini:

**namespace A**pp\**H**ttp\**M**iddleware; **use Closure**;

**class M**yMiddleware

{

/\*\*

* Run the request filter.

\*

* ***@param \Illuminate\Http\Request*** $request
* ***@param \Closure*** $next
* ***@return*** mixed

\*/

**public function** handle(**$request**, **Closure $next**)

{

//logic

}

}

Seperti yang terlihat pada kode di atas, dalam setiap *Middleware* terdapat sebuah *method* khusus yang bernama handle() . *Method* tersebut memiliki dua buah parameter yaitu Illuminate\Http\Request $request dan

Closure $next . *Method* ini akan dipanggil secara otomatis oleh Lumen ketika kita meregistrasikan *middleware* tersebut. Lalu, kode seperti apakah yang harus kita letakkan di dalam method handle() tersebut. Kita bisa meletakkan logika yang kita inginkan. Misal, seperti berikut ini.

**public function** handle(**$request**, **Closure $next**)

{

**if** (**$request**->input('age') < 17) {

**return** redirect('home');

}

**return $next**(**$request**);

}

Contoh kode di atas merupakan kode sederhana yang melakukan pengecekan terhadap usia dari *user*. Apabila *user* mengisikan usia lebih dari 17 tahun, maka kita akan meneruskan *request* yang masuk dan memberikannya kepada *Controller* yang dituju (dengan memanggil 𝑛𝑒𝑥𝑡(*request)*). Namun, apabila *user* yang melakukan *request* usia kurang dari 17 tahun maka kita akan me-redirect *user* tersebut ke halaman *home*. Dengan cara ini kita dapat mem-filter setiap *request* yang masuk dengan mudah.

##### Before/After Middleware

Secara umum, *Middleware* pada lumen dapat digolongkan kedalam dua kelompok yaitu *After Middleware* dan *Before Middleware. After Middleware* merupakan *Middleware* yang diproses setelah *request* masuk kedalam *Controller*, sedangkan *Before Middlware* merupakan *Middleware* yang diproses sebelum *request* masuk kedalam *Controller*.

Berikut ini adalah contoh mendefinisikan kedua buah *Middlware* tersebut.

**namespace A**pp\**H**ttp\**M**iddleware; **use Closure**;

**class B**eforeMiddleware

{

**public function** handle(**$request**, **Closure $next**)

{

//lakukan sesuatu terhadap request yang masuk..

**return $next**(**$request**)

}

}

Contoh kode di atas merupakan contoh bagaimana mendefinisikan sebuah *Before Middleware* pada Lumen. Pada kode tersebut terlihat bahwa *request* yang masuk akan diproses terlebih dahulu sebelum diteruskan ke *Controller* yang dituju.

**namespace A**pp\**H**ttp\**M**iddleware; **use Closure**;

**class A**fterMiddleware

{

**public function** handle(**$request**, **Closure $next**)

{

**$response** = **$next**(**$request**);

//lakukan sesuatu terhadap response yang diperoleh

**return $response**

}

}

Berbeda dengan *Before Middleware*, pada *After Middleware* kita meneruskan *request* yang masuk ke *Controller* yang dituju terlebih dahulu hingga mendapatkan *response* dari *Controller* tersebut. Setelah mendapatkan *response* yang dimaksud, kita akan memprosesnya lebih lanjut sebelum nantinya dikembalikan ke *user*.

**Registrasi *Middlware* Secara Global**

Setiap *Middleware* yang dibuat dapat diregistrasikan secara *global* sehingga *Middleware* tersebut akan selalu dipanggil setiap ada *request* yang masuk. Untuk dapat meregistrasikan *Middleware* yang dibuat secara *global*, kita dapat menambahkannya di dalam *file* bootstrap/app.php seperti contoh di bawah ini:

**$app**->middleware([ **A**pp\**H**ttp\**M**iddleware\**M**yMiddleware::**class**

]);

**Registrasi *Middlware* pada *Route***

Selain dapat didefinisikan secara *global*, komponen *Middleware* yang dibuat juga dapat diregistrasikan secara spesifik untuk digunakan pada *Routes* tertentu dengan cara menambahkan nama *Middleware* yang dibuat kedalam *file* bootstrap/app.php seperti contoh dibawah ini:

**$app**->routeMiddleware([

'auth' => **A**pp\**H**ttp\**M**iddleware\**A**uthenticate::**class**,

]);

Setelah meregistrasikan *Middleware* kita kedalam $routeMiddleware seperti contoh diatas, berikutnya adalah memberitahu Lumen, *Routes* mana saja yang akan menggunakan *Middleware* tersebut dengan cara menambahkan *key middlware* pada bagian *route option* yang berada di *file* routes/web.php seperti contoh dibawah ini:



**$router**->get('admin/profile', ['middleware' => 'auth', **function** () {

//

}]);

Menggunakan *multiple middleware* sekaligus pada *route*:

**$router**->get('/', ['middleware' => ['first', 'second'], **function** () {

//

}]);

**Parameter pada *Middleware***

**namespace A**pp\**H**ttp\**M**iddleware; **use Closure**;

**class R**oleMiddleware

{

/\*\*

* Run the request filter.

\*

* ***@param \Illuminate\Http\Request*** $request
* ***@param \Closure*** $next
* ***@param string*** $role
* ***@return*** mixed

\*/

**public function** handle(**$request**, **Closure $next**, **$role**)

{

**if** (! **$request**->user()->hasRole(**$role**)) {

// Redirect...

}

**return $next**(**$request**);

}

}

Contoh kode di atas terlihat bahwa pada *method* handle() terdapat penambahan paramater $role . Untuk penerapan pada *route* bisa seperti kode di bawah ini.

**$router**->put('post/{id}', ['middleware' => 'role:editor', **function** (**$id**) {

//

}]);

Parameter *middleware* dapat ditentukan saat menentukan *route* dengan memisahkan nama *middleware* dan parameter bisa diletakkan setelah simbol :

Contoh kode di atas terlihat bahwa parameter dapat diletakkan diantara tanda : .

**Penggunaan *method terminate()***

**namespace I**lluminate\**S**ession\**M**iddleware; **use Closure**;

**class S**tartSession

{

**public function** handle(**$request**, **Closure $next**)

{

**return $next**(**$request**);

}

**public function** terminate(**$request**, **$response**)

{

// Store the session data...

}

}

Contoh kode di atas *method* terminate harus menerima *request* dan *response*. Setelah kita menentukan

*middleware*.

Saat memanggil *method* terminate di *middleware*, Lumen akan menyelesaikan *instance* baru *middleware*

dari *service container*. Jika ingin menggunakan *instance middleware* yang sama saat *method* handle dan

terminate dipanggil, daftarkan *middleware* dengan *container* menggunakan *method* singleton .

### Controller

Dalam konsep **MVC** biasanya sebuah *URL* dipetakkan ke sebuah *controller*, dan karena konsep **MVC** sudah mendarah daging di kalangan *Developer*, maka kita akan mencoba membuat satu halaman dengan *url* /halo-dunia

.

Pertama kita harus daftarkan *controller* ke *routes*.

Buka kembali *file* routes/web.php , lalu tambahkan *route* baru seperti kode di bawah ini:

**$router**->get('halo-dunia', 'HaloController@index');

Penjelasan dari kode di atas, jika ada yang *request url* /halo-dunia , maka lumen akan mengeksekusi fungsi

index() di dalam HaloController . Setelah itu maka langkah selanjutnya adalah membuat *controller*. Buat *file* baru dengan nama HaloController pada direktori app/Http/Controllers/HaloController.php :

**namespace A**pp\**H**ttp\**C**ontrollers;

**class H**aloController

{

**public function** index()

{

**return** 'Halo Dunia';

}

}

Jalankan aplikasi lumen dan lalu buka http://localhost:8000/halo-dunia dan lihat hasilnya. Pada tahap ini kita sudah berhasil membuat halaman pada lumen dengan mamanfaatkan *route* dan *controller*.

Ada tip menarik yang bisa kita terapkan pada lumen atau laravel. Ketika kita membuat *controller* dengan

satu *method* atau *function* maka kita cukup menggunakan *magic function* bawaan **PHP** yaitu .

invoke()

Lakukan perubahan pada *file* routes/web.php seperti berikut ini:

**$router**->get('halo-dunia', 'HaloController');

dan lakukan juga perubahan pada *file controller* app/Http/Controllers/HaloController.php menajdi seperti berikut ini:

**namespace A**pp\**H**ttp\**C**ontrollers;

**class H**aloController

{

**public function** invoke()

{

**return** 'Halo Dunia';

}

}

Kita coba akses kembali, buka http://localhost:8000/halo-dunia dan lihat hasilnya. Contoh kode pada *route* dan

*controller* lebih ringkas dan mudah untuk kita baca.

### Request

*HTTP Request* digunakan untuk mengambil *field* input atau string apapun dalam bentuk metode GET dan

POST .

Caranya sangat mudah untuk menerapkan pada Lumen. Cukup kita mendeklarasikan Illuminate\Http\Request

ke dalam *controller*.

Contohnya seperti kode di bawah ini :

**namespace A**pp\**H**ttp\**C**ontrollers; **use I**lluminate\**H**ttp\**R**equest;

**class U**serController

{

/\*\*

* Store a new user.

\*

* ***@param Request*** $request
* ***@return*** Response

\*/

**public function** store(**R**equest **$request**)

{

**$name** = **$request**->input('name');

//

}

}

Kode di atas terlihat deklarasi use Illuminate\Http\Request , lalu pada *method* store() terlihat class Request dipanggil. Dengan kata lain kita sudah bisa mengambil apa yang dikirim oleh *user*. Pada contoh tersebut kita hanya mengambil name yang dikirim oleh *user*.

Seperti yang terdapat pada Laravel, Lumen juga mendukung *method-method* berikut ini:

# Mengambil semua data yang dikirim oleh pengguna

**$input** = **$request**->all();

# Mengambil hanya data `username` dan `password` yang dikirim oleh pengguna

**$input** = **$request**->only(['username', 'password']);

**$input** = **$request**->only('username', 'password');

# Mengambil kecuali data `credit\_cart` yang dikirim oleh pengguna

**$input** = **$request**->except(['credit\_card']);

**$input** = **$request**->except('credit\_card');

### Response

Laravel menyediakan beberapa cara berbeda untuk mengembalikan respon. Responnya bisa dikirim baik dari *route* maupun *controller*. Semua *route* dan *controller* harus mengembalikan tanggapan untuk dikirim kembali ke pada *user*.

##### String & Array

Respons paling dasar adalah mengembalikan *string* dari *route* dan *controller*. Ini secara otomatis akan mengkon- versi *string* ke *Http* respon.

**$router**->get('halo', **function**() {

**return** 'Halo semua';

});

Contoh kode di atas tambahkan pada routes/web.php dan ketika kita akses http://localhost:8000/halo akan mengembalikan respon Halo semua .

##### JSON

Selain mengembalikan *string* dari *route* dan *controller*, kita juga dapat mengembalikannya ke *array*. Secara otomatis lumen akan mengubah *array* menjadi *response json*.

**$router**->get('halo', **function**() {

**return** response()->json([ 'first\_name' => 'Yudi', 'last\_name' => 'Purwanto'

]);

});

### Eloquent

#### Pengantar

Eloquent adalah **ORM** (*Object Relational Model*) yang bisa kita gunakan di dalam Lumen. *Eloquent* sendiri mengimplementasikan *active record* dan digunakan untuk berinteraksi dengan *database*.

**Penamaan Tabel** Konvensi menggunakan *plural* “snake\_case” untuk nama tabel dan *singular* “StudlyCase” untuk nama model. Mari kita coba jabarkan, sebagai contoh: - Tabel *cats* akan memiliki nama model *Cat* - Tabel *jungle\_cats* akan memiliki nama model *JungleCat* - Tabel *users* akan memiliki nama model User - Tabel *people* akan memiliki nama model *Person*

*Eloquent* secara otomatis akan mencoba membuat model kita dengan tabel yang memiliki bentuk jamak dari nama model, seperti yang dijelaskan di atas.

Namun, kita bisa menentukan nama tabel untuk mengganti konvensi *default* tersebut.

**class U**ser **extends M**odel

{

**protected $table** = 'customers';

}

Pada contoh kode diatas menunjukkan bahwa model *User* akan menghubungkan ke pada table *customers*, penamaan ini bisa kita lakukan dengan memanfaatkan definisi variabel protected $table = 'nama\_tabel' .

***Insert* Data** Selain membaca data dengan *Eloquent*, Kita juga dapat menggunakannya untuk memasukkan atau memperbarui data dengan *method* save() .

Dalam contoh ini kita akan mencoba membuat data *User*:

**$user** = **new U**ser();

**$user**->first\_name = 'Yudi';

**$user**->last\_name = 'Baim';

**$user**->email = 'yudi.baim@example.com';

**$user**->password = bcrypt('password');

**$user**->save();

Kita juga bisa menggunakan *method* create() .

**U**ser::create([ 'first\_name'=> 'John', 'last\_name' => 'Doe',

'email' => 'john.doe@example.com', 'password' => bcrypt('changeme'),

]);

Namun, perlu diingat sebelum menggunakan *method* create() , kita harus mendeklarasikan $fillable pada model:

**class U**ser **extends M**odel

{

**protected $fillable** = [ 'first\_name', 'last\_name',

'email', 'password',

];

}

Sebagai alternatif lain, jika kita ingin membuat semua atribut dapat ditetapkan secara umum, kita dapat mendeklarasikan $guarded pada Model. Seperti kode di bawah ini:

**class U**ser **extends M**odel

{

/\*\*

* The attributes that aren't mass assignable.

\*

* ***@var*** *array*

\*/

**protected $guarded** = [];

}

***Select* Data** Berikut ini adalah contoh menampilkan semua data pada tabel users dengan menggunakan

*method* all() :

**$users** = **U**ser::all();

dd(**$users**);

Untuk menampilkan data berdasarkan id kita bisa menggunakan *method* find() atau findOrFail()

**$user** = **U**ser::find(1);

//or

**$user** = **U**ser::findOrFail(2);

dd(**$user**);

***Update* Data** Berikut ini adalah contoh memperbarui *data* dengan menggunakan *method* find() :

**$user** = **U**ser::find(1);

**$user**->password = bcrypt('my\_new\_password');

**$user**->save();

Atau seperti ini

**$user**->update([

'password' => bcrypt('my\_new\_password'),

])->where('id', 1);

**Delete Data** Kita dapat menghapus data dengan menggunakan *method* delete() .

**$user** = **U**ser::find(12);

**$user**->delete();

Alternatif lainnya, kita juga bisa menggunakan *method* destroy() seperti berikut ini.

**U**ser::destroy(1); **U**ser::destroy([1, 2, 3]);

***Soft Delete*** Terkadang Kita tidak ingin menghapus data secara permanen, tetapi menyimpannya untuk tujuan audit atau tujuan *reporting*. *Eloquent* menyediakan fungsionalitas penghapusan data secara tidak permanen.

Untuk menambahkan fungsionalitas penghapusan ini ke model, kita perlu *import* class SoftDeletes dan menambahkannya ke *model*.

Seperti berikut ini.

**namespace I**lluminate\**D**atabase\**E**loquent\**M**odel; **namespace I**lluminate\**D**atabase\**E**loquent\**S**oftDeletes; **class U**ser **extends M**odel

{

**use S**oftDeletes;

}

Namun, sebelum menggunakan fungsionalitas ini, kita perlu pastikan untuk membuat kolom deleted\_at di tabel terlebih dahulu. Atau dalam migrasi kita harus memanggil *method* softDeletes() .

Contoh:

**S**chema::table('users', **function** (**$table**) {

**$table**->softDeletes();

});

Jika tabel users sudah terbuat, dan kita ingin menambahkan funsionalitas penghapusan ini. Kita bisa membuat satu *file migration* dengan perintah ini php make:migration add\_deleted\_at\_to\_users . Perintah tersebut akan menghasilkan *file* migration/\*add\_deleted\_at\_to\_users.php .

Lalu, kita bisa mengubah *file* tersebut seperti kode di bawah ini.

use **I**lluminate\**D**atabase\**M**igrations\**M**igration; use **I**lluminate\**D**atabase\**S**chema\**B**lueprint; use **I**lluminate\**S**upport\**F**acades\**S**chema;

**class A**ddDeletedAtToUsers **extends M**igration

{

/\*\*

* Run the migrations.

\*

* ***@returnvoid***

\*/

**public function** up()

{

**S**chema::table('users', function(**B**lueprint **$table**) {

**$table**->softDeletes();

});

}

/\*\*

* Reverse the migrations.

\*

* ***@returnvoid***

\*/

**public function** down()

{

**S**chema::table('users', function(**B**lueprint **$table**) {

**$table**->dropColumn(['deleted\_at']);

});

}

}

Setiap *query* akan menghilangkan data yang dihapus secara sementara. Kita dapat melihat data yang telah terhapus tersebut dengan menggunakan *method* withTrashed() :

**U**ser::withTrashed()->get();

Kita juga bisa mengembalikan data yang sudah terhapus, menggunakan *method* restore .

**$user** = **U**ser::find(1);

**$user**->delete();

**$user**->restore();

**Merubah *primary key* & *timestamps*** Secara *default*, *primary key* pada *Model Eloquent* diberi nama id , namun kita bisa merubah primary key tersebut dengan mendefinisikan variabel $primaryKey .

**class U**ser **extends M**odel

{

**protected $primaryKey** = 'id\_user';

// ...

}

Selain itu, *primary key* pada *Model Eloquent* integer dan *auto-increment*. Jika *primery key* kita bukan integer (mis. GUID), kita perlu mendefinisikan pada *Model Eloquent* dengan variabel $incrementing diberi nilai

false :

**class U**ser **extends M**odel

{

**protected $primaryKey** = 'uid\_user';

**public $incrementing** = **false**;

// ...

}

Secara *default*, *Model Eloquent* mengharuskan adanya kolom create\_at dan updated\_at ada di tabel. Jika kita tidak ingin kolom ini secara otomatis dikelola oleh *Eloquent*, kita cukup mendefinisikan variabel $timestamps pada model dengan nilai false :

**class U**ser **extends M**odel

{

**public $timestamps** = **false**;

// ...

}

Namun, jika kita perlu menyesuaikan nama kolom yang digunakan untuk menyimpan timestamps , kita dapat mengatur dengan cara merubah dan mendefinisikan CREATED\_AT dan UPDATED\_AT pada model:

**class U**ser **extends M**odel

{

**const CREATED\_AT** = 'tanggal\_pembuatan';

**const UPDATED\_AT** = 'tanggal\_update';

// ...

}

### Eloquent : Relationship

Dalam memanfaatkan *database*, dikenal ada relasi antar *entity* atau antar tabel. Penggunaan fitur ini pada *database* konvesional membutuhkan penggunaan perintah *SQL* yang relatif panjang dan juga membutuhkan penanganan dari program yang membutuhkan program yang rumit pula.

Berikut adalah beberapa jenis relasi *database* yang dikenal pada umumnya dan telah diakomodasi oleh *Elo- quent*.

1. Relasi *one to one* dimana sebuah data pada sebuah tabel hanya memiliki relasi ke sebuah data pada tabel yang lain. Misalnya, sebuah data tabel users memiliki relasi 1 nomor telepon di tabel contacts .
2. Relasi *one to many* dimana sebuah data pada sebuah tabel memiliki relasi ke beberapa data pada tabel yang lain. Misalnya, sebuah data tabel categories memiliki relasi banyak data di tabel transactions . Atau dengan kata lain, 1 kategori memiliki banyak data.
3. Relasi *many to one (One to many Inverse)* dimana merupakan kebalikan dari relasi one to many. Misalnya kita ingin mengetahui data di table transactions memiliki kategori apa, maka relasi ini yang akan digunakan.
4. Relasi *many to many* dimana banyak data pada sebuah tabel memiliki relasi ke banyak data juga pada tabel yang lainnya.

##### One to One

Relasi *One to One* adalah relasi yang mana setiap satu baris data pada tabel pertama hanya berhubungan dengan satu baris pada tabel kedua. Agar tidak bingung, mari kita coba terapkan pada *eloquent*.

**class U**ser **extends M**odel

{

/\*\*

Get the phone record assosiated with the user.

\*/

**public function** phone()

{

**return $this**->hasOne('App\Phone');

}

}

Pada contoh kode di atas, *model User* berelasi dengan *model Phone*, secara *default primary key* pada *eloquent* berupa id dan *foreign\_key* berupa user\_id . Yang mana *foreign key* itulah yang digunakan sebagai penghubung antara kedua tabel tersebut.

Definisi *Relationship* sebaliknya. Untuk mendapatkan siapa *user* yang memiliki sebuah *phone*, perlu didefinisikan

*relationship* sebaliknya (*Inverse Relationships*), dengan menggunakan *method* belongsTo

**class P**hone **extends M**odel

{

/\*\*

Get the user that owns the phone.

\*/

**public function** user()

{

**return $this**->belongsTo('App\User');

}

}

##### One to Many

Relasi *One to Many* adalah relasi yang mana setiap satu baris data pada tabel pertama berhubungan dengan lebih dari satu baris pada tabel kedua. Agar tidak bingung, mari kita coba terapkan pada *eloquent*.

**class P**ost **extends M**odel

{

/\*\*

Get the comments for the blog post.

\*/

**public function** comments()

{

**return $this**->hasMany('App\Comment');

}

}

Pada contoh kode di atas, sebuah *post* akan memiliki banyak *comments*. *Foreign key* pada contoh ini: post\_id . Sesuai dengan *snake case* nama model diikuti \_id .

##### Many to Many

Relasi *Many to Many* adalah relasi yang mana setiap lebih dari satu baris data dari tabel pertama berhubungan dengan lebih dari satu baris data pada tabel kedua. Artinya, kedua tabel masing-masing dapat mengakses banyak data dari tabel yang direlasikan. Dalam hal ini, relasi *Many to Many* akan menghasilkan tabel ketiga sebagai perantara tabel kesatu dan tabel kedua sebagai tempat untuk menyimpan *foreign key* dari masing-masing tabel. Agar tidak bingung, mari kita coba terapkan pada *eloquent*.

**class U**ser **extends M**odel

{

/\*\*

The roles that belong to the user.

\*/

**public function** roles()

{

**return $this**->belongsToMany('App\Role');

}

}

* Relasi ini juga banyak digunakan. Penggunaannya sedikit lebih kompleks dibanding 2 tipe relasi sebelum- nya.
* Contoh berikut adalah di mana terdapat users , roles dan role\_user . Seorang *user* dapat memiliki banyak role dan sebuah role dapat dimiliki banyak *user*.
* role\_user merupakan *default* tabel perantara (*intermediate*) atau *pivot*. Nama tabel adalah dari kedua tabel role dan *user*
* Relasi didefinisikan dengan menggunakan *method* belongsToMany .

Penting, Jika kita ingin menggunakan *Eloquent*, kita harus menghapus komentar panggilan

$app->withEloquent() pada *file* bootstrap/app.php .

## Kesimpulan

Pada tahap ini, kita sudah berkenalan dengan lumen beserta fitur-fitur dasar yang ada pada Lumen itu sendiri. Seharusnya kita sudah bisa menjadikan pemahaman dasar ini untuk siap melanjutkan ke tahap selanjutnya.

Penulis tekankan kembali bahwa penjelasan di atas hanya mencakup dasar-dasar dan penggunaan secara umum dari Lumen. Selebihnya bisa membaca di halaman resmi. <https://lumen.laravel.com/docs>

# Instalasi

## Kebutuhan Server

Sebelum memulai instalasi, lumen membutuhkan beberapa *extension php* yang harus kita instal, pastikan kembali

*extension* berikut ini sudah kita instal:

* PHP >= 7.3
* OpenSSL PHP Extension
* PDO PHP Extension
* Mbstring PHP Extension

## Intalasi Lumen

Perlu kita ketahui bahwa untuk menginstal Lumen terdapat dua cara, yaitu cara pertama menggunakan Lumen

*Installer* dan cara kedua menggunakan *Composer Create-Project*.

1. Melalui Lumen *Installer*

composer global require "laravel/lumen-installer"

Pastikan untuk mendaftarkan atau mendefinisikan direktori ~/.composer/vendor/bin di **PATH**. sehingga,

 lumen dapat dieksekusi dan dapat ditemukan oleh sistem.

Setelah terinstal, perintah lumen akan membuat penginstalan Lumen baru di direktori yang ditentukan. Misalnya kita akan membuat proyek baru bernama todo , lumen akan membuat direktori bernama todo yang berisi kerangka Lumen secara *default* dengan semua dependensi Lumen yang sudah terinstal. Metode penginstalan ini jauh lebih cepat daripada menginstal melalui composer .

Untuk membuat proyek baru dengan lumen cukup jalankan perintah berikut:

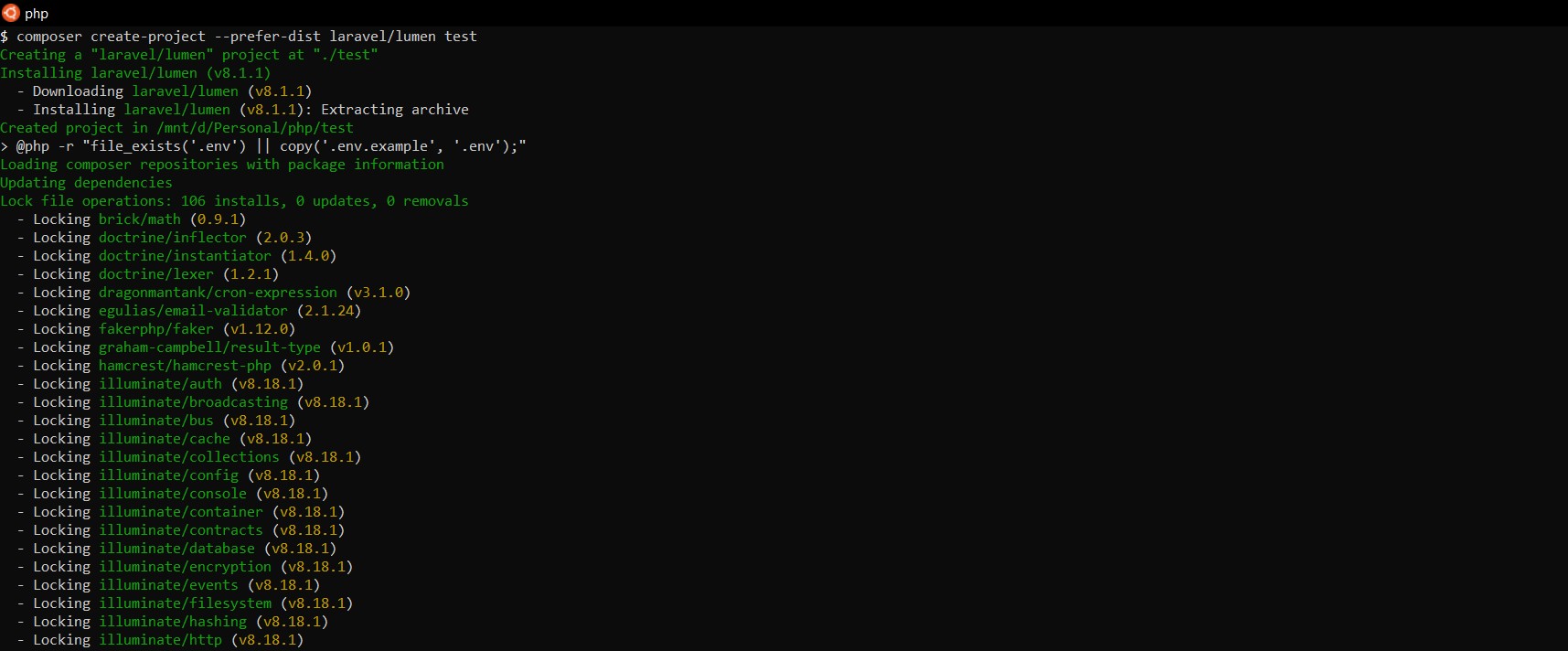
lumen new todo

1. Melalui *Composer Create-Project*

composer create-project --prefer-dist laravel/lumen todo

Setelah kita jalankan perintah di atas, maka secara otomatis akan melakukan *download repository*.

Catatan, lama tidaknya proses instalasi ini tergantung koneksi *internet* kita.

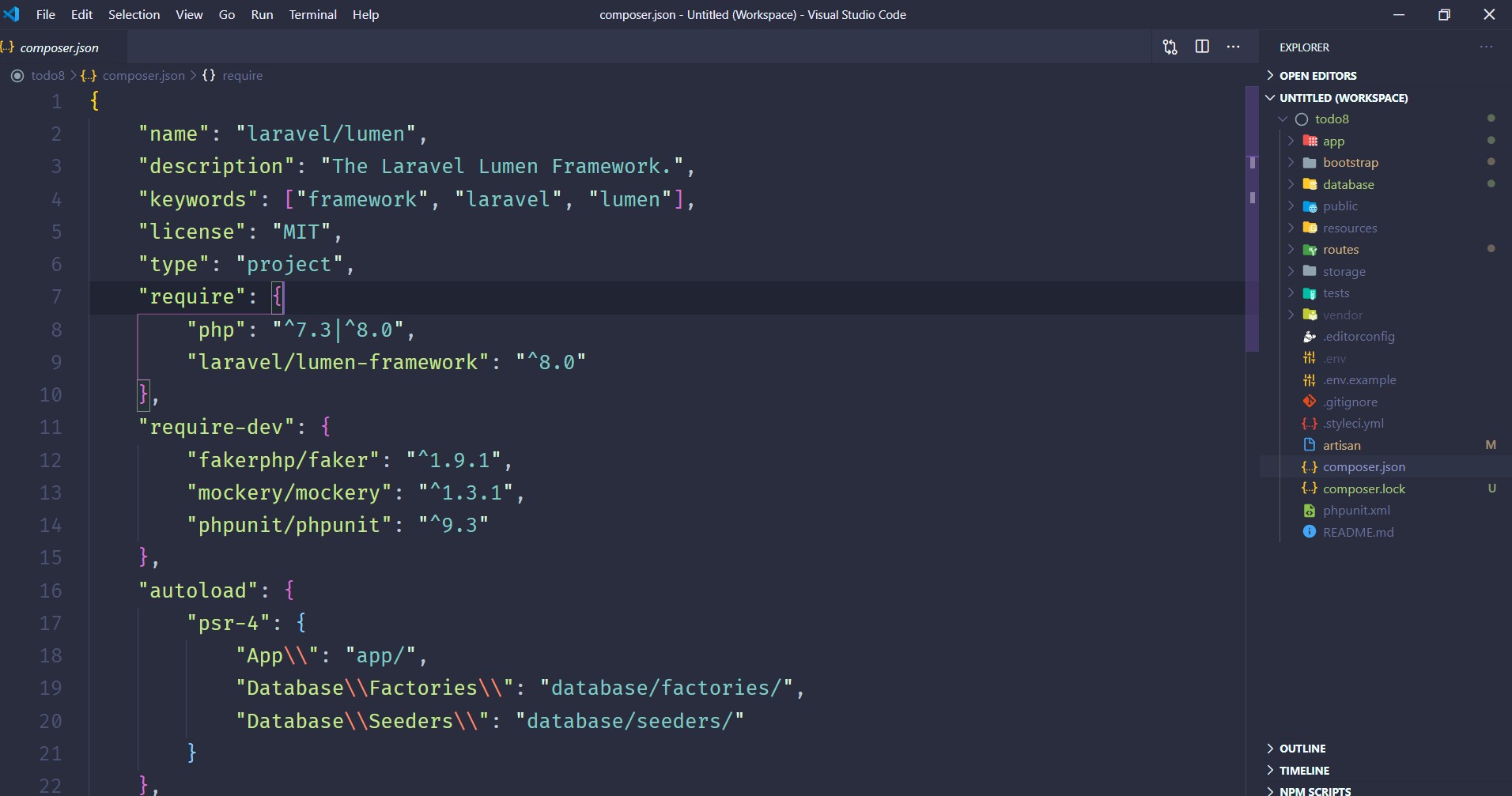


**Figure 15:** Proses download lumen

Gambar di atas menunjukkan bahwa pada Terminal/CMD kita sedang *download repository* lumen.

Setelah selesai maka terdapat *folder* dengan nama *folder* yang kita tuliskan ketika menginstal Lumen, contoh di atas nama projek kita adalah todo .

## Struktur pada Lumen



**Figure 16:** Struktur Pada Lumen

Gambar di atas merupakan direktori projek lumen yang di instal, untuk melihat versi lumen nya bisa dilihat pada

*file* composer.json pada *field laravel/lumen-framework*. Saat ini kita menginstal lumen yang terbaru yaitu

#### ^8.0.

Kita melihat banyak folder di sini, tetapi hanya perlu memperhatikan *folder-folder* berikut.

app/Models Di sini untuk mendefinisikan model yang melakukan operasi *database*.

app/http/controllers: Di sini untuk mendefinisikan logika aplikasi kita.

app/http/middlewares: Di sini untuk mendefinisikan keamanan *API* di sini.

database/migrations: Di sini untuk menentukan skema *database* di sini.

routes: Di sini untuk menentukan tujuan akhir ( end points ) untuk *API*.

## Menjalankan Lumen

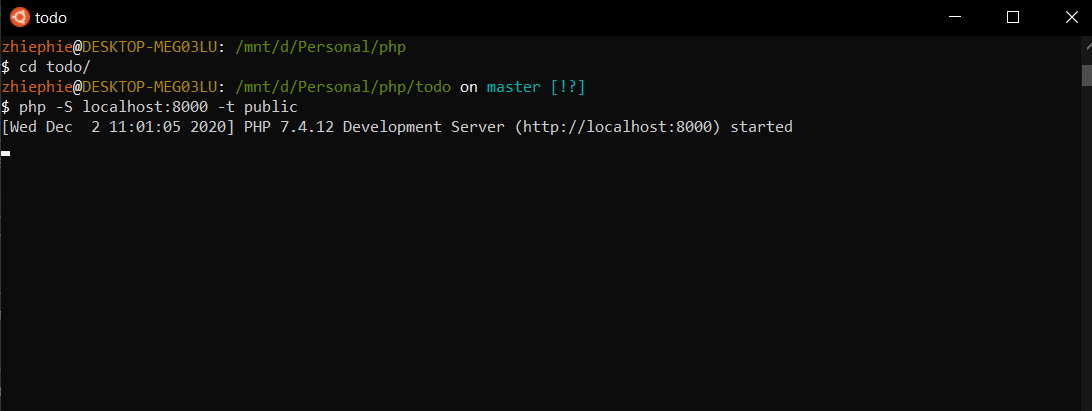
php sendiri sudah bisa menjalankan aplikasi secara *standalone*. Kita akan memanfaatkan server bawaan tersebut. Sehingga dalam proses development kita tidak perlu menggunakan web-server seperti apache, nginx dan web-server sejenisnya.

Untuk menjalankan proyek lumen, pastikan sudah berada didalam direktori aplikasi, selanjutnya jalankan perintah berikut ini:

php -S localhost:8000 -t public

Perintah di atas terdapat angka 8000, artinya lumen berjalan pada *port* 8000. *Port* tersebut juga dapat di ganti, misalnya ingin di ganti 8001.

Berikut hasil dari perintah diatas:



**Figure 17:** Pertama Menjalankan Lumen

Jika muncul seperti pada gambar di atas, maka kita tinggal akses melalui browser pada *url* berikut:

http://localhost:8000





**Figure 18:** Proses Menjalankan Lumen Sukses

## Kesimpulan

Pada tahan ini, kita sudah mempelajari cara instal lumen secara benar dan berjalan sesuai harapan. Pada tahan selanjutkan kita akan membahas konfigurasi pada Lumen.

# Konfigurasi

**Konfigurasi *Database***

Kita akan belajar bagaimana cara menghubungkan proyek Lumen kita dengan *database*. Sekarang silahkan buka proyek Lumen kita dengan *text editor*, kemudian *copy/rename file* .env.example menjadi .env kemudian buka *file* .env dan cari kode berikut ini:

DB\_CONNECTION=mysql DB\_HOST=127.0.0.1 DB\_PORT=3306

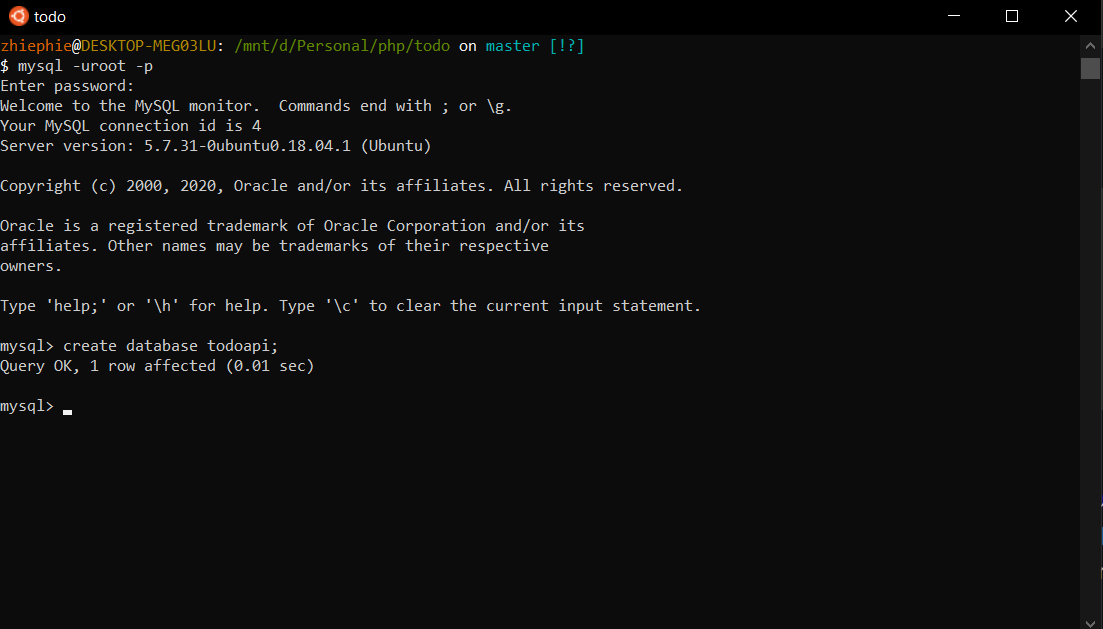
DB\_DATABASE=homestead DB\_USERNAME=homestead DB\_PASSWORD=secret

Kemudian, silahkan rubah dan sesuaikan koneksi *database* berikut ini:

DB\_CONNECTION=mysql DB\_HOST=127.0.0.1 DB\_PORT=3306

DB\_DATABASE=todoapi DB\_USERNAME=homestead DB\_PASSWORD=secret

Di atas, kita akan menggunakan nama *database* todoapi dan untuk *password* silahkan disesuaikan dengan koneksi dari **MySQL** masing-masing, jika menggunakan **XAMPP**, maka secara *default password* adalah kosong atau tidak perlu mengisi-nya. Sekarang, kita lanjutkan membuat *database* bisa menggunakan *terminal*, *phpmyad- min* atau *client gui* seperti *navicat*, kemudian buat *database* baru dengan nama todoapi . Kurang lebih seperti berikut ini:



**Figure 19:** Membuat Database

**Registrasi *Facedes, Eloquent & Authentication***

Pada *file* bootstrap/app.php kita harus menghapus komentar pada bagian-bagian berikut ini:

// $app->withFacades();

// $app->withEloquent();

// $app->routeMiddleware([

// 'auth'=> App\Http\Middleware\Authenticate::class,

// ]);

// $app->register(App\Providers\AppServiceProvider::class);

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

Menjadi seperti berikut ini:

**$app**->withFacades();

**$app**->withEloquent();

**$app**->routeMiddleware([

'auth'=> **A**pp\**H**ttp\**M**iddleware\**A**uthenticate::**class**,

]);

**$app**->register(**A**pp\**P**roviders\**A**ppServiceProvider::**class**);

**$app**->register(**A**pp\**P**roviders\**A**uthServiceProvider::**class**);

Pasti ada yang bertanya-tanya mengenai baris kode di atas itu berguna untuk apa, baris kode di atas untuk mengaktifkan fitur Facede , Eloquent , authentication dan middleware .

***Generate* APP\_KEY**

Di Laravel memang ada fitur untuk *generate* secara otomatis dengan perintah php artisan key:generate , namun berbeda dengan Lumen, Lumen sendiri tidak dibekali dengan fitur tersebut. Oleh karena itu, penulis memberikan cara mudah dengan cara berikut ini. Salin kode di bawah ini ke *file* routes/web.php .

use **I**lluminate\**S**upport\**S**tr;

**$router**->get('/key', function() {

**return S**tr::random(32);

});

Jalankan perintah berikut ini:

php -S localhost:8000 -t public

Kemudian, akses http://localhost:8000/key , nanti akan menghasilkan nilai seperti 0qu89f2zHQVdDhJuSLe4acxfH5ASg2vr

Setelah itu, buka *file* .env rubah bagian APP\_KEY= dengan *key* yang dihasilkan oleh fungsi di atas menjadi seperti APP\_KEY=0qu89f2zHQVdDhJuSLe4acxfH5ASg2vr

## Kesimpulan

Pada tahap ini kita sudah belajar konfigurasi pada Lumen. Pada tahap selanjutkan kita akan melajutkan ke tahap bejalar *CRUD (create, read, update & delete)*

# Studi Kasus - TodoManager

Pada tahan ini menjelaskan secara rinci tentang penerapan Operasi **CRUD** pada Lumen. *Create, Read, Update, & Delete* adalah operasi paling dasar dan penting yang harus dimiliki setiap aplikasi. Membuat Operasi **CRUD** dasar di lumen adalah hal yang sangat sederhana.

Untuk tahap ini kita akan membuat *TodoManager*, di mana kita dapat membuat data baru, menampilkannya sebagai daftar di halaman depan dan mengubah atau menghapusnya.

**Membuat Model dan *Migration***

Di tahap sebelumnya kita telah belajar bagaimana cara menghubungkan proyek Lumen yang kita buat dengan *database*. Dan kita juga sudah belajar membuat *database*, maka sekarang kita akan lanjutkan untuk membuat model dan juga *migration*.

**Membuat Model dan *Migration Todo***

Mari kita jalankan perintah berikut ini melalu terminal atau CMD :

php artisan make:migration create\_todos\_table --create=todos

Perintah di atas akan membuat *migration* untuk tabel todos .

Jika perintah di atas berhasil, kita akan mendapati 1 *file* baru yang ada pada direktori database/migrations/\*\_create\_todos\_table.

, oke mari kita buka *file* tersebut dan merubah seperti kode di bawah ini:

**public function** up()

{

**S**chema::create('todos', function(**B**lueprint **$table**) {

**$table**->id();

**$table**->**string**('name');

**$table**->**string**('description')->nullable();

**$table**->timestamps();

});

}

/\*\*

* Reverse the migrations.

\*

* ***@returnvoid***

\*/

**public function** down()

{

**S**chema::dropIfExists('todos');

}

Selanjutnya kita jalankan perintah berikut ini:

php artisan migrate

Jika perintah di atas berhasil, maka akan muncul pesan seperti berikut ini.

Migration table created successfully.

Migrating: 2020\_12\_03\_092145\_create\_todos\_table

Migrated: 2020\_12\_03\_092145\_create\_todos\_table (247.49ms)

Sekarang kita akan membuat model Todo pada direktori app/Models/Todo.php , dan rubah sesuai kode di bawah ini:

namespace **A**pp\**M**odels;

use **I**lluminate\**D**atabase\**E**loquent\**M**odel; **class T**odo **extends M**odel

{

/\*\*

* The attributes that are mass assignable.

\*

* ***@vararray***

\*/

**protected $fillable** = [ 'name', 'description'

];

}

**Membuat *URL* atau *API Endpoint***

Tambahkan kode di bawah ini, ke dalam file routes/web.php

// API route group prefix /api

**$router**->group(['prefix' => 'api'], function() **use** (**$router**) {

/\*

* Matches
* /api/todo (post, get method)
* /api/todo/id (get, put, delete method)

\*/

**$router**->post('todo','TodoController@store');

**$router**->get('todo', 'TodoController@index');

**$router**->get('todo/{id}', 'TodoController@show');

**$router**->put('todo/{id}', 'TodoController@update');

**$router**->delete('todo/{id}', 'TodoController@destroy');

});

Kita akan memberikan *prefix* atau awalan pada setiap *url* berupa api di semua *route*, untuk mengurangi pengulangan. Kita bisa menggunakan *method* $router->group untuk melakukan hal itu.

**Membuat *Controller***

#### Membuat Controller TodoManager

Di sini kita akan membuat controller dengan nama TodoController yang digunakan untuk manajemen data pada tabel todos .

Mari kita buat *file* app/Http/Controllers/TodoController.php dan sesuaikan kodenya seperti di bawah ini:

namespace **A**pp\**H**ttp\**C**ontrollers;

use **I**lluminate\**H**ttp\{**R**equest, **J**sonResponse}; use **A**pp\**M**odels\**T**odo;

**class T**odoController **extends C**ontroller

{

**protected $todo**;

**public function** construct(**T**odo **$todo**)

{

**$this**->todo = **$todo**;

}

}

**Membuat *Create* Todo *API*** Di tahap sebelumnya kita sudah membahas bagaimana membuat *controller*, sekarang kita coba pelajari bagaimana membuat sebuah *method create* data pada tabel todos .

Kita buat function store() , yang isinya seperti kode di bawah ini:

...

**public function** store(**R**equest **$request**): **J**sonResponse

{

// validate incoming request

**$data** = **$this**->validate(**$request**, [ 'name' => 'required|max:100', 'description' => 'nullable'

]);

try {

**$todo** = **$this**->todo->create(**$data**);

//return successful response

**return** response()->json([ 'status' => **true**,

'message' => 'Data todo berhasil disimpan.', 'data' => **$todo**

], 201);

} catch(\**Exception $e**) {

//return error message

**return** response()->json([

'status' => **false**,

'message' => 'Create data todo gagal'

], 409);

}

}

**Membuat *List* Todo *API*** Di sini, kita akan mencoba membuat atau menampilkan semua data yang ada pada tabel todos , kita hanya perlu membuat function index() , yang mana isinya seperti di bawah ini.

...

**public function** index(): **J**sonResponse

{

**$todos** = **$this**->todo->all();

**return** response()->json( ['data' => **$todos**], 200

);

}

Karena kita sudah mendefinisikan variabel $todo pada *construct* yang kita kembalikan pada model Todo , maka kita tinggal memanggilnya dengan perintah $this->todo->method(); . Untuk penggunaan JsonResponse ini optional, kita bisa menghapus atau memberikan spesifikasi balikan yang akan kita inginkan.

**Membuat *Get* Todo *API*** Di tahap sebelumnya kita sudah membahas bagaimana menampilkan semua data pada tabel todos , sekarang kita coba pelajari bagaimana menampilkan satu data todo .

Kita buat function show() , yang isinya seperti kode di bawah ini:

...

**public function** show(**int $id**): **J**sonResponse

{

**$todo** = **$this**->todo->findOrFail(**$id**);

**return** response()->json(['data' => **$todo**], 200);

}

Tidak jauh beda fungsi pada *method show* di atas, yang membedakan hanya pada bagian findOrFail($id) , yang mana *method* ini akan menghasilkan *query* berupa where clause . Dengan kata lain, data yang dihasilkan hanya satu.

**Membuat *Update* Todo *API*** Di tahap sebelumnya kita sudah membahas bagaimana membuat data baru, sekarang kita coba pelajari bagaimana memperbaharui data pada tabel todos .

Kita buat function update() , yang isinya seperti kode di bawah ini:

...

**public function** update(**R**equest **$request**, int **$id**): **J**sonResponse

{

//validate incoming request

**$data** = **$this**->validate(**$request**, [ 'name' => 'required|max:100', 'description' => 'nullable'

]);

try {

**$todo** = **$this**->todo->findOrFail(**$id**);

**$todo**->fill(**$data**);

**$todo**->save();

//return successful response

**return** response()->json([ 'status' => **true**,

'message' => 'Data todo berhasil diupdate', 'data' => **$todo**

], 201);

} catch(\**Exception $e**) {

//return error message

**return** response()->json([ 'status' => **false**,

'message' => 'Update data todo gagal'

], 409);

}

}

**Membuat *Delete* Todo *API*** Di tahap sebelumnya kita sudah membahas bagaimana memperbaharui data, sekarang kita coba pelajari bagaimana menghapus data pada tabel todos .

Kita buat function destroy() , yang isinya seperti kode di bawah ini:

...

**public function** destroy(**int $id**): **J**sonResponse

{

try {

**$todo** = **$this**->todo->findOrFail(**$id**);

**$todo**->delete();

//return successful response

**return** response()->json([ 'status' => **true**,

'message' => 'Data todo berhasil dihapus', 'user' => **$todo**

], 201);

} catch(\**Exception $e**) {

//return error message

**return** response()->json([ 'status' => **false**,

'message' => 'Hapus data todo gagal'

], 409);

}

}

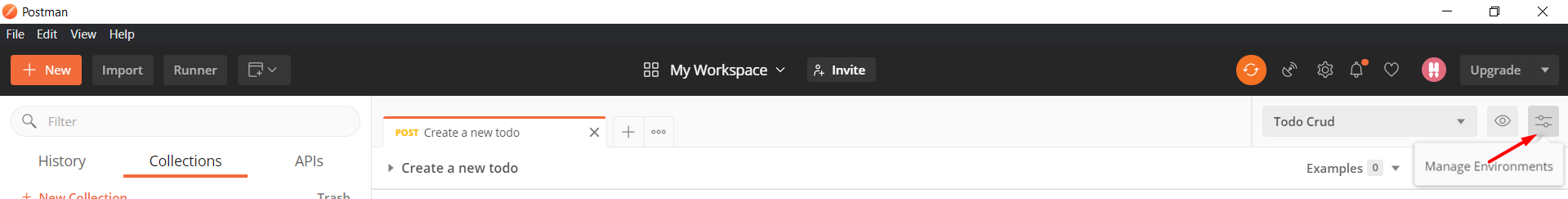
**Menguji dengan *Postman***

Pada tahap ini kita akan membahas tentang pengujian *API* yang sudah kita buat dengan **Postman**.

**Postman** ini merupakan *tool* wajib bagi para *developer* yang berkutat pada pembuatan *API*, fungsi utama *postman* ini adalah sebagai *GUI API Caller* namun sekarang *postman* juga menyediakan fitur lain yaitu *Sharing Collection API for Documentation (free), Testing API (free), Realtime Collaboration Team (paid), Monitoring API (paid), Integration (paid)*.

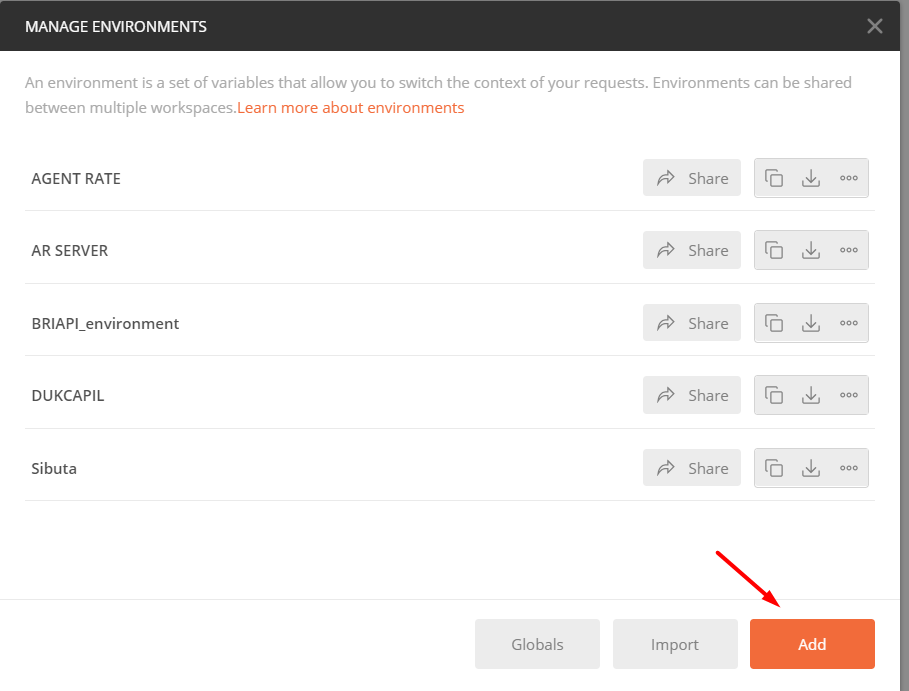
***Setting Environment* dan *Collection*** Pastikan *postman* sudah terhubung dengan *account google* yang kita miliki karena setiap perubahan di *collection*, *postman* akan melakukan *sync* secara berkala. *setting environment* ini akan memudahkan kita dalam membedakan *environment API development* dan *environment API production*. berikut caranya:

1. Klik bagian yang dipanah merah, nantinya akan memuncul sebuah dialog.



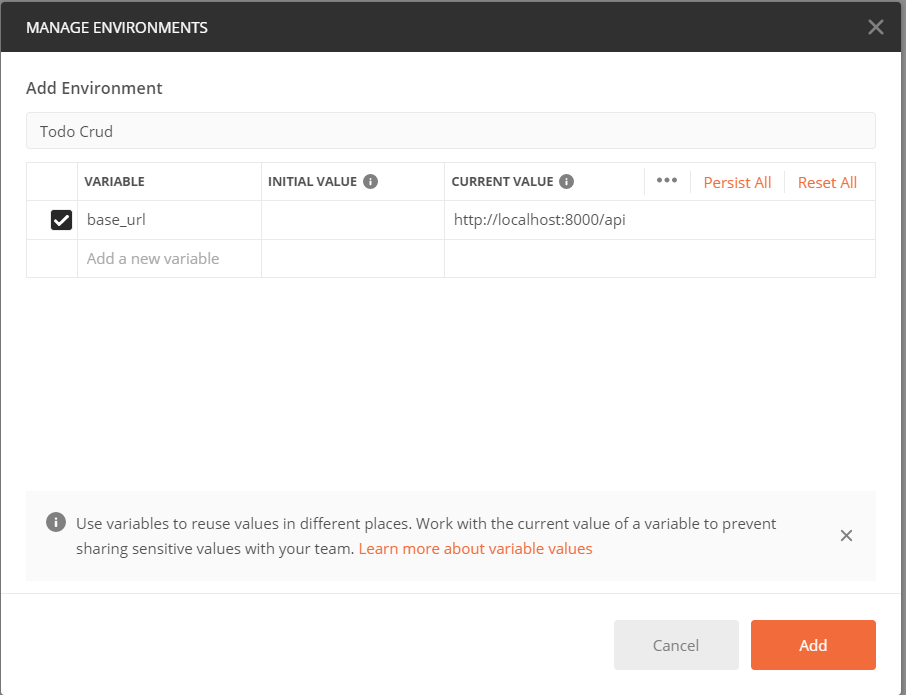
**Figure 20:** Environment pada Postman

1. Klik tombol *Add*.



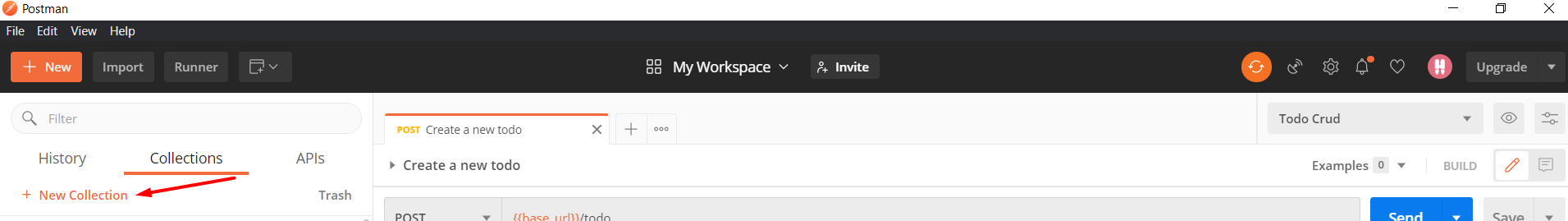
**Figure 21:** add env postman

1. Tambahkan variabel dan *url* seperti pada gambar di bawah ini.



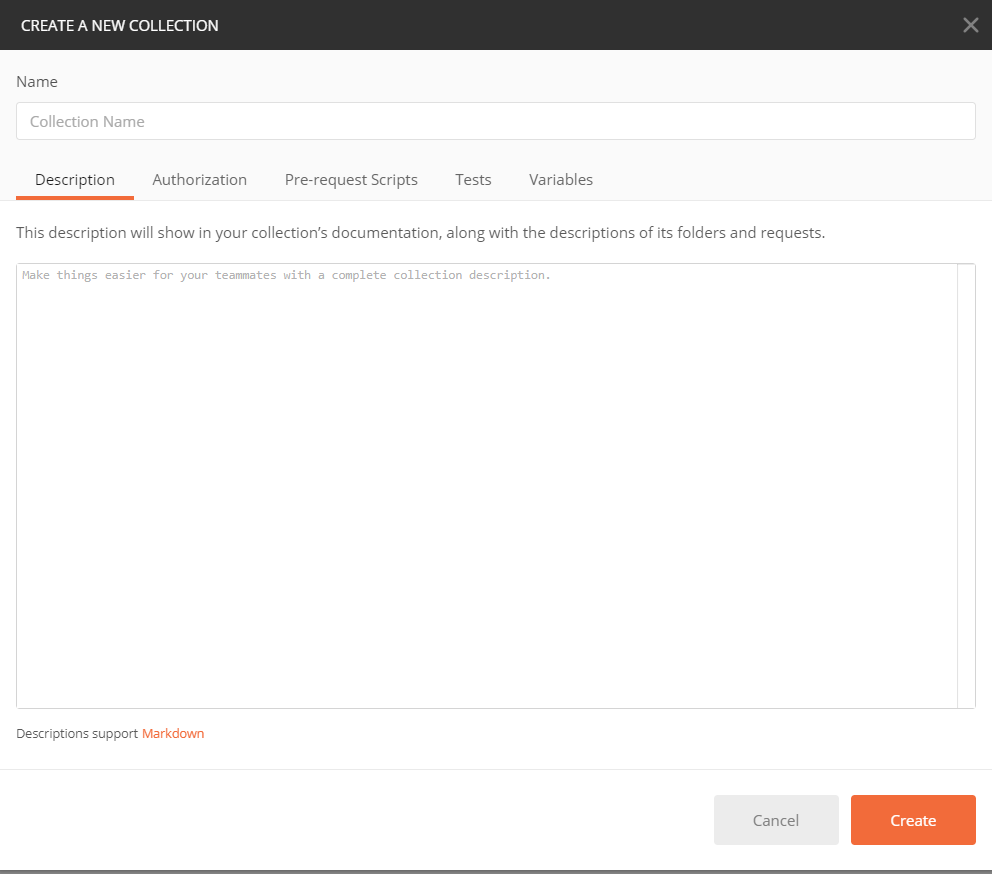
**Figure 22:** add new env postman

1. Klik *New Collecton*



**Figure 23:** new collection postman

1. Isi nama *Collection* dan klik tombol *Create*, kita juga bisa memberikan sebuah deskripsi pada bagian ini.



**Figure 24:** create\_a\_collection\_postman

Baik, konfigurasi *postman* kita sudah berhasil, sekarang kita akan ketahap pengujian *Restful API* untuk memas- tikan fungsi berjalan dengan lancar.

**Menguji *Create* Todo API**

##### Endpoint

Method URL

POST {{base\_url}}/todo

##### HTTP Header

Header Name Required Values

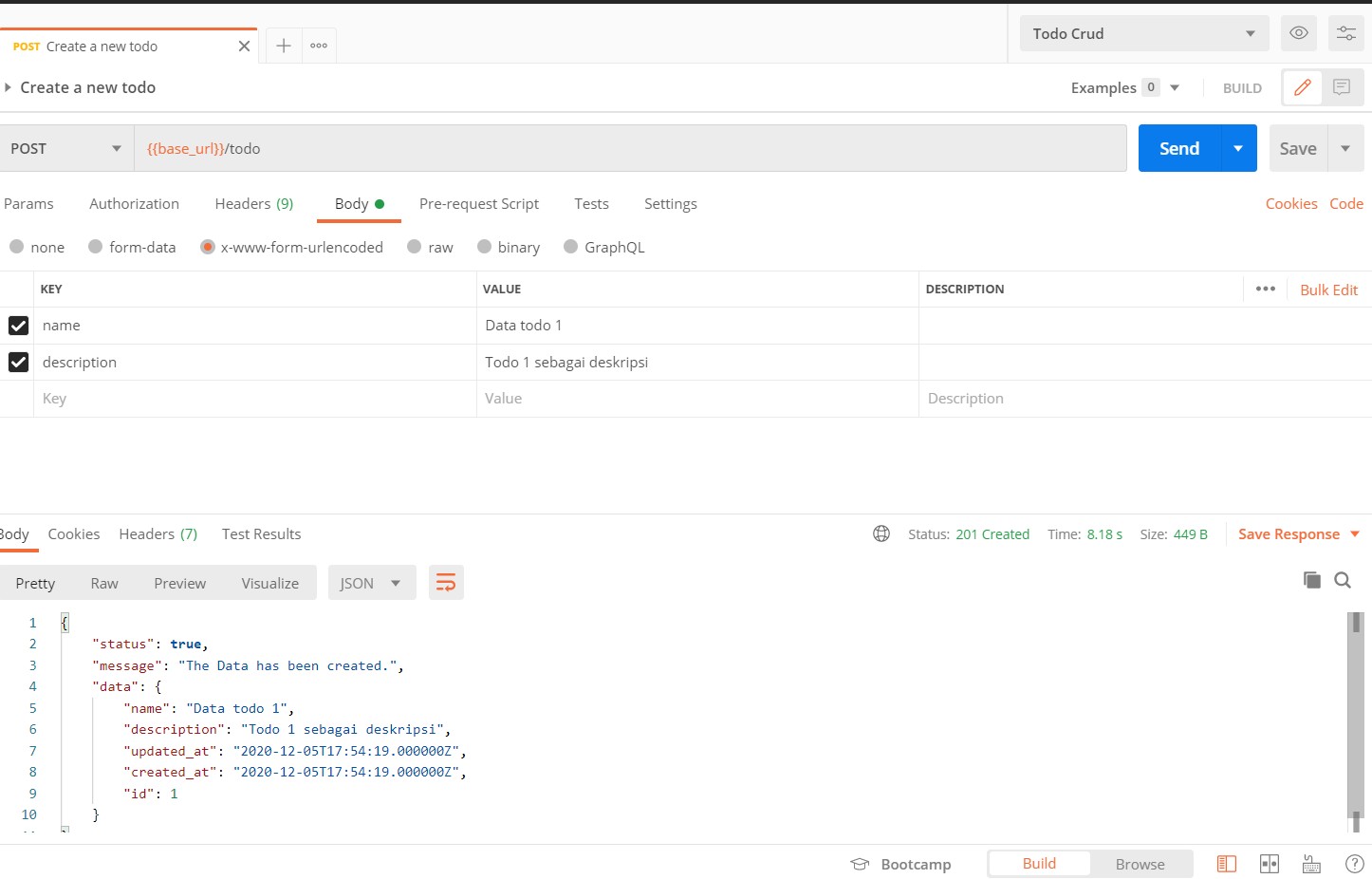
Accept ya application/json Content-type ya application/json

##### HTTP Body (x-www-form-urlencoded)

Parameter Required Description

name ya nama todo description ya keterangan todo

Contoh hasil dari pengujian di atas.



**Figure 25:** create new todo

**Menguji *List* Todo API**

##### Endpoint



Method URL

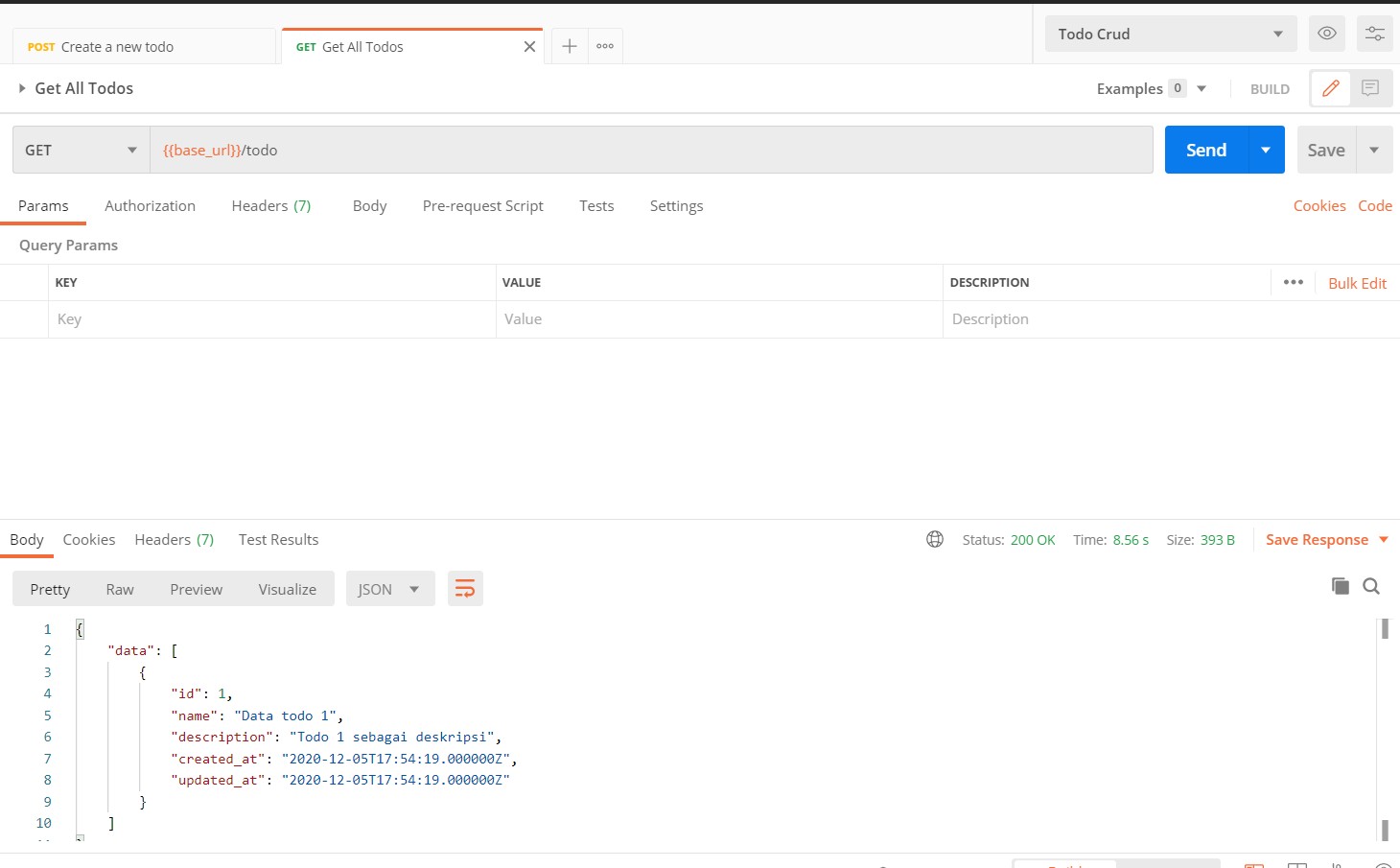
GET {{base\_url}}/todo

##### HTTP Header

Header Name Required Values

Accept ya application/json

Contoh hasil dari pengujian di atas.



**Figure 26:** List All Todo

#### Menguji *Get* Todo API

##### Endpoint

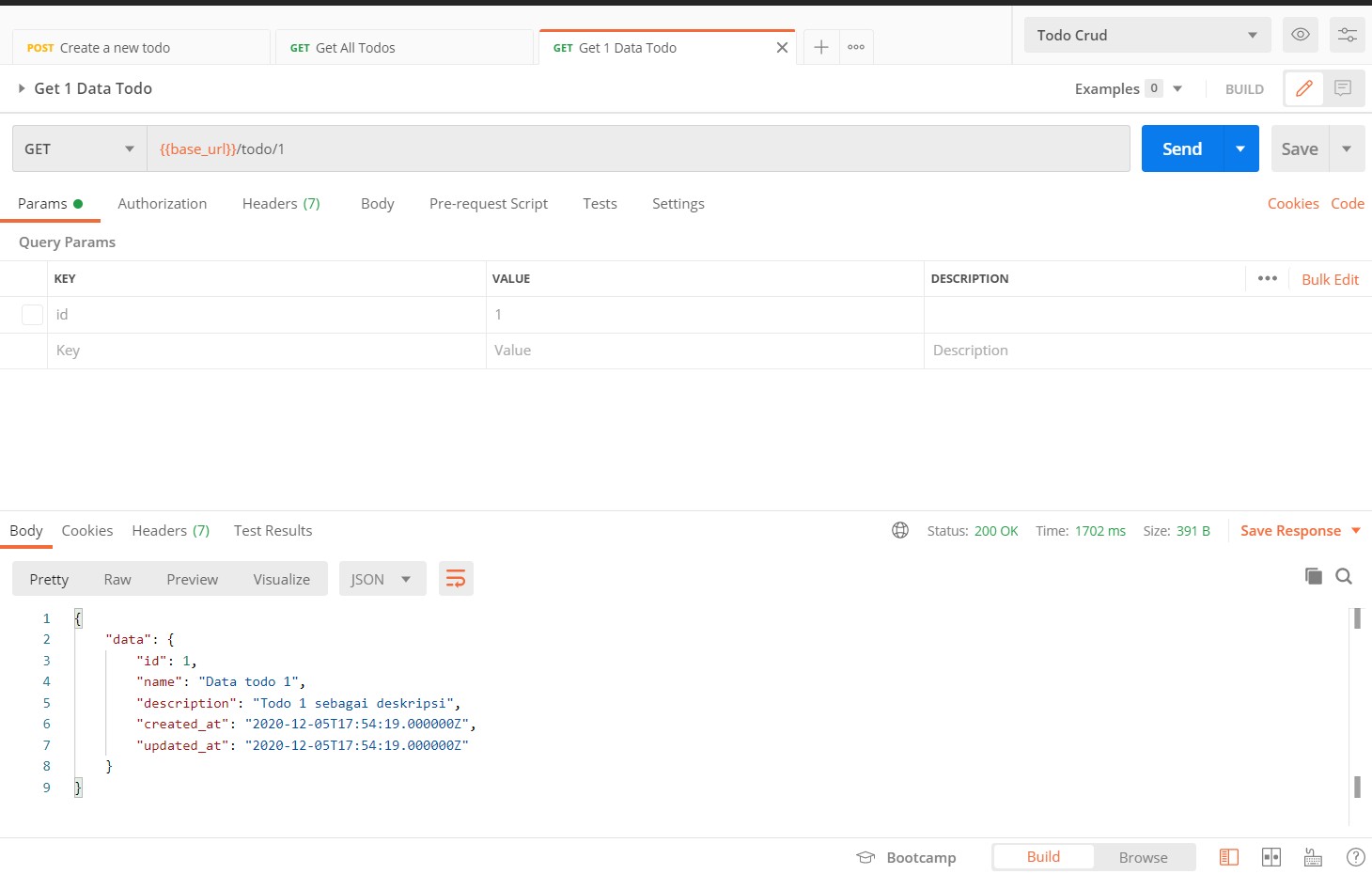
Method URL

GET {{base\_url}}/todo/1

##### HTTP Header

|  |  |  |
| --- | --- | --- |
| Header Name | Required | Values |
| Accept | ya | application/json |

Contoh hasil dari pengujian di atas.



**Figure 27:** Get Todo By ID

**Menguji *Update* Todo API**

##### Endpoint

Method URL

PUT {{base\_url}}/todo/1

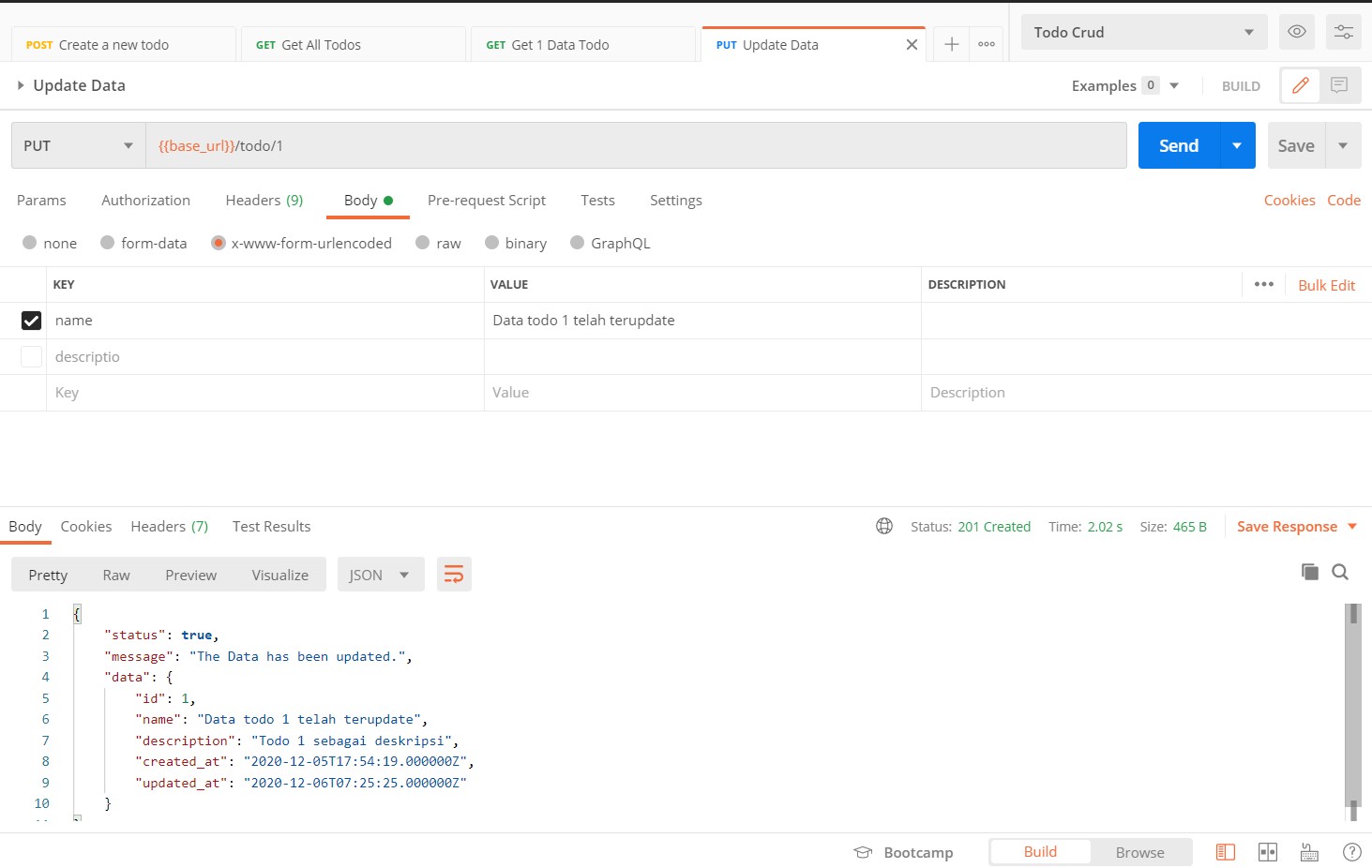
##### HTTP Header

Header Name Required Values

Accept ya application/json Content-type ya application/json

|  |  |  |
| --- | --- | --- |
| ***HTTP Body (x-www-form-urlencoded)*** |  | |
| Parameter | Required | Description |
| name | ya | nama todo |
| description | ya | keterangan todo |

Contoh hasil dari pengujian di atas.



**Figure 28:** Update Todo Data

**Menguji *Delete* Todo API**

##### Endpoint

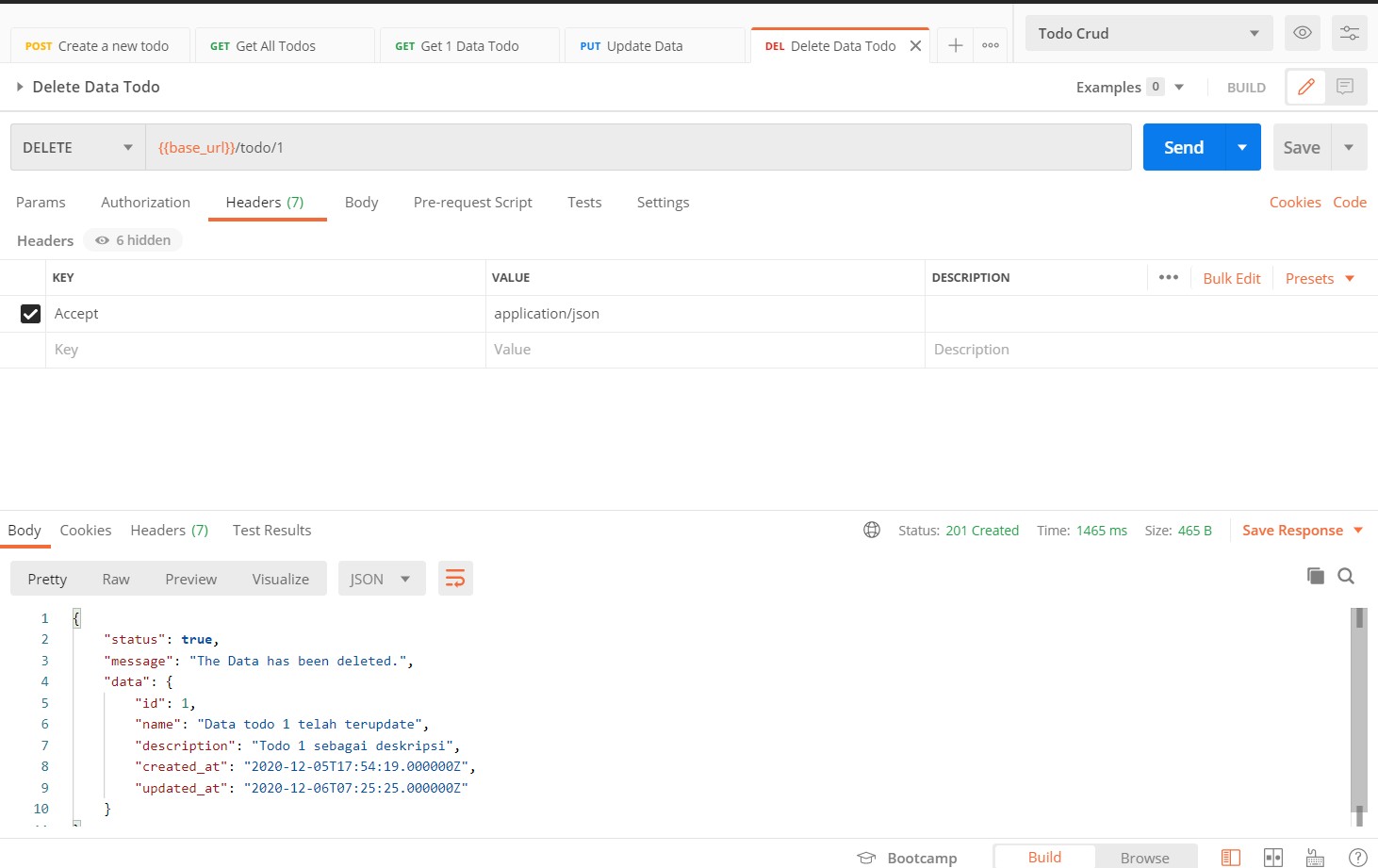
Method URL

DELETE {{base\_url}}/todo/1

##### HTTP Header

|  |  |  |
| --- | --- | --- |
| Header Name | Required | Values |
| Accept | ya | application/json |

Contoh hasil dari pengujian di atas.



**Figure 29:** Delete Data Todo

#### Kesimpulan

*Restful API* menjadi bagian penting dalam menghubungkan antara *backend* dengan *frontend* dan *mobile*. Kualitas *Restful API* bisa dilihat dari *response body, status code, error message* yang dihasilkan. Pastikan ketika *develop API*, kita tidak lupa untuk membuat *design* dan *documentation* antara *backend* dan *frontend*.

Dari tahap ini kita sudah mempelajari membuat *CRUD Todo* menggunakan *Restful API* dengan lumen. Dan kita juga sudah melalukan pengujian *Restful API* ini dengan *Postman*.