LARAVEL

**Fitur utama laravel** : MVC, Template Engine, Artisan Console, Eloquent ORM, Authentication & Authorization, Testing, Packaging System, Multiple File System, Task Scheduling, Websocket Programming.

*\*Folder public digunakan untuk menyimpan folder css, js, img milik kita.*

# Blade Template

Blade adalah fitur yang disediakan Laravel untuk proses templating sederhana namun sangat bermanfaat dalam proses pengembangan tampilan halaman web. Tidak seperti fitur templating PHP populer lainnya, Blade tidak membatasi pemrogram untuk menggunakan kode PHP biasa dalam membuat kode untuk tampilan.

Jika menggunakan php

*<?php echo $nama; ?>*

Jika menggunakan blade

*{{ $nama }}*

Fungsi dari Laravel Blade templating adalah penggunaan layout, agar tampilan yang berulang misalnya header, footer, sidebar dan sebagainya tidak perlu dibuat berkali-kali, seperti kita menggunakan @extends, @include, @yield, @foreach, dll

*@dd, digunakan untuk melakukan debugging dan tidak akan menjalankan code di bawahnya.*

# Eloquent Model

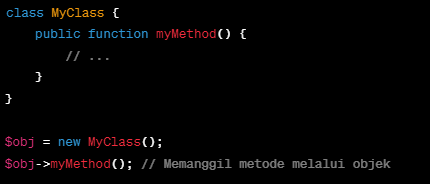
Model merupakan representasi dari object dalam database yang memungkinkan kita melakukan interaksi dengan efisien. Eloquent, sebuah object-relational (Object Relational Model) yang membuatnya mudah dalam bertintegrasi dengan database. Setiap database memiliki ‘model’ terkait yang digunakan untuk berinteraksi dengan tabel tersebut.

Static adalah function yang terkait dengan class itu sendiri, bukan instasiasi dari object class tersebut. Fungsi static dapat diakses langsung dengan menggunakan sintaks ::

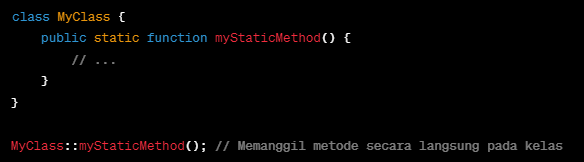
Jika static harus menggunakan keyword ***self::*** bukan this

Untuk static:: biasanya digunakan ketika mengakses variabel di dalam classnya

Tanpa Static



Dengan Static



# Collection

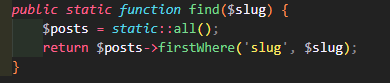
Collection adalah fitur laravel untuk memanipulasi data array. Collection bisa disebut pembungkus dari sebuah array.

Menggunakan Collection cukup dengan

***collect(self::$blog\_posts);***

*biasanya digunakan saat mengembalikan data, perlu diingat bahwa ada self:: yang menandakan bahwa $blog\_posts adalah array static yang ada di dalam class.*

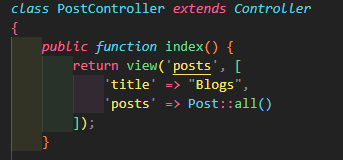
Dengan menggunakan collection membuat kita bisa menggunakan method2 yang disediakan laravel. Misalkan seperti berikut



*\*berikut adalah contoh bahwa kita akan mencari post berdasarkan slug, disini kita bisa menggunakan method firstWhere (cari yang pertama ketemu) lalu di return.*

# Controller

Controller dalam laravel adalah untuk menangani closure yang di buat di dalam route sebagai pemisah yang lebih baik.



# .env

.env digunakan untuk menyimpan settingan pada lingkungan pengembangan pada aplikasi kita.

Banyak file yang akan saling terhubung dengan file .env ini.

Contoh di dalam file config/database.php di dalamnya terdapat code berikut

***‘default’ => env(‘DB\_CONNECTION’, ‘mysql’)***

*\*artinya secara default database kita akan menyesuaikan dengan database yang telah kita set di env DB\_CONNECTION, jika tidak kita set maka secara default database yang digunakan adalah mysql.*

* Buat Database Baru, lalu di laravel lakukan migration.

*\*migration itu sama seperti version control (contoh: git) dimana kita bisa melihat perubahannya namun ini ke dalam database.*

php artisan make:migration posts\_table = membuat migrasi

php artisan migrate = melakukan migrasi table

php artisan migrate:rollback = menarik migrasi table

php artisan migrate:fresh = mere-migrate tabel dan isi datanya

# Eloquent ORM & Tinker

Eloquent ORM (Object Relation Mapper), adalah sebuah fungsi untuk memetakkan tiap tiap data yang ada di dalam table database ke dalam sebuah object yang kemudian memungkinkan kita untuk melakukan crud.

Active Record Pattern adalah seperti kita membuat object instansiasi baru untuk membuat data baru, selanjutnya kita melakukan crud di dalamnya.

***Laravel Tinker***

Laravel Tinker adalah sebuah REPL (Read-Eval-Print Loop) interaktif yang terintegrasi secara langsung dengan Laravel. Ini memungkinkan Anda berinteraksi dengan aplikasi Laravel Anda melalui baris perintah (command line) untuk menjalankan kode PHP dan menjelajahi model, database, dan komponen lainnya secara langsung.

php artisan tinker = untuk memulai tinker

$user = new App\Models\User; = untuk inisialisasi & menginstansiasi user baru.

$user->name = ‘Luthfi’; = mengisikan data

$user->al(); = menampilkan semua data

\*jangan lupa di modelnya harus diberikan akses mana field yang bisa diisi mana yang tidak.

App\Models\Post::create([‘title’ => ‘Lorem Ipsum 1’, ‘body’ => ‘Lorem Ipsum’]);

App\Models\Post::find(1)->update([‘title’ => ‘Title Update’, ‘body’ => ‘Lorem Ipsum’]);

App\Models\Post::where(‘title’, ‘Title Update’)->update([‘body’ => ‘Body Update’]);

Php artisan help make:model

php artisan help make:model -m Post

# Collection Notation

Pada dasarnya Collection mengembalikan data berupa object namun karena laravel mengimplementasi antarmuka ArrayAccess sehingga kita dapat menggunakan 2 notasi, seperti berikut

$post->nama; = notasi object

$post[‘nama’]; = notasi array

# Route Model Binding

Route Model Binding, saat kita menyuntikkan sebuah id ke dalam controller maka akan melakukan query berdsarkan id yang kita cari.

Route Model Binding ini akan meng-skip cara untuk melakukan querynya jadi secara otomatis akan dilakukan oleh laravel supaya dapat mencari data apa yang kita cari. Misalkan kita akan mencari post berdasarkan slug

*Route::get(‘/post/id’, function() { … } );*

*Route::get(‘/post/{post:slug}’, function (Post $post) { … } );*

*\*dengan menggunakan cara ke dua kita bisa langsung menggunakan slug*

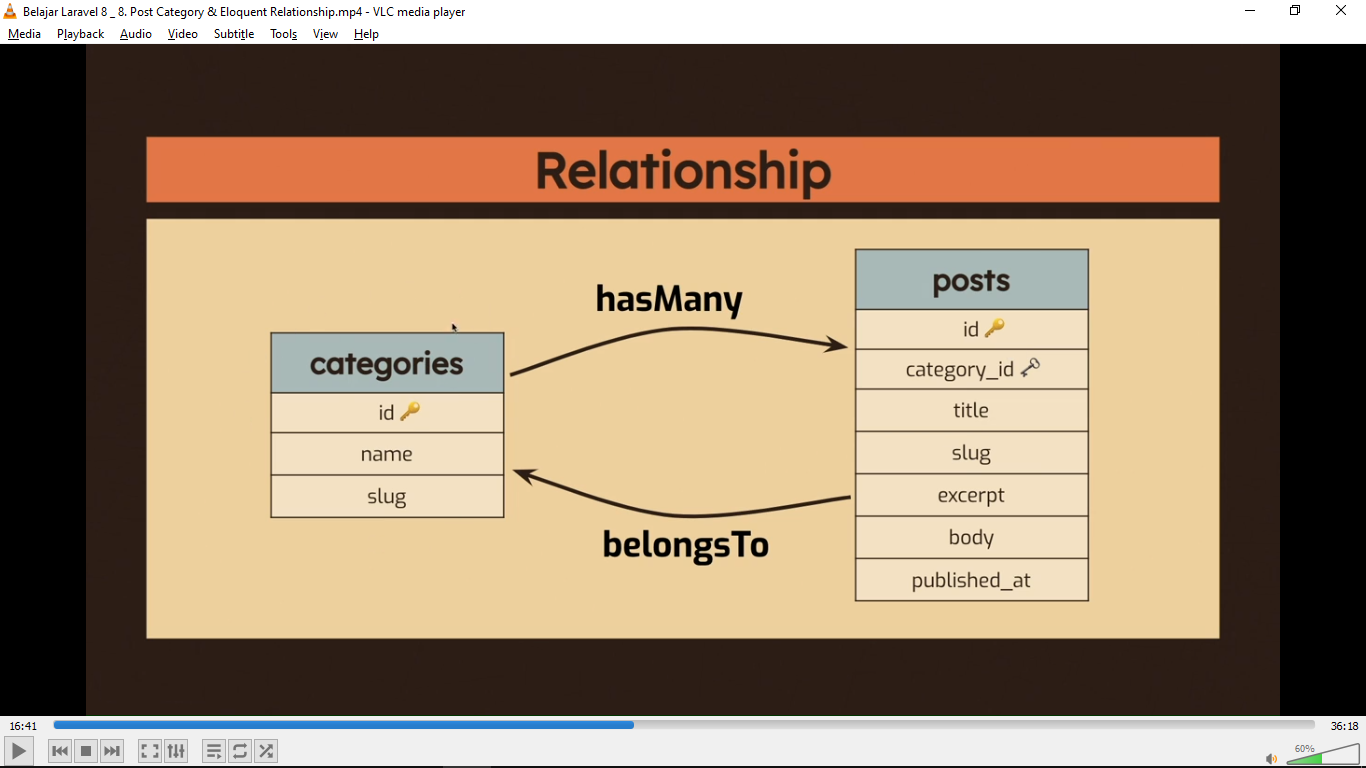
*{post} itu harus sama dengan $post.*

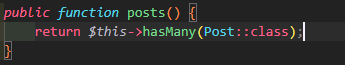
# Relationship Database One to Many

Primary Key = Setiap tabel hanya memiliki 1 primary key yang dimana nilainya harus unik dan tidak boleh sama.

Foreign Key = Sebuah kolom yang menyatakan adanya hubungan antar tabel dengan primary key atau unique key di dalam tabel lain.

*\*Setiap relasi antar tabel harus memiliki id / primary\_key untuk merujuk ke nilai tersebut dari tabel lain.*





*\*disini di dalam modelnya setiap Categori memiliki banyak postingan dan setiap postingan pasti memiliki 1 category*

# Database Seeder

Seeding, mengisi otomatis database kita dengan class seed.

Factories, pabrik untuk membuat data secara otomatis.

*php artisan db:seed = digunakan untuk menjalankan seeder*

Melakukan seeding sama seperti kita melakukan di tinker, kita bisa membuatnya di file DatabaseSeeder.php

# Factories & Faker

Buat factory baru untuk dapat membuat data secara otomatis.

*php artisan make:factory PostFactory*

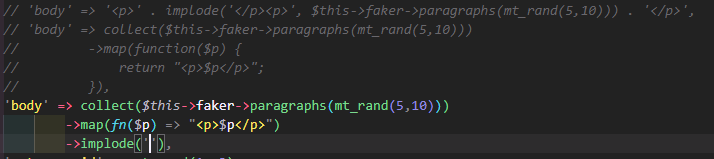
Install Faker, untuk apa? Agar kita bisa melakukan data diisi otomatis dengan data dummy / generate data random.

Pergi ke bagian config/app.php cari faker ubah faker\_locale

'faker\_locale' => env('FAKER\_LOCALE', 'en\_US'),

*php artisan migrate:fresh –seed = digunakan untuk merefresh database sekalian melakuan seeding.*

Merubah agar setiap data di factory diawali dengan huruf p per paragrafnya



# N + 1 Problem

N+1 disini artinya adalah 1 query untuk parent dan N adalah jumlah record pada table child. Untuk mengecek / melakukan debugging kita akan menggunakan Clockwork untuk laravelnya.

*Lakukan installasi clockwork “****composer require itsgoingd/clockwork****”*

*Install Extension Chrome “****clockwork****”*

Untuk memahami problemnya, misalkan kita punya posts dengan menghubungkan ke relasinya category & user. Saat melakukan pemanggilan data biasa maka akan melakukan pemanggilan ke 3 tabel di databasenya seperti *Select \* from ‘users’ limit 1*.

Lalu bagaimana jika kita melakuan foreach atau looping, maka yang terjadi pemanggilan data nya juga akan di looping beserta pemanggilan ke 3 tabel di databasenya.

* **Lazy Loading**

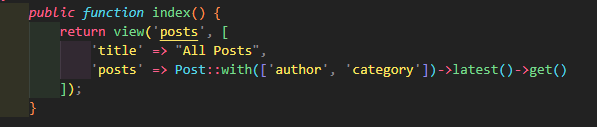
Kita melakuan pemanggilan data jika dibutuhkan saja.

(Penyebab N + 1 Problem jika datanya banyak).

* **Eager Loading**

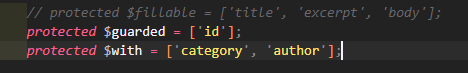
Kita Lakukan semuanya di awal ‘sehingga sudah punya datanya di awal’, sehingga saat looping tidak perlu melakukan pemanggilan ke database lagi. Dengan Eager Loading kita cukup gunakan keyword **with()**.

*PostController.php*



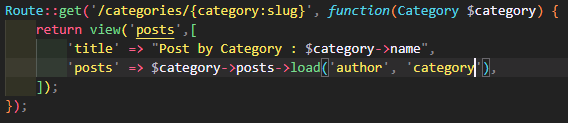
Kita juga bisa memindahkan with() ke dalam model post

*bahwa ketika kita memanggil tabel post panggil juga sekalian dengan category & author*



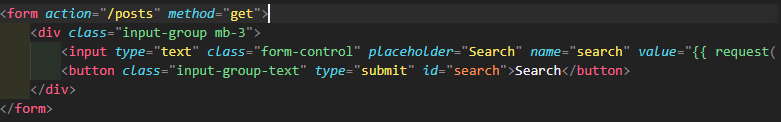
* **Lazy Eager Loading**

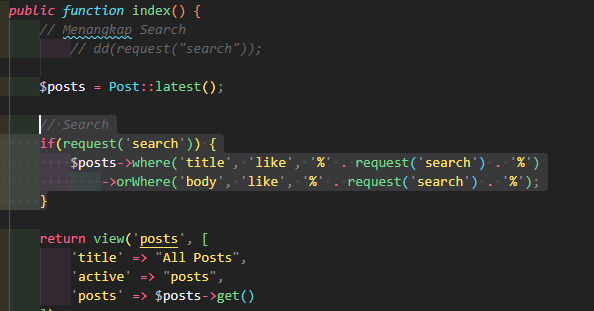
Terkadang kita butuh untuk melakukan eager loading ***\*pada relationship tapi setelah parent nya di dapatkan.*** Untuk menggunakan Lazy Eager Loading pada relationship disini kita gunakan method **load()**.



# Search & Pagination

Untuk kita Membuat Fitur Search, bisa kita simpan di dalam Controller.

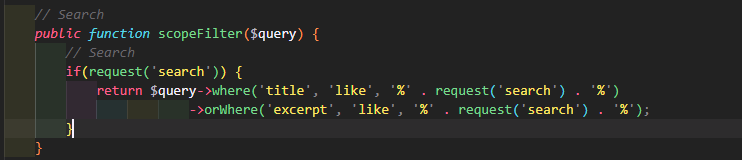




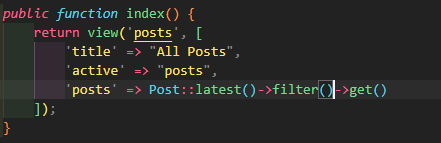
Namun, karena search ini terhubung langsung ke database sebaiknya kita menyimpannya di dalam Model.

***Eloquent Query Scope***, adalah metode dalam model Eloquent yang memungkinkan Anda mengatur kriteria query umum dan kemudian menggunakan kriteria tersebut secara bersamaan di beberapa tempat dalam aplikasi Anda. Terdapat 2 ya itu Local Scope dan Global Scope

***Local Scope,*** query scopes yang spesifik untuk model tertentu dan hanya dapat digunakan di model tersebut / query umum yang bisa digunakan kembali. Untuk mendedinisikannya kita gunakan kata kunci **scope’NamaScope’** pada awal method.

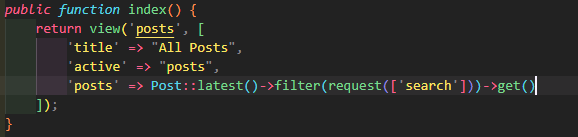


Disini kita namakan scopeFilter, maka di dalam index kita jalankan method filter()



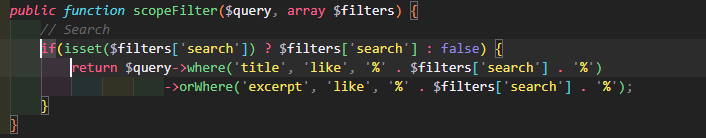
*Query ini adalah kerjaanya sebuah Model, sedangkan untuk request() itu adalah kerjaanya Controller.*

Kita pindahkan requestnya ke dalam Controller yang ditujukan ke method Filter.



Disini kita dapat gunakan ***array $filters***

Agar Parameter $filters yang ditangkap bisa berupa array dimana tujuannya agar kita bisa melakukan pencarian untuk category & authornya nanti.



*Search berdasarkan title & excerpt*

Kita dapat mengganti if dengan keyword ***when()*** dimana akan menjalankan jika argument yang diterima bernilai true. Ini akan melihat dari Collection yang kita punya.

***Null coalesing operator***, digunakan untuk mengecek ternary dan isset.

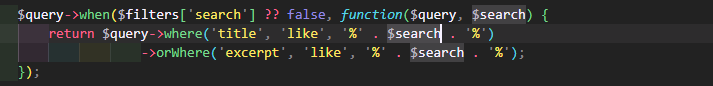
***Tanpa null coalesing operator***

If (isset($filters[‘search’]) ? $filters[‘search’] : false) { … }

***Dengan null coalesing operator***

When ($filters[‘search’] ?? false, function($query, $search) { … }

*\*kedua ini sama sama saja*



Spesifik pencarian dengan menggunakan **whereHas()**, digunakan untuk menyaring query berdasarkan hubungan yang terkait. Karena kita melakukan join table maka kita gunakan **whereHas()** untuk menunjukan pencarian berdasarkan relationship.

Karena tadi request parameternya berupa array maka dapat kita tambahkan di dalam controllernya method model yang sudah didefinisikan.

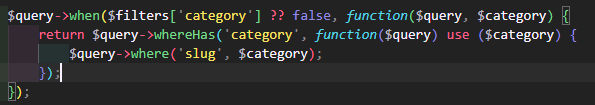


Lalu kita buat hal yang sama namun kita menggunakan whereHas(‘category’) yang merujuk ke method category hasil dari relationship post. Yang kemudian menjalankan function untuk mencari slug berdasarkan $category.

Disini kita gunakan **use** untuk mengarahkan ke $category yang pertama kali.

*$query akan masuk ke parameter $query*

*Dan $filters[‘category’] akan masuk ke parameter $category*



Perbaiki url nya agar tidak lagi mengarah ke **/categories/{{ … }}** Tapi menjadi **/posts?category=** dengan bergitu kita dapat mencari postingan di dalam category tertentu. Lakukan juga untuk di post

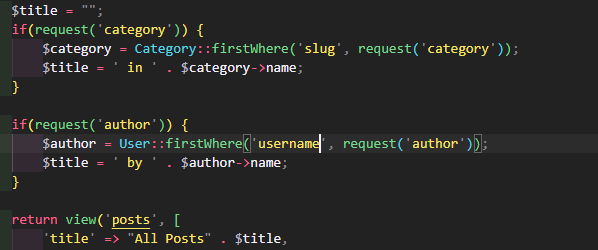


Lalu kita sisipkan ke form pencariannya



Maka nanti jika kita sedang berada di postingan category urlnya akan menjadi **/posts?category=** kemudian kita melakukan search maka akan ditambahkan dengan **&search=**

Di dalam route nya kita buat agar title akan berubah ketika kita berhasil mencari categori berdasarkan yang kita cari



Pagination

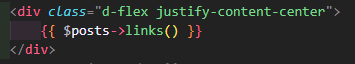
Di dalam Controller saat melakukan pemanggilan kita cukup gunakan

**paginate(n)** = digunakan untuk membatasi jumlah yang tampil perhalaman.

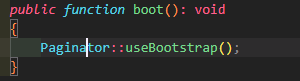
**withQueryString()** = digunakan agar paginate tetap jalan saat melakukan pencarian



Di Tampilannya kita cukup berikan



Jangan lupa kita setting agar pagination kita menggunakan bootstrap, ada di AppServiceProvider.php



# Login & Registration

***@csrf***

Validasi

Rule validation laravel

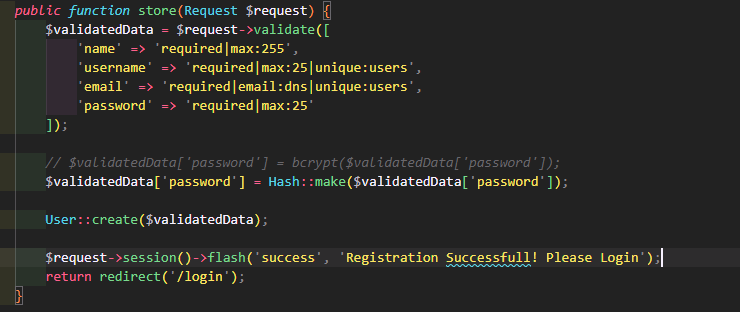
Memberikan error & Displaying the validation error

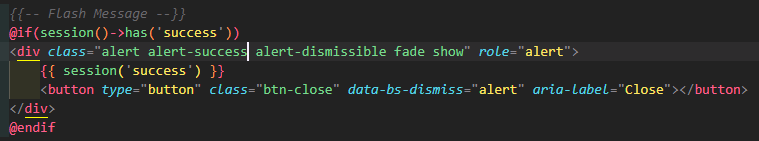
Old(), isian lama kita berdasarkan value / inputan lama (sebelumnya)



Jangan lupa lakukan bcrypt / hashing pada password.

Flash Message / Flash Data





# User Login Logout & Middleware

Authentication, adalah metode untuk menentukan atau memastikan bahwa seseorang (atau sesuatu) adalah asli atau benar. Di dalam laravel terdapat *Starter Kit* atau *Plugin* yang khusus untuk menangani authentication seperti ***Laravel Breeze*** dan juga ***Laravel JetStream.***

***Manually Authenticate Laravel***

Kita tidak diharuskan untuk selalu menggunakan *Starter Kit* milik laravel, kita bisa bisa mengakses layanan authentikasi dengan melalui **Auth** dari *facades*.

Disini kita akan melakukan pengecekkan, dimana kita bisa saja lolos dari validasi namun harus ada yang kita cek lagi, karena lolos dari validasi belum tentu berhasil login, bisa saja emailnya belum terdaftar atau password yang digunakan salah. Maka sekarang kita akan menggunakan fungsi **attempt()** secara khusus, **attempt()** digunakan untuk mencoba mengotentikasi pengguna berdasarkan kredensial yang diberikan.

Setelah kita menggunakan fungsi **attempt()**, kita regenerate sessionnya, karena setiap informasi login itu akan mendapatkan key, kita regenerate sessionnya jika ada yang mencoba akses cookie login kita dan sessionnya tidak sama maka tidak akan bisa.



*\*jika kita memberikan validasi dan menampikan field error, misalkan ‘email anda salah’ atau email belum terdaftar / passwordnya salah (ini akan menjadi celah keamanan).*

*\*Melakukan Regenerate pada session, untuk menghindari teknik hacking* ***session fixsession.*** *Oleh karena itu untuk menghindarinya kita generate ulang.*

*\*****intended()****, Metode intended yang disediakan oleh redirector Laravel akan mengarahkan pengguna ke URL yang mereka coba akses sebelum dicegat oleh middleware otentikasi. URI cadangan dapat diberikan pada metode ini jika tujuan yang dituju tidak tersedia.*

Middleware Authentication, middleware otentikasi digunakan untuk melindungi rute atau aksi tertentu agar hanya dapat diakses oleh pengguna yang sudah login. Kita dapat melihat global middleware di *App/Http/Kernel.php*

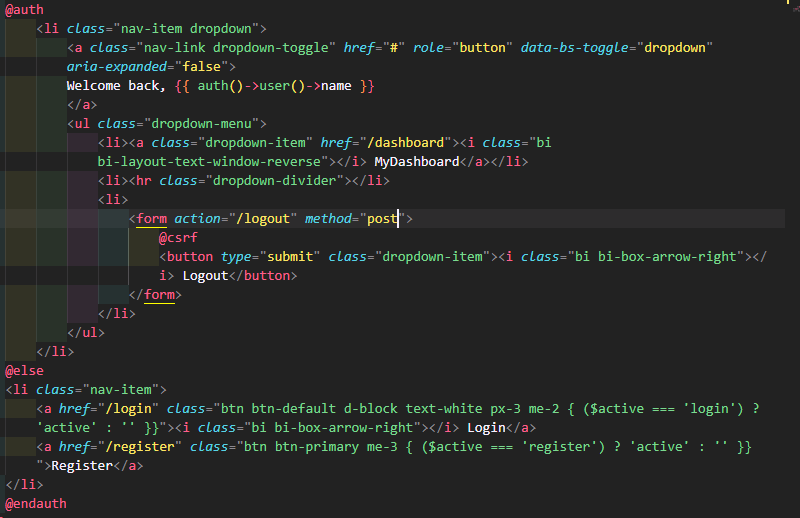
Menggunakan middleware, guest artinya hanya bisa diakses bagi guest / yang tidak melakukan login.



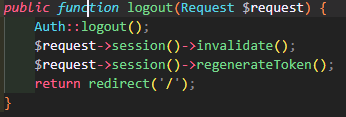
Jika ada user yang sudah login, dan memaksakan untuk bisa kembali ke route /login, maka akan terjadi error dan akan mengarah ke /home, oleh karena itu kita ubah **RouteServiceProvider.php** agar jika ada user yang sudah login dan memaksa ke halaman /login maka akan dilempar ke route **/** .



Blade Directive Laravel, seperti @auth, @guest itu akan menampilkan halaman / bagian khusus yang sudah login saja misalkan.



**Logging Out**, Untuk mengeluarkan pengguna secara manual dari aplikasi Anda, Anda dapat menggunakan *logout* metode yang disediakan oleh **Auth** *facades*. Ini akan menghapus informasi otentikasi dari sesi pengguna sehingga permintaan selanjutnya tidak diautentikasi.



*\*invalidate() akan menghapus semua data sesi untuk sesi saat ini.*

*\*regenerateToken() akan menghapus semua data sesi untuk sesi saat ini.*

Defaultnya jika misalnya ada user yang tidak terauthentikasi berusaha masuk ke halaman yang di authentikasi maka redirect dia ke route yang namanya login. Kita bisa mengeceknya disini App/http/midleware/authenticate.php

***Name Routes***, merujuk pada memberikan nama (alias) tertentu kepada suatu rute dalam aplikasi Anda.



Menampilkan nama yang sudah login, kita bisa gunakan fungsi auth() yang sudah disediakan laravel dan juga cari tahu siapa user yang login.

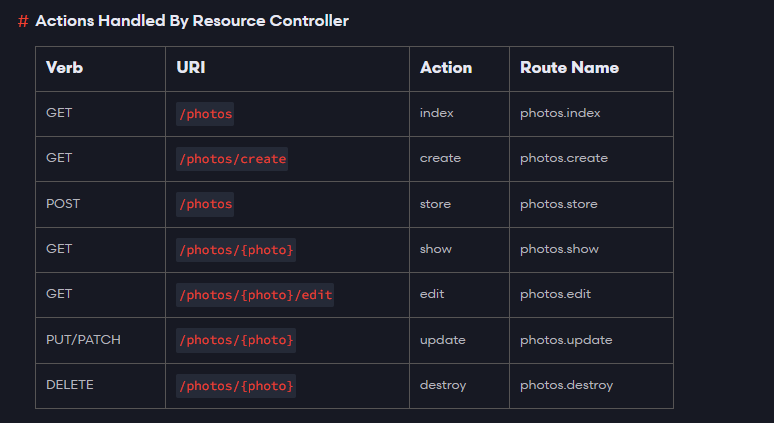


# Resources Controller

Resource Controller, konsep dalam Laravel yang menyediakan pendekatan terstruktur untuk mengelola logika CRUD (Create, Read, Update, Delete) terkait suatu sumber daya (resource) dalam aplikasi web Anda



***php artisan route:list***

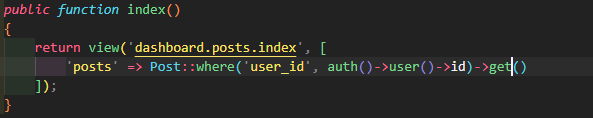


**Specifying The Resource Model**, kita bisa langsung menentukan route model binding untuk controller baru kita, Kita juga akan memberitahukan controller yang kita punya selain dia bentuknya resource dia juga terhubung ke model yang mana.

*Misalkan resource untuk DashboardPostController yang terhubung ke Post Model*



Menampilkan Post berdasarkan user id / user yang lagi login / pemilik dari postnya.



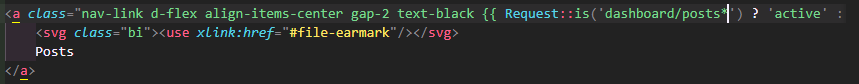
**Customizing The Key**, Terkadang eloquent yang kita butuhkan kita bisa mencari kolom selain dari id, misalkan yang kita butuhkan adalah slug. Kita akan timpa dengan menggunakan sebuah ***getRouteKeyName()*** di dalam Model Post nya sebagai menandakan bahwa ketika kita mencari akan otomatis berdasarkan slug.



Dengan begini saja laravel sudah tau bahwa kita akan melihat detail post berdasarkan slugnya.



Apapun link setelahnya misalkan posts adalah utama, di dalamnya ada posts/info posts/detail, itu dapat dibuat active secara otomatis jika yang kita arahkan itu adalah posts utama nya, cukup berikan tanda **\***.



## Create Post Form

***Create***

Ketika kita membuat Form untuk mengarah ke halaman create dengan menggunakan resource, kita hanya cukup arahkan ke halaman yang dituju.



Secara default Laravel sudah mengetahui method yang dituju saat routing adalah get, artinya tujuannya untuk pindah ke halaman dan diberikan /create menandakan halaman yang dituju adalah create.

***Form Create***

Ketika kita membuat sebuah form untuk mengirimkan data ke dalam resource controller, kita hanya cukup arahkan ke route utamanya adalah /dashboard/posts dengan method post, karena laravel akan mengetahui data yang dikirimkan dan ditangkap oleh method store.



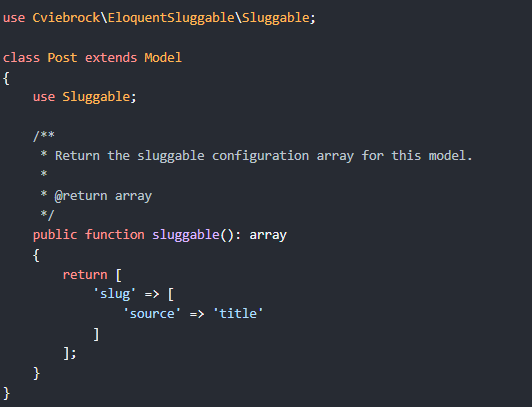
***Eloquent-slugable*** ( <https://github.com/cviebrock/eloquent-sluggable> )

Slug atau kata unik itu seharusnya tidak bisa dibuat oleh user, karena bisa saja ada kesamaan satu sama lain, oleh karena itu kita gunakan library Eloquent-Slugable, dengan tujuan kita akan membuat slug secara otomatis dengan mengambil data dari inputan judul yang ditulis.

Langkah pertama dalam installation **Library Eloquent Slugable**, dengan mengetikkan

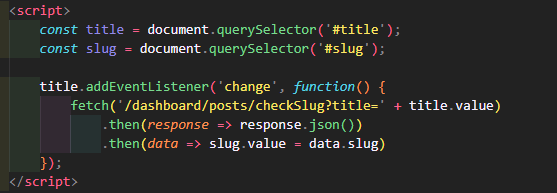
***composer require cviebrock/eloquent-sluggable***

Kemudian Tuliskan function sluggable() ke dalam model yang akan kita berikan slug secara otomatis. Ketikkan code tersebut dan jangan lupa untuk import Eloquent Sluggable-nya.



Langkah berikutnya kita akan melakukan fetch API

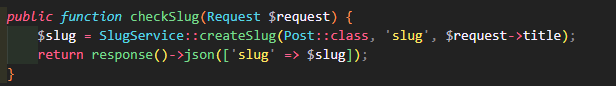
*Fetch API adalah antarmuka JavaScript yang menyediakan cara untuk mengambil (fetch) sumber daya dari jaringan (seperti file atau data JSON) melalui HTTP.*



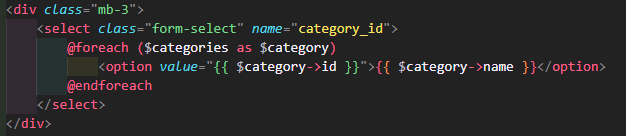
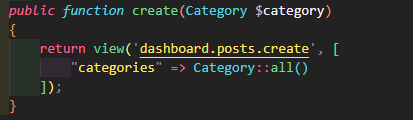
Ketika ada inputan di dalam title / ketika berubah, maka lakukan fetch yang dimana tujuannya untuk mengambil data inputan yang kemudian dikirimkan ke checkSlug sambil check ada tidak yang mengirimkan judul ke url? Jika ada ambil valuenya dan kirimkan dalam bentuk json ke route /dashboard/posts/checkSlug?title=



Kemudian di dalam controller kita jalankan function checkSlug.



***Mengirimkan data kategori ke dalam inputan***



***Membuat TextArea dengan Trix Editor*** ( <https://github.com/basecamp/trix> )

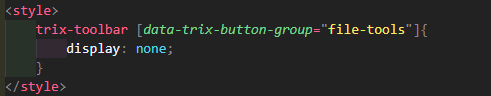
Langkah pertama tambahkan 2 baris CDN (css, js) trix editor ke dalam head.

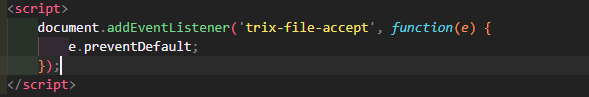
Kemudian di dalam form kita berikan inputan untuk trix editor



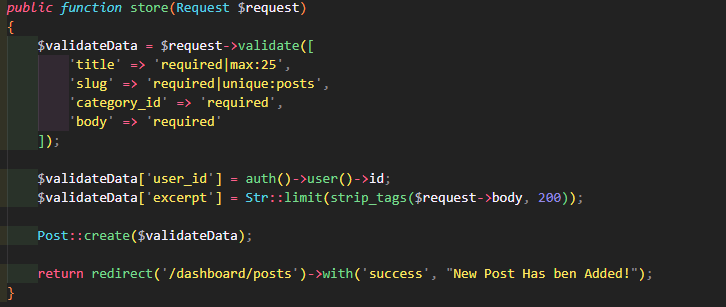
*Tambahan:*

1. *Menghapus fitur upload file di trix editor*





## Validation Insert Post

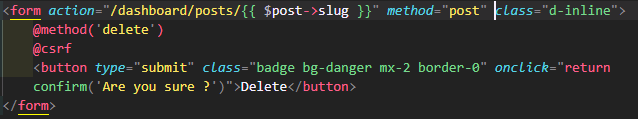


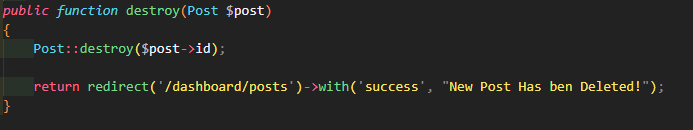
Disini kita mengisikan user\_id cukup panggil method **auth()**, karena informasi user yang sedang login akan tersimpan disitu.

Lalu untuk excerpt, kita gunakan helper laravel yaitu **Str::limit** dimana digunakan untuk membatasi panjang text yang diambil dari body. Format data yang yang diambil dari trix akan ada tag htmlnya, maka kita gunakan **strip\_tags()** dari php untuk menghilangkan tag html-nya.

## Validation Edit & Delete

***Delete Data***





Untuk membuat delete data kita harus buat sebuah form terlebih dahulu, lalu kita kirimkan data slugnya, Jangan lupa juga untuk menggunakan blade directives **@method(‘delete’)** untuk merubah menjadi delete data. Dari slug yang dikirimkan kita bisa mengetahui informasi 1 postingan itu,kita ambil id nya lalu lakukan destroy.

***Edit Data***

Kita ambil slug dari postingannya dan arahkan ke /edit.



Di halaman edit kita bisa mengisikan value dengan menggunakan old() sekalian juga mengirimkan data yang sudah ada.



Validasi Edit, sebelumnya kita bungkus terlebih dahulu ke dalam $rules, karena slug tidak boleh sama maka kita lakukan pengecekkan terlebih dahulu. *Jika request slug tidak sama dengan slug yang sudah ada* maka lakukan validasi tapi jika sama tidak perlu melakukan validasi, sisanya sama seperti method store.

*Eloquent ORM, where (id nya berapa) jalankan method update().*



## Upload Image

Di dalam form kita tambahkan sebuah input bertipe file dengan name ***image***. Namun karena di dalam database kita belum ada kolom image kita buat kembali terlebih dahulu, dan untuk form nya kita berikan ***enctype=”multipart/form-data”*** karena jika ada string akan dispisahkan ke dalam request dan jika ada file maka akan dipisahkan ke dalam files.

Kita lakukan debugging untuk mengetahui lokasi penyimpanan file image itu dimana ?

***ddd($request);***



Request akan berisi inputan kita yang bertipe string.

Files akan berisi inputan kita yang bertipe file.

Kita bisa mencoba menangkap gambar dan menyimpannya ke dalam storage dengan cara :

*return $request->file(‘image’)->store(‘post-images’);*

Kita ambil request yang berupa file dengan name nya image, lalu lakukan store ke dalam direktori bernama post-images, jika tidak ada maka akan dibuatkan secara otomatis.

**store(),** adalah bagian dari Laravel's Filesystem API dan digunakan untuk menyimpan file yang diunggah ke sistem penyimpanan yang telah dikonfigurasi.

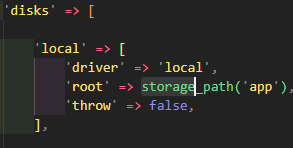
Setelah melakukan store(), file image yang di upload akan berada di dalam folder storage/app/post-images, image yang diupload akan tersimpan disitu, namun sayangnya image disini tidak akan bisa kita akses. Kita dapat mengaksesnya dengan kita konfigurasi terlebih dahulu.

***File Storage***

Kita lakukan konfigurasi terlebih dahulu di *config/filesystem.php*, secara default file storage kita adalah local.



Jika file storage local maka akan tersimpan disini, di dalam storage/app.



Agar bisa kita tampilkan gambarnya, kita harus merubahnya menjadi public. Sebelumnya kita buat FILESYSTEM\_DRIVER di .env karena sebaiknya kita ubah di env atau tambahkan di env terlebih dahulu, karena jika tidak ada maka akan mengambil yang local.





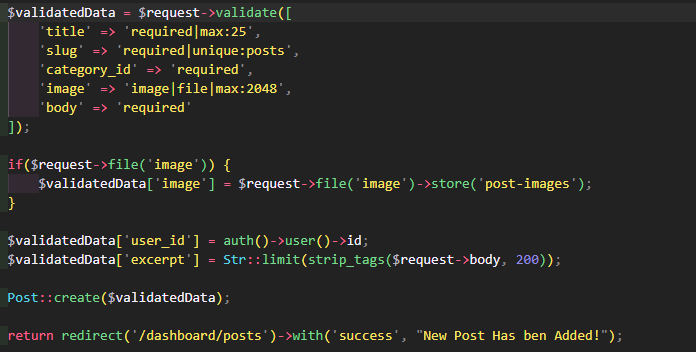
Jika file storage public maka akan tersimpan di dalam storage/app/public/post-images, meskipun begini kita tetap saja belum bisa mengakses file di dalamnya. Agar bisa kita lakukan langkah berikutnya.



Hubungkan folder public di dalam storage dengan folder public milik aplikasi kita, kita akan gunakan *symbolic link.*

***php artisan storage:link***

Dengan begini file public milik kita akan bisa mengakses ke dalam folder post-images di storage kita.



Validasi image, kita berikan keyword image karena di dalam laravel ada validasi untuk image maka itu akan menerima file dengan ekstensi (jpg, jpeg, png, bmp, gif, svg, or webp).

Lalu File, jika kita tidak memberikan file pada validasinya maka nama image akan dianggap string karakter biasa bukan string image.

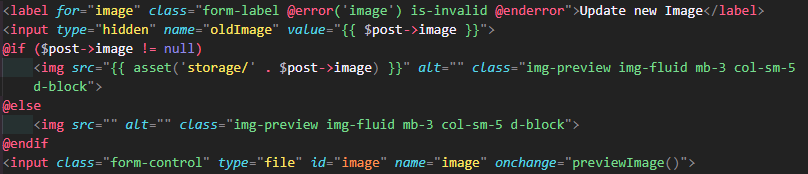
## Preview Image



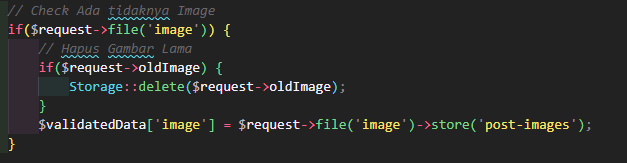
Sebelumnya di dalam inputan image kita tambahkan javascript ***onchange*** jadi ketika ada perubahan / ada data diupload, jalankan fungsi **previewImage()**. Kita buat sebuah fungsi previewImage di javascriptnya.



## Delete Image When Update



Kita buat sebuah file input hidden dengan nama *oldImage* yang akan berisikan imagenya, nanti akan ditangkap oleh $request.



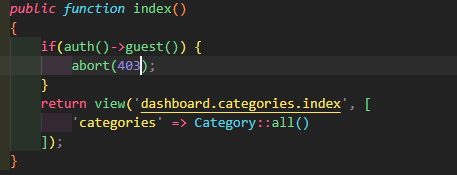
Di dalam method update, kita cek ada tidak image barunya? Jika ada maka cek lagi ada ga request dengan nama *oldImage* kalo ada hapus dari storage. Lalu kemudian masukan image ke dalam strore(‘post-image’).

# Authorization

Authorisasi, proses memberikan hak atau izin kepada entitas tertentu (seperti pengguna atau sistem) untuk melakukan tindakan atau mengakses sumber daya tertentu. Ada 2 cara utama untuk melakukan Otorisasi dengan menggunakan ***gates*** atau ***policies***. Starter Kit Laravel misalkan menggunakan JetStream / Breeze.

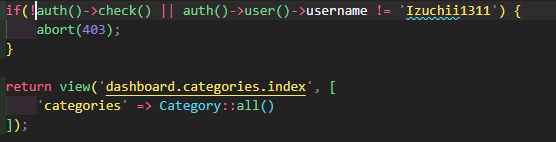
## Authorization with Middleware

Di Routenya kita tidak lagi menggunakan middleware(‘auth’), secara manual kita bisa membatasi authorisasi dengan menggunakan middleware dengan mengirimkan ke dalam setiap method di controller.



Disini kita menggunakan **guest()** artinya akan mengecek jika sudah login akan menghasilkan true, jika tidak akan menghasilkan false, tapi kita juga bisa menggunakan **check()** bedanya check akan menghasilkan nilai true meskipun belum login.

Kita lakukan kembali pengecekkan !auth() cek jika tidak ada authentikasi dari login maka jangan beri akses, begitu pula jika ada yang login tapi username tidak sama jangan beri akses.



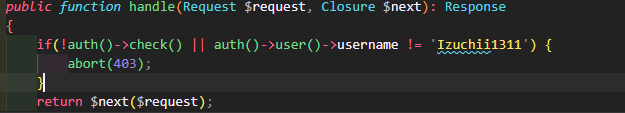
Disini cara itu berhasil namun masih ada problem, seperti :

* Kita harus copy paste ke dalam setiap method di controller.
* Jika usernamenya bukan Izuchii kita tidak akan bisa akses.

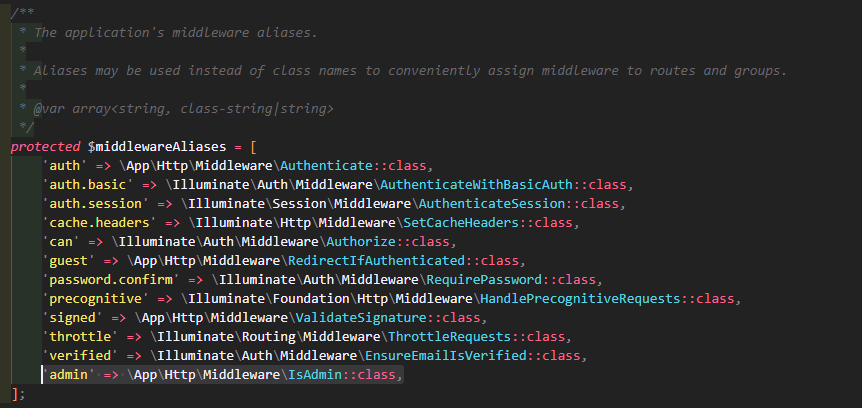
Kita Buat cara tersebut menjadi *middleware* kita sendiri

***php artisan make:middleware IsAdmin***

Kemudian kita pindahkan pengecekan logika tersebut ke dalam method handle di middleware yang telah kita buat.



Middleware ini hanya baru dibuat classnya saja, kita harus mendaftarkan nama middleware yang telah dibuat. Ada di dalam ***kernel.php***, disini kita daftarkan dengan nama middleware ‘admin’.

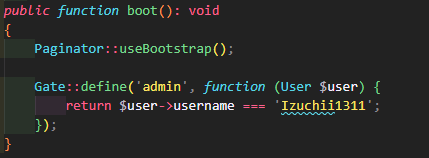


Untuk Menggunakan Middlewarenya cukup panggil nama middleware kita yang sudah didaftarkan.

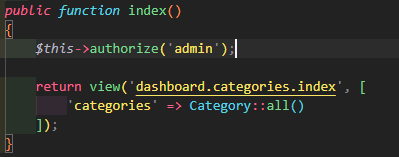


## Authorization with Gates

Fitur Authorisasi dengan menggunakan Gates, pada umumnya gates ini harus kita definisikan terlebih dahulu di dalam method **boot()** di dalam ***AppServiceProvider.php***. Kita definisikan yang boleh akses siapa.

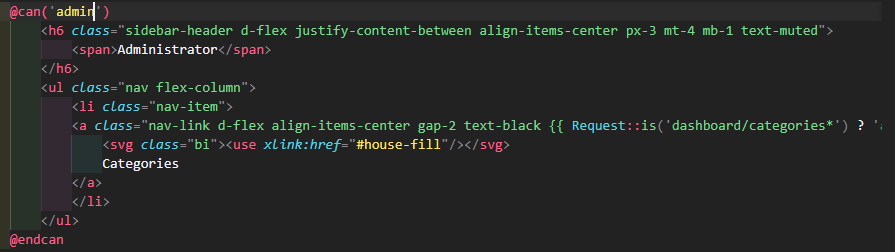


Disini kita definisikan sebuah gate yang namanya admin dimana ini hanya bisa diakses oleh user dengan username ‘Izuchii1311’. Sekarang di dalam Controllernya kita tulis code berikut



**authorize()**, digunakan dalam kebijakan (policy) untuk mengidentifikasi apakah pengguna yang saat ini terautentikasi memiliki izin untuk melakukan suatu tindakan tertentu pada sumber daya tertentu.

Disini kita gunakan blade directives yaitu **@can()**, maka dengan ini bagian html ini akan bisa diakses oleh user yang memiliki authorisasi admin saja.



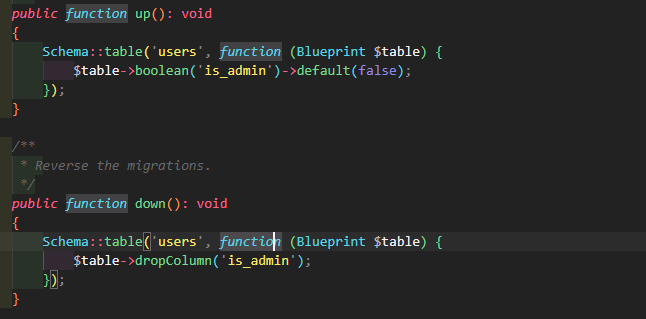
Namun sebenarnya masih bisa saja ada problem, bagaimana jika kita ingin menambah admin baru? Bagaimana jika usernamenya berubah / diganti?

Oleh karena itu kita akan mengecek authorisasi tidak lagi dari username tapi akan mengecek ke field baru di database dengan nama *is\_admin* yang akan bertipe boolean.

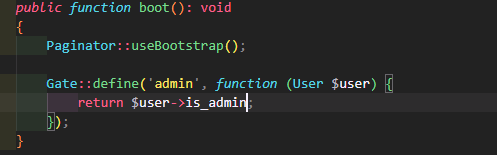
***Menyisipkan Field Baru ke Database***

Kita akan menyisipkan field is\_admin ke dalam tabel users. Jangan lupa untuk mengisikan function up dan down nya.

**php artisan make:migration add\_is\_admin\_to\_users\_table --table=users**

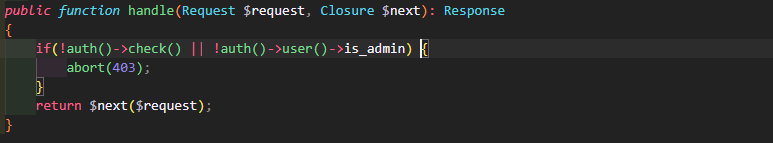


Di dalam *AppServiceProvide.php*sekarang kita ganti menjadi is\_admin, karena jika bernilai true maka berikan akses admin.



Sekarang di dalam middleware *IsAdmin.php*

Disini kita cek terlebih dahulu user login nya, jika tidak ada user login maka jangan beri akses, dan jika tidak ada user yang login dengan is\_admin tidak true maka jangan beri akses juga.



# Laravel Authentication

## Laravel Breeze

Laravel Breeze adalah salah satu package resmi yang dikembangkan oleh Laravel untuk memberikan kemudahan dan kecepatan bagi website developer dalam membangun aplikasi web yang membutuhkan sistem autentikasi pengguna. Package laravel jetstream itu lebih kuat dalam mengelola level seperti role role yang sangat banyak.

Laravel Breeze adalah paket otentikasi yang lebih terintegrasi, khususnya melalui Jetstream. Ini menawarkan pengalaman pengembangan yang lebih cepat dan lengkap, tetapi mungkin terlalu bertenaga jika Anda hanya membutuhkan otentikasi dasar.

Perbedaan diantara keduanya, laravel ui lebih identik menggunakan bootstrap sedangkan laravel breeze ui nya lebih identik menggunakan tailwindCSS.

Install Laravel Breeze

***composer require laravel/breeze –dev***

***php artisan breeze:install misalkan pilih : blade***

***php artisan migrate***

***npm install***

***npm run dev***

## Laravel UI

Laravel UI adalah paket resmi dari Laravel yang menyediakan alat bantu untuk menghasilkan kerangka kerja pengguna (UI) dasar untuk aplikasi Laravel.

Laravel UI memberikan kerangka kerja UI dasar dan dukungan untuk pemasangan kerangka kerja UI populer. Ini memberikan fleksibilitas lebih besar tetapi memerlukan konfigurasi manual untuk otentikasi.

Install Laravel UI

***composer require laravel/ui***

***php artisan ui bootstrap --auth***

***php artisan migrate***

***npm install***

***npm run dev***

Selain dari Laravel Breeze dan Laravel UI ada juga seperti Laravel Jetstream dan Laravel Socialite.

# Rest API Laravel

*API (Application Programming Interface),* interface yang dapat menghubungkan satu aplikasi dengan aplikasi lainnya. Ada 4 jenis API yaitu: SOAP API, RPC API, Websocket API, dan REST API. Jenis API juga ada 4 ada Public API, Private API, Partner API, dan Composite API.

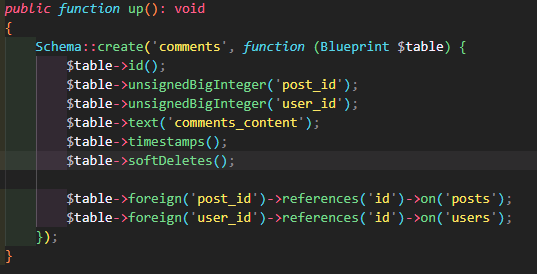
Prerequisite : Format JSON, HTTP status, Database DBMS.

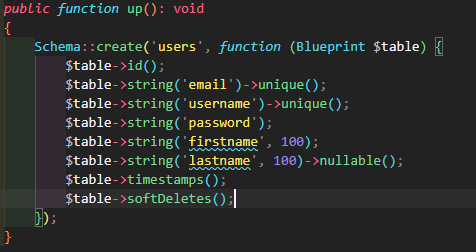
Tools : Postman

## Get JSON

Pertama buat terlebih dahulu table database, posts, users, dan comments.



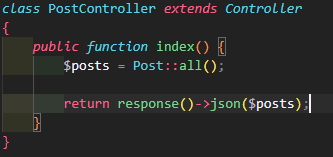




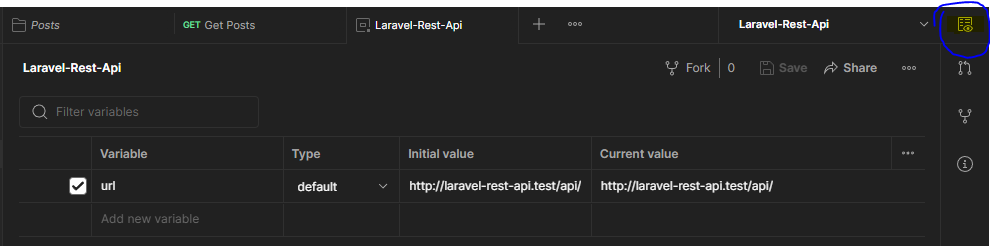
Lalu kemudian buat route api baru untuk melakukan get data.

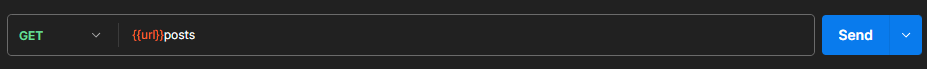


Buat Model Post dan juga Controller dimana Model melakukan guarded pada id dan Controller Mengirimkan Data berupa Json.



Tips Postman, kita bisa membuat variabel agar tidak perlu menuliskan secara berulang, seperti berikut. Kemudian kita cukup panggil dengan menggunakan {{ nama variabel }} lalu karena kita hanya melakukan get, pilih method get dan arahkan urlnya.





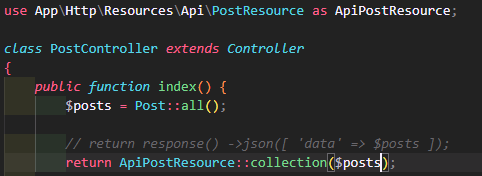
## Eloquent: API Resource

API Resource dalam konteks Laravel adalah cara untuk membentuk dan mengekspose data model Anda dalam bentuk yang dapat dengan mudah diubah menjadi format JSON atau array. Cara Membuat Resource

***php artisan make:resource Api/PostResource***

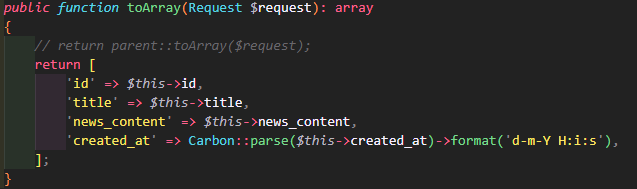
*\*sebenarnya tidak perlu membuat / dimasukkan ke dalam folder API.*

Di dalam Controller Kita panggil untuk mengembalikan apa yang dilakukan oleh resource. Disini kita mengembalikan data dari model Post yang akan dikelola terlebih dahulu oleh PostResource yang nantinya akan dikembalikan dalam bentuk collection.



*\*****collection*** *ini adalah pembungkus dari array, yang membuat kita bisa dengan mudah untuk mengelola data. Hasil dari collection ini akan berupa Object yang memiliki key dan Value tapi perlu diketahui ini tidak sama dengan array assosiatif.*

Di dalam Resource kita lakukan ini, untuk mengembalikan data apa saja yang kita inginkan misalkan hanya menampilkan id, title, news\_content, dan created\_at.



## $This

$this digunakan untuk menunjukkan kepada object, ketika kita mengaksesnya di dalam class. <https://www.duniailkom.com/tutorial-belajar-oop-php-pengertian-dan-fungsi-variabel-this-dalam-pemrograman-objek/>

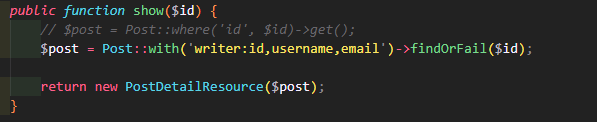
* + *$posts* adalah kumpulan (*collection*) objek dari model *Post*.
  + *PostResource::collection($posts)* mengambil setiap objek *Post* dalam koleksi tersebut dan menerapkannya ke dalam resource *PostResource*. Ini mengembalikan koleksi resource yang berisi hasil dari metode toArray() dari setiap objek Post dalam koleksi*.*
  + *$this* dalam metode *toArray()* merujuk pada instance objek resource saat ini, yang merupakan instance dari kelas *PostResource*.

*$this di dalam toArray() itu mengembalikan data dari object yang ada di dalam class PostResource. PostResource itu menerima data collection dari Controller dimana data itu diambil dari Model Post*

Cara diatas adalah method index() untuk menampilkan semua data array, untuk method show() kita tidak akan menggunakan collection() karena itu akan menghasilkan banyak data, kita gunakan keyword **new** karena kita mencari berdasarkan id dan kita buat collection yang hanya menampilkan 1 data.

***\*collection() = akan menghasilkan banyak data.***

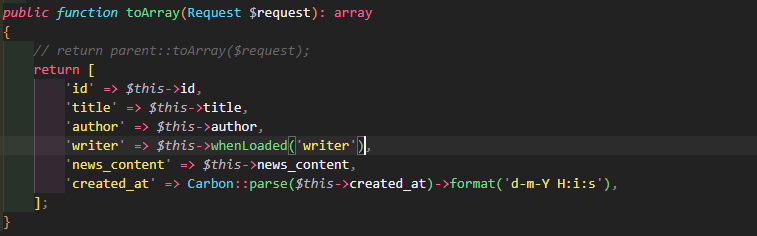
***\*new = akan menghasilkan data baru (1).***



**Eager Loading**, Jika kita ingin memanggil relationship di dalamnya bisa kita sisipkan di dalam **with()**, namun sebelumnya perlu ditambahkan di dalam Model kita.



Lalu kemudian kita buat ApiResource Terpisah untuk method show dengan nama PostDetailResource. Dimana kita hanya akan menampilkan 1 data dengan data data berikut



## Authentication API

Laravel API Authentication Services, ada 2 yang digunakan di Laravel yaitu pasport dan sanctum.Keduanya menggunakan token untuk authentikasinya.

### Laravel Sanctum

Pertama, Sanctum adalah paket sederhana yang dapat Anda gunakan untuk menerbitkan token API kepada pengguna Anda tanpa kerumitan OAuth.

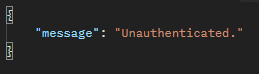
Kita bisa cek di composer.json apakah ada laravel sanctum, jika belum ada lakukan installasi terlebih dahulu

***composer require laravel/sanctum***

Untuk Menggunakan Sanctum kita cukup gunakan di middleware seperti berikut :



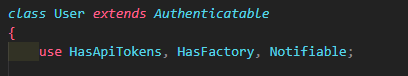
Maka jika ada user yang mencoba akses tanpa login akan di halangi oleh auth:sanctum.



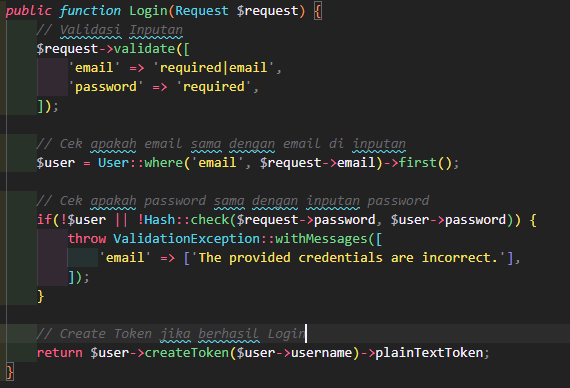
***Issuing API Token***

Pemberian token kepada pengguna yang mengauthentikasi permintaan HTTP. Simpelnya user akan diberikan token setelah berhasil melakukan login dan token tersebut dapat digunakan untuk mengakses Sebuah Post atau bahkan melakukan CRUD.

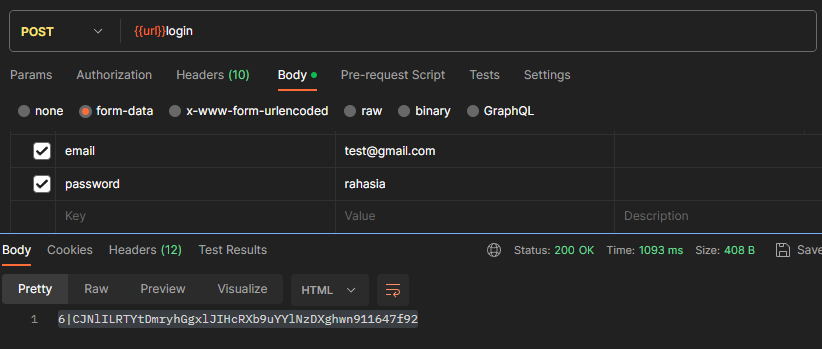
Biasanya di dalam Model kita sudah ada pemanggilan HasApiTokens, untuk meregenerate token.







Jika kita berhasil melakukan Login maka akan mengembalikkan sebuah token



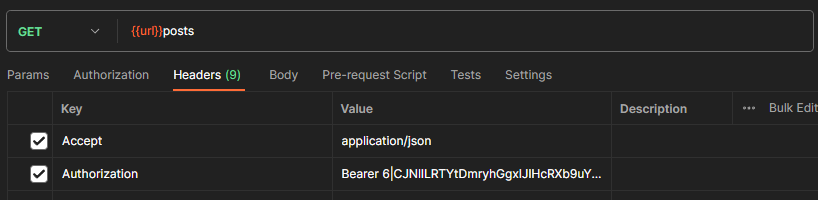
Token ini juga akan ada di dalam tabel database *personal\_access\_token*



Selanjutnya kita batasi akses untuk melihat data post, agar hanya user yang sudah login saja yang bisa melihat.



Di dalam Postman nanti jika user belum Login tidak akan bisa mengaksesnya, kita bisa mengakses posts jika kita sudah login, sebelumnya jangan lupa untuk menambahkan ini di dalam header



Di dalam Header kita tambahkan

***Accept: application/json*** = menyatakan bahwa klien meminta respons dari server dalam format JSON

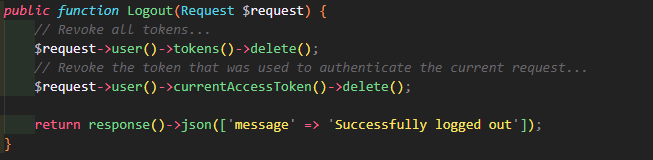
***Authorization: Bearer (kode token)*** = memberikan akses authorisasi dan menyatakan bahwa ia sudah melakukan login.

Dengan begini kita bisa melihat Post yang sudah diberikan keamanan login, selanjutnya adalah Logout dimana kita akan menghapus informasi token dan token yang sudah dihapus tidak akan berlaku lagi.

***Revoking Tokens*** (Mencabut Token dari database)



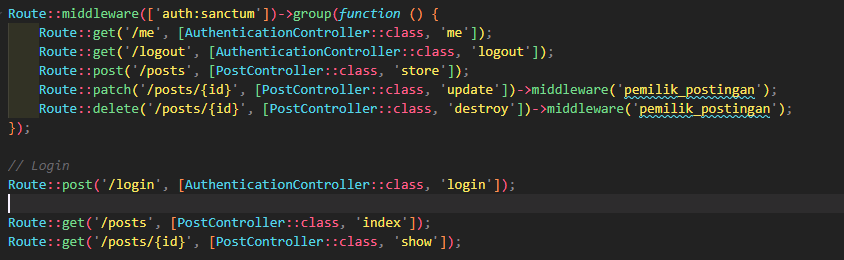
Di dalam controllernya kita lakukan delete pada token



*\*Perlu diketahui jika route diberikan middleware maka di dalam header postman harus diberikan Authorization: Bearer dan tokennya.*

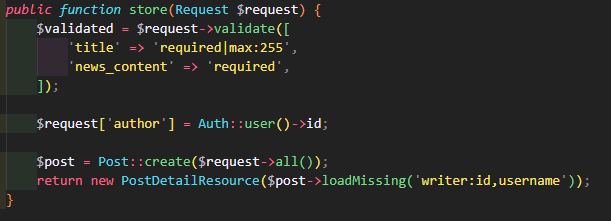
### CRUD API / (RESTful API)

Untuk Route Kita bungkus semuanya ke dalam route group karena banyak yang sama menggunakan middleware yang sama.



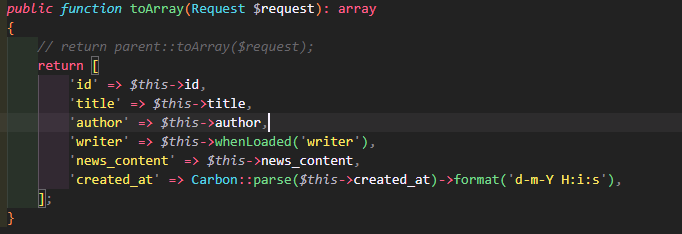
**Create Post**

Controller untuk melakukan Store, kita mengambil author dari user yang sedang login dan kita kembalikan Resource sambil mengirimkan data relationship writer

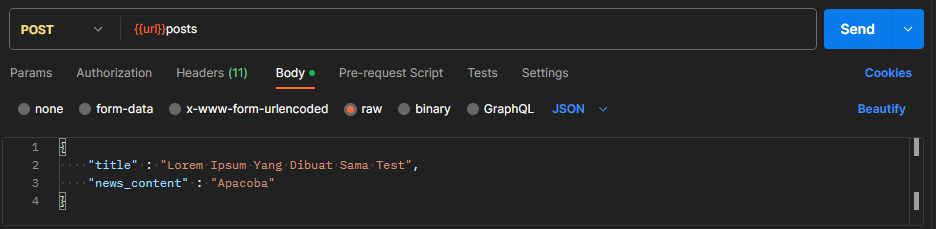




Data di resource nya akan menampilkan berupa data berikut



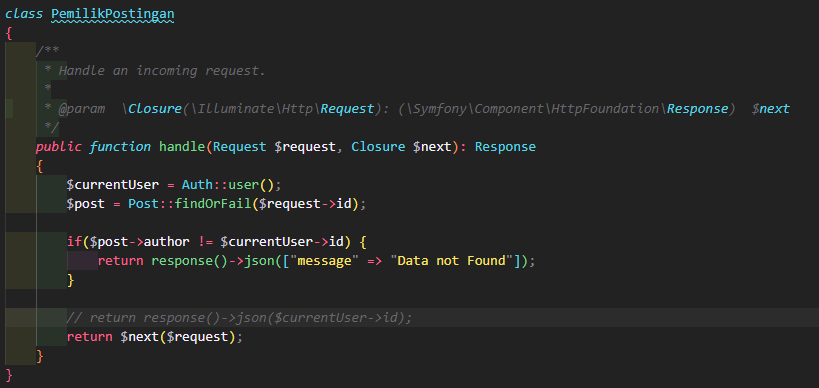
Untuk di Postman kita lakukan hal yang sama di dalam header nya. Untuk Menambahkan datanya kita bisa ke **body > raw**  data yang dituliskan juga harus dengan format **JSON.**



**Update Post**

Sebelumya kita buat sebuah middleware terlebih dahulu supaya, user yang bisa mengedit / bahkan menghapus datanya, adalah user asli yang membuat postingan itu. Sebagai contoh kita buat middleware seperti berikut

***php artisan make:middleware PemilikPostingan***



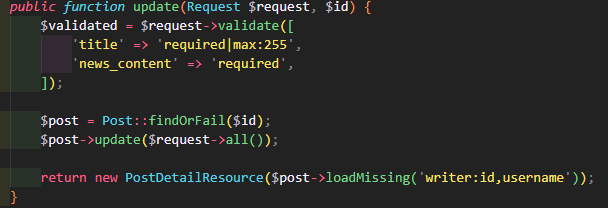
Jangan lupa di dalam kernel kita daftarkan middlware yang sudah kita buat

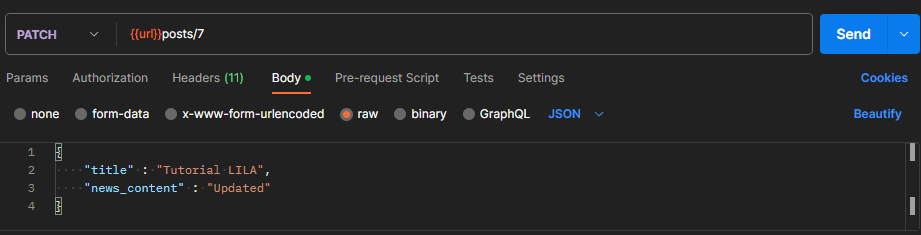


Dengan begitu kita bisa menggunakan middleware yang baru saja kita buat.



Untuk update kita lakukan hal yang hampir serupa dengan store, jangan lupa method nya diubah menjadi patch.

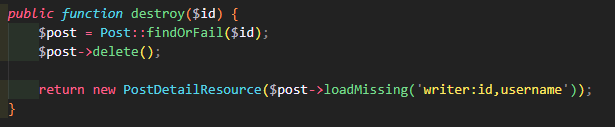


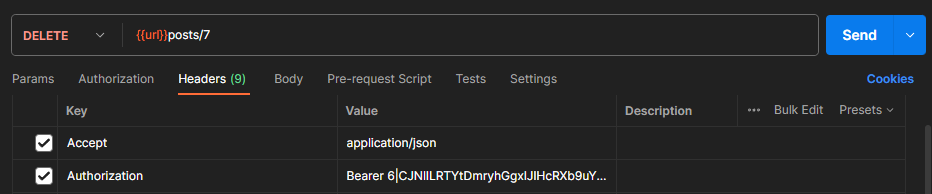


**Delete Post**

Delete hanya bisa dilakukan oleh user yang bersangkutan / user itu yang membuat postingan tersebut. Karena sebelumnya kita sudah membuat middleware dan mendaftarkannya kita cukup gunakan middleware di routenya.

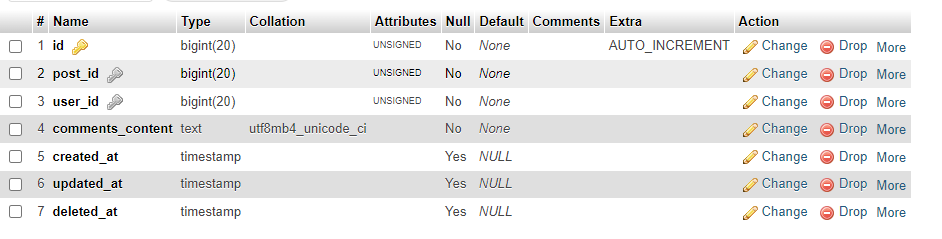
Berikut adalah Controller untuk melakukan delete



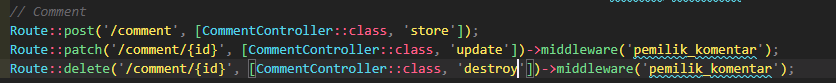


### CRUD Relationship

Sebelumnya kita sudah mempunyai rancangan database kita juga sudah ada tabel comments yang berupa pivot table, dimana itu akan digunakan sebagai relasi many to many.

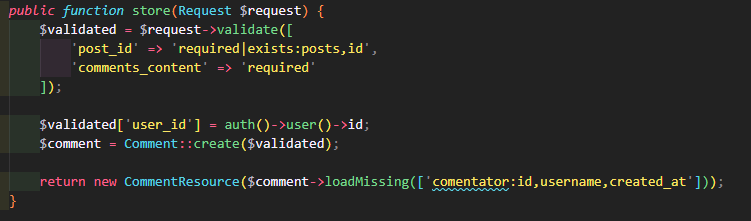


Hal ini hampir sama dengan melakukan crud sebelumnya, pertama kita buat routenya dan buat juga Controller untuk mengelola Komentar.



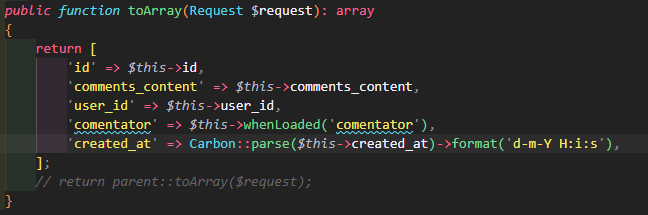
**Store**

Kita lakukan store untuk menambahkan data ke dalam tabel comments, disini kita mereturn resource baru sambil mengirimkan data Comentator.

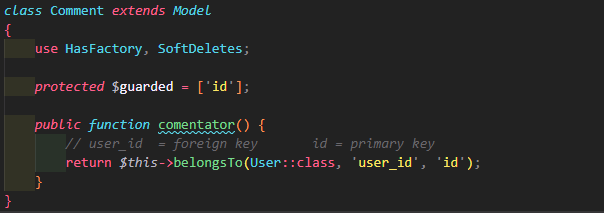


***php artisan make:resource CommentResource***

Di dalam Resource kita hanya menampilkan data berikut.



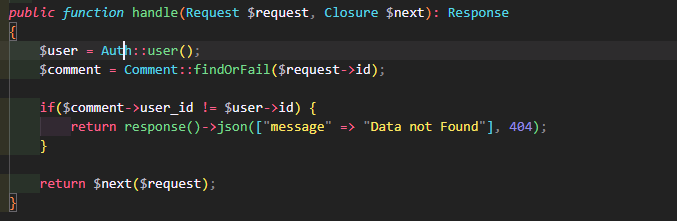
Sebelumnya jangan lupa kita untuk membuat Model Comment. Commentator adalah method yang mendefinisikan users disini, jadi 1 komentar itu dimiliki oleh setiap 1 user.



Karena di dalam Route terdapat middleware baru maka kita buat terlebih dahulu

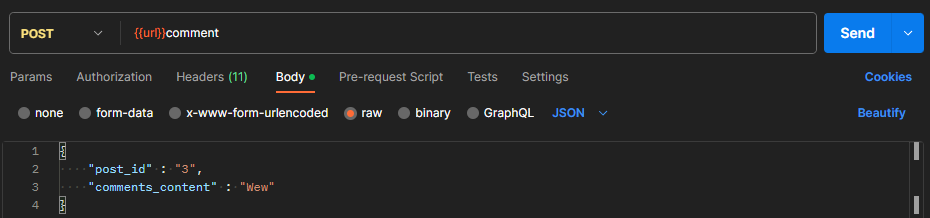
***php artisan make:middleware PemilikKomentar***

Disini kita lakukan pengecekkan, apakah user\_id komentar sama dengan user yang sedang login? Kalo iya maka beri akses agar dapat bisa melakukan CRUD. Dan jangan lupa untuk mendaftarkannya juga di kernel.





Disini bisa kita lakukan seperti biasa



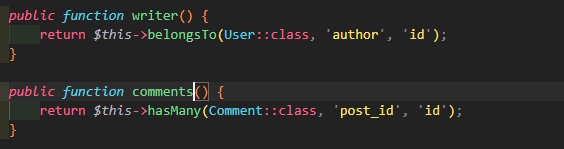
**Update & Delete**

Sebenarnya ini hampir sama dengan controller sebelumnya ketika kita ingin melakukan CRUD di Controller Posts.

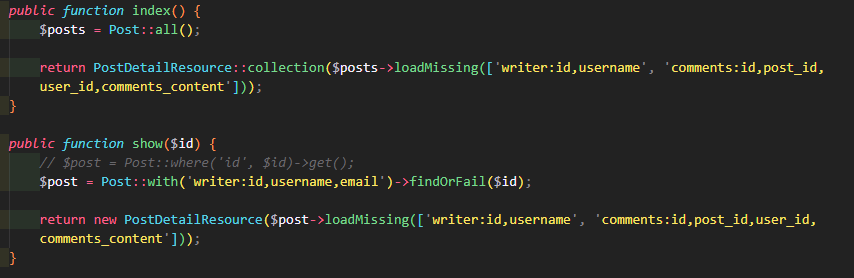


***Menampilkan Post dengan komentar & user-nya***

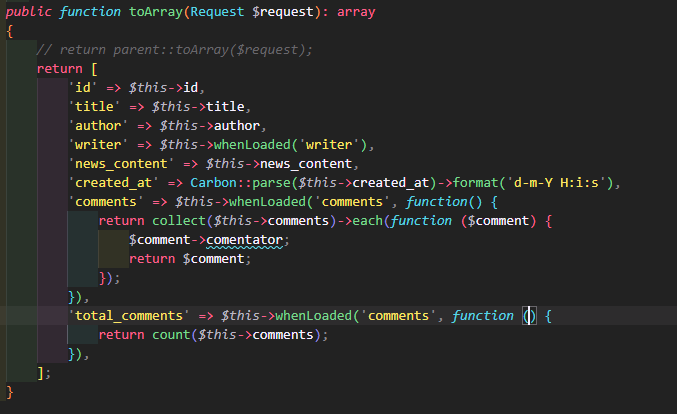
Di dalam model Post tambahkan method comments yang merepresentasikan komentar, dengan post\_id nya adalah foreign key dan id adalah primary key dari tabel Comment



Ketika kita ingin menampilkan post beserta komentarnya kita bisa kirimkan juga ke dalam Resource nya.



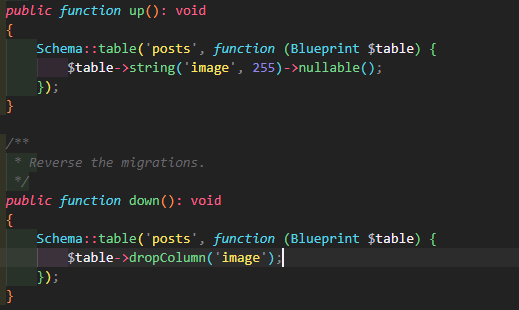
Di dalam Resource nya dapat kita ubah menjadi seperti berikut agar bisa menampilkan detail yang kita butuhkan.



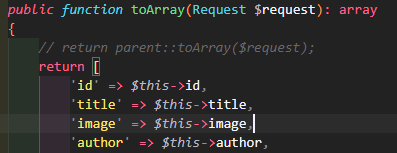
**Upload Image**

Karena di dalam database kita tidak ada kolom untuk menyimpan gambar maka kita sisipkan ke dalam tabel posts

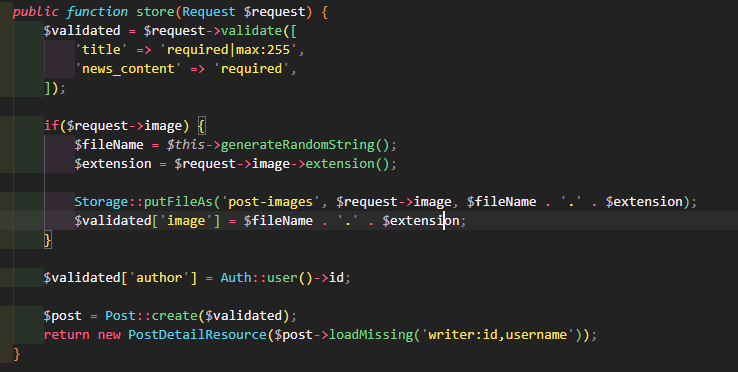
***php artisan make:migration add\_image\_to\_posts\_table –table=posts***

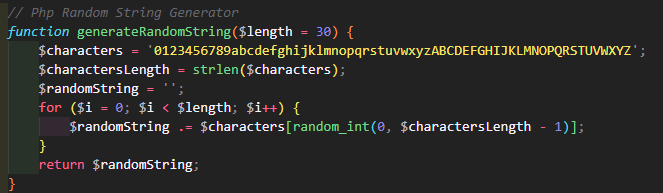


Langkah selanjutnya di dalam resource kita panggil, agar dapat ditampilkan hasilnya

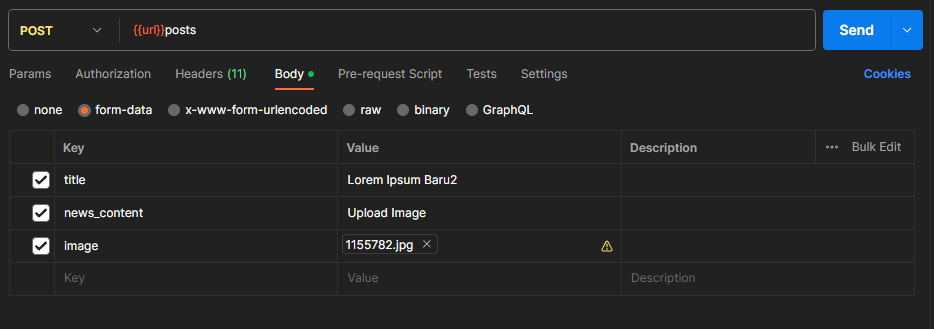


Sebelum itu kita harus menerima request untuk upload image nya terlebih dahulu seperti berikut.Dan menyimpan lokasi penyimpanan image ke folder yang kita buat





Lalu di postman kita cukup buat data baru dengan menggunakan form data bukan raw, lalu upload image.

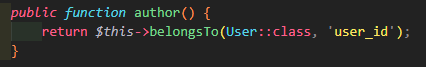


# Catatan :

Di dalam Models kita melakukan relasi dan memberikan nama pada method yang kita relasikan, misalkan: Kita mengidentifikasinya sebagai category bahwa ini sebagai penghubung relasi dari Model Category



Saat nanti di view nya kita memanggil relasinya $post->category->name itu akan bisa dan menampilkan nama category, jika nama method tidak sama dengan yang di panggil di viewnya maka akan error.



Sedangkan ini adalah contoh yang harusnya relasi dengan user diberi nama methodnya user. Karena jika nama method nya sama dengan nama modelnya maka secara otomatis akan didefinisikan oleh laravel, sehingga kita bisa akses semua.

Tapi jika kita melakukan costumize pada methodnya kita harus menambahkan parameter ‘user\_id yang menandakan bahwa ini merujuk ke foreignId yang ada di database / foreignId(‘user\_id’);

Biasanya 1 controller merepresentasikan 1 folder

Menambahkan class active di sidebar / navbar

