

LARAVEL MODEL & DATABASE

KELAS PROGRAMMING FULL STACK DEVELOPER

MITRA PELATIHAN



Jabar Digital Academy

digitalacademy.jabarprov.go.id

BAB V

LARAVEL MODEL DAN DATABASE

1. Tujuan

- a. Peserta didik dapat mengetahui apa itu model dalam Laravel.
- b. Peserta didik dapat mengetahui cara mengatur relasi antar tabel di dalam model.
- c. Peserta didik dapat mengetahui penggunaan model dalam controller.
- d. Peserta didik dapat mengetahui apa itu ORM eloquent.
- e. Peserta didik dapat mengetahui cara kerja migration database.
- f. Peserta didik dapat mempraktekan query sederhana menggunakan ORM.
- g. Peserta didik dapat mempraktekkan Create, Update dan Delete menggunakan ORM dan Query Builder.
- h. Peserta didik dapat mengetahui cara menghubungkan antara tabel yang berelasi.
- i. Peserta didik dapat mempraktekan studi kasus sederhana.

2. Perlengkapan

- a. Modul V. LARAVEL MODEL DAN DATABASE.
- b. IDE atau Teks Editor (Visual Studio Code, Notepad++, Sublime)
- c. Xampp.

3. Materi

Dalam pengembangan perangkat lunak berbasis web, Model adalah bagian dari pola desain MVC (Model-View-Controller) yang bertanggung jawab untuk mengelola data aplikasi. Model merepresentasikan struktur data, logika bisnis, dan aturan validasi yang terkait dengan data tersebut. Dengan menggunakan Model, pengembang dapat memisahkan logika aplikasi dari tampilan (View) dan kontrol (Controller), sehingga memungkinkan pengembangan yang lebih terorganisir dan terkelola.

A) Apa Itu Model.

Model adalah sebuah representasi dari sebuah tabel yang ada didalam database. Dengan model kita dapat melakukan operasi seperti menyimpan, membaca, menghapus, dan mengubah data yang ada di dalam tabel kita. Selain itu, model juga memungkinkan Anda untuk mendefinisikan hubungan antara tabel, sehingga Anda dapat mengakses data yang terkait dengan cara yang efisien. Semua model yang kita

miliki berada didalam folder `app/models`. Contoh model yang sudah dibuatkan secara default yaitu model `User`, untuk membuat model kita dapat menggunakan perintah artisan, contohnya seperti ini.

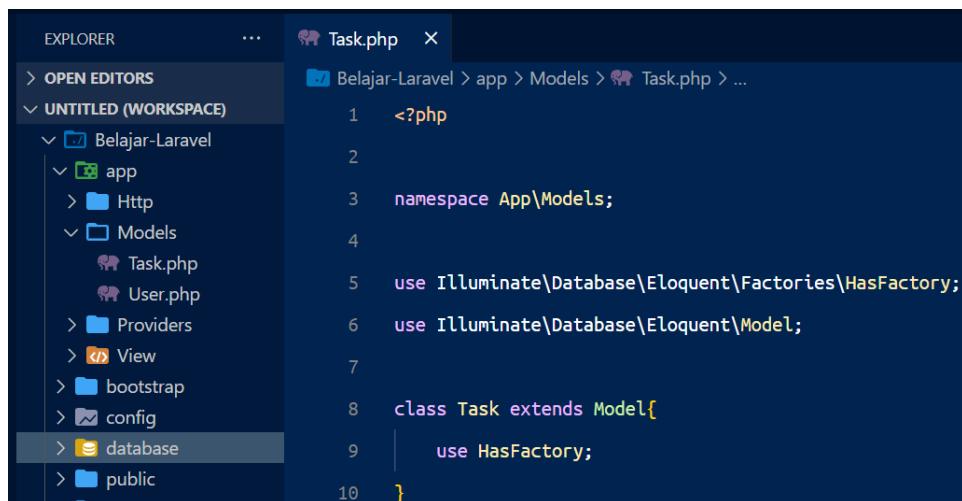
```
php artisan make:model (nama_model)
```

Kita coba buat sebuah aplikasi task management atau management tugas. Pertama kita buat model namanya `task`, nantinya kita akan mencoba hubungkan tabel ini dengan tabel lain, seperti ini.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan make:model Task
```

Perintah Artisan Untuk Membuat Model.

Jika sudah, bisa kita lihat di folder `app/models`, secara otomatis ditambahkan di dalam folder tersebut.



```
Task.php
Belajar-Laravel > app > Models > Task.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Task extends Model{
9      use HasFactory;
10 }
```

Model Yang Sudah Dibuat.

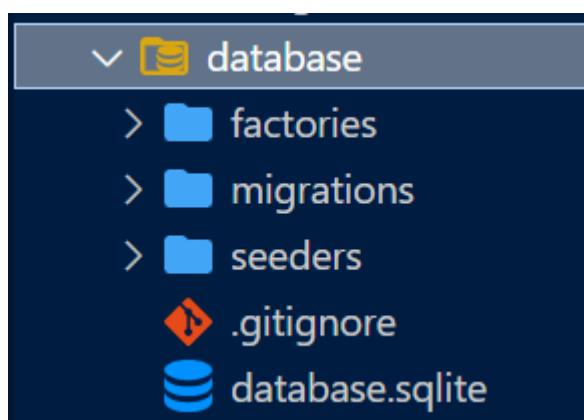
Ketika kita membuat sebuah model baru, maka secara otomatis akan ditambahkan `HasFactory` di dalamnya, model ini nantinya akan kita gunakan untuk menyimpan, mengambil, mengubah dan menghapus data didalam controller kita.

Di dalam model, terdapat beberapa variabel yang bisa kita gunakan untuk berbagai kebutuhan kita.

B) Database.

Pada saat kita pertama kali install Laravel, kita diminta untuk memilih database yang akan digunakan, ada banyak database SQL yang sudah didukung oleh laravel, contohnya MySQL, PostgreSQL, SQLite, SQL Server, dan Oracle.

Kita dapat atur konfigurasi koneksi Database di dalam file `.env` . Semua konfigurasi yang ada di dalam `env` akan diproses di dalam `config/database.php` pada aplikasi. Pada file tersebut, kita dapat menentukan semua koneksi database kita, serta menentukan koneksi mana yang harus digunakan secara default.



Folder Didalam Folder Database.

Terdapat 3 folder didalam folder database, yaitu migrations, seeders, dan factories. Migration adalah tempat kita mengatur tabel yang ada didalam database kita. Factories digunakan untuk membuat data dummy atau data palsu untuk kepentingan testing atau keperluan lainnya. Seeder digunakan untuk menentukan memasukkan data kedalam database yang bisa digunakan bersama dengan factories.

C) Migration.

Didalam laravel kita dapat membuat sebuah schema yang nantinya akan dijadikan sebagai tabel didalam database, kita bisa menentukan atribut beserta tipe data di dalamnya. Selain itu kita juga bisa mengubah struktur tabel yang sudah ada, sehingga setiap perubahan di dalam database terekam. Semua migration bisa kita lihat di dalam folder `database/migrations`.

Untuk membuat sebuah migration kita bisa menggunakan perintah artisan seperti ini.

```
php artisan make:migration (nama_migration)
```

contohnya kita coba buat tabel baru di dalam Task menggunakan perintah artisan seperti ini.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programming/Belajar-Laravel
$ php artisan make:migration task
```

Ketika kita jalankan maka bisa kita lihat pada file terbaru di dalam database/migrations. Format penamaan file untuk migration yaitu diawali dengan timestamps saat dimana migration dibuat lalu dibarengi dengan nama migrationnya, contohnya seperti ini.

```
// database/migrations/2024_04_22_072100_task.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration{
    public function up(): void{
        //
    }

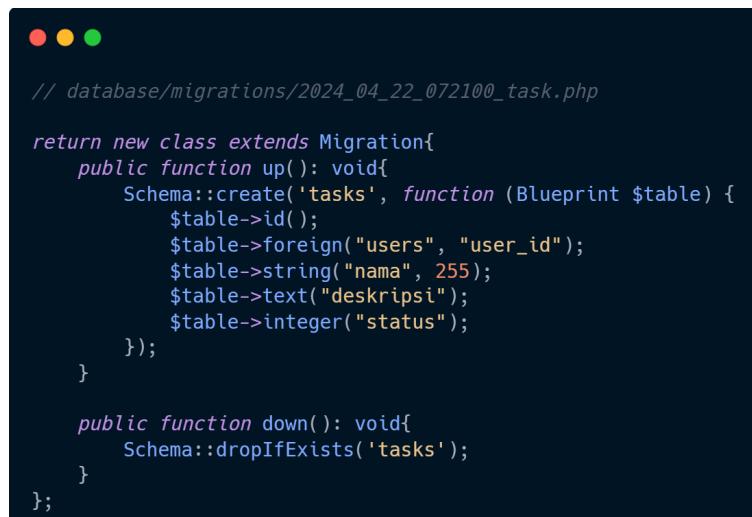
    public function down(): void{
        //
    }
};
```

Contoh Code Yang Berada Di Dalam Migration.

Didalam function run digunakan untuk menjalankan perintah migration seperti membuat tabel, menambahkan kolom, mengubah kolom. Sedangkan pada function

down biasanya diisi dengan perintah yang berlawanan dengan yang ada didalam up, contohnya menghapus tabel, atau kolom.

Di dalamnya kita definisikan tabel yang akan kita buat, seperti ini.



```
// database/migrations/2024_04_22_072100_task.php

return new class extends Migration{
    public function up(): void{
        Schema::create('tasks', function (Blueprint $table) {
            $table->id();
            $table->foreign("users", "user_id");
            $table->string("nama", 255);
            $table->text("deskripsi");
            $table->integer("status");
        });
    }

    public function down(): void{
        Schema::dropIfExists('tasks');
    }
};
```

Pembuatan Tabel Dengan Migration.

Pertama kita buat atribut yang berhubungan dengan table users, yaitu dengan nama atribut user_id. Selain itu beri juga nama dengan tipe string dan limit panjangnya yaitu 255, deskripsi kita beri text agar bisa diisi dengan nilai yang banyak, dan status kita beri integer namun nilai didalamnya yaitu 0 / 1.

Lalu didalam function down kita tambahkan dropIfExists agar ketika kita ingin melakukan rollback maka tabel akan dihapus, Kita juga bisa menambahkan tabel lebih dari satu didalam satu migration yang sama.

Lalu untuk menjalankan migration sudah kita buat yaitu dengan menjalankan perintah artisan di dalam terminal seperti ini.

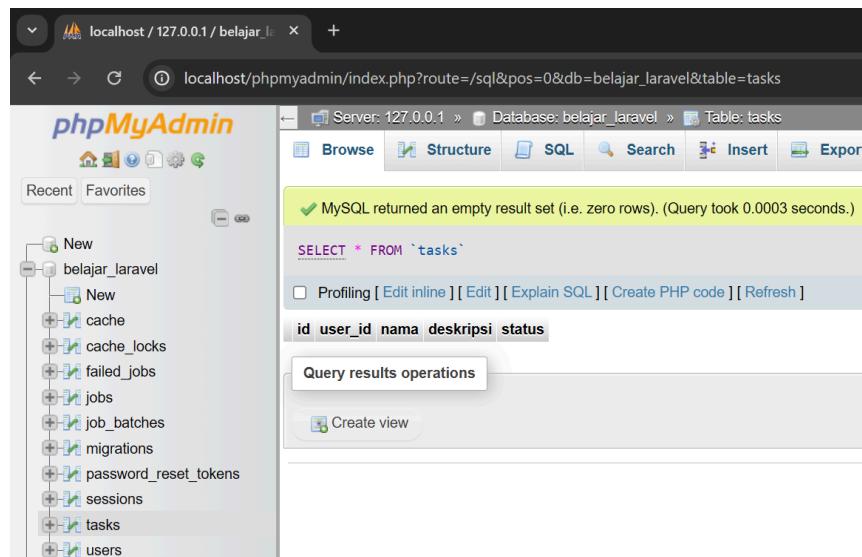
```
php artisan migrate
```

Jika sudah dijalankan maka akan terlihat seperti ini.

```
INFO Running migrations.  
2024_04_22_072100_task ..... 18.02ms DONE
```

Tampilan Ketika Berhasil Melakukan Migrate.

Lalu kita periksa di dalam database kita dengan menjalankan phpmyadmin kita didalam browser kita.



Memastikan Tabel Berhasil Ditambahkan.

Sekarang kita coba rollback tabel yang sudah kita tambah menggunakan migration di dalam database kita dengan perintah artisan seperti ini.

```
php artisan migrate:rollback
```

Perintah ini akan mengembalikan migrate terakhir, jadi jika sebelumnya hanya menambahkan tabel task saja. Harap perhatikan sebelum kita rollback jika terdapat data didalam tabel, maka ketika kita rollback maka data didalamnya akan hilang, Jika berhasil dijalankan maka akan terlihat seperti ini.

```
INFO Rolling back migrations.  
2024_04_22_072100_task ..... 14.39ms DONE
```

Kita dapat melakukan rollback semua tabel dengan perintah ini.

```
php artisan migrate:reset
```

Kita dapat melakukan rollback semua tabel lalu melakukan migrate ulang dengan perintah ini.

```
php artisan migrate:fresh
```

Jika kita jalankan maka akan terlihat seperti ini.

```
Dropping all tables ..... 138.32ms DONE
INFO Preparing database.

Creating migration table ..... 11.15ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 94.66ms DONE
0001_01_01_000001_create_cache_table ..... 18.70ms DONE
0001_01_01_000002_create_jobs_table ..... 69.41ms DONE
2024_04_22_072100_task ..... 8.96ms DONE
```

Tampilan Setelah Menjalankan Perintah Migrate Fresh.

Jika kita ingin melihat migration yang sudah dijalankan atau belum bisa menggunakan perintah ini.

```
php artisan migrate:status
```

Jika statusnya ran maka sudah dijalankan, kalau pending maka belum dijalankan.

Migration name	Batch / Status
0001_01_01_000000_create_users_table	Pending
0001_01_01_000001_create_cache_table	Pending
0001_01_01_000002_create_jobs_table	Pending
2024_04_22_072100_task	Pending

Tampilan Migration Yang Belum Dijalankan.

Migration name	Batch / Status
0001_01_01_000001_create_users_table	[1] Ran
0001_01_01_000001_create_cache_table	[1] Ran
0001_01_01_000002_create_jobs_table	[1] Ran
2024_04_22_072100_task	[1] Ran

Tampilan Migration Yang Sudah Dijalankan.

D) Factories.

Ketika kita melakukan testing aplikasi, terkadang kita perlu data yang harus ditampilkan, daripada kita membuat data manual kita dapat memanfaatkan factories dan seeder yang ada di dalam laravel untuk membuat data ke dalam database kita. Kita hanya perlu mendefinisikan data apa yang akan ditambahkan di dalam kelas yang ada di dalam factories.

Semua factory yang kita punya bisa kita lihat di dalam folder database/factories, Untuk membuat factory kita bisa buat dengan perintah artisan ini.

```
php artisan make:factory (nama_factory)
```

atau kita bisa tambahkan --model untuk mendefinisikan model yang akan dibuatkan factory.

```
php artisan make:factory (nama_factory) --model=(nama_model)
```

Kita coba membuat factory task dengan perintah artisan diatas.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan make:factory TaskFactory --model=Task
```

Perintah Artisan Untuk Membuat Factory.

Factory yang sudah kita buat secara otomatis dibuatkan didalam folder database/factories.

```
// database/factories/TaskFactory.php

use Illuminate\Database\Eloquent\Factories\Factory;

class TaskFactory extends Factory{
    public function definition(): array{
        return [
            //
        ];
    }
}
```

Class Factory Setelah Dibuat.

Kita bisa mendefinisikan jenis data yang akan ditambahkan kedalam database didalam return yang ada di dalam function definition. Sesuaikan key yang ada di dalam array dengan kolom atau atribut yang ada di dalam tabel, contohnya seperti ini.

```
// database/factories/TaskFactory.php

use Illuminate\Database\Eloquent\Factories\Factory;

class TaskFactory extends Factory{
    public function definition(): array{
        return [
            "user_id" => 1,
            "nama" => $this->faker->words(rand(5, 15), true),
            "deskripsi" => $this->faker->sentence(10),
            "status" => 0,
        ];
    }
}
```

Data Yang Akan Ditambahkan Kedalam Database.

Untuk memudahkan kita untuk membuat data, kita bisa buat menggunakan library faker agar kita tidak perlu tentukan nilainya sendiri. Factory dapat dijalankan di dalam seeder atau didalam controller.

E) Seeder.

Seeder adalah sebuah class yang digunakan untuk membuat data secara otomatis kedalam database. Dengan seeder kita bisa membuat banyak data di berbagai tabel hanya dengan menjalankan satu seeder, semua seeder yang kita miliki bisa kita lihat di dalam folder database/seeders. Jika ingin melihat contoh seeder, kita bisa melihatnya di dalam file database/seeders/DatabaseSeeder.php. Untuk membuat seeder bisa dengan perintah artisan seperti ini.

```
php artisan make:seeder (nama_seeder)
```

Kita coba buat seeder yang nantinya ketika kita jalankan data didalam tabel user dan task akan kita tambahkan data.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan make:seeder UserTaskSeeder
```

Perintah Artisan Untuk Membuat Seeder.

Jika sudah kita jalankan perintah artisan untuk membuat seeder, class seeder yang baru saja kita buat akan terlihat seperti ini.

```
// database/seeders/UserTaskSeeder.php

namespace Database\Seeders;

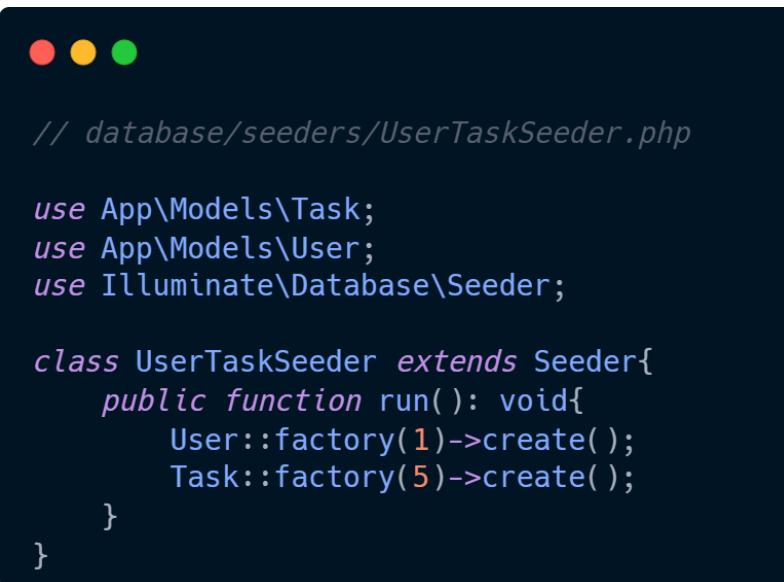
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class UserTaskSeeder extends Seeder{
    public function run(): void{
        //
    }
}
```

Contoh Class Seeder.

Ketika kita jalankan seeder maka code didalam function run yang akan dijalankan, maka dari itu kita dapat memanggil factory didalam function run untuk dibuatkan

data dan ditambahkan kedalam database. Kita juga dapat menjalankan banyak seeder dalam satu perintah, seperti ini.



```
// database/seeders/UserTaskSeeder.php

use App\Models\Task;
use App\Models\User;
use Illuminate\Database\Seeder;

class UserTaskSeeder extends Seeder{
    public function run(): void{
        User::factory(1)->create();
        Task::factory(5)->create();
    }
}
```

Pemanggilan Factory Didalam Seeder.

Kita akan coba buat satu data user dan lima data task didalam seeder ini. Untuk menjalankan seeder kita bisa gunakan perintah artisan seperti ini.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan db:seed UserTaskSeeder
```

Perintah Untuk Menjalankan Seeder Dengan Artisan.

Ketika sudah dijalankan, jika terlihat seperti ini maka data sudah ditambahkan.

```
INFO  Seeding database.
```

Informasi Seeder Yang Sedang Dijalankan.

Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.)

SELECT * FROM `tasks`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

	<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None			
Extra options										
	Edit	Copy	Delete	id	user_id	nama	deskripsi	status	created_at	updated_at
1	beatae quas quod porro odio qui dolorum unde	Atque illo debitis nisi reiciendis impedit quasi n...	0	2024-04-23 06:53:04	2024-04-23 06:53:04					
2	impedit asperiores nobis quae tempora amet et impe...	Fugit repellat iste aut aut voluptatem sed provide...	0	2024-04-23 06:53:04	2024-04-23 06:53:04					
3	1 quae dolorum quis nihil natus	Consequuntur in aut aut eligendi quibusdam nemo re...	0	2024-04-23 06:53:04	2024-04-23 06:53:04					
4	repellendus neque et autem quam at voluptatibus pr...	Voluptatem placeat impedit et officiis optio et au...	0	2024-04-23 06:53:04	2024-04-23 06:53:04					
5	1 laudantium aut delectus officiis voluptatem deleni...	Dolor iste non non id earum delectus.	0	2024-04-23 06:53:04	2024-04-23 06:53:04					

Data Didalam Tabel Tasks.

Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

SELECT * FROM `users`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

	<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table				
Extra options									
	Edit	Copy	Delete	id	name	email	email_verified_at	password	remember_token
1	Zoie D'Amore	kareem31@example.net	2024-04-23 06:53:03	\$2y\$12\$XtJLOuugGZeZdTpAF7AKkuawhqY0IogJPQNDaVFXNU... K7Yu1Sr.					

Data Didalam Tabel Users.

Kita sudah mencoba menjalankan seeder yang di dalamnya menambahkan data kedalam tabel users dan tasks menggunakan factory.

Didalam seeder kita bisa menjalankan banyak kelas seeder dalam satu kali perintah, untuk menjalankan banyak seeder di dalam satu perintah kita bisa menggunakan fungsi bernama call lalu di dalamnya beri array yang berisikan class seeder. Contohnya di dalam class DatabaseSeeder kita jalankan class UserTaskSeeder Seperti ini.

```
// database/seeders/DatabaseSeeder.php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder{
    public function run(): void{
        $this->call([
            UserTaskSeeder::class
        ]);
    }
}
```

Penggunaan Fungsi Call Didalam Seeder.

Saat kita jalankan, akan terlihat status setiap seeder sedang dijalankan, berbeda dengan yang sebelumnya tidak menampilkan status seeder yang sedang dijalankan karena didalamnya hanya memanggil factory.

```
[INFO] Seeding database.
Database\Seeders\UserTaskSeeder ..... RUNNING
Database\Seeders\UserTaskSeeder ..... 343 ms DONE
```

Tampilan Setelah Menjalankan Database Seeder.

Kita coba pisahkan seeder user dan task, lalu di dalamnya isi dengan factory seperti ini.

```
// database/seeders/UserSeeder.php

use App\Models\User;
use Illuminate\Database\Seeder;

class UserSeeder extends Seeder{
    public function run(): void{
        User::factory(1)->create();
    }
}
```

Tampilan Dari Class User Seeder.

```
// database/seeders/TaskSeeder.php

use App\Models\Task;
use Illuminate\Database\Seeder;

class TaskSeeder extends Seeder{
    public function run(): void{
        Task::factory(5)->create();
    }
}
```

Tampilan Dari Class Task Seeder.

Lalu didalam DatabaseSeeder kita panggil dua seeder yang sudah kita buat seperti ini.

```
// database/seeders/DatabaseSeeder.php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder{
    public function run(): void{
        $this->call([
            UserSeeder::class,
            TaskSeeder::class,
        ]);
    }
}
```

Tampilan Dari Class User Seeder.

Ketika jalankan maka akan terlihat seperti ini.

```
[INFO] Seeding database.

Database\Seeders\UserSeeder ..... RUNNING
Database\Seeders\UserSeeder ..... 340 ms DONE

Database\Seeders\TaskSeeder ..... RUNNING
Database\Seeders\TaskSeeder ..... 11 ms DONE
```

Kita bisa jalankan migration dan seeder secara bersamaan, dengan menambahkan --seeder, lalu diisi dengan class seeder yang akan dijalankan. Contohnya seperti ini.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan migrate:fresh --seeder=DatabaseSeeder
```

Perintah Artisan Migration Yang Dijalankan Bersamaan Dengan Seeder.

Ketika kita jalankan maka akan terlihat seperti ini.

```
Dropping all tables ..... 104.36ms DONE
INFO Preparing database.

Creating migration table ..... 13.85ms DONE
INFO Running migrations.

0001_01_01_000000_create_users_table ..... 95.90ms DONE
0001_01_01_000001_create_cache_table ..... 15.38ms DONE
0001_01_01_000002_create_jobs_table ..... 64.96ms DONE
2024_04_22_072100_task ..... 8.07ms DONE

INFO Seeding database.

Database\Seeders\UserSeeder ..... RUNNING
Database\Seeders\UserSeeder ..... 340 ms DONE

Database\Seeders\TaskSeeder ..... RUNNING
Database\Seeders\TaskSeeder ..... 11 ms DONE
```

Tampilan Setelah Perintah Artisan Dan Seeder Dijalankan.

F) Query Builder.

Query Builder adalah fitur yang memungkinkan kita untuk membuat sebuah query SQL untuk berinteraksi antara aplikasi kita dengan database. Query Builder dalam Laravel sangat fleksibel dan kuat, dan bisa digunakan dalam berbagai situasi, baik untuk kueri sederhana maupun kompleks.

Untuk membuat query builder kita bisa menggunakan library dari facades yaitu DB, dengan library ini kita bisa menentukan mulai dari tabel mana yang akan ditampilkan lalu diikuti dengan penambahan kondisi hingga mengurutkan dari terbesar atau terkecil. Contohnya kita ingin mengambil data yang ada didalam tabel users dengan library DB dari facades seperti ini.

```
// routes/web.php

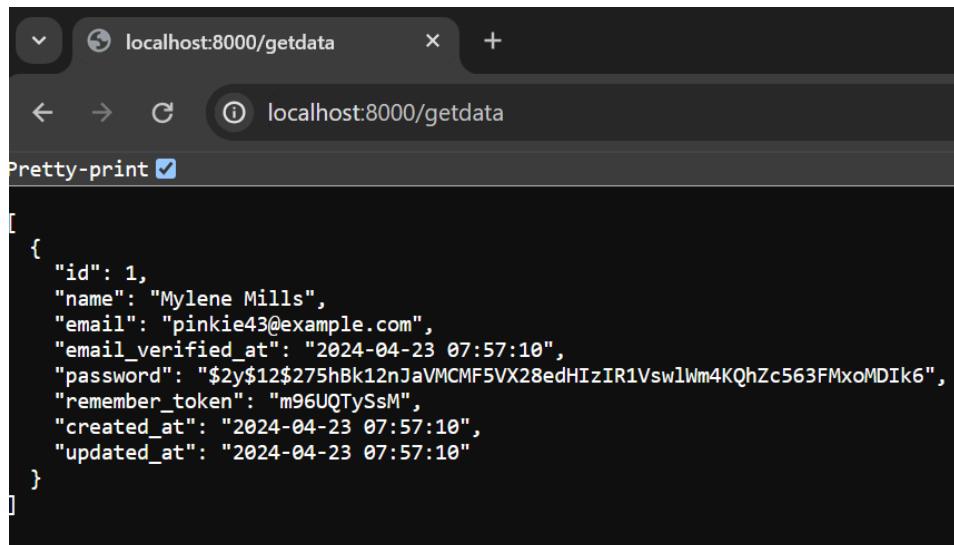
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $users = DB::table('users')->get();

    return $users;
});
```

Penggunaan Library DB Facades Untuk Mengambil Data.

Jika kita perhatikan kita memanggil fungsi get untuk mendapatkan datanya lalu disimpan didalam variabel. Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



A screenshot of a web browser window. The address bar shows "localhost:8000/getdata". The page content is a JSON response with "Pretty-print" checked. The JSON data is:

```
[{"id": 1, "name": "Mylene Mills", "email": "pinkie43@example.com", "email_verified_at": "2024-04-23 07:57:10", "password": "$2y$12$275hBk12nJaVMCMF5VX28edHIZIR1VswlWm4KQhZc563FMxoMDIk6", "remember_token": "m96UQTySsM", "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}]
```

Tampilan Data Yang Didapatkan Dari Database.

Kita sudah mencoba mengambil data dengan library DB facades. Kita bisa menambahkan kondisi agar data yang didapatkan sesuai dengan kondisi yang diberikan, contohnya kita coba seperti ini.

```
// routes/web.php

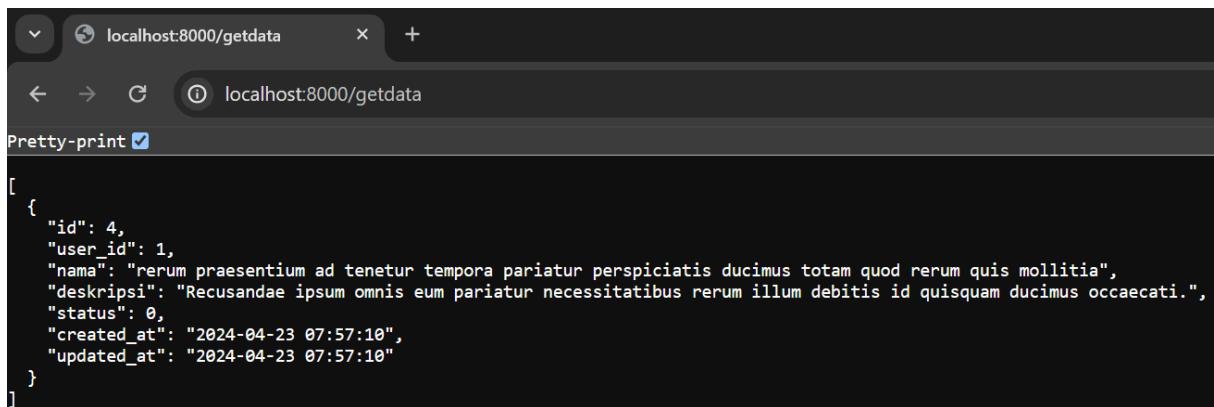
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $tasks = DB::table('tasks')->where("id", 4)->get();

    return $tasks;
});
```

Penambahan Kondisi Pada Pengambilan Data Dari Database.

Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



A screenshot of a web browser window. The address bar shows 'localhost:8000/getdata'. The page content is a JSON response with 'Pretty-print' checked. The JSON data is:

```
[{"id": 4, "user_id": 1, "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}]
```

Tampilan Data Yang Didapatkan Setelah Diberi Kondisi.

Selain kita bisa menambahkan kondisi, kita juga bisa mengurutkan dari terkecil atau terbesar dengan menggunakan fungsi order.

```
// routes/web.php

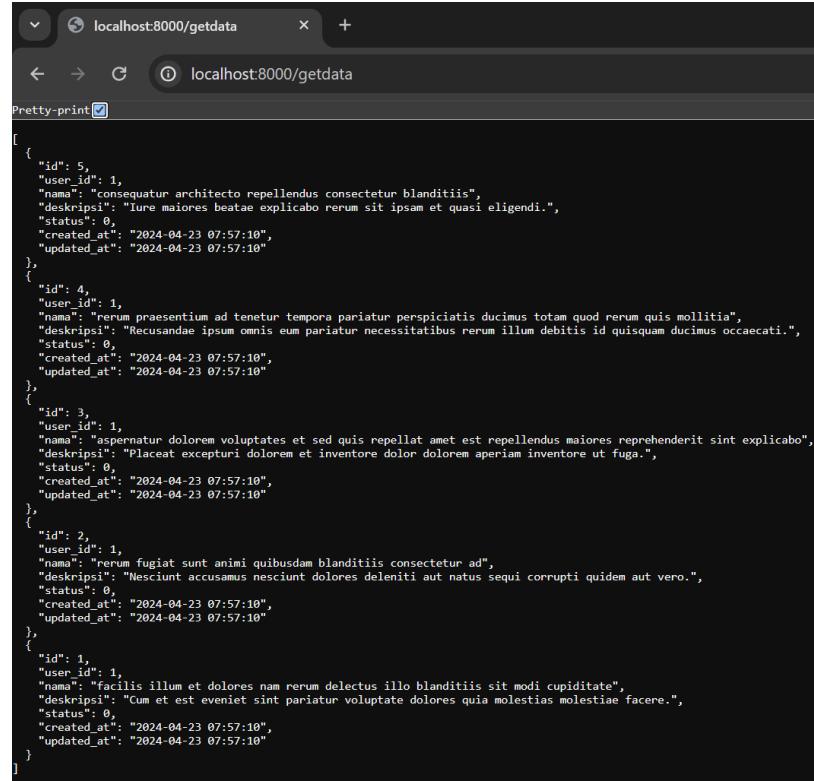
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $tasks = DB::table('tasks')->orderBy("id", "desc")->get();

    return $tasks;
});
```

Mengurutkan Data Dari Terbesar Ke Terkecil Dari Tabel ID.

Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



```
[{"id": 5, "user_id": 1, "nama": "consequatur architecto repellendus consectetur blanditiis", "deskripsi": "lure maiores beatae explicabo rerum sit ipsam et quasi eligendi.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 4, "user_id": 1, "nama": "orum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 3, "user_id": 1, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 2, "user_id": 1, "nama": "rerum fugiat sunt animi quibusdam blanditiis consectetur ad", "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 1, "user_id": 1, "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}]
```

Tampilan Setelah Diurutkan Berdasarkan ID Dari Terbesar.

Kita juga bisa memilih data apa saja yang akan diambil, yaitu dengan select lalu di dalamnya diisi dengan kolom mana yang ingin ditampilkan. Contohnya seperti ini.



```
// routes/web.php

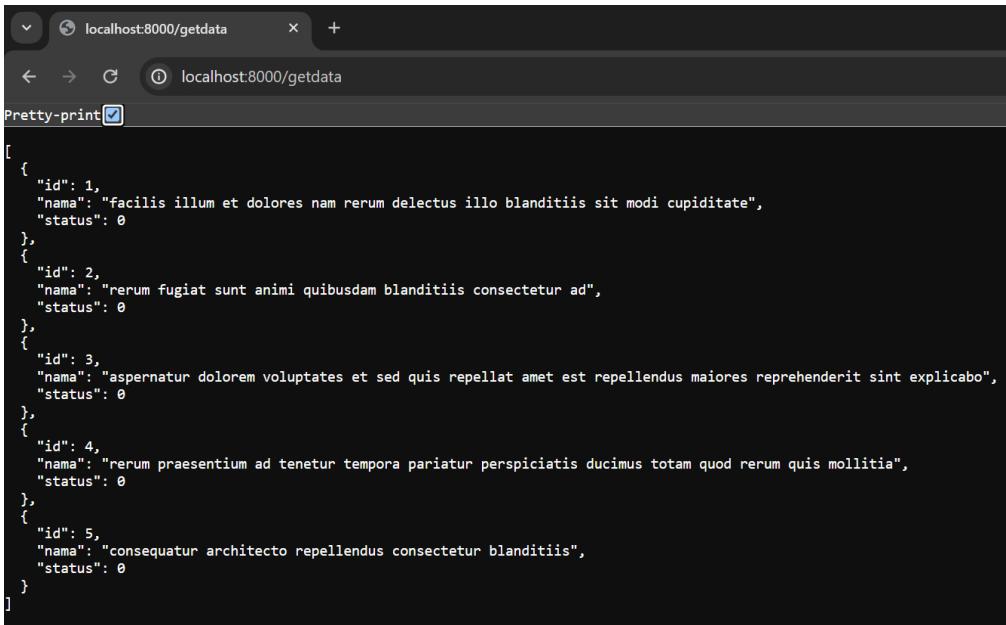
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $tasks = DB::table("tasks")->select(["id", "nama", "status"])->get();

    return $tasks;
});
```

Penambahan Select Pada Saat Mengambil Data.

Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



A screenshot of a web browser window titled "localhost:8000/getdata". The browser has a dark theme. In the address bar, it shows "localhost:8000/getdata". Below the address bar, there is a "Pretty-print" checkbox which is checked. The main content area displays a JSON array with five elements, each representing a task with fields id, nama, and status. The JSON is formatted with indentation and line breaks for readability.

```
[{"id": 1, "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "status": 0}, {"id": 2, "nama": "rerum fugiat sunt animi quibusdam blanditiis consectetur ad", "status": 0}, {"id": 3, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "status": 0}, {"id": 4, "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "status": 0}, {"id": 5, "nama": "consequatur architecto repellendus consectetur blanditiis", "status": 0}]
```

Tampilan Setelah Diberi Select.

Selain kita bisa menggunakan Query Builder yang telah disediakan Laravel, kita dapat menggunakan Raw Query untuk menuliskan Query SQL secara langsung ke database. Penggunaan Raw Query berguna ketika Anda perlu menulis kueri yang kompleks. Penulisan Raw query bisa kita lakukan dengan menggunakan fungsi select. contohnya seperti ini.



A screenshot of a terminal window with a dark background. At the top, there are three colored window control buttons (red, yellow, green). The terminal window contains PHP code for a route definition. It includes the use statement for the Illuminate\Support\Facades\DB facade, the Route::get() method, and the DB::select() method to execute a raw SQL query.

```
// routes/web.php

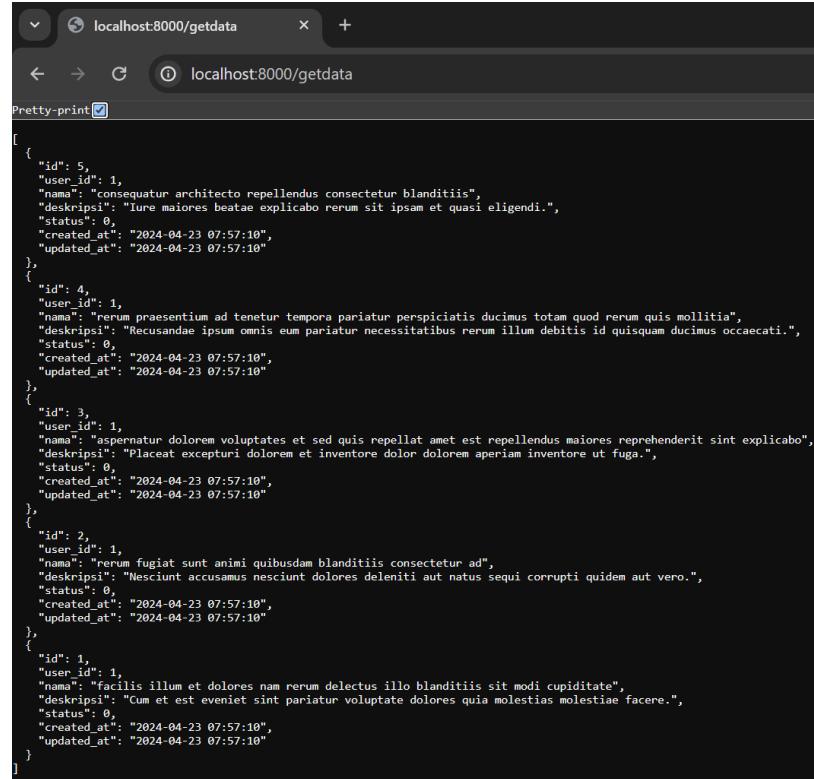
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $tasks = DB::select("SELECT * FROM tasks");

    return $tasks;
});
```

Pengambilan Data Dari Database Menggunakan Raw

Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



```
[{"id": 5, "user_id": 1, "nama": "consequatur architecto repellendus consectetur blanditiis", "deskripsi": "lure maiores beatae explicabo rerum sit ipsam et quasi eligendi.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 4, "user_id": 1, "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 3, "user_id": 1, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 2, "user_id": 1, "nama": "rerum fugiat sunt animi quibusdam blanditiis consectetur ad", "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}, {"id": 1, "user_id": 1, "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}]
```

Hasil Dari Pengambilan Data Menggunakan Raw Query.

Kita juga bisa menambahkan kondisi di dalamnya sama seperti kita menuliskan Query SQL, seperti ini.



```
// routes/web.php

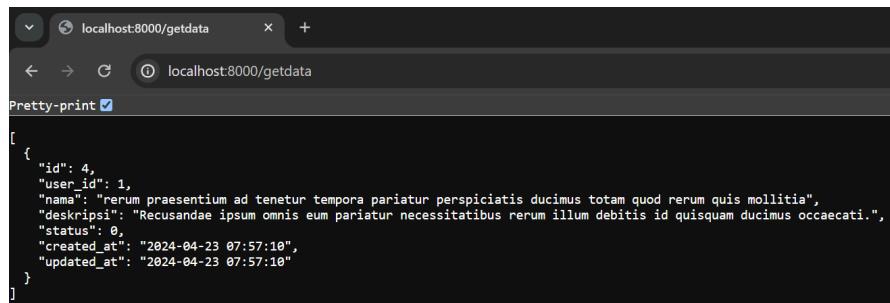
use Illuminate\Support\Facades\DB;

Route::get("/getdata", function(){
    $tasks = DB::select("SELECT * FROM tasks WHERE id = 4");

    return $tasks;
});
```

Penambahan Kondisi Pada Query Raw.

Untuk melihat data yang kita dapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



A screenshot of a web browser window titled "localhost:8000/getdata". The page content is a JSON array with one element, displayed in a "Pretty-print" format. The JSON data is as follows:

```
[{"id": 4, "user_id": 1, "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23 07:57:10", "updated_at": "2024-04-23 07:57:10"}]
```

Pengambilan Data Menggunakan Raw Query Setelah Diberi Kondisi.

G) Eloquent ORM.

Untuk berinteraksi dengan database, selain menggunakan Query Builder, kita bisa memanfaatkan Eloquent ORM berinteraksi dengan database kita. Dengan Eloquent kita bisa memanfaatkan model untuk berinteraksi dengan tabel yang ada didalam database.

Untuk menggunakan Eloquent ORM kita cukup panggil modelnya lalu jalankan perintah yang ingin didapatkan, contohnya kita ingin mengambil semua data yang ada di dalam tabel tasks, kita bisa gunakan model tasks lalu panggil fungsi `all` untuk mengambil semua datanya.



```
// routes/web.php

use App\Models\Task;

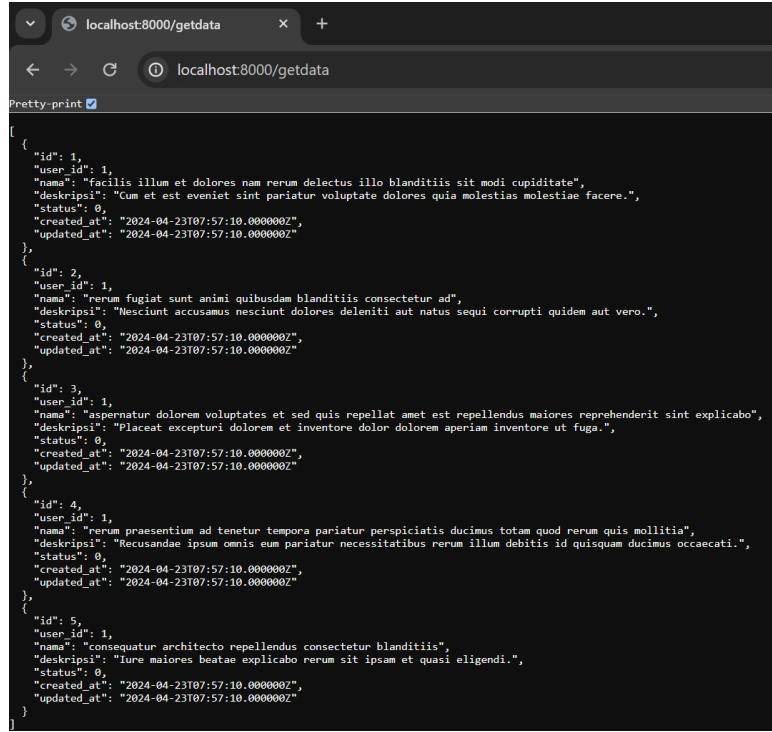
Route::get("/getdata", function(){
    $tasks = Task::all();

    return $tasks;
});
```

Pengambilan Data Menggunakan Eloquent ORM.

Dengan eloquent kita tidak perlu definisikan tabel mana yang akan kita ambil datanya, karena sudah didefinisikan secara otomatis di dalam model yang kita

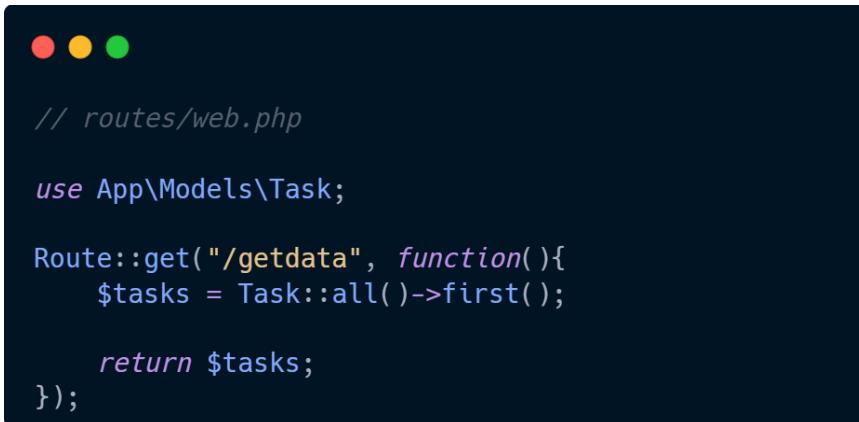
panggil. Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



```
[{"id": 1, "user_id": 1, "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 2, "user_id": 1, "nama": "rerum fugiat sunt animi quibusdam blanditiis consectetur ad", "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 3, "user_id": 1, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 4, "user_id": 1, "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 5, "user_id": 1, "nama": "consequatur architecto repellendus consectetur blanditiis", "deskripsi": "Iure maiores beatae explicabo rerum sit ipsam et quasi eligendi.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}]
```

Hasil Dari Pengambilan Data Menggunakan Eloquent.

Kita bisa dapatkan data pertama dari data yang kita dapatkan dari `all` dengan menggunakan fungsi `first`.



```
// routes/web.php

use App\Models\Task;

Route::get("/getdata", function(){
    $tasks = Task::all()->first();

    return $tasks;
});
```

Pengambilan Data Menggunakan First.

Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.

kita bisa membuat Query Builder di dalam Eloquent ORM, yaitu dengan menambahkan fungsi `query` setelah model dipanggil, contoh pengambilan semua data menggunakan query builder dalam eloquent seperti ini.



```
// routes/web.php

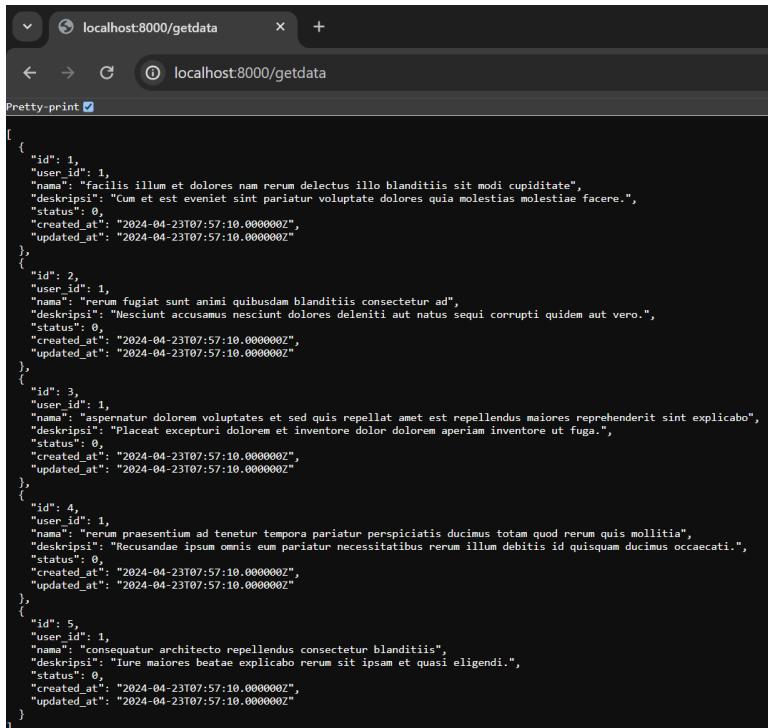
use App\Models\Task;

Route::get("/getdata", function(){
    $tasks = Task::query()->get();

    return $tasks;
});
```

Membuat Query Builder Di Dalam Eloquent.

Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



```
[{"id": 1, "user_id": 1, "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 2, "user_id": 1, "nama": "rerum fugiat sunt animi quibusdam blanditiis consequetur ad", "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 3, "user_id": 1, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 4, "user_id": 1, "nama": "ut rerum praesentium ad tenetur tempora pariatur perspicatis ducimus totam quod rerum quis mollitia", "deskripsi": "Reculsandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 5, "user_id": 1, "nama": "consequatur architecto repellendus consequetur blanditiis", "deskripsi": "Iure maiores beatae explicabo rerum sit ipsam et quasi eligendi.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}]
```

Hasil Dari Pengambilan Data Menggunakan Query Builder Di Dalam Eloquent.

Sama seperti kita membuat Query Builder kita bisa tambahkan banyak fungsi seperti where, orderBy, dan lain lainnya. Contohnya seperti ini.



```
// routes/web.php

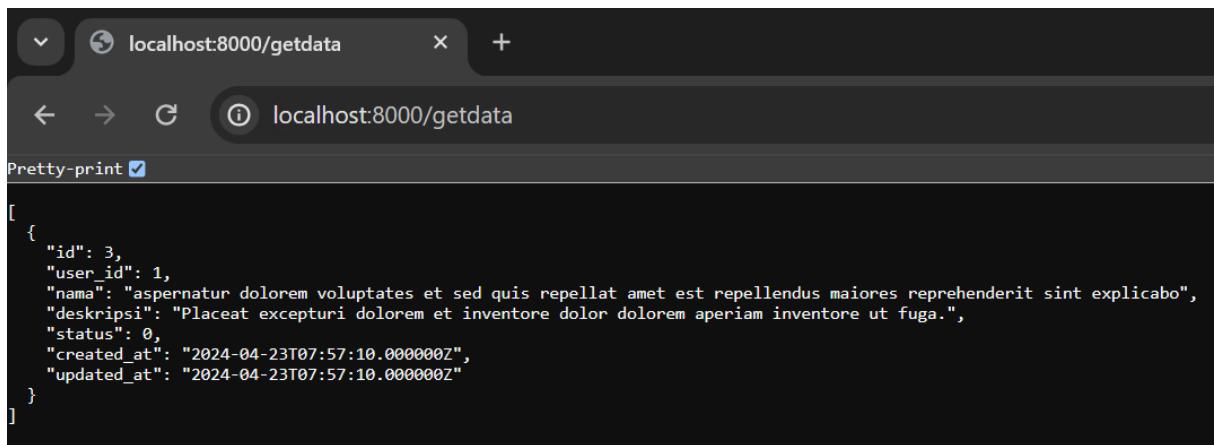
use App\Models\Task;

Route::get("/getdata", function(){
    $tasks = Task::query()->where("id", 3)->get();

    return $tasks;
});
```

Penambahan Kondisi Didalam Query Builder.

Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



Pretty-print

```
[{"id": 3, "user_id": 1, "nama": "aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}]
```

Hasil Dari Pengambilan Data Menggunakan Query Builder Setelah Diberi Kondisi.

H) Insert, Update, Delete

Terkadang kita perlu menyimpan data kita ke dalam database, ada banyak cara untuk menyimpan data didalam laravel, kita bisa menggunakan query builder atau eloquent. Contoh menyimpan data menggunakan query builder seperti ini.

```
// routes/web.php

use Illuminate\Support\Facades\DB;

Route::get("/insertdata", function(){
    $data = [
        "name" => "Fulan",
        "email" => "fulan@gmail.com",
        "password" => "12345678"
    ];

    $users = DB::table("users")->insert($data);

    return "Data Telah Ditambahkan!!";
});
```

Menyimpan Data Menggunakan Query Builder.

Ketika kita jalankan route maka akan terlihat pesan bahwa data telah ditambahkan,

The screenshot shows a MySQL database interface with the 'users' table selected. The table has columns: id, name, email, email_verified_at, password, remember_token, created_at, and updated_at. There are two rows of data:

	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input checked="" type="checkbox"/> Delete	1	Mylene Mills	pinkie43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5VX28edHlzlR1VswIWm4KQhZc56...	m96UQTySsM	2024-04-23 07:57:10	2024-04-23 07:57:10
	<input type="checkbox"/>	<input checked="" type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input checked="" type="checkbox"/> Delete	2	Fulan	fulan@gmail.com	NULL	12345678	NULL	NULL	NULL

Data Yang Sudah Disimpan Menggunakan Query Builder.

Kita sudah mencoba menyimpan data menggunakan query builder. Untuk menyimpan data menggunakan eloquent kita bisa menggunakan fungsi bernama save. Contohnya seperti ini.

```
// routes/web.php

use Illuminate\Support\Facades\DB;

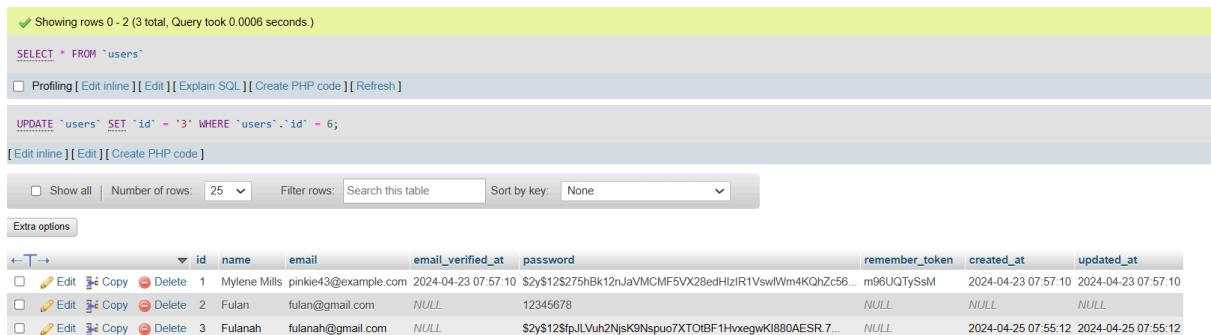
Route::get("/insertdata", function(){
    $user = new User();

    $user->name = "Fulanah";
    $user->email = "fulanah@gmail.com";
    $user->password = "12345678";

    $user->save();
    return "Data Telah Ditambahkan!!";
});
```

Menyimpan Data Menggunakan Eloquent.

Ketika kita jalankan route maka akan terlihat pesan bahwa data telah ditambahkan,



The screenshot shows a MySQL database interface with the following details:

- Showing rows 0 - 2 (3 total, Query took 0.0006 seconds.)**
- SELECT * FROM `users`**
- UPDATE `users` SET `id` = '3' WHERE `users`.`id` = 6;**
- Number of rows: 25**
- Filter rows: Search this table**
- Sort by key: None**
- Extra options:** Show all, Number of rows: 25, Filter rows: Search this table, Sort by key: None
- Table Headers:** id, name, email, email_verified_at, password, remember_token, created_at, updated_at
- Data Rows:**

<input type="checkbox"/>	Edit	Copy	Delete	1	Mylene Mills	pinkie43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5VX28edHlzlR1VswlWm4KQhZc56... m96UQTySsM	2024-04-23 07:57:10	2024-04-23 07:57:10
<input type="checkbox"/>	Edit	Copy	Delete	2	Fulan	fulan@gmail.com	NULL	12345678	NULL	NULL
<input type="checkbox"/>	Edit	Copy	Delete	3	Fulanah	fulanah@gmail.com	NULL	\$2y\$12\$pJLVuh2NjsK9Nspuo7XT0IBF1hvsegwKI880AESR.7...	NULL	2024-04-25 07:55:12

Data Yang Sudah Disimpan Query Builder.

Kita sudah mencoba menyimpan data menggunakan Query Builder dan Eloquent. Selain kita bisa menambahkan atau menyimpan data, kita juga bisa mengubah data yang ada di dalam database dengan menggunakan Query Builder dan Eloquent. Untuk melakukan Update data dengan Query Builder kita bisa tambahkan kondisi untuk mengidentifikasi data mana yang akan diubah, lalu kita gunakan fungsi update untuk mengubah data, contohnya seperti ini.

```
// routes/web.php

use Illuminate\Support\Facades\DB;

Route::get("/updatedata", function(){
    $data = [
        "name" => "fulan updated",
        "email" => "fulan@gmail.com",
        "password" => '11223344',
    ];

    DB::table("users")->where("id", 2)->update($data);

    return "Data Telah Diubah!!";
});
```

Mengubah Data Menggunakan Query Builder.

Ketika kita jalankan route yang telah kita buat di browser maka akan terlihat pesan dan data akan diubah.

	<input type="checkbox"/>	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	1	Mylene Mills	pinkie43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5VX28edHlzIR1VswlWm4KQhZc56... m96UQTySsM	2024-04-23 07:57:10	2024-04-23 07:57:10
	<input type="checkbox"/>	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	2	fulan updated	fulan@gmail.com	NULL	11223344	NULL	NULL
	<input type="checkbox"/>	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	3	Fulanah	fulanah@gmail.com	NULL	\$2y\$12\$pJLVuh2NjsK9Nspuo7XTObF1HxegwKi880AESR.7...	NULL	2024-04-25 07:55:12
	<input type="checkbox"/>	<input type="checkbox"/> Check all	With selected:	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	<input type="checkbox"/> Export				

Tampilan Data Yang Sudah Diubah Menggunakan Query Builder.

```
// routes/web.php

use App\Models\User;

Route::get("/updatedata", function(){
    $user = User::find(3);

    $user->name = "fulanah updated";
    $user->email = "fulanah@gmail.com";
    $user->password = "11223344";

    $user->save();

    return "Data Telah Diubah!!";
});
```

Mengubah Data Menggunakan Eloquent.

Untuk mengubah data kita fungsi yang sama seperti kita menambahkan data, yaitu fungsi `save` untuk menyimpan perubahan data yang terjadi, namun untuk mengidentifikasi data mana yang akan diubah kita ambil menggunakan fungsi `find`.

Ketika kita jalankan route yang telah kita buat di browser maka akan terlihat pesan dan data akan diubah.

Showing rows 0 - 2 (3 total). Query took 0.0008 seconds.

SELECT * FROM `users`											
<input type="checkbox"/> Profiling Edit inline Edit Explain SQL Create PHP code Refresh											
<input type="checkbox"/> Show all		Number of rows:		25		Filter rows:		Search this table		Sort by key: None	
Extra options											
	#	id	name	email	email_verified_at	password	remember_token	created_at	updated_at		
<input type="checkbox"/>	1	Mylene Mills	pinkas43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5Vx28edHlzIR1VswWm4KQhZc56...	m96UJQtYsS...	2024-04-23 07:57:10	2024-04-23 07:57:10			
<input type="checkbox"/>	2	fulan updated	fulan@gmail.com		NULL	11223344	NULL	NULL	NULL		
<input type="checkbox"/>	3	fulanah updated	fulanah@gmail.com		NULL	\$2y\$12\$Zr23WsIUF.mTxxtSWeaRuXYBd.vfEPrxwPp/agVl0...	NULL	2024-04-25 07:55:12	2024-04-25 08:45:45		
Check all With selected Edit Copy Delete Export											

Data Yang Sudah Diubah Menggunakan Eloquent.

Kita sudah mencoba mengubah dan menambahkan data, Selain menambah dan mengubah data, kita juga dapat menghapus data yang ada dengan menggunakan Query Builder dan Eloquent. Contoh menghapus data menggunakan Query Builder itu seperti ini.

```
// routes/web.php

use Illuminate\Support\Facades\DB;

Route::get("/updatedata", function() {
    DB::table("users")->delete(3);

    return "Data Telah Dihapus!!";
});
```

Menghapus Data Menggunakan Query Builder.

Saat kita menghapus data, jika kita menggunakan ID untuk menentukan data mana yang akan hapus, kita bisa tentukan ID nya di dalam argumen di dalam fungsi delete, kita juga bisa menambahkan where untuk memberikan kondisi tertentu sebelum delete dijalankan. Ketika kita jalankan route yang telah kita buat di browser maka akan terlihat pesan dan data akan dihapus.

Showing rows 0 - 1 (total, Query took 0.0004 seconds.)

[SELECT * FROM `users`](#)

Profiling | [Edit inline](#) | [Edit](#) | [Explain SQL](#) | [Create PHP code](#) | [Refresh](#)

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	Edit	Copy	Delete	id	name	email	email_verified_at	password	remember_token	created_at	updated_at
<input type="checkbox"/>	Edit	Copy	Delete	1	Mylene Mills	pinkie43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5VX28edHzlR1VswlWm4KQhZc56... m96UQTySsM		2024-04-23 07:57:10	2024-04-23 07:57:10
<input type="checkbox"/>	Edit	Copy	Delete	2	fulan updated	fulan@gmail.com	NULL	11223344	NULL	NULL	NULL

[With selected...](#) [Edit](#) [Copy](#) [Delete](#) [Export](#)

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

[Print](#) [Copy to clipboard](#) [Export](#) [Display chart](#) [Create view](#)

Tampilan Data Yang Telah Dihapus Menggunakan Query Builder.

Selain kita menggunakan Query Builder, kita juga bisa menggunakan Eloquent ORM. Contohnya seperti ini.

```
// routes/web.php

use App\Models\User;

Route::get("/deletedata", function(){
    User::destroy(2);

    return "Data Telah Dihapus!!";
});
```

Menghapus Data Menggunakan Eloquent.

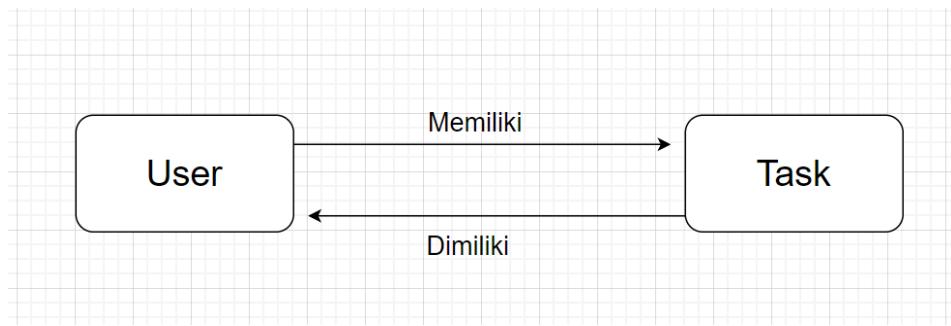
Sama seperti delete pada Query Builder, kita masukkan id dari data yang akan dihapus di dalam fungsi bernama `destroy`, Ketika kita jalankan route yang telah kita buat di browser maka akan terlihat pesan dan data akan dihapus.

	<input type="checkbox"/> Edit	<input checked="" type="checkbox"/> Copy	<input type="checkbox"/> Delete	id	name	email	email_verified_at	password	remember_token	created_at	updated_at
				1	Mylene Mills	pinkie43@example.com	2024-04-23 07:57:10	\$2y\$12\$275hBk12nJaVMCMF5VX28edHlzIR1VswlWm4KQhZc56...	m96UQTySsM	2024-04-23 07:57:10	2024-04-23 07:57:10

Tampilan Data Yang Telah Dihapus Menggunakan Eloquent.

I) Relationship.

Saat kita mengembangkan aplikasi web, seringkali kita akan menemukan situasi di mana tabel dalam database memiliki hubungan satu sama lain. Didalam laravel kita dapat mengatur relasi antar tabel di dalam model. Kita coba buat hubungan relasi antara User dengan Task, jadi satu User akan memiliki banyak Task, sedangkan satu Task hanya dimiliki satu user saja.



Gambaran Hubungan Relasi Antar Model.

untuk menghubungkan relasi antar model kita dapat buat sebuah function lalu beri nama sesuai dengan nama model yang akan dihubungkan, Didalamnya kita tentukan kolom mana yang ingin dijadikan `foreign key`, dan juga tentukan `id` yang kita jadikan sebagai `key` yang ingin disandingkan dengan `foreign key`. lalu di dalamnya beri `return` yang memanggil fungsi relasi yang telah ditentukan, seperti ini.

```

// app/Models/User.php

class User extends Model{
    ...

    public function task(){
        return $this->hasMany(Task::class, "user_id", "id");
    }
}
  
```

Menghubungkan Antara Model User Dengan Task.

Lalu di dalam model task kita panggil function `belongsTo` untuk menentukan hubungan sebaliknya dengan model yang ingin dihubungkan. Didalamnya kita tentukan kolom mana yang ingin dijadikan `foreign key`, dan juga tentukan `id` yang kita jadikan sebagai `key` yang ingin disandingkan dengan `foreign key`, id disini adalah `id` milik task. Function `belongsTo` dapat digunakan untuk hubungan relasi one to many, dan one to one.

```
// app/Models/task.php

class Task extends Model{
    ...

    public function user(){
        return $this->belongsTo(User::class, "user_id", "id");
    }
}
```

Menghubungkan Antara Model Task Dengan User.

Setelah kita hubungkan relasi antar model, untuk kita bisa mengambil data dari tabel yang berelasi kita cukup panggil function yang sudah dibuat, contohnya kita ingin mengambil data task berdasarkan user, maka kita ambil data dari tabel task seperti ini.

```
// routes/web.php

use App\Models\User;

Route::get("/getdata", function(){
    $users = User::find(1);
    $users->task;

    return $users;
});
```

Contoh Pengambilan Data Dari Tabel Berelasi.

Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.

```

{
  "id": 1,
  "name": "Mylene Mills",
  "email": "pinkie4@example.com",
  "email_verified_at": "2024-04-23T07:57:10.000000Z",
  "created_at": "2024-04-23T07:57:10.000000Z",
  "updated_at": "2024-04-23T07:57:10.000000Z",
  "task": [
    {
      "id": 1,
      "user_id": 1,
      "nama": "facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate",
      "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.",
      "status": 0,
      "created_at": "2024-04-23T07:57:10.000000Z",
      "updated_at": "2024-04-23T07:57:10.000000Z"
    },
    {
      "id": 2,
      "user_id": 1,
      "nama": "rerum fugiat sunt animi quibusdam blanditiis consectetur ad",
      "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.",
      "status": 0,
      "created_at": "2024-04-23T07:57:10.000000Z",
      "updated_at": "2024-04-23T07:57:10.000000Z"
    },
    {
      "id": 3,
      "user_id": 1,
      "nama": "aperiam dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo",
      "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.",
      "status": 0,
      "created_at": "2024-04-23T07:57:10.000000Z",
      "updated_at": "2024-04-23T07:57:10.000000Z"
    },
    {
      "id": 4,
      "user_id": 1,
      "nama": "rerum praesentium ad tenetur tempora pariatur perspiciatis ducimus totam quod rerum quis mollitia",
      "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitisi id quisquam ducimus occaecati.",
      "status": 0,
      "created_at": "2024-04-23T07:57:10.000000Z",
      "updated_at": "2024-04-23T07:57:10.000000Z"
    },
    {
      "id": 5,
      "user_id": 1,
      "nama": "consequatur architecto repellendus consectetur blanditiis",
      "deskripsi": "Iure maiores beatae explicabo rerum sit ipsam et quasi eligendi.",
      "status": 0,
      "created_at": "2024-04-23T07:57:10.000000Z",
      "updated_at": "2024-04-23T07:57:10.000000Z"
    }
  ]
}

```

Hasil Pengambilan Data Yang Berelasi.

Semua data task ada di dalam objek dengan key task. Kita menggunakan fungsi `find` untuk mengambil data user berdasarkan ID, namun `find` hanya bisa digunakan untuk kondisi berdasarkan nilai ID saja, jika kita ingin memberikan kondisi selain ID, kita bisa berikan fungsi `where` untuk memberikan kondisi berdasarkan kolom selain ID, lalu untuk mendapatkan datanya kita gunakan fungsi `first`, contohnya seperti ini.

```

// routes/web.php

use App\Models\User;

Route::get("/getdata", function(){
    $users = User::where("name", "Mylene Mills")->first();

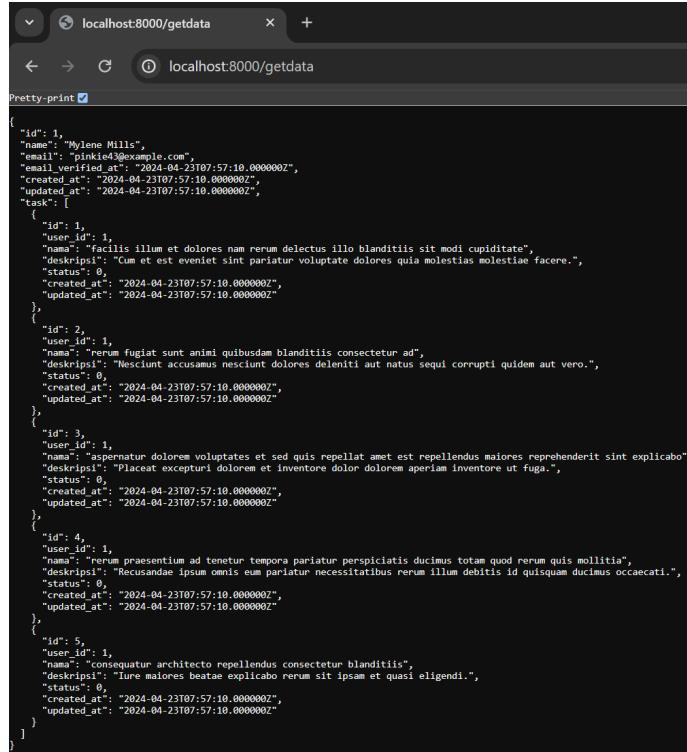
    $users->task;

    return $users;
});

```

Penggunaan `Where` Sebagai Penambahan Kondisi.

Untuk melihat data yang didapatkan bisa kita lihat dengan mengunjungi route yang sudah dibuat.



```
[{"id": 1, "name": "Mylene Mills", "email": "pinkie43@example.com", "email_verified_at": "2024-04-23T07:57:10.000000Z", "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z", "task": [{"id": 1, "user_id": 1, "name": "Facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate", "deskripsi": "Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 2, "user_id": 1, "name": "Rerum fugiat sunt animi quibusdam blanditiis consectetur ad", "deskripsi": "Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 3, "user_id": 1, "name": "Aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo", "deskripsi": "Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 4, "user_id": 1, "name": "Rerum praesentium ad tenetur tempora pariatur perspicatis ducimus totam quod rerum quis mollitia", "deskripsi": "Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}, {"id": 5, "user_id": 1, "name": "Consequatur architecto repellendus consectetur blanditiis", "deskripsi": "Ture maiores beatae explicabo rerum sit ipsam et quasi eligendi.", "status": 0, "created_at": "2024-04-23T07:57:10.000000Z", "updated_at": "2024-04-23T07:57:10.000000Z"}]
```

Hasil Pengambilan Data Yang Berelasi Setelah Diberikan Where.

J) Studi Kasus

kita sudah mengaplikasikan struktur Laravel yaitu View dan Controller. Sekarang, kita akan mengaplikasikan struktur yang lain, yaitu Model. Dalam hal ini, Model berhubungan langsung dengan Database. Kita akan coba membuat sebuah studi kasus CRUD (Create, Read, Update, Delete) sederhana untuk menambahkan task dan user.

Kita tidak perlu membuat migration lagi karena tabel yang akan gunakan yaitu tabel yang sudah kita buat sebelumnya, dan juga beberapa model yang telah kita buat akan kita gunakan dalam studi kasus ini.

Pertama kita buat terlebih dahulu Controller untuk user dan task. Kita bisa gunakan perintah artisan seperti ini seperti ini.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan make:controller UserController
```

Perintah Untuk Membuat User Controller.

Lalu didalam UserController kita tambahkan fungsi index yang didalamnya menampilkan semua daftar user, seperti ini.

```
// app/Http/Controllers/UserController.php

use App\Models\User;

class UserController extends Controller{
    public function index(){
        $users = User::all();

        return view("users.index", [
            "users" => $users,
        ])
    }
}
```

Menambahkan Fungsi Index Dalam User Controller.

Lalu kita buat route agar user dapat mengakses halaman yang akan kita buat. Karena kita akan membuat route yang mana akan menggunakan controller yang sama. Maka kita dapat dikelompokkan route yang akan kita buat berdasarkan controller. Seperti ini.

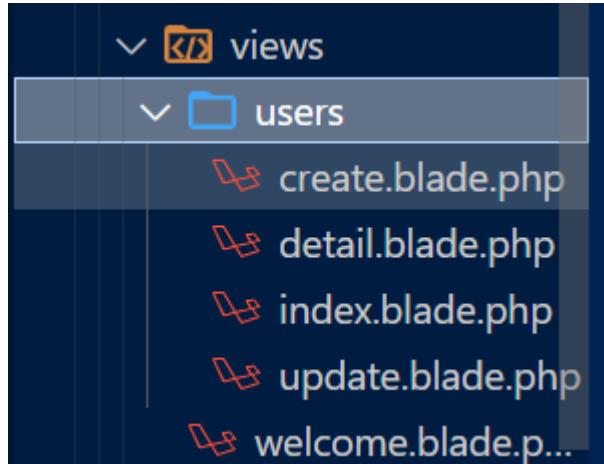
```
// app/Http/Controllers/UserController.php

use App\Http\Controllers\UserController;

Route::controller(UserController::class)->prefix("users")->name("users.")->group(function(){
    Route::get("/", "index")->name(".index");
});
```

Pembuatan Route Yang Dikelompokkan.

Lalu kita buat view nya untuk menampilkan index data yang sudah didapatkan dari controller. halaman user yang kita buat yaitu index, create, detail, dan update. Semua view ini kita buat di dalam folder users.



Semua View Yang Akan Ditampilkan Sebagai Halaman User.

Lalu kita buat view app sebagai layout, di dalamnya kita tambahkan dua tag yield untuk title dan content supaya setiap halaman content yang ditampilkan akan dinamis, seperti ini.

```
// resources/views/app.blade.php

<html>
<head>
    <title>@yield("title")</title>
</head>
<body>
    @yield("content")
</body>
</html>
```

Struktur Layout Didalam View App.

Lalu didalam view index yang ada di dalam folder users, kita akan tampilkan semua data yang kita dapatkan dari controller ke dalam sebuah tabel, lalu di bagian nama kita buat ketika kita klik namanya maka akan diarahkan ke halaman detail user.

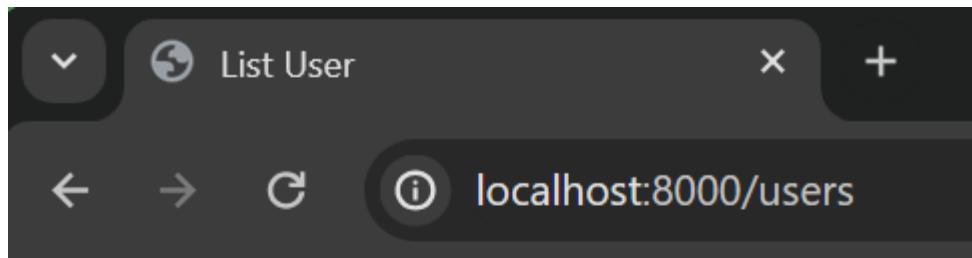
```
// resources/views/users/index.blade.php

@extends("app")

@section("title","List User")
@section("content")
    <a href="">Tambah User</a>
    <table border="1">
        <thead>
            <tr>
                <th>No</th>
                <th>Nama</th>
                <th>Jumlah Task</th>
                <th colspan="2">aksi</th>
            </tr>
        </thead>
        <tbody>
            @foreach($users as $index => $user)
            <tr>
                <td>{{ $index+1 }}</td>
                <td><a href="">{{ $user->name }}</a></td>
                <td>{{ count($user->task) }}</td>
                <td><a href="">Edit</a></td>
                <td><a href="">hapus</a></td>
            </tr>
            @endforeach
        </tbody>
    </table>
@endsection
```

Code Untuk Membuat Tabel Dan Data Didalamnya.

Lalu kita jalankan di browser maka akan terlihat seperti ini.



Tambah User

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit hapus

Tampilan Ketika Dijalankan Di Browser.

Lalu kita ingin membuat halaman yang akan menampilkan data detail dari user ketika kita klik nama yang ada di dalam tabel. Pertama kita akan buat fungsi untuk menampilkan data user berdasarkan id yang ditentukan dari url.

```
// resources/views/users/create.blade.php

use Illuminate\Http\Request;

class UserController extends Controller{
    ...

    public function show($id){
        $user = User::find($id);

        return view("users.detail", [
            "user" => $user,
        ]);
    }
}
```

Menambahkan Method Show Untuk Menampilkan Detail User.

Lalu kita buat route agar bisa mengakses halaman detail user, seperti ini.

```
// routes/web.php

Route::controller(UserController::class)->prefix("users")->name("users.")->group(function(){
    Route::get("/", "index")->name("index");
    Route::get("/{id}", "show")->name("show");
});
```

Menambahkan Route Untuk Menggunakan.

Lalu kita atur tampilannya didalam view detail yang ada didalam folder users, kita tambahkan if juga kita tambahkan tabel untuk daftar task yang dia kerjakan, seperti ini.

```
<!-- resources/views/users/detail.blade.php -->

@extends("app")
@section("title", $user->name)
@section("content")
    image) }}" alt="">
    <h1>Nama : {{ $user->name }}</h1>
    <h3>Email : {{ $user->email }}</h3>

    <h3>Daftar Task</h3>

    @if(count($user->task) > 0)
        <table border="1">
            <thead>
                <tr>
                    <th>No</th>
                    <th>Judul</th>
                    <th>Deskripsi</th>
                    <th>Status</th>
                    <th colspan="2">Aksi</th>
                </tr>
            </thead>
            <tbody>
                @foreach($user->task as $index => $task)
                    <tr>
                        <td>{{ $index+1 }}</td>
                        <td>{{ $task->nama }}</td>
                        <td>{{ $task->deskripsi }}</td>
                        <td>{{ $task->status }}</td>
                        <td><a href="">Edit</a></td>
                        <td><a href="">hapus</a></td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    @else
        <p>Kamu belum memiliki tugas apapun.</p>
    @endif
@endsection
```

Code Untuk Menampilkan Detail User Beserta Task Yang Dia Miliki.

Lalu didalam kita tambahkan route yang mengarah ke halaman detail di dalam hyperlink pada nama user di dalam tabel.

```

<!-- resources/views/users/index.blade.php -->

@extends('app')

@section('title', 'List User')
@section('content')
Tambah User

| No              | Nama                                                                         | Jumlah Task               | aksi                 |                       |
|-----------------|------------------------------------------------------------------------------|---------------------------|----------------------|-----------------------|
| {{ \$index+1 }} | <a href="{{ route('users.show', \$user-&gt;id) }}">{{ \$user-&gt;name }}</a> | {{ count(\$user->task) }} | <a href="#">Edit</a> | <a href="#">hapus</a> |


@endsection

```

Menambahkan Route Yang Mengarah Ke User Detail Pada Href Didalam Tabel.

Ketika kita jalankan di browser, maka akan terlihat seperti ini.



Nama : Mylene Mills

Email : pinkie43@example.com

Daftar Task

Tambah Task

No	Judul	Deskripsi	Status	Aksi
1	facilis illum et dolores nam rerum delectus illo blanditiis sit modi cupiditate	Cum et est eveniet sint pariatur voluptate dolores quia molestias molestiae facere.	0	Edit hapus
2	rerum fugiat sunt animi quibusdam blanditiis consectetur ad	Nesciunt accusamus nesciunt dolores deleniti aut natus sequi corrupti quidem aut vero.	0	Edit hapus
3	aspernatur dolorem voluptates et sed quis repellat amet est repellendus maiores reprehenderit sint explicabo	Placeat excepturi dolorem et inventore dolor dolorem aperiam inventore ut fuga.	0	Edit hapus
4	rerum praesentium ad tenetur tempora pariatur perspicatis ducimus totam quod rerum quis mollitia	Recusandae ipsum omnis eum pariatur necessitatibus rerum illum debitis id quisquam ducimus occaecati.	0	Edit hapus
5	consequatur architecto repellendus consectetur blanditiis	Iure maiores beatae explicabo rerum sit ipsam et quasi eligendi.	0	Edit hapus

Tampilan Detail User.

Kita sudah membuat halaman untuk menampilkan tabel user dan halaman detail user, sekarang kita buat halaman untuk buat sebuah form untuk menambahkan user baru. Pertama kita buat method baru di dalam controller untuk menampilkan halaman form pembuatan user, seperti ini.

```
// app/Http/Controllers/UserController.php

class UserController extends Controller{
    ...
    public function create(){
        return view("users.create");
    }
}
```

Penambahan Method Baru Untuk Menampilkan Halaman Create.

Lalu kita buat method baru untuk memproses data yang dikirim dari form dan disimpannya kedalam database. Method beserta code di dalamnya untuk memproses data yang dikirim dari form seperti ini.

```
// app/Http/Controllers/UserController.php

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller{
    ...
    public function store(Request $request){
        $payload = $request->all();

        $user = New User();
        $user->name = $payload["name"];
        $user->email = $payload["email"];
        $user->password = 12345678;
        $user->save();

        return redirect()->route("users.index");
    }
}
```

Proses Penyimpanan Data Yang Dikirim Dari Form Tambah User.

Kita buat password setiap user yang kita buat nilainya sama sehingga kita tidak perlu tambahkan input password di halaman tambah user. Lalu setelah data berhasil ditambahkan kita arahkan browser untuk kembali ke halaman user index.

Lalu kita buat route yang mengarah ke halaman tambah user yang sudah kita buat, seperti ini.

```
// routes/web.php

Route::controller(UserController::class)->prefix("users")->name("users.")->group(function(){
    Route::get("/", "index")->name("index");
    Route::get("/create", "create")->name("create");
    Route::get("/{id}", "show")->name("show");

    Route::post("/", "store")->name("store");
});
```

Penambahan Route Baru Untuk Menampilkan Halaman Tambah User.

Lalu didalam view create yang ada didalam folder users, lalu didalam form kita tambahkan input untuk name dan email, dan juga kita sisipkan csrf. Lalu pada tag form kita tambahkan action yang didalamnya mengarah ke route untuk memproses data yang dikirim dari form, dan tambahkan method POST, seperti ini.

```
<!-- resources/views/users/create.blade.php -->

@extends("app")
@section("title", "Tambah User")
@section("content")

<form action="{{ route('users.store') }}" method="POST">
    @csrf
    <label for="name">Nama</label>
    <input type="text" id="name" name="name"><br>
    <label for="email">Email</label>
    <input type="email" id="email" name="email">
    <button type="submit">Submit</button>
</form>
@endsection
```

Pembuatan Form Di Dalam View Create Users.

Kita coba buat user baru melalui form yang telah kita buat, seperti ini.

Nama

Email

Tampilan Form Untuk Membuat User Baru.

Ketika kita submit maka browser akan mengarahkan ke halaman sebelumnya yaitu halaman daftar user, seperti ini.

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit hapus
2	Riko	0	Edit hapus

Tampilan Setelah Data Telah Ditambahkan.

Kita sudah mencoba membuat tambah user, sekarang kita coba tambahkan upload file, jadi pada tabel user kita tambahkan kolom baru untuk menyimpan gambar. Pertama kita buat terlebih dahulu migration baru untuk menambahkan kolom baru bernama gambar di dalam user.

```
advan@DESKTOP-K34IR2I MINGW64 /d/Programing/Belajar-Laravel
$ php artisan make:migration AddKolomImage --table=users
```

Perintah Artisan Untuk Membuat Migration.

Lalu didalam migration yang sudah kita buat, secara otomatis sudah didefinisikan tabel yang sudah kita buat. Di Dalam fungsi run kita definisikan column yang akan dibuat. Lalu didalam fungsi down kita definisikan juga column yang akan dihapuskan.

```
// database/migrations/add_kolom_image.php

return new class extends Migration{
    public function up(): void{
        Schema::table('users', function (Blueprint $table) {
            $table->text("image");
        });
    }

    public function down(): void{
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn("image");
        });
    }
};
```

Penambahan Column Image Didalam Migration.

Jika sudah, maka kita jalankan migration yang sudah kita buat.

```
[INFO] Running migrations.
2024_04_29_090639_add_kolom_image ..... 198.50ms DONE
```

Tampilan Setelah Menjalankan Migration.

Kita sudah menambahkan kolom baru didalam tabel users. Kita tambahkan input baru di dalam form pada halaman create user, dan juga kita tambahkan enctype di dalam tag form.

```
<!-- resources/views/users/create.blade.php -->
@extends("app")
@section("title", "Tambah User")
@section("content")

<form action="{{ route('users.store') }}" method="POST" enctype="multipart/form-data">
    @csrf
    <label for="name">Nama</label>
    <input type="text" id="name" name="name"><br>
    <label for="email">Email</label>
    <input type="email" id="email" name="email"><br>
    <label for="image">Image</label>
    <input type="file" id="image" name="image">
    <button type="submit">Submit</button>
</form>
@endsection
```

Penambahan Enctype Dan Input Gambar Di Dalam Form.

Lalu didalam method yang kita gunakan untuk memproses data yang dikirim dari form, file yang dikirim kita pindahkan ke dalam folder public/image menggunakan fungsi move, lalu penamaan file kita acakan agar tidak ada nama file yang sama. Lalu nama file yang sudah di acak kita simpan kedalam database, dan file yang dikirim akan disimpan kedalam folder public/image. Lalu jika ketika kita tambah user namun tidak memberikan gambar, maka kita buat nilai di gambarnya menjadi gambar default, seperti ini.



```
// app/Http/Controllers/UserController.php

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller{
    ...

    public function store(Request $request){
        $payload = $request->all();

        if($request->hasFile("image")){
            $image = $request->file("image");
            $imageName = time() . "-" . $image->hashName();
            $image->move("image/", $imageName);
        } else{
            $imageName = "default.jpg";
        }

        $user = New User();

        $user->image = $imageName;
        $user->name = $payload["name"];
        $user->email = $payload["email"];
        $user->password = 12345678;

        $user->save();
        return redirect()->route("users.index");
    }
}
```

Menambahkan Code Untuk Menangani File Yang Dikirim.

Lalu didalam halaman detail user, kita tampilkan gambar yang sudah kita tambahkan.

```
<!-- resources/views/users/detail.blade.php -->

@extends("app")
@section("title", $user->name)
@section("content")
    image) }}" alt="">
    <h1>Nama : {{ $user->name }}</h1>
    <h3>Email : {{ $user->email }}</h3>

    <h3>Daftar Task</h3>

    @if(count($user->task) > 0)
        <table border="1">
            <thead>
                <tr>
                    <th>No</th>
                    <th>Judul</th>
                    <th>Deskripsi</th>
                    <th>Status</th>
                </tr>
            </thead>
            <tbody>
                @foreach($user->task as $index => $task)
                    <tr>
                        <td>{{ $index+1 }}</td>
                        <td>{{ $task->nama }}</td>
                        <td>{{ $task->deskripsi }}</td>
                        <td>{{ $task->status }}</td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    @else
        <p>Kamu belum memiliki tugas apapun.</p>
    @endif
@endsection
```

Menambahkan Image Agar Menampilkan Gambar Pada Halaman Detail User.

Ketika kita jalankan di browser maka gambar yang kita upload di halaman tambah user akan disimpan didalam database dan bisa kita tampilkan di halaman detail user, contohnya seperti ini.

Tambah User

localhost:8000/users/create

Nama: Surya

Email: Surya@gmail.com

Image: Choose File contoh.jpg

Submit

Tampilan Ketika Kita Menambahkan Data User.

List User

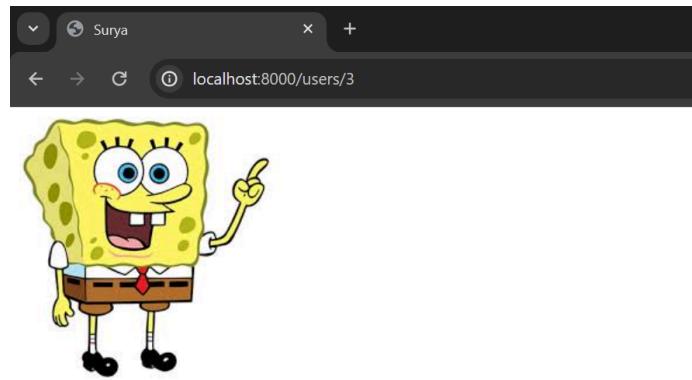
localhost:8000/users

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit hapus
2	Riko	0	Edit hapus
3	Surya	0	Edit hapus

[Tambah User](#)

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit hapus
2	Riko	0	Edit hapus
3	Surya	0	Edit hapus

Tampilan Setelah Data Ditambahkan.



Nama : Surya

Email : Surya@gmail.com

[Daftar Task](#)

[Tambah Task](#)

Kamu belum memiliki tugas apapun, tambah tugas untuk membuat tugas pertama mu

Tampilan Di Halaman Detail User.

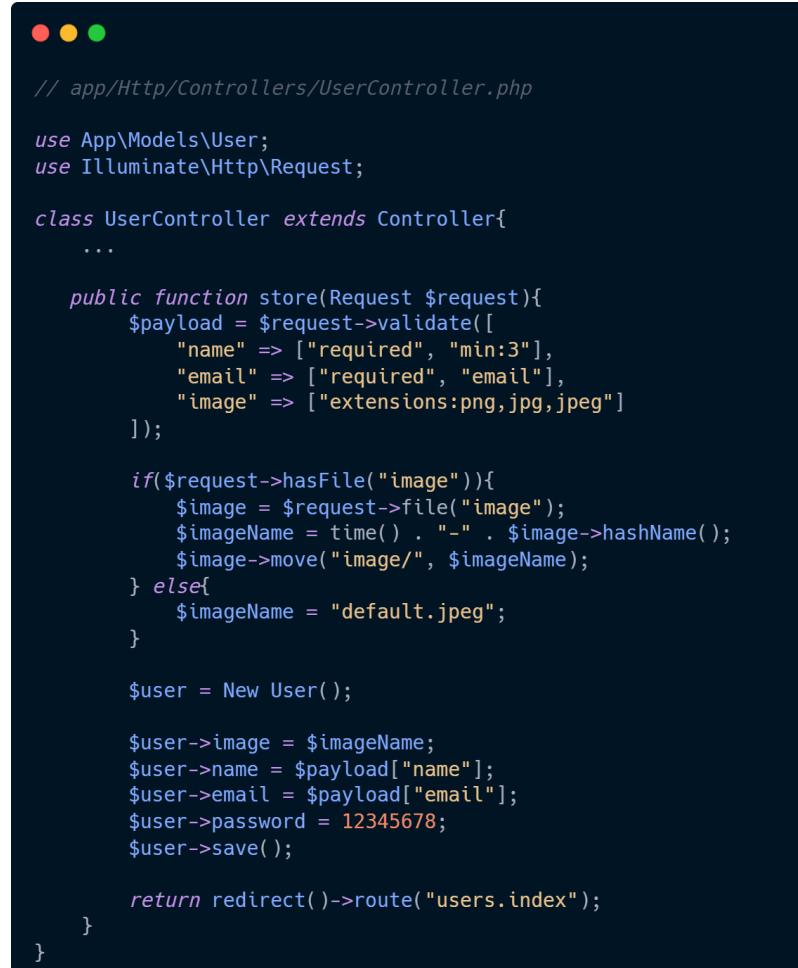
Gambar yang ditambahkan akan disimpan kedalam folder public/image dan gambar yang ditampilkan yaitu berasal dari folder ini.

```
> app
> bootstrap
> config
> database
< public
  < image
    1714391169-HzHfsbDGWOtWrlXOJLrsVjbtmIWpf6zSf7xGiqZ2.jpg
    default.jpeg
    .htaccess
    favicon.ico
    index.php
    robots.txt
```

Tampilan Folder Untuk Menyimpan Gambar Yang Kita Tambahkan.

Kita tambahkan validasi di halaman tambah user. Jadi ketika kita menginputkan data yang tidak sesuai maka akan muncul pesan error tergantung dengan input apa

yang tidak sesuai dengan validasi yang kita berikan. Contohnya kita tambahkan validasi input gambar harus jpg, jpeg, atau png.



```
// app/Http/Controllers/UserController.php

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller{
    ...

    public function store(Request $request){
        $payload = $request->validate([
            "name" => ["required", "min:3"],
            "email" => ["required", "email"],
            "image" => ["extensions:png,jpg,jpeg"]
        ]);

        if($request->hasFile("image")){
            $image = $request->file("image");
            $imageName = time() . "-" . $image->hashName();
            $image->move("image/", $imageName);
        } else{
            $imageName = "default.jpeg";
        }

        $user = New User();

        $user->image = $imageName;
        $user->name = $payload["name"];
        $user->email = $payload["email"];
        $user->password = 12345678;
        $user->save();

        return redirect()->route("users.index");
    }
}
```

Penambahan Validasi Data Yang Dikirim Dari Form.

Lalu di halaman tambah user kita tampilkan pesan error ketika data yang diinputkan terdapat kesalahan atau tidak valid.

```
<!-- resources/views/users/create.blade.php -->

@extends("app")
@section("title", "Tambah User")
@section("content")

@if($errors->any)
    @foreach($errors->all() as $error)
        <p style="color: red;">{{ $error }}</p>
    @endforeach
@endif

<form action="{{ route('users.store') }}" method="POST"
enctype="multipart/form-data">
    @csrf
    <label for="name">Nama</label>
    <input type="text" id="name" name="name"><br>
    <label for="email">Email</label>
    <input type="email" id="email" name="email"><br>
    <label for="image">Image</label>
    <input type="file" id="image" name="image">
    <button type="submit">Submit</button>
</form>
@endsection
```

Menambahkan Pesan Error Saat Terjadi Kesalahan.

Ketika kita coba kirim filenya yang tidak sesuai dengan yang telah kita validasi, seperti ini.

The screenshot shows a web browser window with a dark theme. At the top, the title bar displays "Tambah User". Below it, the address bar shows the URL "localhost:8000/users/create". The main content area contains a form with three fields:

- Nama:** An input field containing "ri".
- Email:** An input field containing "ri@gmail.com", which is highlighted with a red border, indicating an validation error.
- Image:** A file input field with the placeholder "Choose File" and the file name "contoh.pdf".

To the right of the form, there is a "Submit" button.

Percobaan Input Data Yang Tidak Sesuai.

The name field must be at least 3 characters.

The image field must have one of the following extensions: png, jpg, jpeg.

Nama

Email

Image Choose File No file chosen

Tampilan Pesan Error Ketika Data Tidak Sesuai.

Lalu kita akan buat edit user. Pertama kita buat method baru untuk menampilkan halaman edit user, di dalam methodnya kita tambahkan id di dalam parameteranya untuk menentukan data mana yang akan diubah.

```
// app/Http/Controllers/UserController.php

use App\Models\User;

class UserController extends Controller{
    ...

    public function edit($id){
        $user = User::all()->find($id);

        return view("users.update", [
            'user' => $user,
        ]);
    }
}
```

Method Untuk menampilkan Halaman Edit User.

Lalu kita buat juga method untuk memproses data yang dikirim dari form edit user, dan juga kita akan coba timpa gambar jika terdapat gambar yang dikirim. Code untuk memproses data yang kirim dari form edit user. Lalu kita tambahkan juga validasi sama seperti kita menyimpan data, seperti ini.

```
// app/Http/Controllers/UserController.php

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller{
    ...

    public function update(Request $request, $id){
        $payload = $request->validate([
            "name" => ["required", "min:3"],
            "email" => ["required", "email"],
            "image" => ["extensions:png,jpg,jpeg"]
        ]);

        $user = User::find($id);

        if($request->hasFile("image")){
            $image = $request->file("image");
            $imageName = time() . "-" . $image->hashName();

            if(file_exists(public_path("image/" . $user->image)) && $user->image != "default.jpeg"){
                unlink(public_path("image/" . $user->image));
            }

            $user->image = $imageName;
            $image->move("image/", $imageName);
        }

        $user->name = $payload["name"];
        $user->email = $payload["email"];
        $user->save();

        return redirect()->route("users.index");
    }
}
```

Proses Yang Dijalankan Melalui Method Update.

Kita menggunakan fungsi unlink untuk menghapus image yang ada didalam folder public/image, namun sebelum kita hapus periksa terlebih dahulu apakah file yang sebelumnya disimpan apakah masih ada atau tidak ada, dan juga agar foto default tidak hilang kita tambahkan hanya nama gambar yang diacak saja yang dihapus.

Lalu kita buat route untuk mengakses halaman edit user dan proses data yang dikirim dari halaman edit user.

```
// routes/web.php

Route::controller(UserController::class)->prefix("users")->name("users.")->group(function(){
    Route::get("/", "index")->name("index");
    Route::get("/create", "create")->name("create");
    Route::get("/{id}", "show")->name("show");
    Route::get("/{id}/edit", "edit")->name("edit");

    Route::post("/", "store")->name("store");
    Route::put("/{id}", "update")->name("update");
});
```

Menambahkan Route Edit Dan Update.

Lalu kita atur tampilan untuk edit user di dalam view update seperti ini, dan juga berikan method dengan nilai put agar data yang dikirim dari form methodnya put.

```

<!-- resources/views/users/update.blade.php -->

@extends("app")
@section("title", "Edit User")
@section("content")

@if($errors->any)
    @foreach($errors->all() as $error)
        <p style="color: red;">{{ $error }}</p>
    @endforeach
@endif

<form action="{{ route('users.update', $user->id) }}" method="POST" enctype="multipart/form-data">
    @csrf
    @method("PUT")
    <label for="name">Nama</label>
    <input type="text" id="name" name="name" value="{{ $user->name }}><br>
    <label for="email">Email</label>
    <input type="email" id="email" name="email" value="{{ $user->email }}>
    <label for="image">Gambar</label>
    <input type="file" id="image" name="image">
    <button type="submit">Submit</button>
</form>
@endsection

```

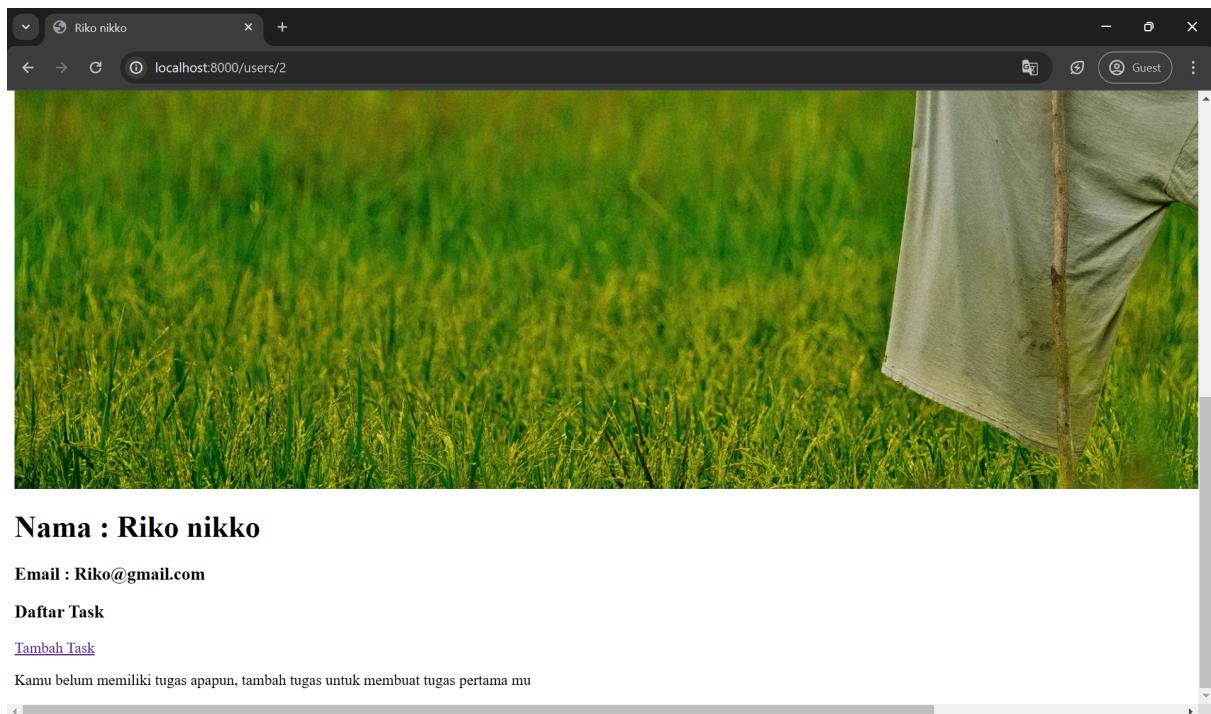
Form Beserta Pesan Error Di Halaman Edit User.

Ketika kita jalankan maka akan terlihat seperti ini.

Mengubah Data Di Halaman Edit.

No	Nama	Jumlah Task	aksi	
1	Mylene Mills	5	Edit	hapus
2	Riko nikko	0	Edit	hapus
3	Surya	0	Edit	hapus

Tampilan Data Setelah Diubah.



Tampilan Detail User.

Kita sudah berhasil membuat user dan edit beserta. Sekarang kita coba hapus data user, pertama kita buat method baru didalam controller untuk proses hapus data, seperti ini.

```
// app/Http/Controllers/UserController.php

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller{
    ...

    public function destroy($id){
        $user = User::find($id);

        if(file_exists(public_path("image/" . $user->image)) && $user->image != "default.jpeg"){
            unlink(public_path("image/" . $user->image));
        }

        User::destroy($id);

        return redirect()->route("users.index");
    }
}
```

Lalu kita buat route untuk agar method yang baru saja dibuat bisa kita jalankan.

```
// routes/web.php

Route::controller(UserController::class)->prefix("users")->name("users.")->group(function(){
    Route::get("/", "index")->name("index");
    Route::get("/create", "create")->name("create");
    Route::get("/{id}", "show")->name("show");
    Route::get("/{id}/edit", "edit")->name("edit");

    Route::post("/", "store")->name("store");
    Route::put("/{id}", "update")->name("update");

    Route::delete("/{id}", "destroy")->name("destroy");
});
```

Menambahkan Route Baru Hapus Data User.

Lalu pada tabel users kita tambahkan form yang didalamnya beri method delete dan button untuk submit data yang ingin dihapus.

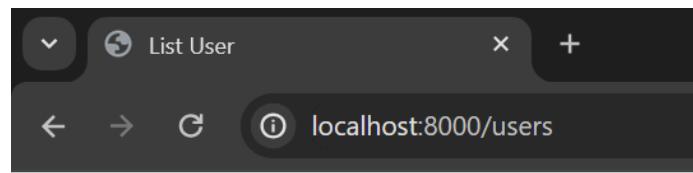
```
<!-- resources/views/users/index.blade.php -->

@extends("app")

@section("title","List User")
@section("content")
    <a href="{{ route("users.create") }}>Tambah User</a>
    <table border="1">
        <thead>
            <tr>
                <th>No</th>
                <th>Nama</th>
                <th>Jumlah Task</th>
                <th colspan="2">aksi</th>
            </tr>
        </thead>
        <tbody>
            @foreach($users as $index => $user)
            <tr>
                <td>{{ $index+1 }}</td>
                <td><a href="{{ route("users.show", $user->id) }}>{{ $user->name }}</a></td>
                <td>{{ count($user->task) }}</td>
                <td><a href="{{ route("users.edit", $user->id) }}>Edit</a></td>
                <td><form action="{{ route("users.destroy", $user->id) }}" method="POST">
                    @csrf
                    @method("DELETE")
                    <button type="submit">Hapus</button>
                </form></td>
            </tr>
            @endforeach
        </tbody>
    </table>
@endsection
```

Menambahkan Form Didalam Tabel User.

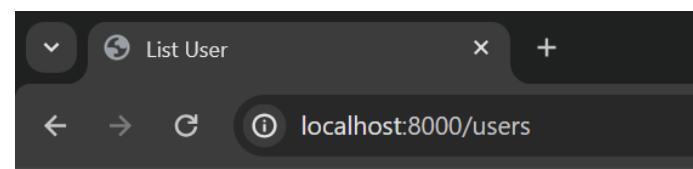
Ketika kita click tombol hapus maka data dibaris tersebut akan terhapus.



Tambah User

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit Hapus
2	Riko nikko	0	Edit Hapus
3	Surya	0	Edit Hapus

Tampilan Tabel Sebelum Dihapus.



Tambah User

No	Nama	Jumlah Task	aksi
1	Mylene Mills	5	Edit Hapus
2	Riko nikko	0	Edit Hapus

Tampilan Tabel Setelah Dihapus.

4. Referensi.

- <https://laravel.com/docs/11.x/queries#running-database-queries/>
- <https://laravel.com/docs/11.x/migrations>
- <https://laravel.com/docs/11.x/seeding>
- <https://laravel.com/docs/11.x/eloquent-relationships>