

# LARAVEL CONTROLLER WITH CASE STUDIES

## KELAS PROGRAMMING FULL STACK DEVELOPER

MITRA PELATIHAN

@lkademi

Jabar Digital Academy

[digitalacademy.jabarprov.go.id](https://digitalacademy.jabarprov.go.id)

2024

## **BAB IV**

### **LARAVEL CONTROLLER WITH CASE STUDIES.**

#### **1. Tujuan**

- a. Peserta didik dapat mengetahui apa itu Controller.
- b. Peserta didik dapat membuat controller sederhana.
- c. Peserta didik dapat membuat grouping berdasarkan controller didalam route.
- d. Peserta didik dapat mengetahui penggunaan form pada Laravel.
- e. Peserta didik dapat membuat form sederhana di dalam Laravel.
- f. Peserta didik dapat membuat validasi pada form yang telah dibuat.
- g. Peserta didik dapat melakukan upload file pada form yang sudah dibuat.

#### **2. Perlengkapan**

- a. Modul IV. Laravel Controller With Case Studies.
- b. IDE atau Teks Editor (Visual Studio Code, Notepad++, Sublime).

#### **3. Materi.**

Dalam pengembangan aplikasi web menggunakan framework Laravel, controller sangatlah penting dalam mengelola logika bisnis dan mengatur interaksi antara model (data) dan tampilan (view). Dengan mengatur berbagai permintaan HTTP yang diterima dari pengguna, controller memandu aplikasi untuk memberikan respons yang sesuai, memastikan pengalaman pengguna yang interaktif dan responsif. Controller dalam Laravel memungkinkan kita untuk membuat aplikasi web yang efisien, terstruktur, dan mudah dikelola, mempercepat proses pengembangan serta meningkatkan fleksibilitas dan skalabilitasnya.

##### **A. Apa itu Controllers.**

Controller adalah pengelolaan permintaan yang diterima dari pengguna melalui HTTP lalu merespon permintaan tersebut dengan menampilkan data yang diperlukan. Controller sangat penting dalam memproses logika yang ada didalam aplikasi kita, dan juga sebagai penghubung antara model dan view.

Controller digunakan untuk mengelompokkan logic yang ada didalam route, contohnya misalnya kita membuat logic untuk user, maka bisa kita lihat logic yang ada di setiap route yang kita buat di bawah ini.

```
// routes/web.php

Route::get('/users', function(){
    $users = [...];

    return view("users.users", [
        "users" => $users
    ]);
})->name("users");

Route::get('/users/{id}', function(int $id){
    $users = [...];

    $result = null;
    foreach($users as $user){
        if($user["id"] == $id){
            $result = $user;
        }
    }

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

Contoh Logic Disetiap Yang Dibuat Jika Tidak Menggunakan Controller.

Lalu kita bisa kelompokkan semua logic yang ada di dalam route kedalam controller, Untuk membuat sebuah controller kita dapat menggunakan perintah artisan, Semua controller yang kita buat ada didalam folder `App/Http/Controllers`. Perintah artisan untuk membuat controller seperti ini.

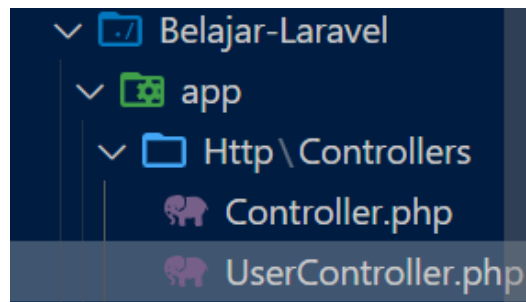
**`php artisan make:controller (nama_controller)`**

Contohnya kita akan buat controller untuk user, perintah seperti ini.

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

Tampilan Perintah Artisan Untuk Membuat Controller.

Kita bisa lihat di dalam folder controllers secara otomatis ditambahkan UserController.



Tampilan Didalam Folder Controller Setelah Menjalankan Perintah Artisan.

Lalu semua logic yang ada di dalam route kita pindahkan ke dalam controller yang sudah kita buat. Didalam controller kita tambahkan function index untuk code dari route users, dan show untuk code dari route users.detail. Seperti ini.

```

// app/Http/Controllers/UserController.php

use Illuminate\Http\Request;

class UserController extends Controller{
    public function index(){
        $users = [
            [
                "id" => 1,
                "nama" => "ronald Alberty",
                "umur" => 20,
            ],
            [
                "id" => 2,
                "nama" => "Kevin Leonardo",
                "umur" => 24,
            ],
            [
                "id" => 3,
                "nama" => "Stevano Michael",
                "umur" => 20,
            ]
        ];

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

    public function show(int $id){
        $users = [
            [
                "id" => 1,
                "nama" => "ronald Alberty",
                "umur" => 20,
            ],
            [
                "id" => 2,
                "nama" => "Kevin Leonardo",
                "umur" => 24,
            ],
            [
                "id" => 3,
                "nama" => "Stevano Michael",
                "umur" => 20,
            ]
        ];

        $result = null;
        foreach($users as $user){
            if($user["id"] == $id){
                $result = $user;
            }
        }

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

```

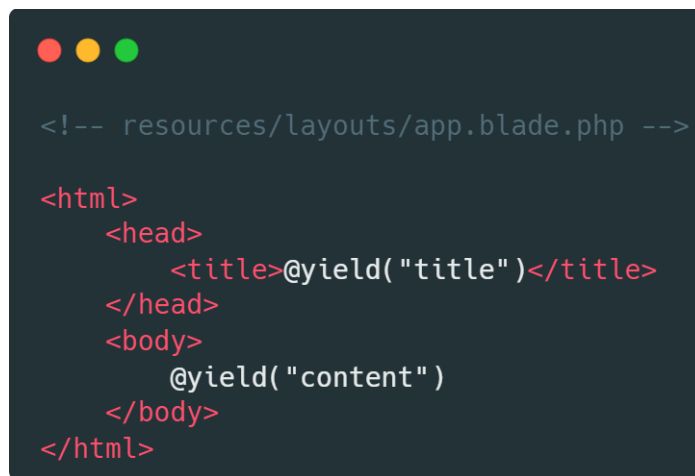
Code Didalam Route Dipindahkan Kedalam Controller.

Didalam return kita panggil view agar view yang ditampilkan sesuai dengan alamat route yang diakses. Kita siapkan view yang akan kita tampilkan di browser. Pertama-tama kita buat layout agar kita dapat menggunakan template HTML secara berulang kali, untuk membuat layout kita buat folder layout di dalam view lalu di dalamnya

beri view nama app.blade.php, untuk melakukan itu kita bisa menggunakan perintah artisan di terminal kita seperti ini.

**php artisan make:view layouts.app**

Jika sudah dijalankan kita dapat melihat view yang sudah kita buat di dalam folder `resources/views/layouts`. Lalu didalam view app kita buat template HTML yang akan kita gunakan di berbagai view, seperti ini.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light-colored font and represents the content of the `resources/layouts/app.blade.php` file. It starts with a comment line, followed by an HTML structure with a head section containing a title placeholder and a body section containing a content placeholder.

```
<!-- resources/layouts/app.blade.php -->

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

Template HTML Yang Ada Di Dalam Layouts.

Lalu kita buat view untuk menampilkan data yang akan ditampilkan di dalam view ketika route dijalankan. untuk view index kita tampilkan semua user di dalam tabel, lalu kita buat nama user ketika di klik akan mengarah ke halaman detail dari user yang diklik. Untuk membuat view kita bisa menggunakan perintah artisan seperti ini.

**php artisan make:view users.users**

**php artisan make:view users.detail**

Lalu disetiap view kita tambahkan extends dan panggil view yang ada di dalam layouts, lalu kita tambahkan dua section di dalamnya untuk memberikan title dan content di dalamnya.

```

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

@extends("layouts.app")
@section("title", "List Users")
@section("content")
    <table border="1">
        <thead>
            <tr>
                <th>no</th>
                <th>nama</th>
                <th>umur</th>
            </tr>
        </thead>
        <tbody>
            @foreach($users as $user)
                <tr>
                    <td>{{ $user["id"] }}</td>
                    <td>
                        <a href='{{ route("users.detail", $user["id"]) }}'>
                            {{ $user["nama"] }}
                        </a>
                    </td>
                    <td>{{ $user["umur"] }}</td>
                </tr>
            @endforeach
        </tbody>
    </table>
@endsection

```

Isi Code Di Dalam View Users.

```

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

@extends("layouts.app")
@section("title", $user["nama"])
@section("content")
    <ul>
        <li><h1>id : {{ $user["id"] }}</h1></li>
        <li><h1>Nama : {{ $user["nama"] }}</h1></li>
        <li><h1>Umur : {{ $user["umur"] }}</h1></li>
    </ul>
@endsection

```

Isi Code Di Dalam View Detail User.

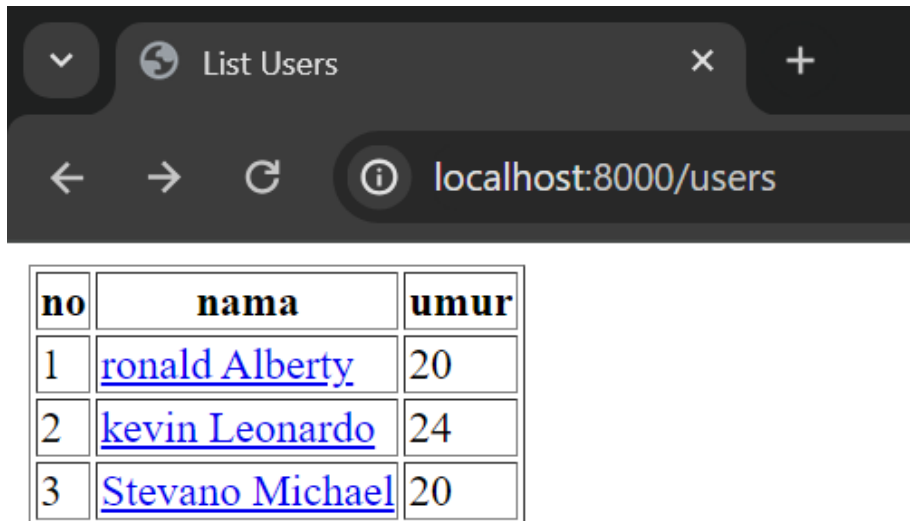
Kita sudah membuat view dan controller agar logic ketika kita buat route bisa kita pisahkan, sehingga di dalam file *routes/web.php* akan difokuskan untuk mendefinisikan route yang ada didalam aplikasi kita. untuk memanggil class UserController yang sudah kita buat kedalam route yaitu dengan memberikan array setelah url, lalu didalam array kita beri class controllernya lalu setelahnya beri nama method yang akan digunakan saat kita akses route tersebut. Contohnya seperti ini.

```
// routes/web.php

Route::get('/users', [UserController::class, "index"]->name("users"));
Route::get('/users/{id}', [UserController::class, "show"]->name("users.detail"));
```

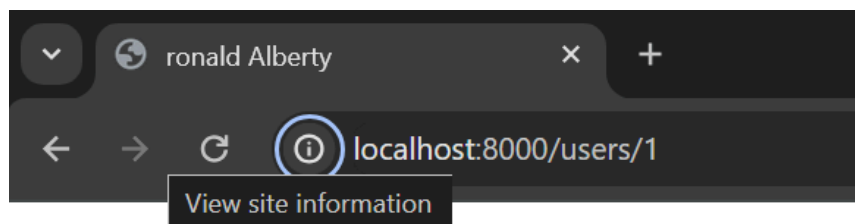
Contoh Penggunaan Controller Dan Method Di Dalamnya.

Ketika kita jalankan di browser akan terlihat seperti ini.



no	nama	umur
1	<a href="#">ronald Alberty</a>	20
2	<a href="#">kevin Leonardo</a>	24
3	<a href="#">Stevano Michael</a>	20

Tampilan Dari Route Users.



**. id : 1**

**. Nama : ronald Alberty**

**. Umur : 20**

Tampilan Dari Route Detail Users.



Kita sudah mencoba buat controller sederhana untuk menampilkan list user di dalam tabel dan detail user.

## B. Grouping By Controller.

Kita dapat melakukan pengelompokkan berdasarkan controller pada saat kita membuat route. Untuk mengelompokkan controller didalam route yaitu kita dapat menggunakan function controller dan definisikan controller yang akan digunakan sebelum function group dijalankan, lalu didalamnya kita cukup definisikan method apa yang akan dipakai dari controller yang kita pakai. Contohnya seperti ini.

```
// routes/web.php

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

Penggunaan Group Controller Didalam Route.

Ketika kita jalankan artisan route list, maka akan terlihat seperti ini.

```
GET|HEAD / ..... index
POST _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionCo...
GET|HEAD _ignition/health-check .... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config . ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD up .....
GET|HEAD users ..... users > UserController@index
GET|HEAD users/{id?} ..... users.detail > UserController@show

Showing [7] routes
```

Tampilan Dari Route List.

## C. Resources.

Terkadang ketika kita membuat sebuah route, code untuk memproses permintaan dari pengguna di dalam route pasti tidak jauh dari membaca, membuat, mengubah atau menghapus data (CRUD). Didalam laravel terdapat sebuah perintah artisan yang akan secara otomatis dibuatkan method yang akan digunakan untuk keperluan

CRUD. Untuk membuatnya kita hanya perlu menambahkan `--resources` setelah nama controller yang akan dibuat, seperti ini.

```
php artisan make:controller (nama_controller) --resources
```

Kita coba buat controller role dengan resource seperti ini.

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

Perintah Membuat Controller Dengan Resources.

Ketika kita lihat controller yang telah kita buat semua method.

```
// app/Http/Controllers/RoleController.php
namespace App\Http\Controllers;
use Illuminate\Http\Request;

class RoleController extends Controller{
    public function index(){}

    public function create(){}

    public function store(Request $request){}

    public function show(string $id){}

    public function edit(string $id){}

    public function update(Request $request, string $id){}

    public function destroy(string $id){}
}
```

Controller Yang Diberi Perintah Resource.

Berikut adalah fungsi method yang biasa digunakan dalam resources.

HTTP Method	Url	Action Method	Route Name	Keterangan
GET	/roles	Index	roles.index	Mengambil Semua Data role

GET	/roles/{role}	show	roles.show	Mengambil Data role berdasarkan data dari url
GET	/roles/{create}	create	roles.create	Menampilkan halaman formulir buat role
POST	/roles	store	roles.store	Memproses data yang dikirim melalui form.
GET	/roles/{role}/edit	edit	roles.edit	Menampilkan halaman formulir edit role
PUT/PATCH	/roles/{role}	update	roles.update	Memproses data yang dikirim melalui form.
DELETE	/roles/{role}	destroy	roles.destroy	Menghapus data berdasarkan data dari url.

Kita bisa buat semua route diatas hanya dengan satu function pada saat membuat route yaitu resource, contohnya seperti ini.

```
// routes/web.php
Route::resources("roles", RoleController::class);
```

Penggunaan Function Resource Didalam Route.

Untuk melihat semua route yang sudah kita buat, jalankan perintah artisan route list di terminal kita,

```

GET|HEAD / ..... index
POST _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolut...
GET|HEAD _ignition/health-check ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigContro...
GET|HEAD roles ..... roles.index > RoleController@index
POST roles ..... roles.store > RoleController@store
GET|HEAD roles/create ..... roles.create > RoleController@create
GET|HEAD roles/{role} ..... roles.show > RoleController@show
PUT|PATCH roles/{role} ..... roles.update > RoleController@update
DELETE roles/{role} ..... roles.destroy > RoleController@destroy
GET|HEAD roles/{role}/edit ..... roles.edit > RoleController@edit
GET|HEAD up .....
GET|HEAD users ..... users > UserController@index
GET|HEAD users/{id?} ..... users.detail > UserController@show

```

Showing [14] routes

Tampilan Ketika Menjalankan Route List.

#### D. Form.

Ketika aplikasi kita membutuhkan form HTML, pada saat mendefinisikan form tersebut, kita harus menyertakan field CSRF tersembunyi pada form, sehingga middleware CSRF dapat memvalidasi permintaan. kita juga dapat menggunakan perintah `@csrf` untuk menghasilkan field token. Contohnya kita buat view baru untuk membuat form di dalam folder user. Perintah untuk membuat view seperti ini.

**php artisan make:view users.create**

View ini akan kita isi dengan form, bentuk form yang akan kita buat seperti ini.

```

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

@extends("layouts.app")

@section("title", "User Form")
@section("content")
<form>
    <label for="">Nama</label>
    <input type="text" name="nama"><br>
    <label for="">Umur</label>
    <input type="text" name="umur">
    <button type="submit">Submit</button>
</form>
<endsection

```

Struktur Form Didalam View User.

Kita buat Method yang akan digunakan untuk menampilkan form didalam controller users.

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

class UserController extends Controller{

    public function create(){
        return view("users.create");
    }
}
```

Method Baru Untuk Menampilkan View Form.

Lalu kita buat method kedua untuk memproses data yang dikirim dari form yang sudah kita buat, code di dalam method seperti ini.

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

use Illuminate\Http\Request;

class UserController extends Controller{
    ...

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

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

Code Untuk Memproses Data Yang Dikirim Dari Form.

Semua data yang dikirim dari form disimpan didalam *request->all()*. Lalu kita simpan semua data yang ada didalam request kedalam variabel, lalu kita tampilkan kedalam view yang akan kita buat, dan di dalam view kita tambahkan isset untuk memeriksa variabel yang akan digunakan pernah didefinisikan atau tidak.

```

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

@extends("layouts.app")

@section("title", "User Form")
@section("content")
<form>
    <label for="">Nama</label>
    <input type="text" name="nama"><br>
    <label for="">Umur</label>
    <input type="text" name="umur">
    <button type="submit">Submit</button>
</form>

@isset($payload)
    <h1>{{ $payload["nama"] }}</h1>
    <h1>{{ $payload["umur"] }}</h1>
@endisset
@endsection

```

Penambahan Isset Di Dalam View.

Kita perlu buat route agar method yang sudah kita buat bisa kita akses.

```

// routes/web.php

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

    Route::get("/users/create", "create")->name("users.create");
    Route::post("/users", "store")->name("users.store");
});

```

Penambahan Route Baru.

Di dalam form yang kita buat kita tambahkan action dan method agar data dari form bisa diproses, sesuaikan dengan route yang telah kita buat untuk memproses data yang dikirim dari form. Sesuaikan method yang ada di dalam form dengan route yang kita gunakan untuk memproses data dari form.

```

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

@extends("layouts.app")

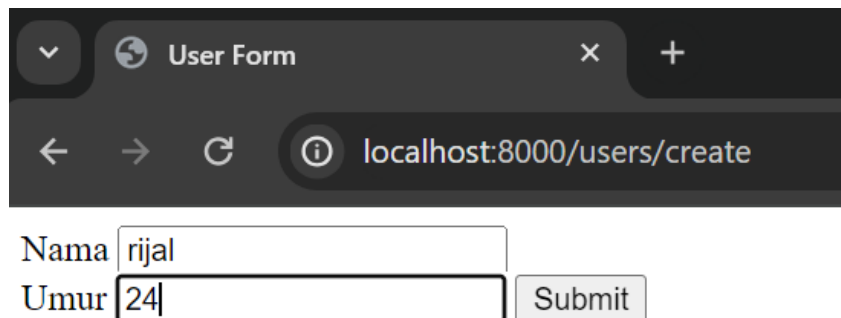
@section("title", "User Form")
@section("content")
<form action="{{ route('users.store') }}" method="POST">
    <label for="">Nama</label>
    <input type="text" name="nama"><br>
    <label for="">Umur</label>
    <input type="text" name="umur">
    <button type="submit">Submit</button>
</form>

@isset($payload)
    <h1>{{ $payload["nama"] }}</h1>
    <h1>{{ $payload["umur"] }}</h1>
@endisset
@endsection

```

Penambahan Action Dan Method Didalam Form.

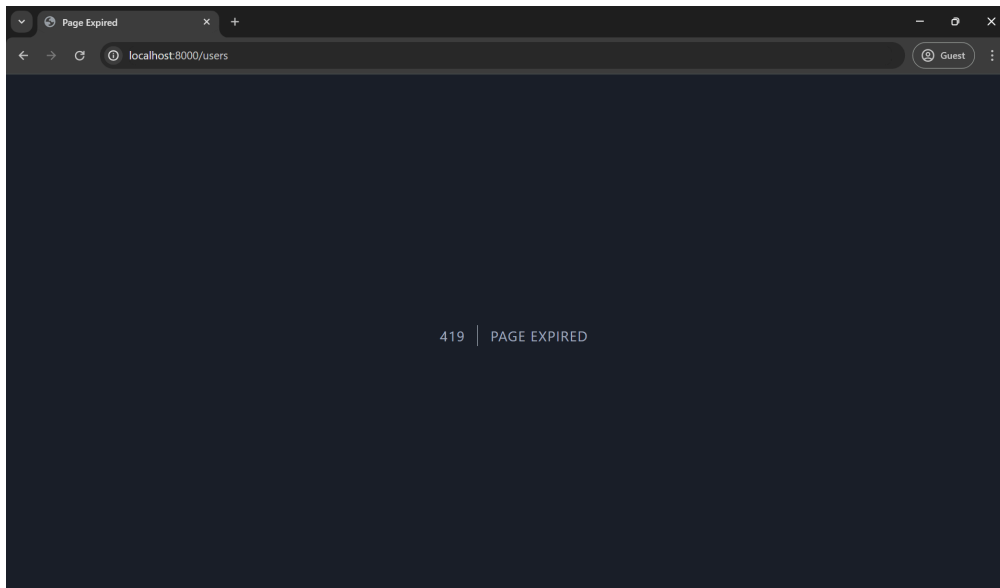
Ketika kita jalankan maka akan terlihat seperti ini.



The screenshot shows a web browser window with a single tab titled "User Form". The address bar displays "localhost:8000/users/create". Below the browser window, a form is visible with two input fields. The first field is labeled "Nama" and contains the text "rijal". The second field is labeled "Umur" and contains the number "24". To the right of the "Umur" field is a button labeled "Submit".

Tampilan Sebelum Submit Form.

Ketika kita submit maka akan diarahkan ke halaman blank dan akan memberikan pesan page expired.



Tampilan Error Dikarenakan Tidak Diberikan Token CSRF Didalam Form.

Ini dikarenakan setiap form yang kita buat, kita memerlukan sebuah token yang di generate oleh laravel, untuk mendapatkan token yang diperlukan kita dapat sisipkan @csrf di dalam form yang telah kita buat. Seperti ini.

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

@extends("layouts.app")

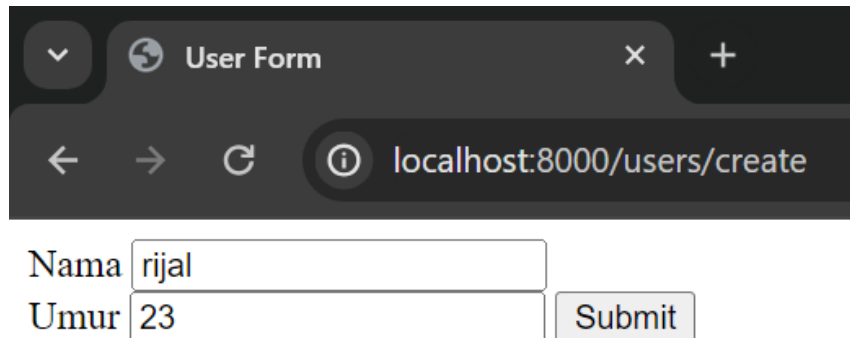
@section("title", "User Form")
@section("content")
<form action="{{ route('users.store') }}" method="POST">
    @csrf
    <label for="">Nama</label>
    <input type="text" name="nama"><br>
    <label for="">Umur</label>
    <input type="text" name="umur">
    <button type="submit">Submit</button>
</form>

@isset($payload)
    <h1>{{ $payload["nama"] }}</h1>
    <h1>{{ $payload["umur"] }}</h1>
@endisset
@endsection
```

Penambahan CSRF Di Dalam Form.

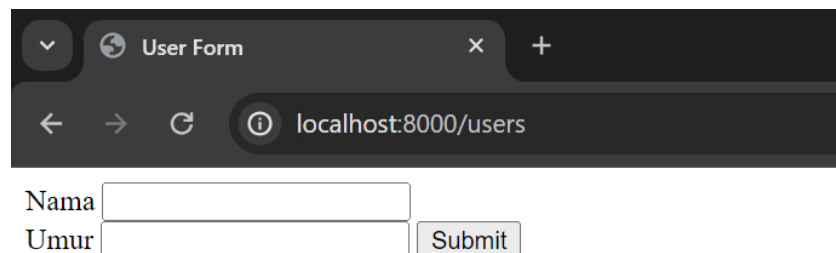


Ketika kita jalankan dan kita submit formnya, maka akan diarahkan ke halaman yang sama namun akan muncul pesan yang pesan dari data yang kita inputkan di dalam form.



A screenshot of a web browser window. The tab is titled 'User Form'. The address bar shows 'localhost:8000/users/create'. Below the address bar, there is a form with two input fields: 'Nama' with the value 'rijal' and 'Umur' with the value '23'. To the right of the 'Umur' field is a 'Submit' button.

Tampilan Sebelum Submit Form.



A screenshot of a web browser window. The tab is titled 'User Form'. The address bar shows 'localhost:8000/users'. Below the address bar, there is a form with two empty input fields: 'Nama' and 'Umur'. To the right of the 'Umur' field is a 'Submit' button.

**Haloo namaku, rijal**

**Umurku sekarang, 24 tahun**

**Kenalan yukk**

Tampilan Setelah Submit Form.

Pada saat kita membuat form, method yang tersedia di dalam HTML hanya POST dan GET, kita tidak bisa membuat permintaan atau request seperti PUT, PATCH,

atau DELETE. Untuk membuat permintaan tersebut, kita perlu menambahkan field tersembunyi `_method` untuk menipu kata kerja HTTP itu. Directive `@method` dapat membuat field tersebut. Kita coba ubah route untuk memproses data kita menjadi patch, seperti ini.

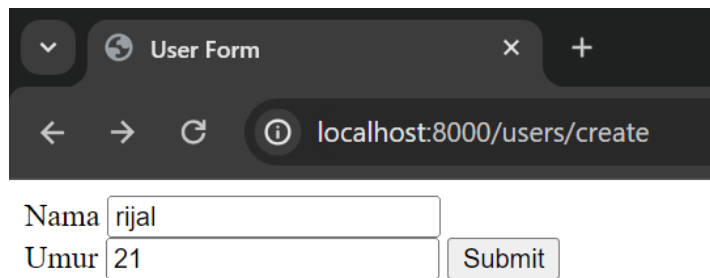
```
// routes/web.php

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

    Route::get('/users/{id}', "show")->name("users.detail");
    Route::patch("/users", "store")->name("users.store");
});
```

Mengubah Route Store Menjadi Patch.

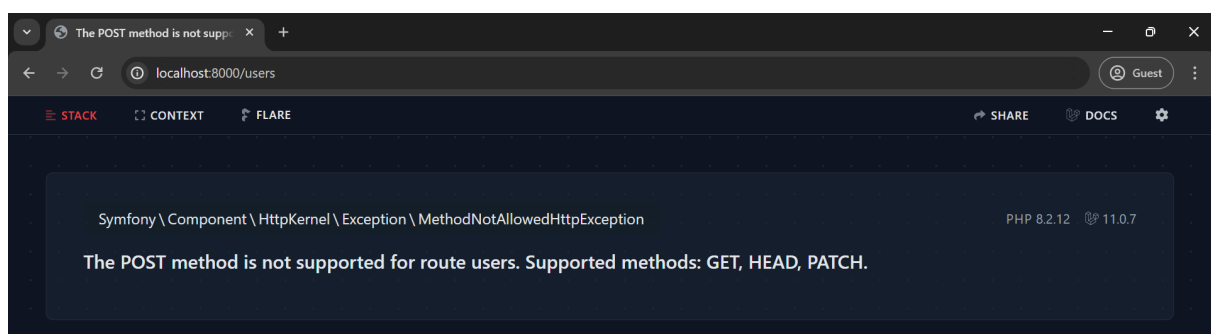
Ketika kita submit form akan muncul error method tidak mendukung.



Form details:

- Form Title: User Form
- URL: localhost:8000/users/create
- Field 1: Nama (rijal)
- Field 2: Umur (21)
- Action: Submit

Tampilan Sebelum Di Submit.



Tampilan Error Setelah Di Submit.

Untuk itu kita bisa menggunakan directive `@method` di dalam form kita, lalu sesuaikan dengan method ada didalam route yang telah kita buat.

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

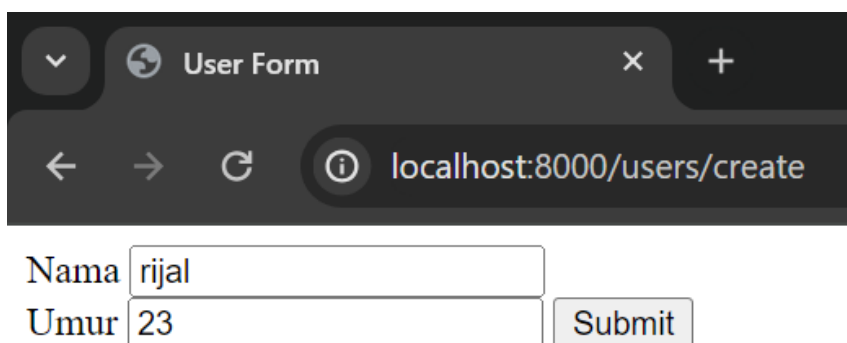
@extends("layouts.app")

@section("title", "User Form")
@section("content")
<form action="{{ route('users.store') }}" method="POST">
    @csrf
    @method("PATCH")
    <label for="">Nama</label>
    <input type="text" name="nama"><br>
    <label for="">Umur</label>
    <input type="text" name="umur">
    <button type="submit">Submit</button>
</form>

@isset($payload["nama"])
    <h1>Halo namaku, {{ $payload["nama"] }}</h1>
    <h1>Umurku sekarang, {{ $payload["umur"] }} tahun</h1>
    <h1>Kenalan yuk</h1>
@endisset
@endsection
```

Penambahan Method Di Dalam Form.

Lalu kita submit form tadi, maka akan muncul pesan dengan data yang sesuai dengan yang ada didalam form.



Browser: User Form

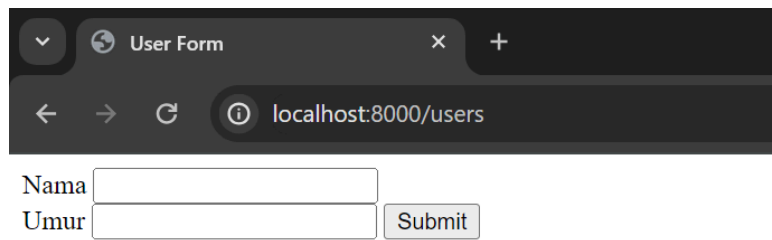
URL: localhost:8000/users/create

Nama: rijal

Umur: 23

Submit

Tampilan Sebelum Submit Form.



The screenshot shows a web browser window with a single tab titled 'User Form'. The address bar displays 'localhost:8000/users'. Below the browser window, there is a form with two input fields. The first field is labeled 'Nama' and the second is labeled 'Umur'. To the right of the 'Umur' field is a button labeled 'Submit'.

**Haloo namaku, rijal**

**Umurku sekarang, 24 tahun**

**Kenalan yukk**

Tampilan Setelah Submit Form.

#### **E. Validation.**

Ketika kita membuat sebuah form, terkadang kita perlu membuat sebuah validasi agar data yang dimasukkan bisa lebih aman, didalam laravel kita bisa buat sebuah validasi untuk form seperti minimal dari suatu input harus terdiri dari beberapa huruf, atau file yang dikirimkan harus berupa gambar.

Untuk membuat validasi pada saat form di kirim yang kita buat bisa kita bisa buat manual seperti ini.

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

use Illuminate\Http\Request;

class UserController extends Controller{
    ...

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

        if($payload["nama"] == null || $payload["umur"] == null){
            return view("users.create", ["error" => "Input salah"]);
        }

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

### Membuat Validasi Secara Manual.

Kita tambahkan kondisi jika nama dan umur dari payload nilainya kosong, maka tampilkan pesan error.

Lalu didalam view create user kita tambahkan isset untuk memeriksa suatu variabel sudah didefinisikan sebelumnya atau belum didefinisikan, lalu didalamnya kita tampilkan pesan errornya. Seperti ini.

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

@extends("layouts.app")
@section("title", "User Form")
@section("content")

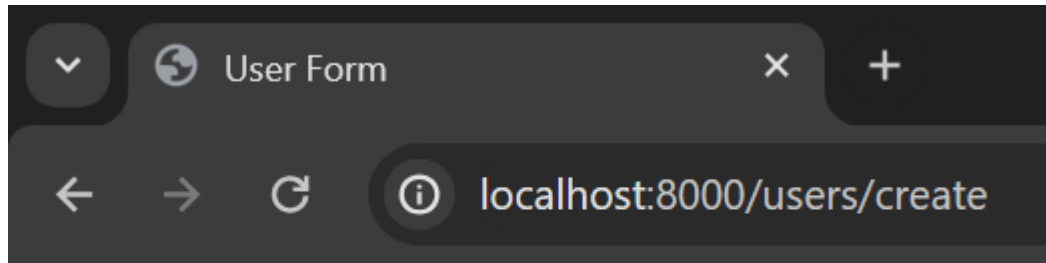
    @isset($error)
        <p style="color: red">{{ $error }}</p>
    @endisset

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

    @isset($payload)
        <h1>Haloo namaku, {{ $payload["nama"] }}</h1>
        <h1>Umurku sekarang, {{ $payload["umur"] }} tahun</h1>
        <h1>Kenalan yukkk</h1>
    @endisset
@endsection
```

Menambahkan Isset Untuk Menampilkan Pesan Error.

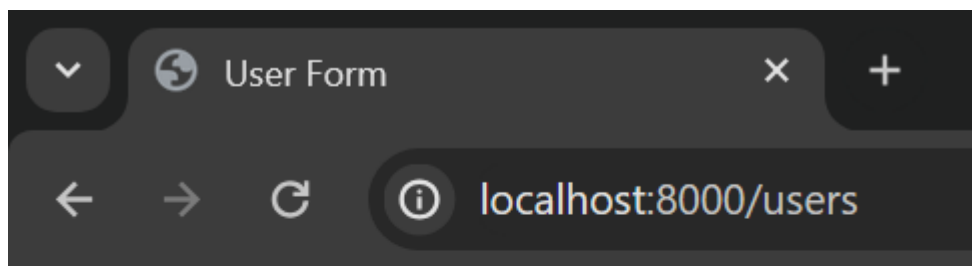
Ketika kita jalankan maka akan terlihat seperti ini.



Nama

Umur

Tampilan Sebelum Submit Form.



**Input salah**

Nama

Umur

Tampilan Setelah Submit Form.

Kita sudah membuat validasi secara manual menggunakan if dan menampilkannya di dalam view. Didalam laravel kita bisa membuat sebuah validasi secara otomatis hanya dengan memanggil fungsi validate yang ada didalam request, seperti ini.

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

use Illuminate\Http\Request;

class UserController extends Controller{
    public function store(Request $request){
        $payload = $request->validate([
            "nama" => ["required", "min:3"],
            "umur" => ["required", "min:1", "integer"],
        ]);

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

Membuat Validasi Dengan Validator Pada Laravel.

Lalu didalam view untuk menampilkan pesan error, kita perlu check terlebih dahulu apakah terdapat ketidaksesuaian dengan validasi yang kita berikan dari data yang dikirim form. Jika ada maka di dalamnya kita lakukan perulangan untuk menampilkan semua pesan error.

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

@extends("layouts.app")
@section("title", "User Form")
@section("content")

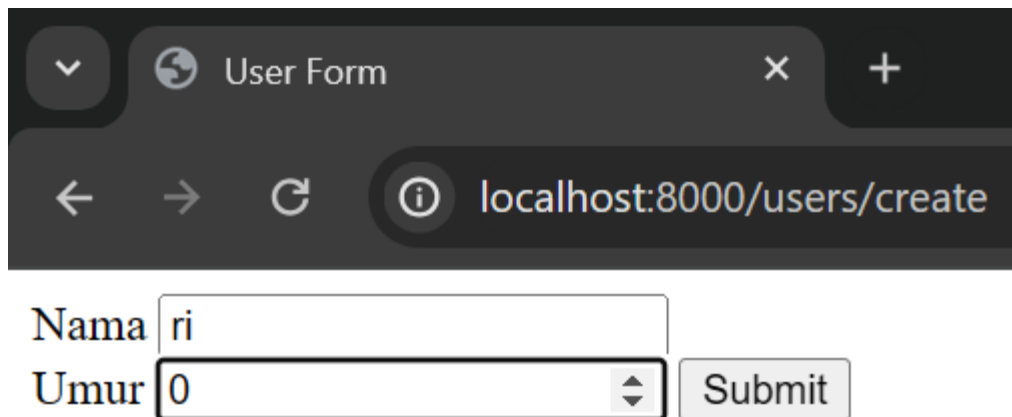
@if($errors->any())
    @foreach ($errors->all() as $error)
        <p style="color: red">{{ $error }}</p>
    @endforeach
@endif

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

@isset($payload)
    <h1>Halo namaku, {{ $payload["nama"] }}</h1>
    <h1>Umurku sekarang, {{ $payload["umur"] }} tahun</h1>
    <h1>Kenalan yuk</h1>
@endisset
@endsection
```

Menambahkan Perulangan Untuk Menampilkan Semua Pesan error.

Ketika kita jalankan maka akan terlihat seperti ini.

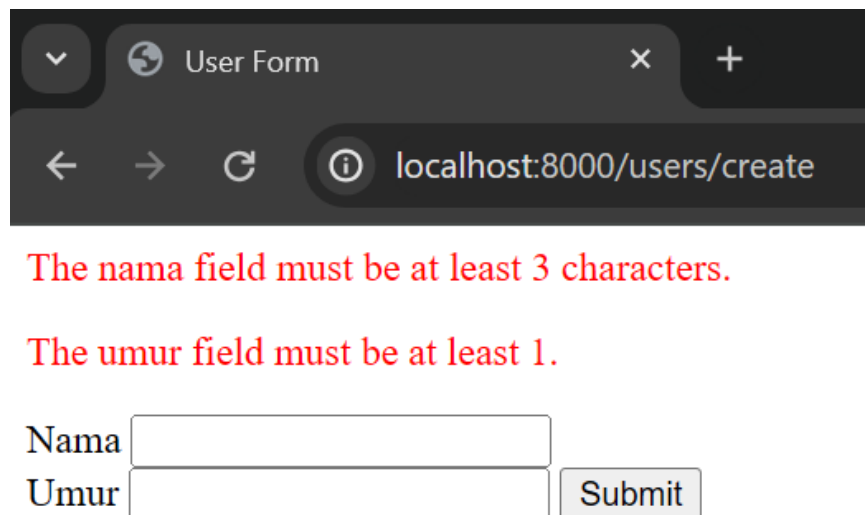


Browser: User Form  
URL: localhost:8000/users/create

Nama

Umur

Tampilan Sebelum Di Submit.



Browser: User Form  
URL: localhost:8000/users/create

The nama field must be at least 3 characters.

The umur field must be at least 1.

Nama

Umur

Tampilan Setelah Di Submit.

## F. Upload File.

Ketika kita membuat halaman form, terkadang kita ingin menyisipkan data berupa berkas atau file. Untuk membuat sebuah Upload File pertama tama kita perlu menambahkan `enctype` pada form yang akan kita tambahkan upload file, lalu beri nilai `multipart/form-data`. Lalu kita tambahkan juga input dengan type file, dan juga kita tampilkan gambar yang sudah dikirim, seperti ini.



```

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

@extends("layouts.app")
@section("title", "User Form")
@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="nama">Nama</label>
    <input type="text" id="nama" name="nama"><br>
    <label for="umur">Umur</label>
    <input type="number" id="umur" name="umur">
    <label for="gambar">Gambar</label>
    <input type="file" id="gambar" name="gambar">
    <button type="submit">Submit</button>
</form>

@isset($payload)
    <h1>Halo namaku, {{ $payload["nama"] }}</h1>
    <h1>Umurku sekarang, {{ $payload["umur"] }} tahun</h1>
    <h1>Kenalan yuk</h1>
@endisset
@endsection

```

Menambahkan Enctype Dan Input File Di Dalam Form.

Lalu didalam controller, untuk menyimpan file yang dikirim dari form, kita dapat menggunakan fungsi move untuk menentukan file yang dikirim disimpan dimana dan diberikan nama seperti apa. Lalu untuk menghindari pemberian nama yang sama pada file yang dikirim, kita buat nama file yang dikirim menjadi berupa huruf acak, untuk membuat huruf acak kita bisa menggunakan fungsi `hashName`. Agar file yang dikirim tidak sembarangan jenis file bisa disimpan, kita bisa memberikan validasi di dalamnya.

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

use Illuminate\Http\Request;

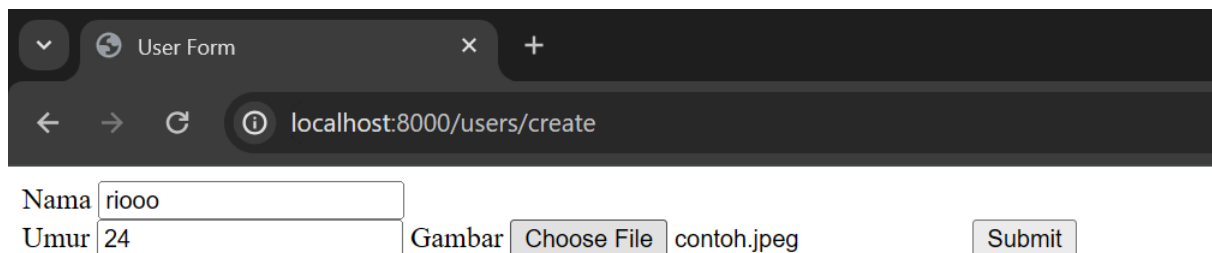
class UserController extends Controller{
    public function store(Request $request){
        $payload = $request->validate([
            "nama" => ["required", "min:3"],
            "umur" => ["required", "min:1", "integer"],
            "gambar" => ["required", "file", "extensions:png,jpg,jpeg"]
        ]);

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

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

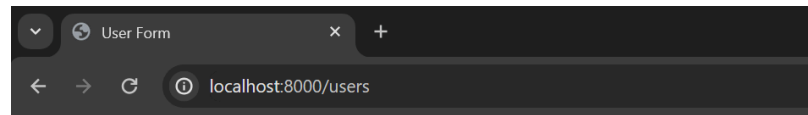
Menambahkan Code Untuk Memproses File Yang Dikirim.

Lalu kita jalankan di browser maka akan terlihat seperti ini.



Nama   
 Umur  Gambar  contoh.jpeg

Tampilan Sebelum Di Submit.



**Haloo namaku, riooo**

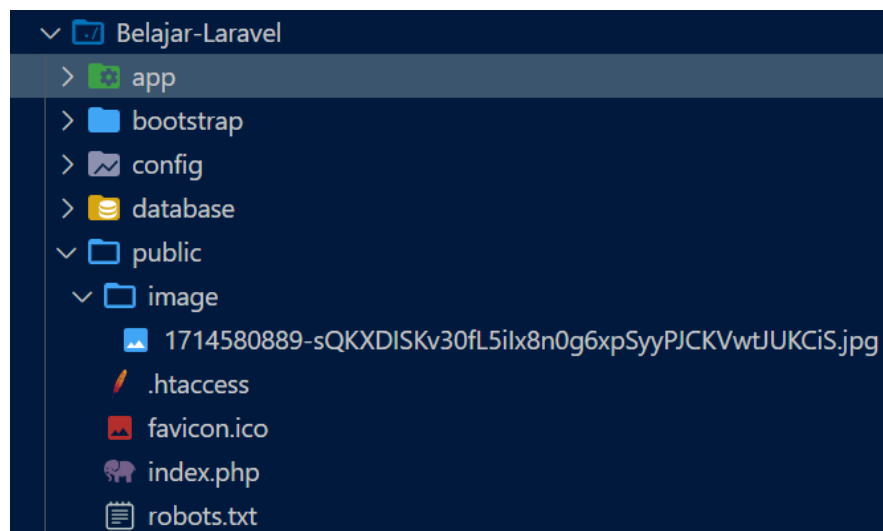
**Umurku sekarang, 24 tahun**

**Kenalan yukk**



Tampilan Setelah Di Submit.

Lalu untuk melihat semua gambar yang telah disimpan bisa kita lihat di dalam folder `public/image`.



Tempat Disimpannya Gambar Yang Dikirim.

#### **4. Referensi.**

- <https://laravel.com/docs/11.x/controllers#writing-controllers>
- <https://laravel.com/docs/11.x/routing#form-method-spoofing>