

# Laporan Modul 6: Model dan Laravel Eloquent

---

**Mata Kuliah:** Workshop Web Lanjut

**Nama:** Ilham Syahdienar

**NIM:** 2024573010029 **Kelas:** TI-2C

---

## Abstrak

Laporan ini membahas tentang bagaimana Laravel 12 mengelola sistem database menggunakan lapisan abstraksi yang kuat, termasuk penggunaan Migration, Seeder, dan Eloquent ORM. Dalam framework Laravel, pengembang dapat dengan mudah melakukan konfigurasi koneksi database melalui file .env, melakukan version control skema menggunakan Migration, mengisi data awal dengan Seeder, serta berinteraksi dengan tabel database menggunakan pendekatan berorientasi objek melalui Eloquent ORM. Dengan adanya fitur-fitur ini, pengelolaan data menjadi lebih efisien, terstruktur, dan konsisten dengan prinsip MVC (Model-View-Controller).

---

## 1. Dasar Teori

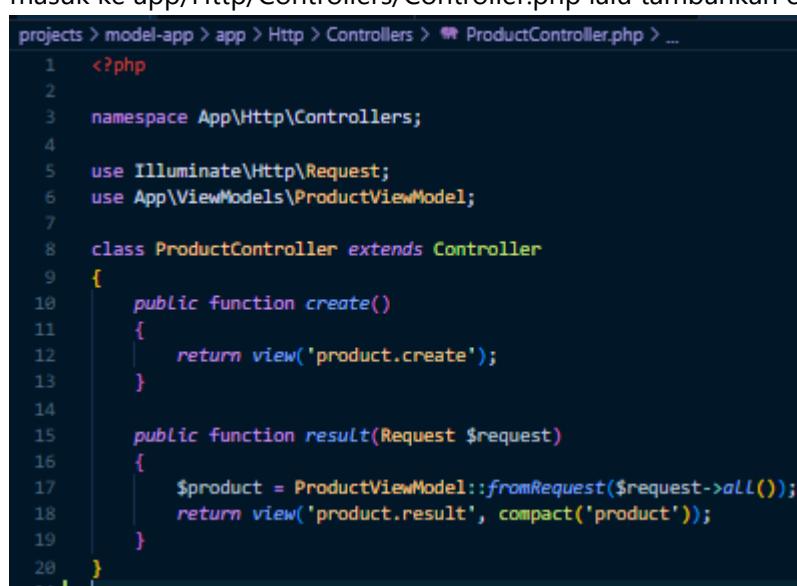
- Apa itu konfigurasi database di Laravel dan bagaimana cara mengurnya?
  - Dalam Laravel, konfigurasi database adalah langkah awal agar aplikasi dapat terhubung dengan sistem manajemen basis data seperti MySQL, PostgreSQL, SQLite, atau SQL Server. Pengaturan koneksi database biasanya dilakukan di file .env sehingga mudah diubah tanpa menyentuh kode program. Di dalam file tersebut, kita mendefinisikan nama database, host, port, username, dan password, misalnya DB\_DATABASE=myapp, DB\_USERNAME=root, dan DB\_PASSWORD=secret. Laravel secara otomatis membaca konfigurasi ini dan menghubungkannya melalui file config/database.php. Untuk memastikan koneksi berhasil, perintah php artisan migrate:status dapat digunakan.
- Apa yang dimaksud dengan Migration dalam Laravel dan apa fungsinya?
  - Migration di Laravel berfungsi untuk mengatur dan memodifikasi struktur tabel pada database menggunakan kode PHP, bukan SQL secara langsung. Dengan Migration, pengembang dapat melacak perubahan struktur database seiring perkembangan aplikasi, mirip dengan sistem version control. Migration dibuat dengan perintah php artisan make:migration, lalu diisi menggunakan class Schema dan objek Blueprint untuk mendefinisikan kolom tabel. Misalnya, Schema::create('users', function (Blueprint \$table) { ... }); digunakan untuk membuat tabel baru. Migration dijalankan menggunakan php artisan migrate, yang secara otomatis menerapkan perubahan struktur ke database.
- Apa fungsi Seeder?
  - Seeder digunakan untuk mengisi tabel database dengan data awal atau data percobaan (dummy data). Hal ini sangat berguna dalam tahap pengembangan agar aplikasi memiliki data contoh tanpa perlu memasukkan secara manual. Untuk membuat seeder, digunakan perintah php artisan make:seeder, lalu di dalamnya dapat ditulis perintah seperti User::create([...]) untuk menambahkan data ke tabel tertentu. Setelah seeder dibuat, pengembang dapat menjalankannya menggunakan php artisan db:seed --class=NamaSeeder. Dengan demikian,

Seeder membantu mempercepat proses pengujian dan memastikan database selalu memiliki data awal yang konsisten

- Apa yang dimaksud dengan Eloquent ORM
    - Eloquent ORM (Object Relational Mapper) adalah fitur Laravel yang memungkinkan pengembang berinteraksi dengan database menggunakan pendekatan berorientasi objek. Setiap tabel di database diwakili oleh sebuah model, dan setiap baris di tabel dianggap sebagai objek.
  - Apa hubungan antara Model, Entity, dan Repository Pattern dalam Laravel?
    - Model di Laravel berfungsi sebagai representasi tabel database dan menjadi tempat logika interaksi data disimpan. Sementara itu, Entity adalah kelas sederhana yang hanya berfungsi untuk membawa data tanpa bergantung pada fitur Laravel. Dalam beberapa arsitektur, digunakan juga DTO (Data Transfer Object) yang bertugas memindahkan data antar lapisan aplikasi secara terstruktur. Di sisi lain, Repository Pattern digunakan untuk memisahkan logika bisnis dari logika akses data. Dengan cara ini, kode menjadi lebih modular, mudah diuji, dan perubahan pada lapisan data tidak akan mempengaruhi lapisan lain dalam aplikasi.
- 

## 2. Langkah-Langkah Praktikum

- **2.1 Praktikum 1 - Menggunakan Model untuk Binding Form dan Display**
  - 1. Buat project baru dengan nama `model-app` di folder projects tempat menyimpan folder project pada pertemuan sebelumnya menggunakan vscode, buka git bash, dan masuk ke direktori projects:
    - `composer create-project laravel/laravel model-app`
    - masuk ke direktori `model-app`
  - 2. Buat Controller dengan nama controller:
    - `php artisan make:controller ProductController`
  - 3. Selanjutnya kita akan buat metode untuk menangani logika pada controller dengan cara masuk ke `app/Http/Controllers/Controller.php` lalu tambahkan code berikut:



```

projects > model-app > app > Http > Controllers > ProductController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\ViewModels\ProductViewModel;
7
8  class ProductController extends Controller
9  {
10     public function create()
11     {
12         return view('product.create');
13     }
14
15     public function result(Request $request)
16     {
17         $product = ProductViewModel::fromRequest($request->all());
18         return view('product.result', compact('product'));
19     }
20 }
21

```

- 4. Setelah membuat controller, selanjutnya kita akan membuat sebuah direktori baru bernama `ViewModels` di direktori `app`, Buat `ProductViewModel.php` di dalam direktori `app/ViewModels` Kemudian buat kelas model sederhana untuk menyimpan data produk. Kelas ini akan memiliki properti untuk nama produk, harga, dan deskripsi:

```

<?php
namespace App\ViewModels;

class ProductViewModel
{
    public string $name;
    public float $price;
    public string $description;

    public function __construct(string $name = '', float $price = 0,
        string $description = '')
    {
        $this->name = $name;
        $this->price = $price;
        $this->description = $description;
    }

    public static function fromRequest(array $data): self
    {
        return new self(
            $data['name']?? '',
            (float)($data['price'] ?? 0),
            $data['description'] ?? '',
        );
    }
}

```

- 4. Setelah membuat controller, selanjutnya definisikan route nya di routes/web.php, tambahkan seperti ini:

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/product/create', [ProductController::class, 'create'])-
>name('product.create');
Route::post('/product/result', [ProductController::class, 'result'])-
>name('product.result');

```

- 5. Di **resources/views** buat direktori baru dengan nama **product**, lalu di direktori tersebut buat dua file yaitu:
  - **create.blade.php**

- result.blade.php isi file tersebut dengan kode program seperti gambar di bawah ini:

- [create.blade.php](#)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Create Product</title>
        <link rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.cs
s">
    </head>
    <body class="container py-5">
        <h2>Create Product (No Database)</h2>
        <form action="{{ route('product.result') }}" method="POST">
            @csrf
            <div class="mb-3">
                <label class="form-label">Name</label>
                <input type="text" name="name" class="form-control"
required>
            </div>
            <div class="mb-3">
                <label class="form-label">Price</label>
                <input type="number" name="price" step="0.01" class="form-
control" required>
            </div>
            <div class="mb-3">
                <label class="form-label">Description</label>
                <textarea name="description" class="form-control">
                </textarea>
            </div>
            <button type="submit" class="btn btn-primary">Submit
Product</button>
        </form>

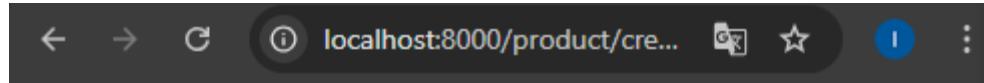
    </body>
</html>
```

- [result.blade.php](#)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Product Result</title>
        <link rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.cs
s">
    </head>
    <body class="container py-5">
        <h2>Submitted Product Details</h2>
        <ul class="list-group">
```

```
<li class="list-group-item"><strong>Name:</strong> {{ $product->name }}</li>
    <li class="list-group-item"><strong>price:</strong> ${{ number_format($product->price, 2) }}</li>
    <li class="list-group-item"><strong>Description:</strong> {{ $product->description }}</li>
</ul>
<a href="{{ route('product.create') }}" class="btn btn-link mt-3">Submit Another Product</a>
</body>
</html>
```

- o 6. Jalankan project laravel yang kita buat dengan mengetik `php artisan serve` diterminal atau git bash
- o 7. Buka browser lalu ketik `http://localhost:8000/product/create`, maka nanti akan keluar seperti gambar dibawah ini:



## Create Product (No Database)

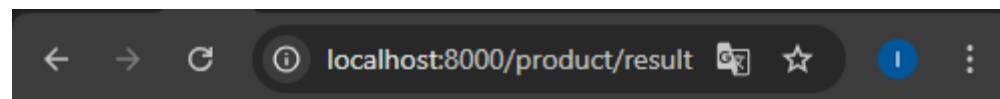
Name

Price

Description

Submit Product

lalu jika sudah mengisi formnya dan kita akan menekan tombol submit maka akan menampilkan data yang dikirim:



## Submitted Product Details

**Name:** Kulkas

**price:** Rp1,000,000.00

**Description:** Kulkas 1 pintu merk LG 250 watt

[Submit Another Product](#)

jika kita mencoba mengirim form tanpa mengisi field yang wajib maka akan melihat error validasi

---

- **2.2 Praktikum 2 – Menggunakan DTO (Data Transfer Object)**

- 1. Buat project baru yaitu **dto-app**
- 2. Selanjutnya kita akan membuat sebuah direktori baru bernama **DTO** di direktori **app**, Buat **ProductDTO.php** di dalam direktori **app/DTO** Kemudian buat kelas model sederhana untuk menyimpan data produk. Kelas ini akan memiliki properti untuk nama produk, harga, dan deskripsi:

```
<?php

namespace App\DTO;
class ProductDTO
{
    public string $name;
    public string $price;
    public string $description;

    public function __construct(string $name, float $price, string
    $description)
    {
        $this->name = $name;
        $this->price = $price;
        $this->description = $description;
    }

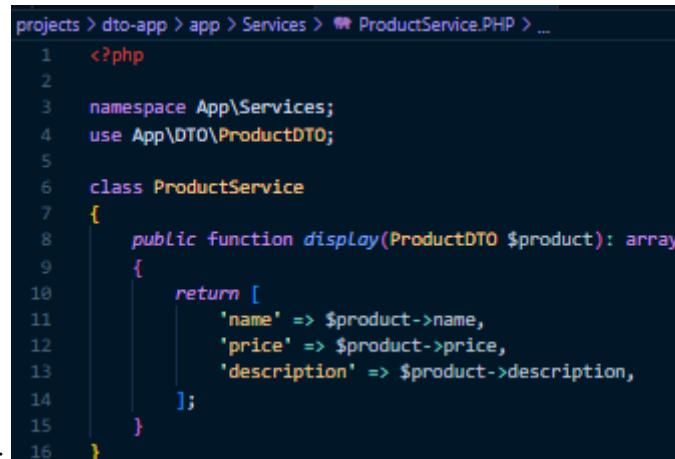
    public static function fromRequest(array $data): self
    {
        return new self(
            $data['name'],
            $data['price'],
            $data['description']
        );
    }
}
```

```

    {
        return new self(
            $data['name'] ?? '',
            (float)($data['price'] ?? 0),
            $data['description'] ?? ''
        );
    }
}

```

- 3. Masih di direktori `app`, Buat direktori baru lagi bernama `Services`, di dalam direktori `app/Services` buat file `ProductServices.php` Kemudian buat kelas yang akan menangani logika bisnis. Kelas ini akan menerima DTO sebagai input dan mengembalikan



```

projects > dto-app > app > Services > ProductService.php > ...
1  <?php
2
3  namespace App\Services;
4  use App\DTO\ProductDTO;
5
6  class ProductService
7  {
8      public function display(ProductDTO $product): array
9      {
10         return [
11             'name' => $product->name,
12             'price' => $product->price,
13             'description' => $product->description,
14         ];
15     }
16 }

```

data yang telah diformat:

- 4. Selanjutnya kita buat controller, dengan nama `ProductController`, untuk membuat controller tersebut kita dapat gunakan perintah:
  - `php artisan make:controller ProductController`
- 5. Lalu pada file controller yang sudah dibuat yaitu `ProductController`, tambahkan logika dengan kode program seperti dibawah ini:

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\DTO\ProductDTO;
use App\Services\ProductService;

class ProductController extends Controller
{
    public function create()
    {
        return view('product.create');
    }

    public function result(Request $request)
    {
        $dto = ProductDTO::fromRequest($request->all());
        $service = new ProductService();
        $product = $service->display($dto);
    }
}

```

```

        return view('product.result', compact('product'));
    }
}

```

- 6. Selanjutnya kita tambahkan route ProductController di `routes/web.php`, ikuti seperti kode program dibawah ini:

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/product/create', [ProductController::class, 'create'])-
>name('product.create');
Route::post('/product/result', [ProductController::class, 'result'])-
>name('product.result');

```

- 5. Setelah menambahkan rute ProductController di routes/web.php, kita buat folder baru di `resources/views` dengan nama `product`, dan isi folder tersebut dengan file-file ini:
  - `create.blade.php`
  - `result.blade.php` isikan kode program disetiap file seperti dibawah ini: `create.blade.php`

```

<!DOCTYPE html>
<html>
<head>
    <title>Create Product DTO</title>
    <link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.m
in.css" rel="stylesheet">
</head>
<body class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <h2 class="mb-4">Create Product</h2>
            <form method="POST" action="{{ route('product.result') }}">
                @csrf
                <div class="mb-3">
                    <label class="form-label">Name</label>
                    <input name="name" class="form-control" required>
                </div>
                <div class="mb-3">
                    <label class="form-label">Price</label>
                    <input name="price" type="number" step="0.01">
                </div>
            </form>
        </div>
    </div>
</body>

```

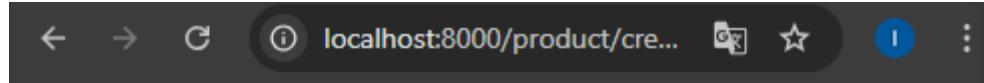
```
        class="form-control" required>
            </div>
            <div class="mb-3">
                <label class="form-label">Description</label>
                <textarea name="description" class="form-control" rows="3"></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Submit Product</button>
        </form>
    </div>
</body>
</html>
```

### result.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Product Result</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <h2 class="mb-4">Product DTO Result</h2>
            <div class="card">
                <div class="card-header">
                    <h5 class="card-title mb-0">Product Details</h5>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item">
                        <strong>Name:</strong> {{ $product['name'] }}
                    <li class="list-group-item">
                        <strong>Price:</strong> ${{ number_format($product['price'], 2) }}
                    <li class="list-group-item">
                        <strong>Description:</strong> {{ $product['description'] }}
                
            </div>
            <a href="{{ route('product.create') }}" class="btn btn-secondary mt-3">Submit Another Product</a>
        </div>
    </div>
</body>
```

```
</html>
```

- o 6. Buka browser lalu ketik <http://localhost:8000/product/create>, maka nanti akan keluar seperti gambar dibawah ini:



## Create Product (No Database)

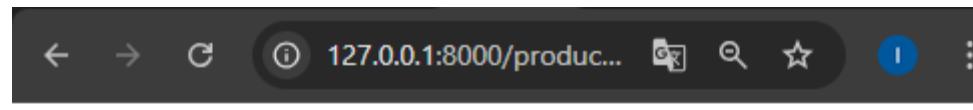
Name

Price

Description

**Submit Product**

lalu jika  
sudah mengisi formnya dan kita akan menekan tombol submit maka akan menampilkan  
data yang dikirim:



## Product DTO Result

Product Details
Name: Lampu
Price: \$20,000.00
Description: Lampu Merk LG 7 watt

Submit Another Product

jika kita mencoba mengirim form tanpa mengisi field yang wajib maka akan melihat error validasi

### • 2.3 Praktikum 3 – Membangun Aplikasi Web Todo Sederhana dengan Laravel 12, Eloquent ORM, dan MySQL

- 1. Buat project baru dengan nama `todo-app-mysql` di folder projects tempat menyimpan folder project pada pertemuan sebelumnya menggunakan vscode, buka git bash, dan masuk ke direktori projects:
  - `composer create-project laravel/laravel todo-app-mysql`
  - masuk ke direktori `todo-app-mysql`
- 2. Buat Database tododb; pada phpadmin, setelah itu install dependency MySQL dengan:  
`composer require doctrine/dbal`
- 3. Konfigurasi pada env dengan menyesuaikan nama database yang digunakan dan jenis databasenya: DB\_CONNECTION=mysql DB\_HOST=127.0.0.1 DB\_PORT=3306 DB\_DATABASE=tododb DB\_USERNAME=database\_anda DB\_PASSWORD=password-database\_jika Ada setelah itu jalankan perintah `php artisan config:clear`
- 4. Lalu kita akan membuat file migrasi, ini berguna untuk mendefinisikan struktur tabel todos, jalankan perintah: `php artisan make:migration create.todos_table`
- 5. di direktori `database/migrations/` cari file yang telah dibuat yaitu `YYYY_MM_DD_create.todos_table.php` dan perbarui file nya seperti kode berikut:

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('todos', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->text('description');
            $table->boolean('completed');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('todos');
    }
};
```

```

*/
public function up(): void
{
    Schema::create('todos', function (Blueprint $table) {
        $table->id();
        $table->string('task');
        $table->boolean('completed')->default(false);
        $table->timestamps();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('todos');
}
;

```

Setelah mengedit file migrasi, kita perlu menjalankan migrasi untuk membuat tabel todos dalam database. jalankan `php artisan migrate`

- 6. Selanjutnya kita akan membuat seeder untuk data dummy, untuk membuat seeder jalankan perintah: `php artisan make:seeder TodoSeeder` Buka file yang dihasilkan di `database/seeders/TodoSeeder.php` dan perbarui dengan kode program:

```

<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Carbon\Carbon;

class TodoSeeder extends Seeder
{
    public function run()
    {
        DB::table('todos')->insert([
            [
                'task' => 'Belanja bahan makanan',
                'completed' => false,
                'created_at' => Carbon::now(),
                'updated_at' => Carbon::now()
            ],
            [
                'task' => 'Beli buah-buahan',
                'completed' => false,
                'created_at' => Carbon::now(),
                'updated_at' => Carbon::now()
            ]
        ]);
    }
}

```

```

        'updated_at' => Carbon::now()
    ],
    [
        'task' => 'Selesaikan proyek Laravel',
        'completed' => true,
        'created_at' => Carbon::now(),
        'updated_at' => Carbon::now()
    ],
]);
}
}

```

lalu jalankan perintah berikut `php artisan db:seed --class=TodoSeeder` perintah ini untuk mengisi database dengan data dummy

- 7. Selanjutnya kita akan membuat model agar bisa berinteraksi dengan tabel todos menggunakan Eloquent ORM, jalankan perintah: `php artisan make:model Todo` lalu buka file yang dihasilkan dari perintah tersebut di direktori `app/Models/Todo.php` lalu isi kode program seperti berikut:

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Todo extends Model
{
    use HasFactory;

    protected $fillable = ['task', 'completed'];
}

```

- 8. Selanjutnya kita akan membuat controller yang akan menangani operasi CRUD untuk tabel todos, buat nama controllernya `TodoController` dengan perintah `php artisan make:controller TodoController`. lalu buat edit code nya seperti ini:

```

<?php

namespace App\Http\Controllers;

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

class TodoController extends Controller
{
    public function index()

```

```

    {
        $todos = Todo::all();
        return view('todos.index', compact('todos'));
    }

    public function create()
    {
        return view('todos.create');
    }

    public function store(Request $request)
    {
        $request->validate(['task' => 'required|string']);
        Todo::create(['task' => $request->task]);
        return redirect()->route('todos.index')->with('success', 'Task added successfully!');
    }

    public function show(Todo $todo)
    {
        return view('todos.show', compact('todo'));
    }

    public function edit(Todo $todo)
    {
        return view('todos.edit', compact('todo'));
    }

    public function update(Request $request, Todo $todo)
    {
        $request->validate(['task' => 'required|string']);
        $todo->update(['task' => $request->task]);
        return redirect()->route('todos.index')->with('success', 'Task updated successfully!');
    }

    public function destroy(Todo $todo)
    {
        $todo->delete();
        return redirect()->route('todos.index')->with('success', 'Task deleted successfully!');
    }
}

```

- 9. Selanjutnya kita tambahkan route ProductController di [routes/web.php](#), ikuti seperti kode program dibawah ini:

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TodoController;

```

```
Route::get('/', [TodoController::class, 'index'])->name('todos.index');
Route::get('/todos/create', [TodoController::class, 'create'])->name('todos.create');
Route::post('/todos', [TodoController::class, 'store'])->name('todos.store');
Route::get('/todos/{todo}', [TodoController::class, 'show'])->name('todos.show');
Route::get('/todos/{todo}/edit', [TodoController::class, 'edit'])->name('todos.edit');
Route::patch('/todos/{todo}', [TodoController::class, 'update'])->name('todos.update');
Route::delete('/todos/{todo}', [TodoController::class, 'destroy'])->name('todos.destroy');
```

- 10. Selanjutnya kita akan membuat tampilan blade dengan bootstrap, di `resources/views` buat direktori baru yaitu: `layouts todos` di `resources/views/layouts` buat file baru yaitu `app.blade.php` kemudian isikan kode program:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield('title', 'Todo App')</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container mt-4">

    <h1 class="text-center mb-4">Laravel 12 Todo App</h1>

    @if(session('success'))
        <div class="alert alert-success">{{ session('success') }}</div>
    @endif

    <nav class="mb-3">
        <a href="{{ route('todos.index') }}" class="btn btn-primary">Todo List</a>
        <a href="{{ route('todos.create') }}" class="btn btn-success">Add New Task</a>
    </nav>

    @yield('content')

</body>
</html>
```

di `resources/views/todos` buat beberapa file blade:

- o `index.blade.php`: Tampilan ini akan menampilkan daftar todos. lalu isi dengan kode berikut:

```

@extends('layouts.app')

@section('title', 'Daftar Todo')

@section('content')
    <h2>Daftar Todo</h2>

    <ul class="list-group">
        @foreach($todos as $todo)
            <li class="list-group-item d-flex justify-content-between align-items-center">
                {{ $todo->task }}
                <div>
                    <form action="{{ route('todos.show', $todo->id) }}" method="GET" class="d-inline">
                        <button type="submit" class="btn btn-info btn-sm">Detail</button>
                    </form>
                    <form action="{{ route('todos.edit', $todo->id) }}" method="GET" class="d-inline">
                        <button type="submit" class="btn btn-warning btn-sm">Edit</button>
                    </form>
                    <form action="{{ route('todos.destroy', $todo->id) }}" method="POST" class="d-inline">
                        @csrf
                        @method('DELETE')
                        <button class="btn btn-danger btn-sm">Hapus</button>
                    </form>
                </div>
            </li>
        @endforeach
    </ul>
@endsection

```

- o `create.blade.php`: Tampilan ini akan menampilkan formulir untuk menambah todo baru. kemudian isi dengan kode program seperti ini:

```

@extends('layouts.app')

@section('title', 'Buat Task Baru')

@section('content')
    <h2>Buat Task Baru</h2>

    <form action="{{ route('todos.store') }}" method="POST" class="mt-3">
        @csrf
        <div class="mb-3">

```

```

        <label for="task" class="form-label">Nama Task</label>
        <input type="text" name="task" id="task" class="form-control"
required>
    </div>
    <button type="submit" class="btn btn-success">Tambah Task</button>
    <a href="{{ route('todos.index') }}" class="btn btn-
secondary">Kembali ke Daftar</a>
</form>

@endsection

```

- **edit.blade.php**: Tampilan ini akan menampilkan formulir untuk mengedit todo yang sudah ada.kemudian isi dengan kode program seperti ini:

```

@extends('layouts.app')
@section('title', 'Edit Task')

@section('content')
    <h2>Edit Task</h2>

    <form action="{{ route('todos.update', $todo->id) }}" method="POST"
class="mt-3">
    @csrf
    @method('PATCH')
    <div class="mb-3">
        <label for="task" class="form-label">Nama Task</label>
        <input type="text" name="task" id="task" class="form-control"
value="{{ $todo->task }}" required>
    </div>
    <button type="submit" class="btn btn-warning">Update Task</button>
    <a href="{{ route('todos.index') }}" class="btn btn-
secondary">Kembali ke Daftar</a>
</form>

@endsection

```

- **show.blade.php**: Tampilan ini akan menampilkan single todo.kemudian isi dengan kode program seperti ini:

```

@extends('layouts.app')
@section('title', 'Detail Task')
@section('content')
    <h2>Detail Task</h2>

    <div class="card mt-3">
        <div class="card-body">
            <h5 class="card-title">{{ $todo->task }}</h5>
            <p class="card-text">Status: {{ $todo->completed ? 'Selesai' :
'Belum Selesai' }}</p>

```

```

        <a href="{{ route('todos.edit', $todo->id) }}" class="btn btn-warning">Edit</a>
            <a href="{{ route('todos.index') }}" class="btn btn-secondary">Kembali ke Daftar</a>
        </div>
    </div>
@endsection

```

- Setelah membuat dan mengedit file blade untuk view, sekarang kita Jalankan aplikasi dengan `php artisan serve` dan coba kunjungi `http://127.0.0.1:8000` Tampilan nya

Daftar Todo			
Belanja bahan makanan	<a href="#">Detail</a>	<a href="#">Edit</a>	<a href="#">Hapus</a>
Beli buah-buahan	<a href="#">Detail</a>	<a href="#">Edit</a>	<a href="#">Hapus</a>
Selesaikan proyek Laravel	<a href="#">Detail</a>	<a href="#">Edit</a>	<a href="#">Hapus</a>

### 3. Hasil dan Pembahasan

- Laravel 12 menyediakan ekosistem yang komprehensif untuk bekerja dengan database melalui kombinasi migrasi, seeder, dan Eloquent ORM. Migrasi memberikan cara yang aman dan mudah untuk mengelola perubahan skema database tanpa menulis SQL mentah. Seeder mempermudah pengisian data awal secara otomatis untuk kebutuhan pengujian. Sementara itu, Eloquent ORM menyederhanakan operasi CRUD dengan gaya berorientasi objek, menjadikannya lebih mudah dibaca dan dipelihara oleh pengembang. Dalam praktik pengembangan aplikasi skala besar, pemisahan antara Model, Entity, DTO, dan penggunaan Repository Pattern membantu menjaga struktur kode tetap modular dan terorganisir. Developer dapat mengubah logika data tanpa harus menyentuh bagian lain dari aplikasi, sehingga meningkatkan fleksibilitas dan skalabilitas sistem

### 4. Kesimpulan

- Kesimpulan dari praktikum ini adalah bahwa framework Laravel terbukti sangat efektif dalam menangani proses form submission secara aman, efisien, dan terstruktur. Melalui penerapan berbagai

HTTP methods seperti POST, PUT/PATCH, dan DELETE, pengiriman data dapat dilakukan dengan cara yang sesuai dengan kebutuhan aplikasi. Laravel juga memiliki sistem CSRF Protection bawaan yang berfungsi melindungi aplikasi dari serangan Cross-Site Request Forgery dengan memastikan setiap permintaan berasal dari sumber yang sah. Selain itu, fitur validasi data di Laravel membantu memastikan bahwa input pengguna memenuhi aturan dan ketentuan yang berlaku, baik dari sisi format, tipe data, maupun kesesuaianya dengan tabel di database. Praktikum ini juga menunjukkan bagaimana validasi kustom dan pesan error spesifik dapat meningkatkan kejelasan komunikasi dengan pengguna. Implementasi multi-step form memperlihatkan fleksibilitas Laravel dalam mengelola data yang dikumpulkan secara bertahap menggunakan session, sehingga proses input menjadi lebih teratur dan tidak membingungkan. Secara keseluruhan, praktikum ini berhasil memperlihatkan bagaimana Laravel dapat digunakan untuk membangun sistem form yang aman, interaktif, dan user-friendly, sekaligus menjaga kualitas serta keamanan data yang diproses.

---

## 5. Referensi

- <https://developerawam.com/artikel/detail/belajar-data-transfer-object-dto-di-laravel>
- <https://hackmd.io/@mohdrzu/ryIIM1a0II>
- <https://buildwithangga.com/tips/mengenal-laravel-eloquent-pengertian-fungsi-dan-penggunaannya>