Laporan Modul 6: Model dan Laravel Eloquent

Mata Kuliah: Workshop Web Lanjut
Nama: Bunga Alfa Zahrah **NIM:** 2024573010023 **Kelas:** TI-2C

Abstrak

- Pada praktikum ini, dilakukan implementasi konsep Model dan Laravel Eloquent dalam membangun aplikasi berbasis web menggunakan framework Laravel. Model berfungsi sebagai representasi data dari database, sementara Eloquent ORM (Object Relational Mapping) memudahkan interaksi antara aplikasi dengan database melalui sintaks yang lebih sederhana dan berorientasi objek.
- Praktikum ini terdiri dari tiga bagian, yaitu penggunaan model sederhana untuk binding data pada form, penerapan DTO (Data Transfer Object) untuk memisahkan logika bisnis, serta implementasi Eloquent ORM untuk membangun aplikasi Todo List sederhana dengan MySQL.

1. Dasar Teori

1.1 Pengertian Model

Model merupakan bagian dari arsitektur MVC (Model-View-Controller) yang bertugas mengelola data dan berinteraksi langsung dengan database. Model juga dapat digunakan untuk mengatur logika bisnis yang berkaitan dengan data.

1.2 Laravel Eloquent ORM

Eloquent ORM adalah fitur bawaan Laravel yang menyediakan cara berinteraksi dengan database menggunakan pendekatan Object-Oriented. Setiap tabel database direpresentasikan oleh satu model, dan setiap record di tabel tersebut direpresentasikan sebagai objek.

1.3 DTO (Data Transfer Object)

DTO adalah objek sederhana yang digunakan untuk membawa data antara lapisan aplikasi tanpa mengandung logika bisnis. Penggunaan DTO membantu memisahkan data mentah dari proses bisnis, membuat kode menjadi lebih terstruktur dan mudah diuji.

1.4 Seeder dan Migration

Migration digunakan untuk membangun struktur tabel database secara otomatis melalui kode PHP, sedangkan Seeder digunakan untuk mengisi data awal (dummy data) ke dalam tabel database untuk keperluan pengujian.

2. Langkah-Langkah Praktikum

2.1 Praktikum 1 – Menggunakan Model untuk Binding Form dan Display

- Buat proyek laravel dengan perintah Laravel new model-app, kemudian masuk ke proyek dengan perintah cd model-app.
- Membuat Model Data Sederhana (POCO) Buat folder ViewModels di dalam direktori app untuk menyimpan kelas model kita: mkdir app/ViewModels.
- Buat ProductViewModel.php di dalam direktori app/ViewModels. Berikut kode untuk kelas model:

```php
<?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;
    }

    Tabnine | Edit | Test | Explain | Document
    public static function fromRequest(array $data): self
    {
        return new self(
            $data['name'] ?? '',
            (float)($data['price'] ?? 0),
            $data['description'] ?? ''
        );
    }
}
```

- Buat Controller Buat controller dengan perintah berikut: php artisan make:controller ProductController

- Edit file : app/Http/Controllers/ProductController.php :

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\ViewModels\ProductViewModel;

class ProductController extends Controller
{
    Tabnine | Edit | Test | Explain | Document
    public function create()
    {
        return view('product.create');
    }

    Tabnine | Edit | Test | Explain | Document
    public function result(Request $request)
    {
        $product = ProductViewModel::fromRequest($request->all());
        return view('product.result', compact('product'));
    }
}
```

- Definisikan Rute edit routes/web.php :

```php
<?php

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

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

Tabnine | Edit | Test | Explain | Document
Route::get('/', function () {
    return view('welcome');
});
```

- Buat tampilan views mkdir resources/views/product
- Kemudian buat dua file : create.blade.php dan result.blade.php

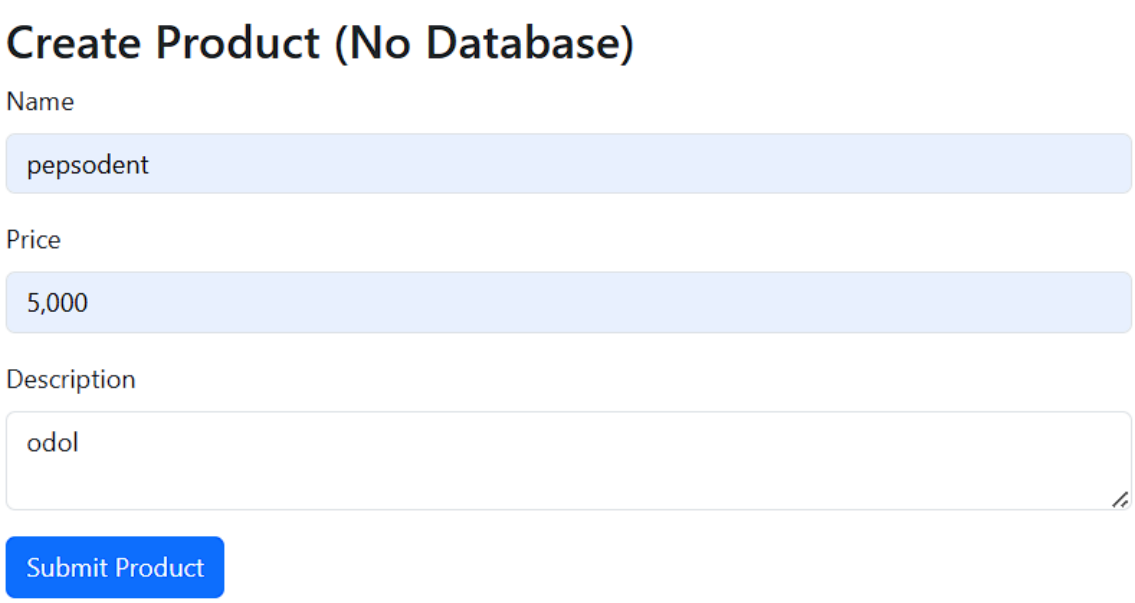
- create.blade.php

```blade
<!DOCTYPE html>
<html>
<head>
    <title>Create Product</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">
    <h2>Create Product (No Database)</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" 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

```blade
<!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">
    <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>
```

- Jalankan aplikasi dan tunjukkan hasil di browser. http://127.0.0.1:8000/product/create



2.2 Praktikum 2 – Menggunakan DTO (Data Transfer Object)

- Buat proyek laravel dengan perintah Laravel new dto-app, kemudian masuk ke proyek dengan perintah cd dto-app.
- Buka kelas DTO. Buat folder DTO di dalam app mkdir app/DTO
- Buat file app/DTO/ProductDTO isi dengan:

```php
<?php

namespace App\DTO;

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

    public function __construct(string $name, float $price, string $description){
        $this->name = $name;
        $this->price = $price;
        $this->description = $description;
    }
    Tabnine | Edit | Test | Explain | Document
    public static function fromRequest(array $data): self{
        return new self(
            $data['name'] ?? '',
            (float)($data['price'] ?? 0),
            $data['description'] ?? ''
        );
    }
}
```

- Buat service layer untuk menangani logika bisnis. DTO sebagai input di kembalikan data yang telah di format buat folder services mkdir app/Services

- Buat file app/Services/ProductServices.php isi dengan

```php
<?php

namespace App\Services;

use App\DTO\ProductDTO;

class ProductService
{
    Tabnine | Edit | Test | Explain | Document
    public function display(ProductDTO $product): array
    {
        return [
            'name' => $product->name,
            'price' => $product->price,
            'description' => $product->description,
        ];
    }
}
```

ini untuk memproses data dari DTO mengembalikannya ke format terstruktur

- Buat controller dengan perintah = php artisan make:controller ProductController

- Edit app/Http/Controllers/ProductController.php

```php
<?php

namespace App\Http\Controllers;

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

class ProductController extends Controller
{
    Tabnine | Edit | Test | Explain | Document
    public function create()
    {
        return view('product.create');
    }
    Tabnine | Edit | Test | Explain | Document
    public function result(Request $request)
    {
        $dto = ProductDTO::fromRequest($request->all());
        $service = new ProductService();
        $product = $service->display($dto);

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

- Definisikan Rute. Rute pertama tampilkan formulir, rute kedua menangani pengiriman dan hasil edit routes/web.php

```php
<?php

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

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

Tabnine | Edit | Test | Explain | Document
Route::get('/', function () {
    return view('welcome');
});
```

- Buat tampilan (Views) dengan Bootstrap buat product di dalam resources/views mkdir resources/views/product

- Setelah membuat direktori, buat dua file: create.blade.php dan result.blade.php.

Berikut adalah konten dari masing-masing file:

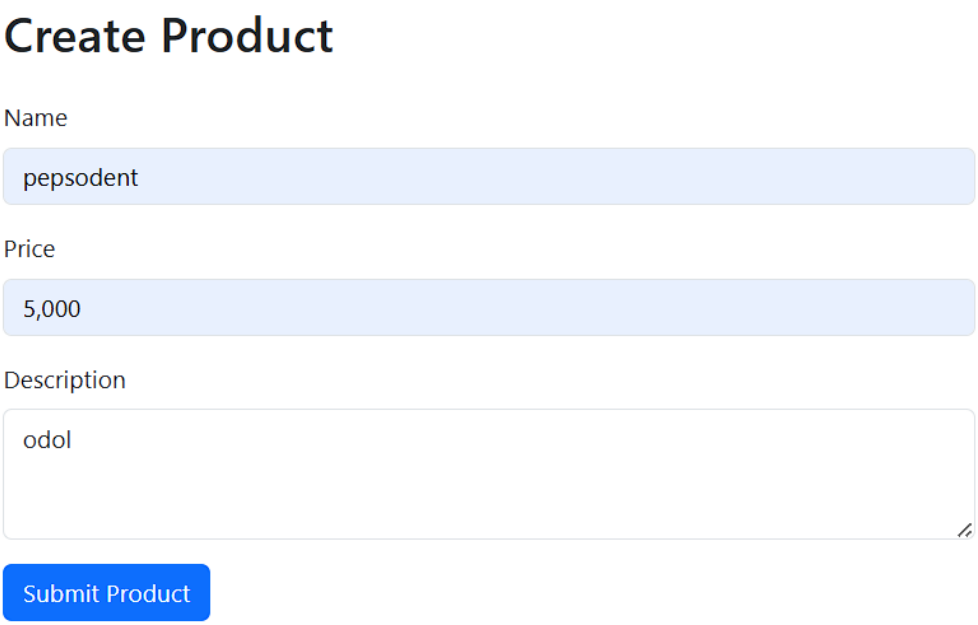
Tampilan 1: resources/views/product/create.blade.php:

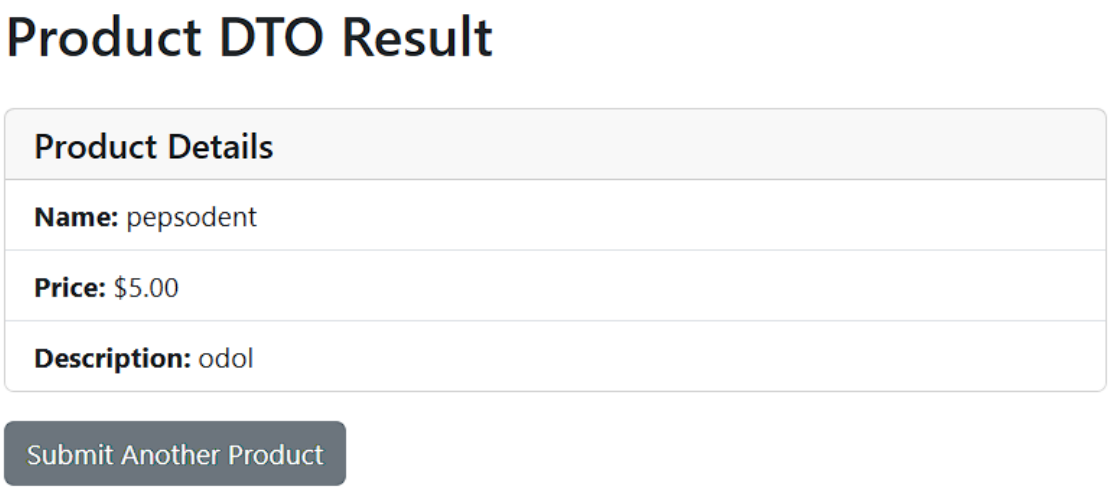
```blade.php
<!DOCTYPE html>
<html>
<head>
    <title>Create Product DTO</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <h2 class="mb-4">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" 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>
    </div>
</body>
</html>
```

Tampilan 2: resources/views/product/result.blade.php:

```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>
                    <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>
            </div>
            <a href="{{ route('product.create') }}" class="btn btn-secondary mt-3">Submit Another Product</a>
        </div>
    </div>
</body>
</html>
```

- Jalankan aplikasi http://127.0.0.1:8000/product/create





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

- Buat proyek laravel baru dengan perintah di terminal: bash laravel new todo-app-mysql masuk ke proyek dengan = cd todo-app-mysql
- Buat database di MySQL phpmyadmin dengan perintah CREATE DATABASE tododb; atau tododb
- Instal dependency MySQL dengan gitbash composer require doctrine/dbal
- Edit file .env edit di bagian DB_DATABASE : sesuaikan dengan nama database yang sudah kita create di phpmyadmin

- Bersihkkan config cache dengan gitbash php artisan config:clear
- Buat Migration untuk Tabel todos php artisan make:migration create_todos_table
- Buka file yang dihasilkan di database/migrations/ YYYY_MM_DD_create_todos_table.php dan perbarui:

```php
<?php

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

return new class extends Migration
{
    Tabnine | Edit | Test | Explain | Document
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('todos', function (Blueprint $table) {
            $table->id();
            $table->string('task');
            $table->boolean('completed')->default(false);
            $table->timestamps();
        });
    }
    Tabnine | Edit | Test | Explain | Document
    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('todos');
    }
};
```

- Jalankan migrasi php artisan migrate
- Buat Seeder untuk Data Dummy php artisan make:seeder TodoSeeder Ini akan membuat file seeder baru di direktori database/seeders. Buka file yang dihasilkan di database/seeders/TodoSeeder.php dan

perbarui:

```php
<?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
{
    Tabnine | Edit | Test | Explain | Document
    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()
            ],
            [
                'task' => 'Selesaikan proyek Laravel',
                'completed' => true,
                'created_at' => Carbon::now(),
                'updated_at' => Carbon::now()
            ],
        ]);
    }
}
```

- Jalankan seeder untuk mengisi database: php artisan db:seed --class=TodoSeeder
- Buat Model Todo Kita akan membuat model untuk tabel todos untuk berinteraksi dengannya menggunakan Eloquent ORM. Jalankan: php artisan make:model Todo
- Buka file yang dihasilkan di app/Models/Todo.php dan perbarui:

```php
<?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'];
}
```

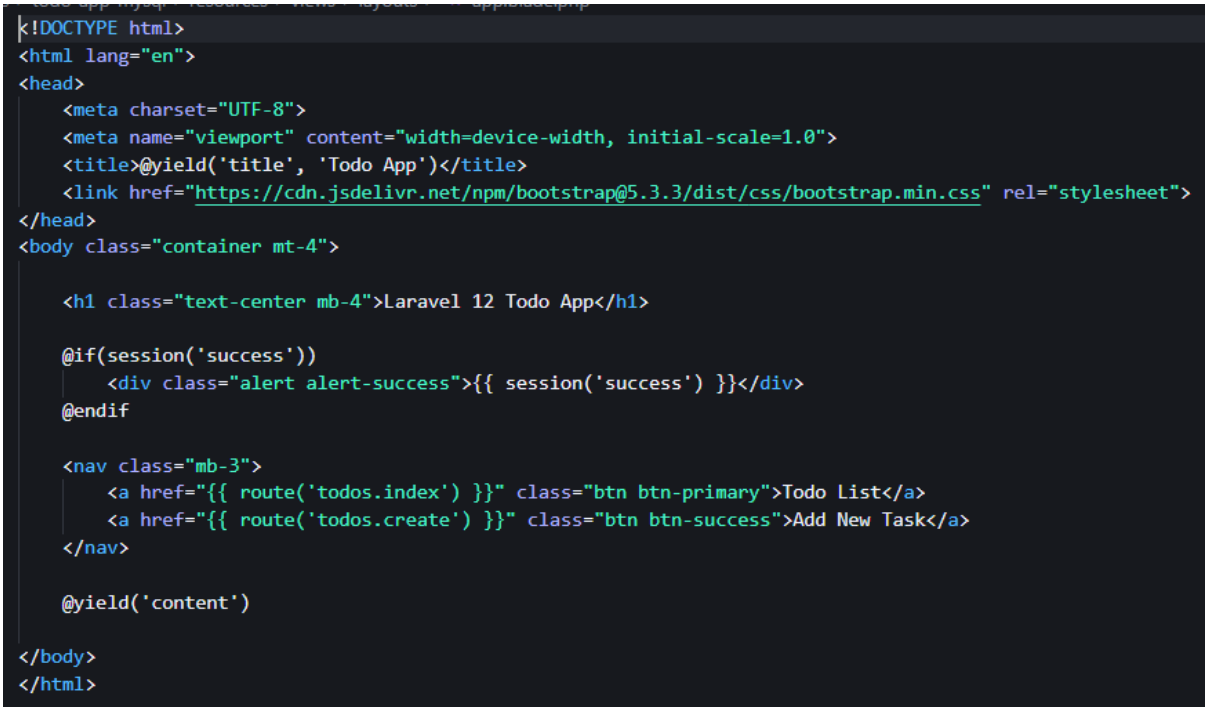
- Buat TodoController untuk Operasi CRUD : php artisan make:controller TodoController
- Perintah diatas akan membuat file controller baru di direktori app/Http/Controllers. Buka file yang dihasilkan di app/Http/Controllers/TodoController.php dan perbarui:

```php
<?php

namespace App\Http\Controllers;

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

class TodoController extends Controller
{
    Tabnine | Edit | Test | Explain | Document
    public function index()
    {
        $todos = Todo::all();
        return view('todos.index', compact('todos'));
    }

    Tabnine | Edit | Test | Explain | Document
    public function create()
    {
        return view('todos.create');
    }

    Tabnine | Edit | Test | Explain | Document
    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!');
    }

    Tabnine | Edit | Test | Explain | Document
    public function show(Todo $todo)
    {
        return view('todos.show', compact('todo'));
    }

    Tabnine | Edit | Test | Explain | Document
    public function edit(Todo $todo)
    {
        return view('todos.edit', compact('todo'));
    }

    Tabnine | Edit | Test | Explain | Document
    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!');
    }

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

- Edit routes/web.php:

```php
<?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');
```

- Buat Tampilan Blade dengan Bootstrap
  - Tampilan Layout Buat folder layouts di resources/views dan buat file baru resources/views/layouts/app.blade.php. File ini akan berfungsi sebagai template dasar untuk semua tampilan:



```
<!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>
```

  - Halaman Todo Sekarang kita akan membuat tampilan untuk aplikasi Todo. Buat folder baru todos di resources/views dan buat file-file berikut:
    - index.blade.php
    - create.blade.php
    - edit.blade.php
    - show.blade.php
    - Edit file resources/views/todos/index.blade.php. Tampilan ini akan menampilkan daftar todo termasuk tautan untuk membuat, mengedit, memperbarui, dan menghapus todo. Tambahkan kode berikut:

```blade
@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
```

Tampilan ini menampilkan formulir untuk menambah todo baru.

- Edit file resources/views/todos/create.blade.php dan tambahkan kode berikut:

```blade
@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
```

Tampilan ini menampilkan formulir untuk mengedit todo yang sudah ada.

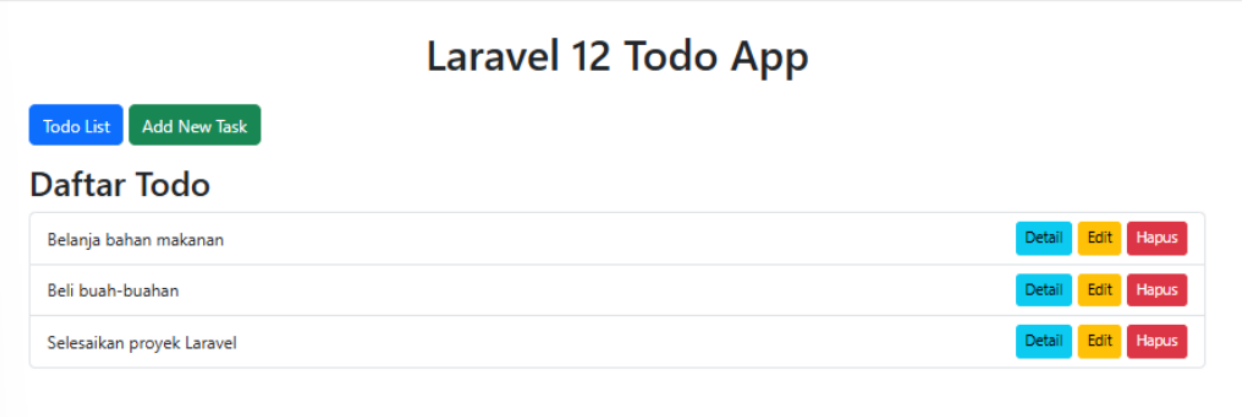
- Edit file resources/views/todos/edit.blade.php dan tambahkan kode berikut:

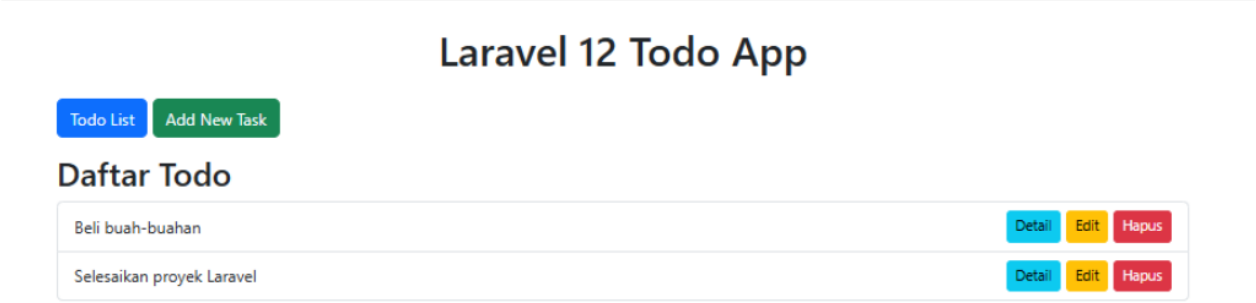
```blade
@extends('layouts.app')
@section('title', 'Edit Task')

@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
```

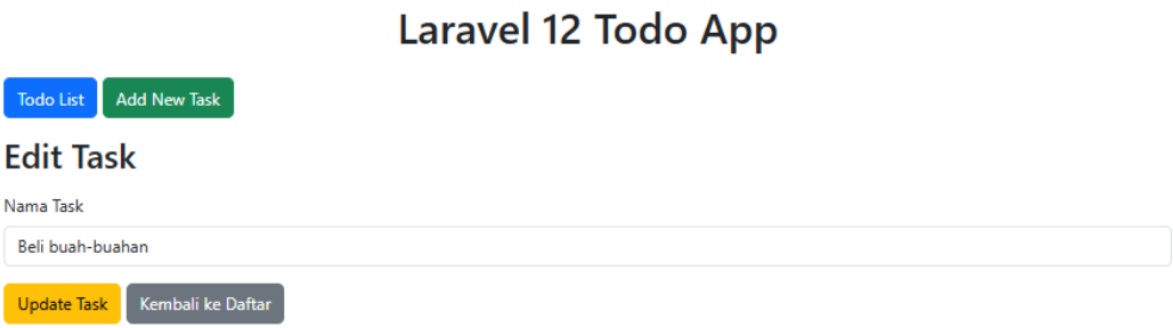
- Terakhir, edit file resources/views/todos/show.blade.php dan tambahkan kode berikut:

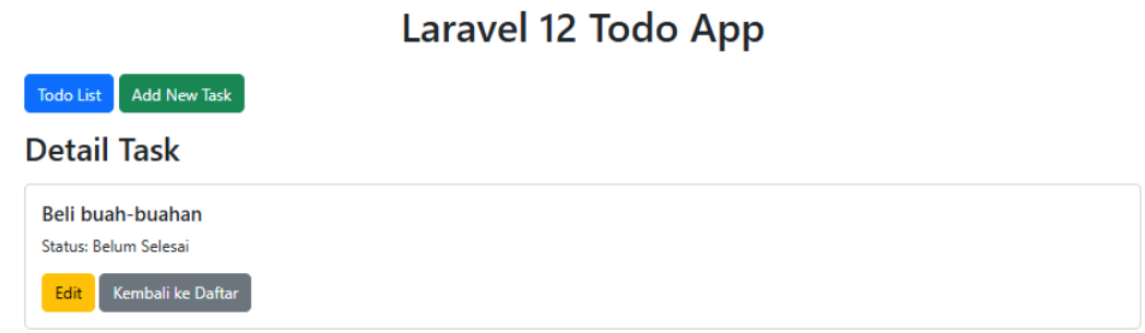
```
@extends('layouts.app')
@section('title', 'Detail Task')
@section('content')
    <h2>Detail Task</h2>

    <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
```

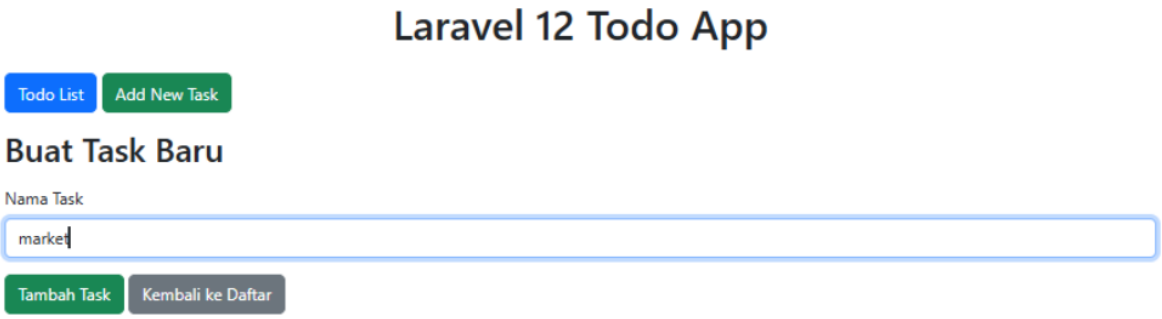
- Jalankan aplikasi http://127.0.0.1:8000











---

3. Hasil dan Pembahasan

Dari hasil praktikum yang dilakukan, diperoleh hasil sebagai berikut:

- Praktikum 1 – Model dan Binding Form: Penggunaan model sederhana (POCO) memungkinkan data dari form dapat ditampilkan kembali dengan mudah di view. Proses binding data menjadi lebih efisien tanpa harus melakukan manipulasi array secara manual.
- Praktikum 2 – DTO dan Service Layer: Dengan menggunakan DTO, alur data antara view, controller, dan logic menjadi lebih rapi. DTO membantu memisahkan data dari logika bisnis yang ditangani oleh Service Layer, sehingga arsitektur aplikasi lebih modular dan mudah dikelola.
- Praktikum 3 – Aplikasi Todo dengan Eloquent dan MySQL: Implementasi Eloquent ORM pada aplikasi Todo List membuat proses CRUD (Create, Read, Update, Delete) lebih sederhana. Developer tidak perlu

menulis query SQL secara langsung, karena Eloquent sudah menyediakan fungsi-fungsi seperti all(), find(), create(), dan update() yang intuitif.

- Selain itu, dengan dukungan migration dan seeder, struktur database dapat dikelola secara konsisten antar lingkungan pengembangan. Tampilan aplikasi juga telah dibuat dengan Blade Template dan Bootstrap, menghasilkan antarmuka yang responsif dan mudah digunakan.

---

4. Kesimpulan

Dari praktikum ini dapat disimpulkan bahwa:

- Laravel menyediakan Eloquent ORM sebagai solusi untuk mengelola data secara efisien dan berorientasi objek.
- Penggunaan Model, DTO, dan Service Layer membantu meningkatkan keteraturan arsitektur aplikasi.
- Melalui Migration dan Seeder, pengaturan database menjadi lebih mudah, konsisten, dan otomatis.
- Penerapan konsep ini memudahkan proses pengembangan aplikasi berbasis data seperti Todo List yang memanfaatkan CRUD secara penuh.

---

5. Referensi

- https://hackmd.io/@mohdrzu/ryIIM1a0ll#Praktikum-2---Menggunakan-DTO-Data-Transfer-Object

---