Laporan Modul 6: Model dan Laravel Eloquent

Mata Kuliah: Workshop Web Lanjut
Nama: Adha Gusti Harmadhan
NIM: 2024573010009
Kelas: 2B TI

Abstrak

Laporan ini menjelaskan hasil praktikum pada Modul 6: *Model dan Laravel Eloquent* dalam mata kuliah Workshop Web Lanjut. Fokus praktikum adalah memahami bagaimana Laravel memetakan data ke dalam model, bagaimana Eloquent ORM bekerja untuk operasi CRUD, serta bagaimana pola-pola pendukung seperti DTO dan Repository dapat dipakai untuk merapikan kode. Praktikum dibagi menjadi tiga: (1) binding form ke model sederhana tanpa database, (2) penggunaan Data Transfer Object (DTO) dan service layer, dan (3) membangun aplikasi Todo CRUD menggunakan Eloquent dan MySQL. Melalui percobaan ini mahasiswa diharapkan mampu menghubungkan teori MVC dengan implementasi Laravel yang riil, khususnya pada sisi *Model* dan interaksinya dengan database.

1. Dasar Teori

1.1 Model dalam Laravel

Dalam arsitektur MVC, **Model** adalah bagian yang mewakili data dan aturan bisnis. Di Laravel, model biasanya berada di folder `app/Models` dan secara default akan terhubung ke tabel yang namanya jamak dari nama model — misalnya model `Product` ke tabel `products`. Model inilah yang berkomunikasi dengan database menggunakan **Eloquent ORM** sehingga kita tidak perlu menulis query SQL mentah setiap kali ingin mengambil atau menyimpan data.

1.2 Eloquent ORM

Eloquent adalah ORM bawaan Laravel yang menyediakan cara berinteraksi dengan database secara *object-oriented*. Setiap baris pada tabel direpresentasikan sebagai objek model. Operasi umum seperti `all()`, `find()`, `create()`, `update()`, dan `delete()` sudah disediakan sehingga kode jadi lebih singkat dan mudah dibaca. Contoh model sederhana:

```php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = ['name', 'price', 'stock'];
}
```

Properti `$fillable` digunakan agar field tersebut boleh di-*mass assign* saat pemanggilan `Product::create([...])`.

1.3 POCO / ViewModel

Sebelum masuk ke database, kadang kita hanya butuh "wadah data" dari form. Pada praktikum pertama, kita memakai kelas PHP biasa (gaya **POCO / ViewModel**) untuk menampung data produk tanpa menyimpannya ke database. Ini berguna kalau kita ingin latihan alur request → controller → view tapi database belum disiapkan.

1.4 Data Transfer Object (DTO)

DTO dipakai untuk memindahkan data dari lapisan request ke lapisan service atau ke controller lain dalam bentuk yang sudah rapi. Keuntungan memakai DTO:

- data yang masuk lebih terkontrol,
- memisahkan data mentah dari logika bisnis,
- kode controller jadi lebih pendek.

1.5 Repository Pattern (sekilas)

Repository dipakai untuk mengabstraksi akses data. Di Laravel, ini cocok kalau nanti aplikasi mulai besar dan kita ingin ganti sumber data (MySQL → API, dsb.) tanpa mengubah controller.

1.6 Migrasi, Seeder, dan Eloquent

Untuk praktikum Todo, kita sudah mulai pakai fitur database Laravel:

- **Migration** → membuat struktur tabel dengan kode
- **Seeder** → mengisi data awal
- **Model Eloquent** → mengakses tabel memakai class
- **Controller** → memanggil model dan melempar ke view

Dengan cara ini, alur *Model layer* Laravel bisa dilihat utuh dari atas ke bawah.

---

2. Langkah-Langkah Praktikum

2.1 Praktikum 1 — Binding Form ke Model Sederhana (tanpa DB)

Tujuan praktik (implisit): latihan mengikat data form ke objek model sederhana agar alur request → controller → view dipahami dulu sebelum pakai Eloquent.

Langkah-langkah:

1. Buat proyek baru:

2. Buat ViewModel: `app/ViewModels/ProductViewModel.php` (kode di laporan utama).

projects > model-app > app > ViewModels > ProductViewModel.php

```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;
    }

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

3. Buat `ProductController`

```
adhag@Adhagusti MINGW64 /c/web_lanjut/web-lanjut-2024573010009/projects/productpagination (main)
$ php artisan make:controller ProductController

  INFO  Controller [C:\web_lanjut\web-lanjut-2024573010009\projects\productpagination\app\Http\Controllers
\ProductController.php] created successfully.
```

dan tambah method `create` & `result`.

projects > model-app > app > Http > Controllers > ProductController.php

```php
<?php

namespace App\Http\Controllers;

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

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

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

4. Daftarkan route di `routes/web.php` untuk form & result.

projects > model-app > routes > web.php

```php
<?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. Buat view form `resources/views/product/create.blade.php`.

projects > model-app > resources > views > product > 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>
```

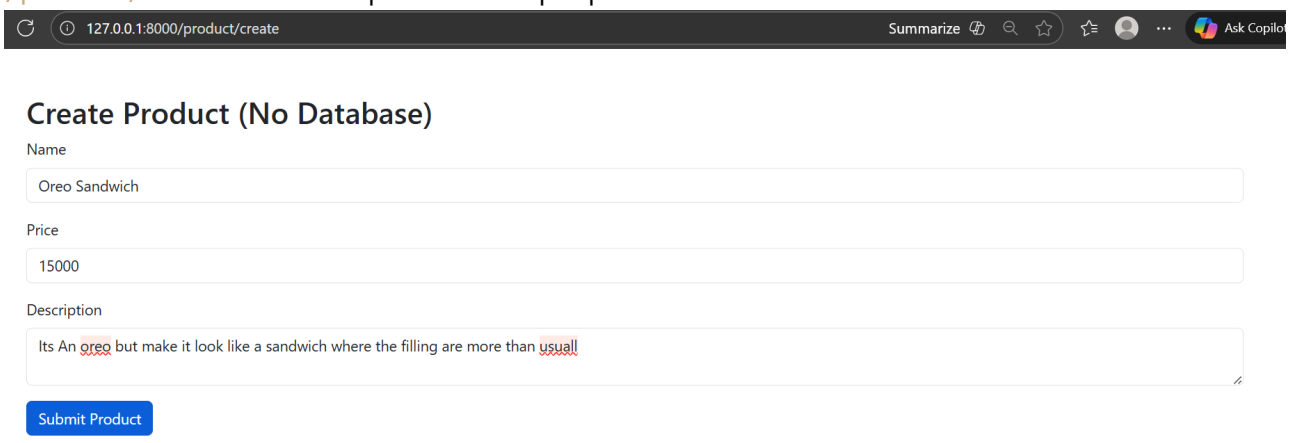
6. Buat view hasil `resources/views/product/result.blade.php`.

projects > model-app > resources > views > product > result.blade.php

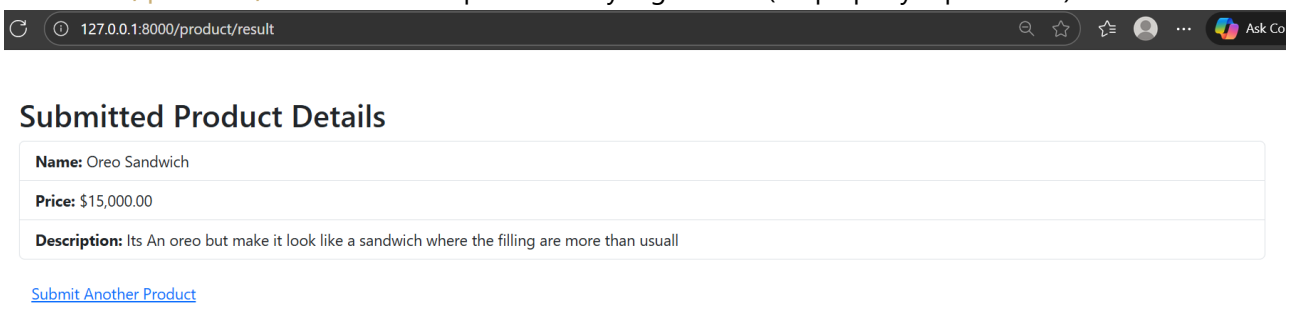
```blade
<!DOCTYPE html>
<html>
<head>
    <title>Product Result</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container py-5">
    <h2>Submitted Product Details</h2>
    <ul class="list-group">
        <li class="list-group-item">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

Hasil Pengujian:

- `/product/create` → menampilkan form input produk.



- Submit → `/product/result` menampilkan data yang dikirim (tanpa penyimpanan DB).

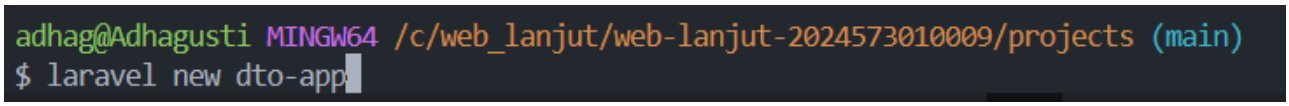


2.2 Praktikum 2 — Menggunakan DTO dan Service

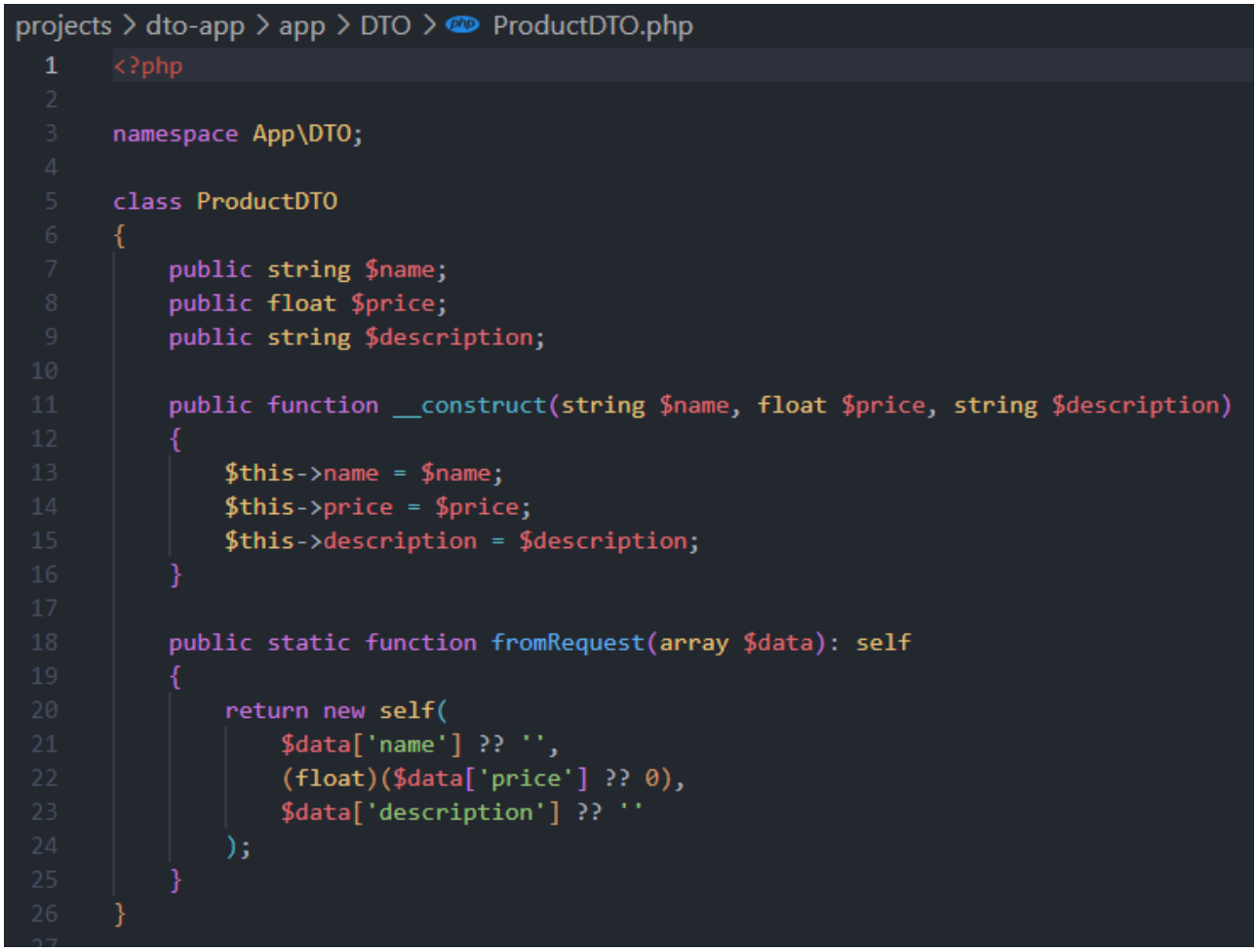
Pada praktik kedua, alurnya mirip praktikum 1, tetapi datanya tidak langsung dipakai oleh view. Data yang dikirim form dimasukkan dulu ke **DTO**, lalu diproses oleh **service** supaya controller tetap tipis.

Langkah-langkah:

1. Buat proyek baru :



2. Buat DTO: `app/DTO/ProductDTO.php`.



3. Buat service: `app/Services/ProductService.php`.

projects > dto-app > app > Services > ProductService.php

```php
<?php

namespace App\Services;

use App\DTO\ProductDTO;

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

4. Buat `ProductController` Update controller untuk menggunakan DTO dan Service.

projects > dto-app > 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
{
    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'));
    }
}
```

5. Route dan view dapat tetap seperti praktikum 1, hanya hasil yang diolah oleh service.

projects > dto-app > routes > web.php

```php
<?php

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

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

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

projects > dto-app > resources > views > product > create.blade.php

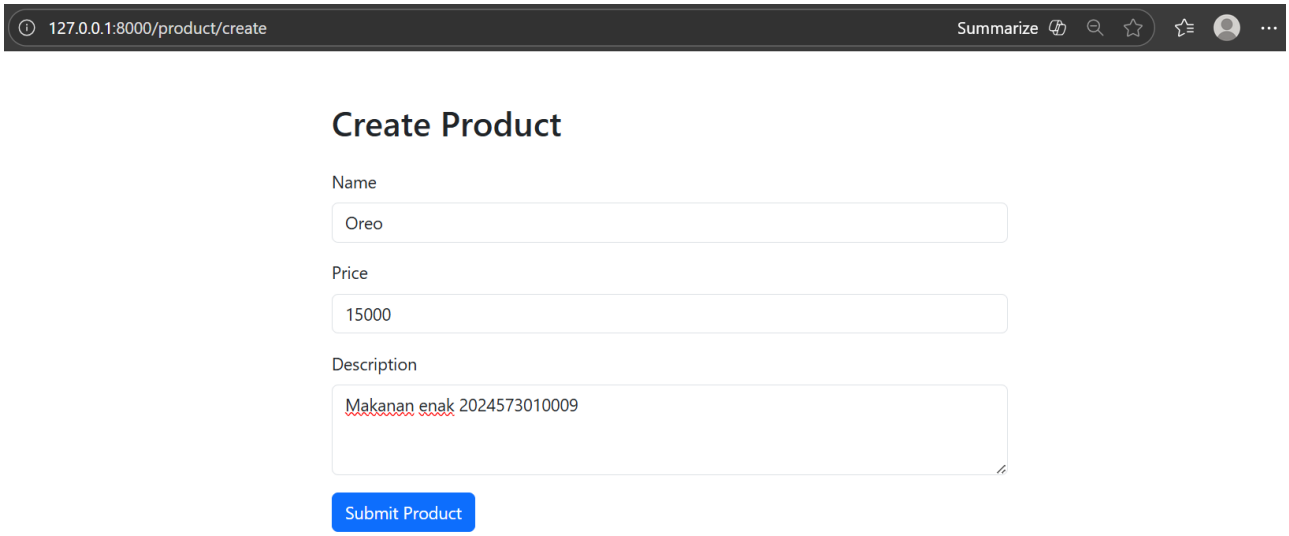
```blade.php
<!DOCTYPE html>
<html>
<head>
    <title>Create Product DTO</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-6">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

projects > dto-app > resources > views > product > result.blade.php

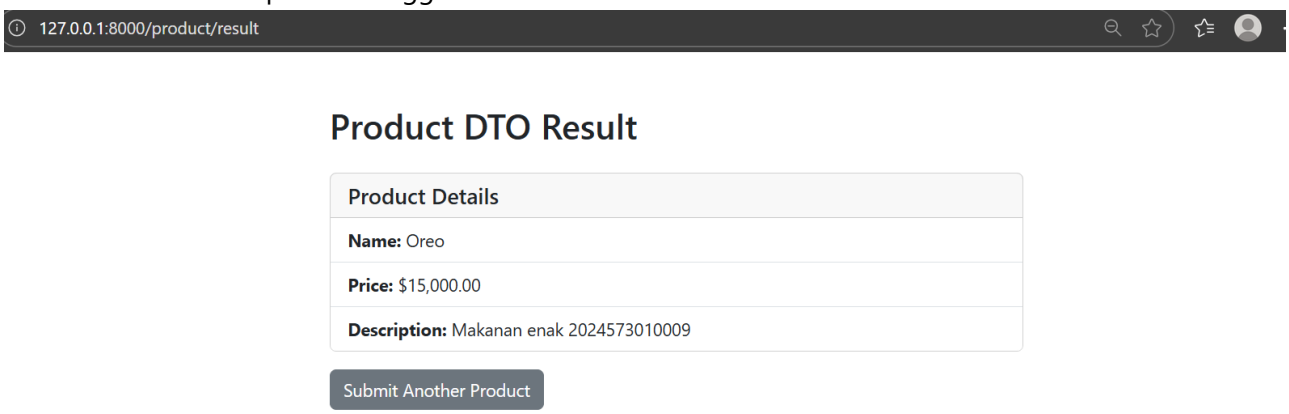
```blade.php
<!DOCTYPE html>
<html>
<head>
    <title>Product Result</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
  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

Hasil Pengujian:

- `/product/create` → form sama.



- Submit → hasil ditampilkan menggunakan data dari `ProductService`.



2.3 Praktikum 3 — Todo CRUD dengan Eloquent dan MySQL

Bagian ini sudah mulai memakai **model Eloquent** plus **migration + seeder**.

Langkah-langkah utama:

1. Buat project baru :

kemudian Install dependency MySQL:

```
composer require doctrine/dbal
```

Tak lupa juga membuat database tododb

2. Atur database di `.env`.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tododb
DB_USERNAME=root
DB_PASSWORD=
```

3. Buat migration `create_todos_table`

```
php artisan make:migration create_todos_table
```

Lalu isi file yang dihasilkan di database/migrations/YYYY_MM_DD_create_todos_table.php dan perbarui

projects > todo-app-mysql > database > migrations > 2025_10_29_023036_create_todos_table.php

```php
<?php

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

return new class extends Migration
{
    public function up()
    {
        Schema::create('todos', function (Blueprint $table) {
            $table->id();
            $table->string('task');
            $table->boolean('completed')->default(false);
            $table->timestamps();
        });
    }

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

lalu jalankan

```
php artisan migrate
```

4. Buat seeder `TodoSeeder` Jalankan perintah ini untuk membuat seeder:

```
php artisan make:seeder TodoSeeder
```

Buka file yang dihasilkan di database/seeders/TodoSeeder.php dan perbarui:

projects > todo-app-mysql > database > seeders > TodoSeeder.php

```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
{
    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()
            ],
        ]);
    }
}
```

Lalu jalankan seeder

```
php artisan db:seed --class=TodoSeeder
```

5. Buat model `Todo`

```
php artisan make:model Todo
```

Buka file yang dihasilkan di app/Models/Todo.php dan perbarui:

projects > todo-app-mysql > app > Models > Todo.php

```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'];
}
```

6. Buat `TodoController`

projects > todo-app-mysql > app > Http > Controllers > TodoController.php

```php
<?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!');
    }
}
```

Dan tambhakan controller tersebut ke route

projects > todo-app-mysql > 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');
```

7. Buat Layout lewat file `app.blade.php` di folder layout

projects > todo-app-mysql > resources > views > layouts > app.blade.php

```blade
<!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>
```

Kemudian buat view di `resources/views/todos/*` (index, create, edit, show).

index

projects > todo-app-mysql > resources > views > todos > create.blade.php

```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
```

create

projects > todo-app-mysql > resources > views > todos > edit.blade.php

```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
```

edit

projects > todo-app-mysql > resources > views > todos > index.blade.php

```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
```

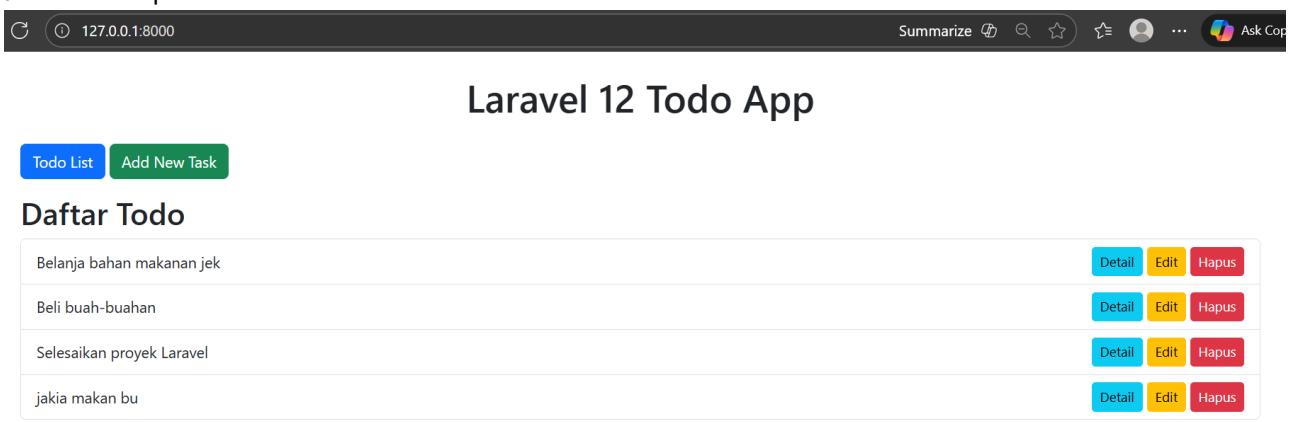
show

projects > todo-app-mysql > resources > views > todos > show.blade.php

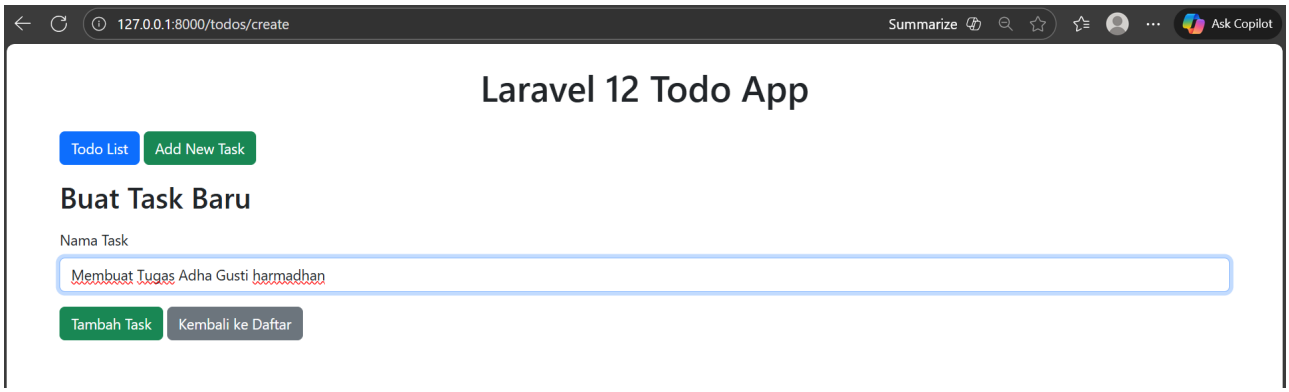
```blade
@extends('layouts.app')
@section('title', 'Detail Task')
@section('content')
    <h2>Detail Task</h2>

    <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
```

Hasil Pengujian:

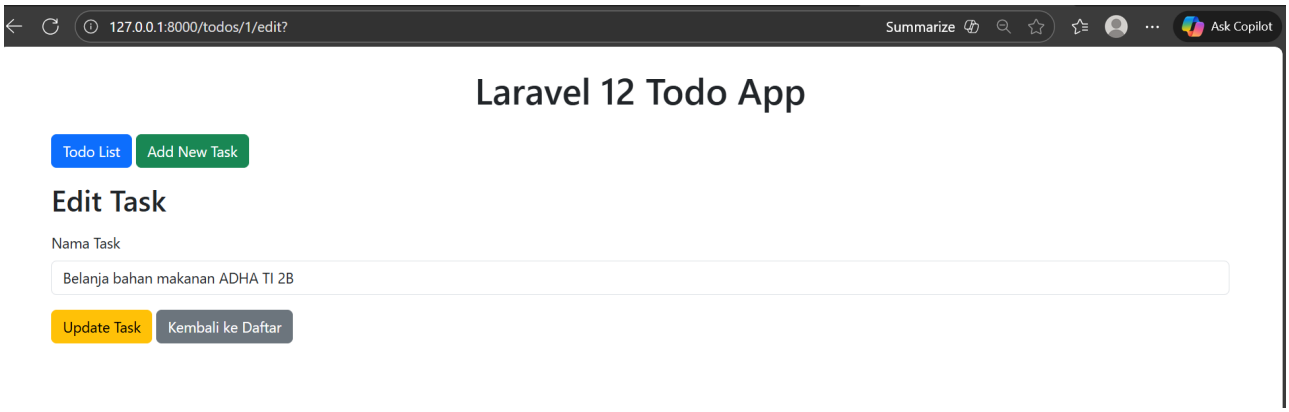
- `/` → menampilkan daftar todo dari database.



- `/todos/create` → form tambah task, submit menyimpan ke DB.



- Edit / Delete bekerja sesuai ekspektasi.



3. Hasil dan Pembahasan

1. **Praktikum 1 (tanpa database)** memperkenalkan konsep model sebagai *data holder* sederhana. `ProductViewModel` membantu memahami bahwa model tidak selalu harus terhubung ke tabel database — fungsinya bisa sekadar menampung data yang dikirim dari form.

2. **Praktikum 2 (DTO dan Service)** memperlihatkan penerapan pola *clean architecture* di Laravel. Dengan memisahkan data ke `ProductDTO` dan logika ke `ProductService`, controller jadi lebih ringkas, mudah diuji, dan mudah dikembangkan jika nanti logikanya bertambah.

3. **Praktikum 3 (CRUD Eloquent dan MySQL)** adalah penerapan konsep *Model* yang sesungguhnya di Laravel. Semua operasi database seperti tambah, ubah, hapus, dan tampil dilakukan lewat Eloquent ORM, tanpa perlu menulis query SQL secara manual.

4. Secara keseluruhan, dari ketiga praktikum ini mahasiswa dapat melihat transisi yang jelas dari model sederhana tanpa database, ke model dengan arsitektur DTO–Service, hingga ke model penuh dengan Eloquent ORM yang terhubung langsung ke database MySQL. Hal ini memperkuat pemahaman konsep MVC Laravel dari sisi Model secara menyeluruh.

---

4. Kesimpulan

Dari praktikum Modul 6 ini dapat disimpulkan bahwa:

1. **Model di Laravel tidak selalu berarti Eloquent** — kita bisa mulai dari kelas PHP biasa (ViewModel/POCO) untuk menampung data form.
2. **DTO membantu merapikan alur data** dari request ke lapisan lain sehingga controller lebih bersih dan mudah diuji.
3. **Eloquent ORM mempermudah CRUD** karena kita cukup memanggil method bawaan tanpa menulis SQL manual.
4. **Migration dan seeder** membuat struktur database dan data awal bisa dikelola lewat kode dan mudah diulang di lingkungan lain.
5. Dengan menggabungkan controller, route, view, dan model, mahasiswa jadi melihat alur penuh MVC Laravel dari form sampai tersimpan ke database.

---

5. Referensi

- Modul 6 - Model dan Laravel Eloquent — (https://hackmd.io/@mohdrzu/ryIIM1a0ll)
- Dokumentasi Resmi Laravel 12 — (https://laravel.com/docs/12.x/eloquent)