

# Laporan Modul 6: Model dan Laravel Eloquent

---

**Mata Kuliah:** Workshop Web Lanjut

**Nama:** Fathan Mubina

**NIM:** 2024573010058

**Kelas:** TI-2C

---

## Abstrak

Modul ini menjelaskan pengertian model dalam framework Laravel serta praktek Eloquent ORM sebagai penghubung aplikasi dan database. Tujuan praktikum ini yaitu mahasiswa memahami bagaimana model bekerja, pengertian dan penggunaan Data Transfer Object (DTO), Repository Pattern, serta penerapan migration dan seeder dalam pengembangan aplikasi berbasis data. Mahasiswa telah mendapatkan pemahaman ini melalui tiga praktikum utama, yaitu pembuatan model sederhana untuk form binding, penggunaan DTO pada service layer, dan pengimplementasian aplikasi Todo List yang melakukan operasi CRUD menggunakan Eloquent ORM dan MySQL. Hasil dari praktikum tersebut membuktikan bahwa Laravel menawarkan sistem pengelolaan data yang efektif dan terstruktur serta mempermudah pengembangan aplikasi dari skala kecil hingga besar.

---

## 1. Dasar Teori

1. Models di Laravel Model adalah komponen penting dalam arsitektur MVC (Model-View-Controller) yang berfungsi sebagai penghubung antara aplikasi dan basis data. Dalam Laravel, model mewakili tabel di dalam database dan berisi logika bisnis serta aturan untuk mengelola data. Setiap model biasanya berhubungan dengan satu tabel di database, dan Laravel menyediakan Eloquent ORM untuk mempermudah interaksi dengan tabel tersebut tanpa perlu menulis query SQL secara manual.
2. Entities dan POCO Entity adalah representasi dari objek nyata dalam sistem yang memiliki atribut dan perilaku tertentu. Dalam konteks pemrograman berorientasi objek, Plain Old CLR Object (POCO) atau Plain Old PHP Object (POPO) di Laravel adalah kelas sederhana yang hanya berisi properti dan konstruktor tanpa bergantung pada framework tertentu. Konsep ini memudahkan pembuatan model atau view-model yang ringan, seperti ProductViewModel, untuk memisahkan logika presentasi dari logika bisnis utama aplikasi.
3. Data Transfer Object (DTO) DTO adalah objek yang digunakan untuk memindahkan data antar lapisan aplikasi (misalnya dari controller ke service). Tujuan utama DTO adalah menghindari pengiriman data mentah dari permintaan pengguna (request) secara langsung, serta menjaga struktur data tetap konsisten dan aman. Dengan DTO, proses validasi dan transformasi data menjadi lebih terkontrol dan mudah dikelola, terutama dalam aplikasi berskala besar.
4. Repository Pattern Repository Pattern adalah pola desain yang berfungsi sebagai lapisan abstraksi antara model dan logika bisnis. Repository menangani interaksi langsung dengan database, sementara controller atau service cukup memanggil metode dari repository tersebut. Dengan pendekatan ini, kode menjadi lebih terorganisir, mudah diuji, dan fleksibel terhadap perubahan sumber data (misalnya dari MySQL ke API eksternal).

5. Migration Migration di Laravel adalah fitur yang digunakan untuk mengatur versi skema database menggunakan file PHP. Setiap perubahan struktur tabel, seperti menambah kolom atau membuat tabel baru, dapat dilakukan melalui file migration yang kemudian dijalankan menggunakan perintah `php artisan migrate`. Migration membantu menjaga konsistensi struktur database antar tim dan memudahkan proses rollback bila terjadi kesalahan.
  6. Seeders Seeder digunakan untuk mengisi database dengan data awal atau data dummy yang berguna dalam proses pengujian dan pengembangan aplikasi. Dengan perintah `php artisan db:seed`, Laravel dapat menambahkan data secara otomatis ke dalam tabel sesuai isi file seeder yang telah dibuat. Seeder sangat berguna untuk mensimulasikan data nyata tanpa harus menginputnya secara manual.
  7. Eloquent ORM Eloquent ORM (Object Relational Mapping) adalah fitur bawaan Laravel yang memungkinkan pengembang untuk berinteraksi dengan database menggunakan sintaks berbasis objek. Setiap model mewakili tabel di database dan setiap instansinya mewakili satu baris data. Dengan Eloquent, operasi CRUD (Create, Read, Update, Delete) dapat dilakukan dengan mudah menggunakan metode seperti `create()`, `all()`, `find()`, `update()`, dan `delete()`. Eloquent juga mendukung relasi antar tabel seperti one-to-many dan many-to-
- 

## 2. Langkah-Langkah Praktikum

### Praktikum 1: Menggunakan Model untuk Binding Form dan Display

- Langkah 1: Buat dan Buka Proyek Laravel  
buat proyek Laravel baru:  

```
laravel new model-app
```

```
cd model-app
```

```
code .
```
- Langkah 2: Membuat Model Data Sederhana (POCO)  
Buat folder ViewModels di dalam direktori app dengan ketik di bash:  

```
mkdir app/ViewModels
```

setelah itu, Buat ProductViewModel.php di dalam direktori app/ViewModels.  
setelah itu ketik kode berikut:

```
<?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'] ?? ''
        );
    }
}

```

- Langkah 3: Buat Controller

Buat controller dengan perintah berikut: `php artisan make:controller ProductController` setelah itu ubah isi file `app/Http/Controllers/ProductController.php` dengan kode berikut:

```

<?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'));
    }
}

```

- Langkah 4: Definisikan Rute isi di dalam `routes/web.php` dengan kode berikut:

```

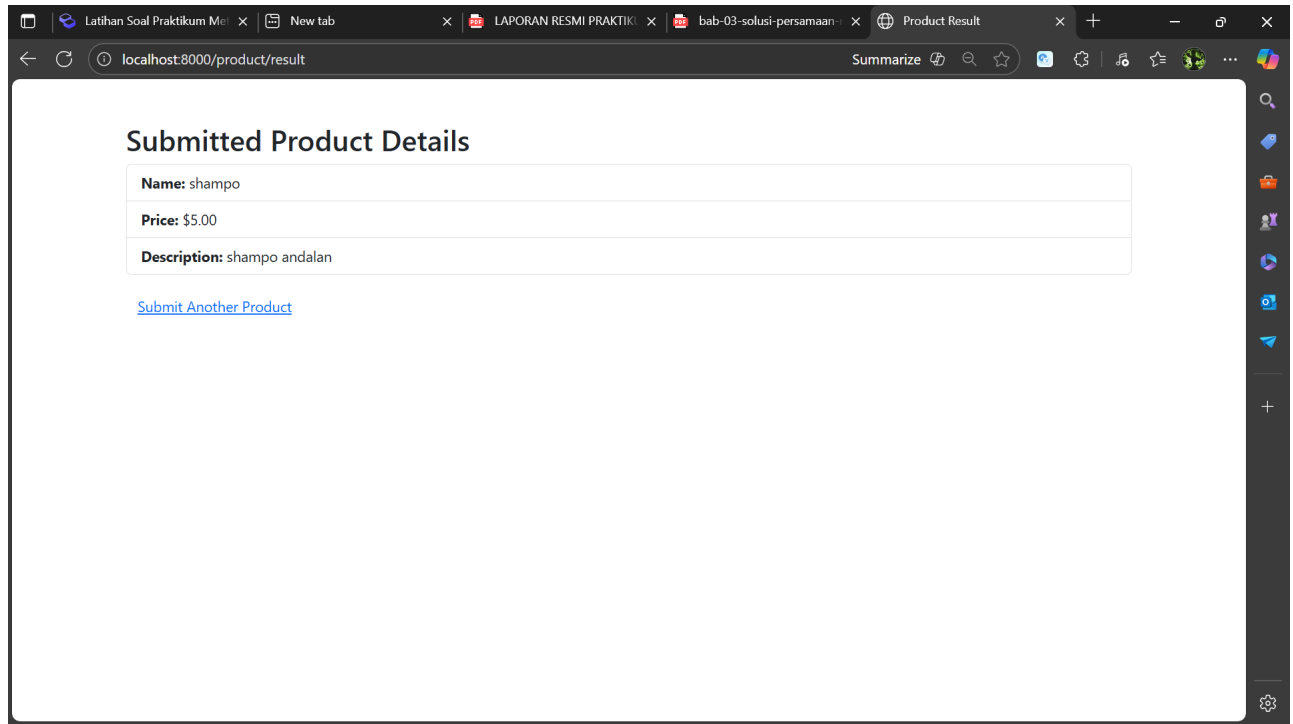
<?php

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');

```

- Langkah 5: Buat Tampilan (Views) dengan Bootstrap  
Buat direktori product di dalam resources/views dengan kode berikut:  
`mkdir resources/views/product`
- Langkah 6: Jalankan php artisan serve di bash, Kemudian Kunjungi:  
`http://localhost:8000/product/create`  
dan test formnya. Hasilnya



## Praktikum 2: Menggunakan DTO (Data Transfer Object)

- Langkah 1: Buat dan Buka Proyek Laravel  
`laravel new dto-app`  
`cd dto-app`  
`code .`
- Langkah 2: Buat Kelas DTO  
Buat folder DTO di dalam app dengan ketik di bash:  
`mkdir app/DTO`  
setelah itu buat file `app/DTO/ProductDTO.php` msukkan kode berikut:

```
<?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;
    }

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

```

- Langkah 3: Buat Service Layer

Buat folder Services di dalam app dengan ketik di bash:

```
mkdir app/Services
```

sekarang, ketik kode berikut di file app/Services/ProductService.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,
        ];
    }
}

```

- Langkah 4: Buat Controller

Buat controller dengan kode berikut: `php artisan make:controller ProductController` masukkan di dalam file app/Http/Controllers/ProductController.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'));
    }
}

```

- Langkah 5: Definisikan Rute  
masukkan kode di dalam routes/web.php  
dengan kode berikut:

```

<?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');

```

- Langkah 6: Buat Tampilan (Views) dengan Bootstrap  
Buat direktori product di dalam resources/views dengan ketik berikut di bash:  
`mkdir resources/views/product`  
buat dua file: create.blade.php dan result.blade.php didalam folder resources/views/product

Setelahnya masukkan kode di dalam masing-masing file:

Tampilan 1: resources/views/product/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.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:

```

<!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>

```

```

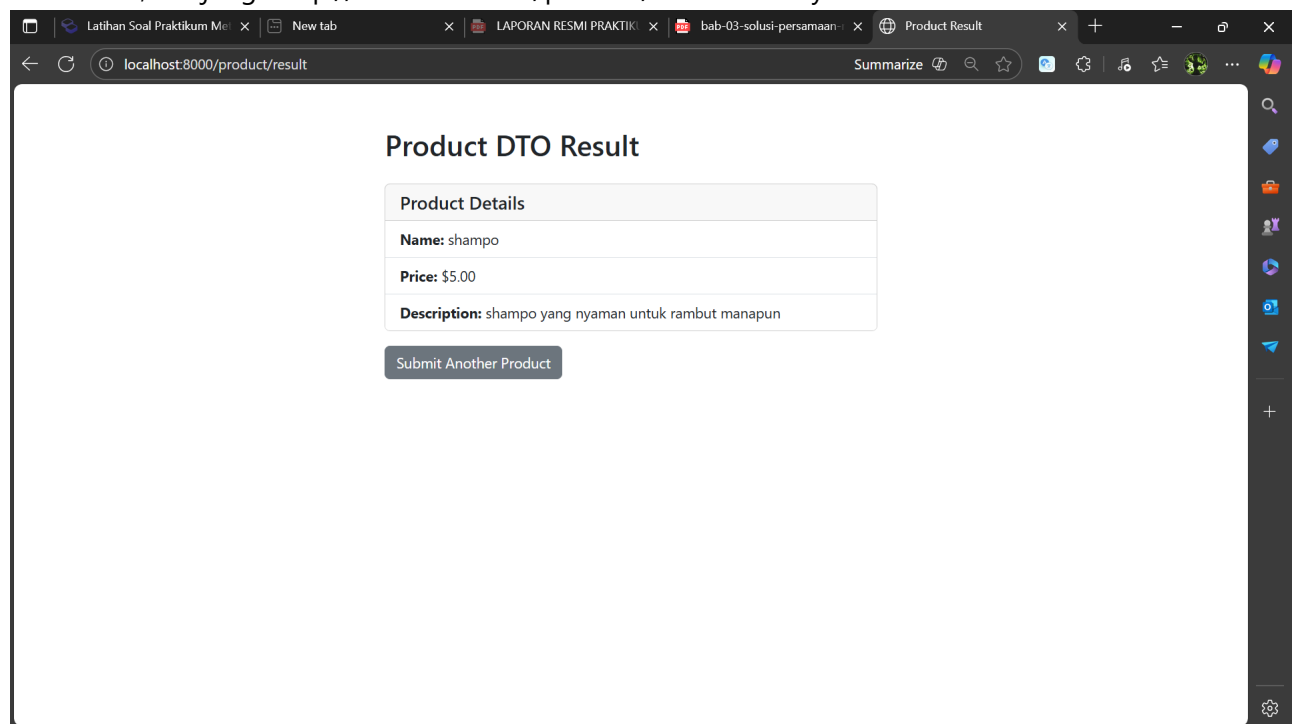
        <li class="list-group-item">
            <strong>Price:</strong> ${{
number_format($product['price'], 2) }}
        </li>
        <li class="list-group-item">
            <strong>Description:</strong> {{ $product['description']
    }}
        </li>
    </ul>
</div>
    <a href="{{ route('product.create')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

```

- Langkah 7: Jalankan dan Uji Aplikasi jalankan aplikasi dengan perintah berikut:

`php artisan serve`

Setelah itu, Kunjungi: <http://localhost:8000/product/create> Hasilnya



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

- Langkah 1: Setup Aplikasi dan Konfigurasi MySQL  
 buat project Laravel baru menggunakan Laravel installer,  
 dengan ketik berikut di bash: `laravel new todo-app-mysql`  
`cd todo-app-mysql`  
`code .`  
 buat database dengan nama berikut: `CREATE DATABASE tododb;`

install dependency MySQL dengan ketik: `composer require doctrine/dbal`

Edit file `.env` seperti berikut:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tododb
DB_USERNAME=<username database anda>
DB_PASSWORD=<password database anda jika ada>
```

Bersihkan config cache: `php artisan config:clear`

- Langkah 2: Buat Migration untuk Tabel todos

Jalankan perintah migrasi dengan ketik:

`php artisan make:migration create_todos_table`

Buka file di `database/migrations/YYYY_MM_DD_create_todos_table.php` dan ketik:

```
<?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');
    }
};
```

Jalankan migrasi: `php artisan migrate`

- Langkah 3: Buat Seeder untuk Data Dummy

Jalankan perintah ini di bash untuk membuat seeder:

`php artisan make:seeder TodoSeeder`

buat file seeder baru di direktori `database/seeder`s.

Buka file yang dihasilkan di `database/seeder/TodoSeeder.php` kemudian ketik kode berikut:

```
<?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()
            ],
        ]);
    }
}
```

Jalankan seeder untuk mengisi database:

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

- Langkah 4: Buat Model Todo

buat model todo dengan ketik di bash: `php artisan make:model Todo` kemudian ketik kode berikut di dalam file `app/Models/Todo.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'];
}
```

- Langkah 5: Buat TodoController untuk Operasi CRUD untuk membuat TodoController, ketik berikut ini di bash:

`php artisan make:controller TodoController` buka file yang dihasilkan di `app/Http/Controllers/TodoController.php` kemuddian ketik kode berikut 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!');
    }
}

```

- Langkah 6: Definisikan Rute Web ketik kode berikut ini di file routes/web.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');

```

beri komentar pada rute default untuk menghindari konflik.

```

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

```

- Langkah 7: Buat Tampilan Blade dengan Bootstrap  
Buat folder layouts di resources/views, setelah folder dibuat, buat file baru resources/views/layouts/app.blade.php.  
kemudian masukkan kode berikut di dalamnya:

```

<!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.cs
s" 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>

```

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 masukkan 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

```

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

```

@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 file `resources/views/todos/edit.blade.php` dan tambahkan kode berikut:

```

@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 file resources/views/todos/show.blade.php dan tambahkan kode berikut:

```

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

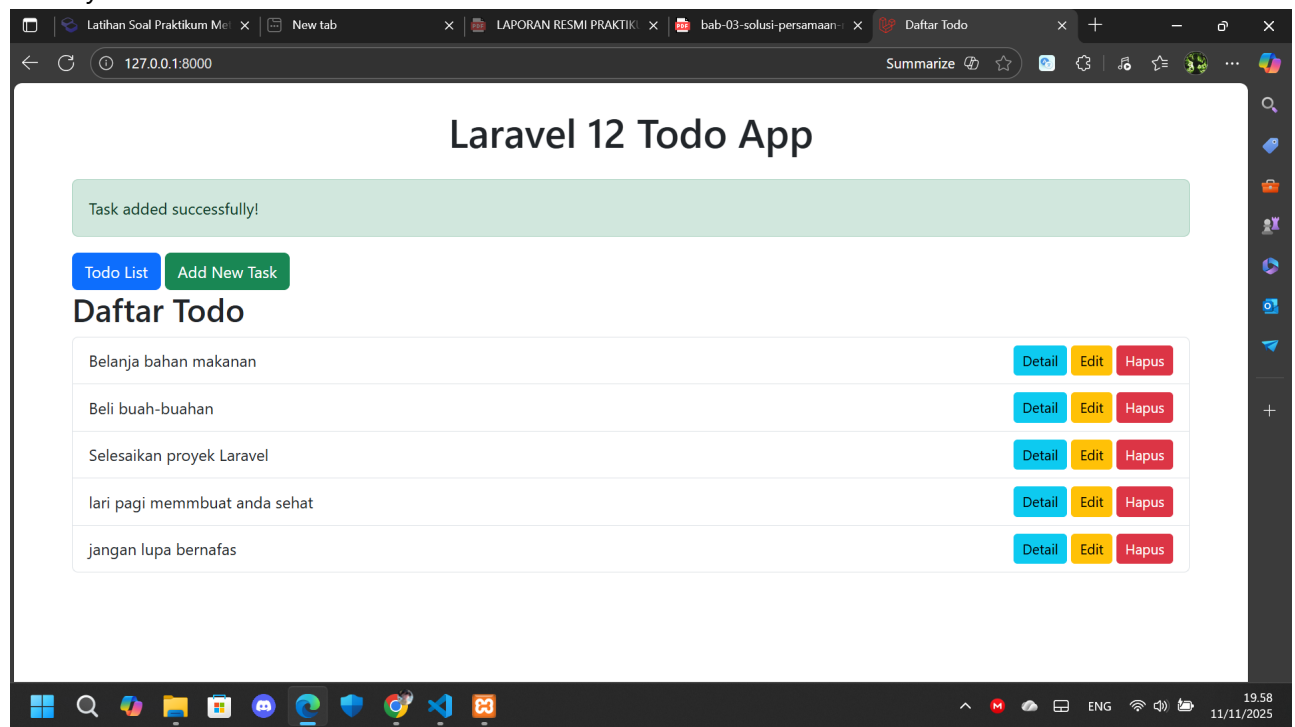
    <div class="card mt-3">
        <div class="card-body">
            <h5 class="card-title">{{ $todo->task }}</h5>
            <p class="card-text">Status: {{ $todo->completed ? 'Selesai' :
'Belum Selesai' }}</p>
            <a href="{{ route('todos.edit', $todo->id) }}" class="btn btn-
warning">Edit</a>
            <a href="{{ route('todos.index') }}" class="btn btn-
secondary">Kembali ke Daftar</a>
        </div>
    </div>

@endsection

```

- Langkah 8: Menguji Aplikasi  
jalankan aplikasi dengan ketik: `php artisan serve`  
Kemudian, kunjungi link `http://127.0.0.1:8000`

Hasilnya:



### 3. Hasil dan Pembahasan

#### Praktikum 1: Menggunakan Model untuk Binding Form dan Display

Pada praktikum pertama, mahasiswa membuat model simple dalam bentuk ViewModel yang disebut ProductViewModel.

Model ini digunakan untuk menampung data dari form input pengguna, misalnya nama produk, harga, dan deskripsi. Data ini lalu dikirim ke controller ProductController yang akan memproses input dan menampilkannya kembali ke halaman hasil.

Pengujian hasil menunjukkan bahwa form berhasil mengirim data ke controller dan menampilkan hasil di view product.result.blade.php. Dengan cara ini, mahasiswa dapat memahami konsep pemisahan presentasi data dengan logika aplikasi menggunakan model simple tanpa database.

#### Praktikum 2: Menggunakan DTO (Data Transfer Object)

Dalam praktik kedua, mahasiswa belajar bagaimana menggunakan Data Transfer Object (DTO) untuk mengelola data antar lapisan aplikasi.

Kelas ProductDTO berperan sebagai penghubung antara controller dan service layer (ProductService). Ketika form dikirim, data dikirim dan dikonversi menjadi objek DTO dengan menggunakan metode fromRequest(). Lalu, service layer mengatur bagaimana data tersebut disiapkan untuk view. Pendekatan ini memisahkan logika bisnis dari controller, dan ini membuat aplikasi menjadi lebih modular dan lebih mudah untuk diuji. Hasil uji coba menunjukkan data produk menggunakan DTO alih-alih di service dan berhasil ditampilkan di halaman hasil dengan rapi menggunakan Bootstrap.

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

dalam praktikum ini kita membangun aplikasi Todo List dilengkapi dengan CRUD (Create, Read, Update, Delete) pada Laravel 12 dan Eloquent ORM pada pengelola datanya pada praktikum ketiga. Langkah ini diawali dengan pembuatan migration untuk tabel todos, seeder untuk data dummy, dan pembuatan model Todo yang bertugas untuk mewakili tabel dan berinteraksi dengan database.

Semua logika CRUD diatur dan dikelola oleh Controller TodoController, sedangkan tampilan responsif sistem dibuat menggunakan Bootstrap dan diolah dengan engine template Blade. Pengujian menunjukkan semua fitur berfungsi dan sistem memenuhi fungsinya, disematkan tasks, pengguna sistem juga dapat menampilkan, mengedit, bahkan menghapus tasks.

Sessional flash message menyediakan dan mengawal fitur notifikasi, dan semua koneksi ke database MySQL terjamin prosesnya tanpa kendala.

---

## 4. Kesimpulan

Berdasarkan hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa model pada Laravel memiliki peran penting dalam menghubungkan antara aplikasi dan basis data. Dengan menerapkan arsitektur MVC, pengelolaan data menjadi lebih terstruktur dan mudah dipahami. Penerapan konsep Plain Old PHP Object (POPO) atau entity sederhana membantu dalam memisahkan logika bisnis dengan data tampilan, sehingga kode menjadi lebih bersih dan terorganisir. Selain itu, penggunaan Data Transfer Object (DTO) terbukti efektif dalam mentransfer data antar lapisan aplikasi secara aman dan konsisten, terutama pada proyek berskala besar.

Fitur migration dan seeder mempermudah pengaturan struktur serta pengisian data awal pada database, memastikan konsistensi antar lingkungan pengembangan. Sementara itu, Eloquent ORM memberikan kemudahan dalam melakukan operasi CRUD dan mengelola relasi antar tabel tanpa perlu menulis query SQL secara manual. Secara keseluruhan, kombinasi antara model, DTO, migration, seeder, dan Eloquent ORM menjadikan pengembangan aplikasi Laravel lebih efisien, terstruktur, dan profesional.

---

## 5. Referensi

- chatgpt.com — <https://chat.openai.com>
  - Laravel Documentation — <https://laravel.com/docs>
  - Modul 6 - Model dan Laravel Eloquent - <https://hackmd.io/@mohdrzu/ryIIM1a0II>
-