

PEMROGRAMAN WEB

BARBASIS FRAMEWORK (LARAVEL)



Arranged By :
Harry Awanda Putra, S.T.

DAFTAR ISI

PENGEMBANGAN PROYEK DENGAN LARAVEL (STUDI KASUS: TO-DO LIST)	3
1. Instalasi Laravel.....	3
2. Konfigurasi Database MySQL di Laravel.....	4
Langkah 1: Buat Database di MySQL	4
Langkah 2: Konfigurasi .env	4
3. Membuat Model, Migration, dan Controller.....	5
Langkah 1: Membuat Model dan Migration Task	5
Langkah 2: Definisikan Struktur Tabel di Migration	6
4. Membuat Blade Templates untuk Tampilan.....	7
Langkah 1: Buat Folder View	7
Langkah 2: Buat Template Layout Utama	7
5. Routing di web.php.....	8
6. Menampilkan Daftar Tugas (index).....	9
Langkah 1: Definisikan Logika Menampilkan Daftar Tugas di TaskController ..	9
Langkah 2: Membuat Halaman index.blade.php	11
7. Menambahkan Data (create dan store).....	14
Langkah 1: Menambahkan <i>Method create</i> di <i>Controller</i>	14
Langkah 2: Menambahkan Halaman create.blade.php	14
Langkah 3: Menambahkan <i>Method store</i> di <i>Controller</i>	15
Langkah 4: Update Model Task	17
8. Menampilkan Detail Tugas (show).....	18
Langkah 1: Menambahkan <i>Method show</i> di <i>Controller</i>	18
Langkah 2: Membuat Halaman show.blade.php	18
9. Mengedit Tugas (edit dan update).....	19
Langkah 1: Menambahkan <i>Method edit</i> di <i>Controller</i>	19
Langkah 2: Membuat Halaman edit.blade.php	19
Langkah 3: Menambahkan <i>Method update</i> di <i>Controller</i>	20
10. Menghapus Tugas (destroy)	21
11. Membuat Fungsi Tombol Status Selesai/Belum Selesai.....	22
Langkah 1: Menambahkan <i>Route</i> untuk <i>Update Status</i>	22
Langkah 2: Perbarui Method index() di <i>Controller</i>	23
Langkah 3: Perbarui Halaman index.blade.php	25
12. Jalankan aplikasi di lokal server.....	28

PENGEMBANGAN PROYEK DENGAN LARAVEL 11

(STUDI KASUS: TO-DO LIST)

1. Instalasi Laravel

Sebelum memulai, pastikan komputer Anda telah terinstal beberapa perangkat lunak berikut:

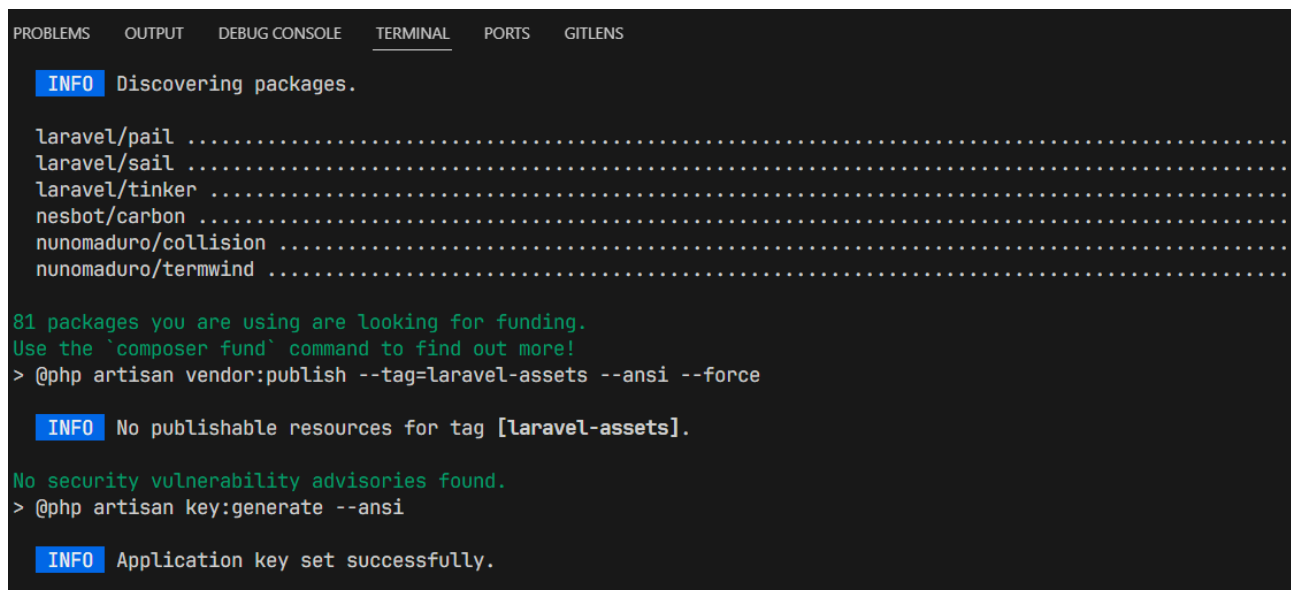
- **PHP** (minimal versi 8.2)
- **Composer** (dependency manager untuk PHP)
- **Database** (MySQL)
- **Laravel Installer** (opsional) atau menggunakan `composer create-project`

Jalankan perintah berikut di terminal atau command prompt untuk menginstal Laravel:

```
composer create-project laravel/laravel to-do_list
```

Masuk ke folder proyek:

```
cd to-do_list
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

INFO Discovering packages.

laravel/pail .....
laravel/sail .....
laravel/tinker .....
nesbot/carbon .....
nunomaduro/collision .....
nunomaduro/termwind .....

81 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

INFO No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
> @php artisan key:generate --ansi

INFO Application key set successfully.
```

2. Konfigurasi Database MySQL di Laravel

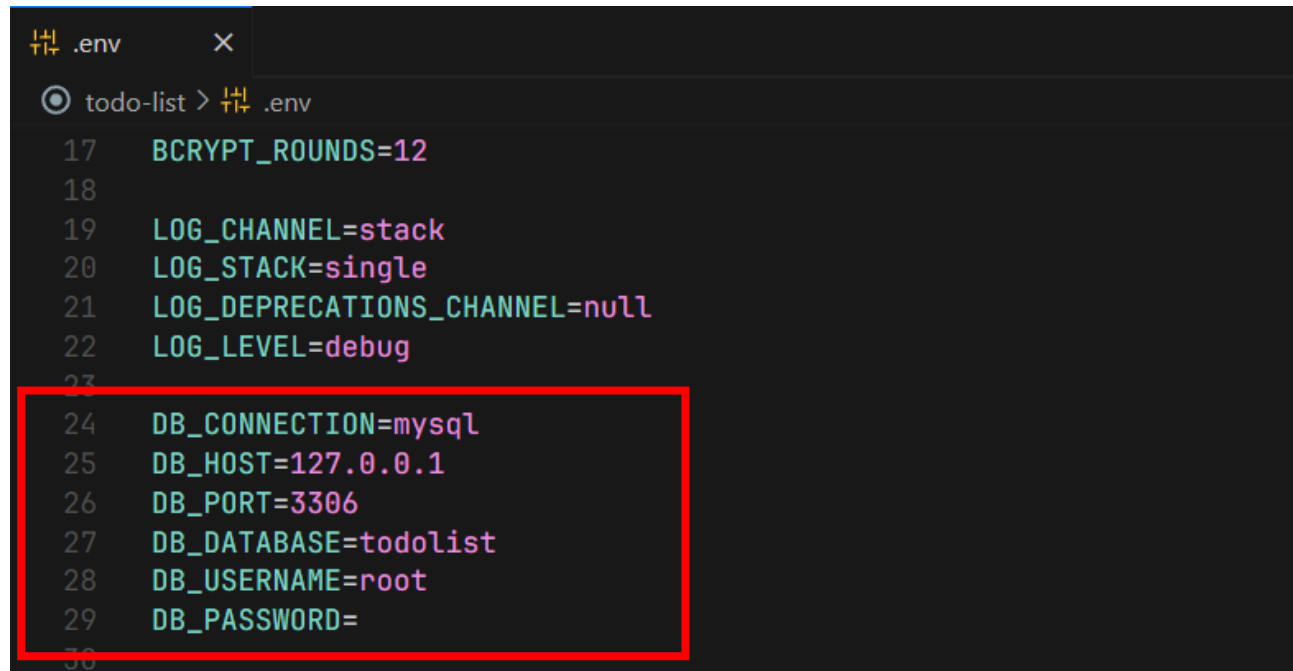
Langkah 1: Buat Database di MySQL

Buka MySQL dan buat *database* baru dengan perintah berikut:

```
CREATE DATABASE todolist;
```

Langkah 2: Konfigurasi .env

Buka file `.env` yang ada di *root* proyek Laravel, lalu ubah bagian berikut sesuai dengan pengaturan MySQL Anda:



```
17 BCRYPT_ROUNDS=12
18
19 LOG_CHANNEL=stack
20 LOG_STACK=single
21 LOG_DEPRECATED_CHANNELS=null
22 LOG_LEVEL=debug
23
24 DB_CONNECTION=mysql
25 DB_HOST=127.0.0.1
26 DB_PORT=3306
27 DB_DATABASE=todolist
28 DB_USERNAME=root
29 DB_PASSWORD=
30
```

3. Membuat Model, Migration, dan Controller

Langkah 1: Membuat Model dan Migration Task

Jalankan perintah berikut untuk membuat model **Task** dengan migration:

```
php artisan make:model Task -mcr
```

Perintah di atas akan menghasilkan tiga *file*:

- Model: `app/Models/Task.php`
- Migration: `database/migrations/xxxx_xx_xx_XXXXXX_create_tasks_table.php`
- Controller: `app/Http/Controllers/TaskController.php` (dengan metode `resource` bawaan Laravel)

Setelah perintah `make:model` kita tambahkan `-mcr` (dibaca *flag mcr* atau *flag (m) migration, (c) controller, (r) resource*) untuk membuat *migration* dan *controller* (dengan metode *resource*) dalam satu perintah. *Controller* dengan metode *resource* memungkinkan Laravel secara otomatis meng-*generate function* `index`, `create`, `store`, `show`, `edit`, `update`, dan `destroy`. Jika tidak menggunakan metode *resource*, maka kita perlu membuat *function* tersebut secara manual.

Perintah `php artisan make:model Task -mcr` adalah bentuk singkat dari perintah `php artisan make:model Task --migration --controller --resource`. Jika membuat satu per satu, maka perintah yang akan dijalankan pada Terminal atau command prompt adalah sebagai berikut.

- Model: `php artisan make:model Task`
- Migration: `php artisan make:migration create_tasks_table`
- Controller: `php artisan make:controller TaskController -resource`

Langkah 2: Definisikan Struktur Tabel di Migration

Buka file migration di

`database/migrations/#date_and_time#_create_tasks_table.php`,

lalu ubah kode di dalam fungsi `up()` seperti berikut:

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration {
8     /**
9      * Run the migrations.
10     */
11     public function up(): void {
12         Schema::create('tasks', function (Blueprint $table) {
13             $table->id();
14             $table->string('name');
15             $table->text('description')->nullable();
16             $table->boolean('is_completed')->default(false);
17             $table->timestamps();
18         });
19     }
20
21     /**
22      * Reverse the migrations.
23     */
24     public function down(): void {
25         Schema::dropIfExists('tasks');
26     }
27 };
```

Jalankan perintah migrasi untuk membuat tabel di database:

```
php artisan migrate
```

4. Membuat Blade Templates untuk Tampilan

Langkah 1: Buat Folder View

Buka folder `resources/views`, lalu buat folder baru bernama `tasks` untuk menyimpan semua tampilan yang berkaitan dengan daftar tugas.

Langkah 2: Buat Template Layout Utama

Buat file `resources/views/layouts/app.blade.php` dan tambahkan kode berikut:

```
1 <!DOCTYPE html>
2 <html lang="id">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>To-Do List</title>
7 </head>
8 <body>
9   <h1>To-Do List</h1>
10
11   @if(session('success'))
12     <p style="color: green;">{{ session('success') }}</p>
13   @endif
14
15   @yield('content')
16
17 </body>
18 </html>
```

Blade Template `app.blade.php` berfungsi sebagai *layout* utama dalam aplikasi To-Do List. Layout ini memastikan bahwa semua halaman yang menggunakannya memiliki struktur HTML yang sama, sehingga kode menjadi lebih rapi dan tidak perlu ditulis ulang di setiap halaman.

- Menampilkan pesan sukses jika ada
 - `@if(session('success'))` : Mengecek apakah ada pesan sukses yang disimpan dalam session (setelah operasi seperti menambah, mengedit, atau menghapus tugas).
 - `{{ session('success') }}` : Menampilkan pesan sukses dengan warna hijau.
- Menentukan tempat konten halaman lain
 - `@yield('content')` : Ini adalah tempat khusus untuk konten halaman lain yang akan menggunakan layout ini.
 - Halaman yang menggunakan layout ini harus mendefinisikan bagian `@section('content')`.

5. Routing di web.php

Buka file `routes/web.php` dan tambahkan baris berikut:

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\TaskController;
5
6 Route::get('/', function () {
7     return redirect()->route('tasks.index');
8 });
9
10 Route::resource('tasks', TaskController::class);
```

Penjelasan:

1. Import Library dan Controller

- `use Illuminate\Support\Facades\Route;`
Mengimpor Route Facade untuk mendefinisikan rute pada aplikasi Laravel.
- `use App\Http\Controllers\TaskController;`
Mengimpor TaskController agar bisa digunakan dalam routing.

2. Redirect Halaman Utama ke `/tasks`

- Ketika pengguna mengakses URL utama (`localhost:8000`) atau (`localhost:8000/`), mereka akan langsung diarahkan (`redirect()`) ke halaman daftar tugas (`tasks.index`).
- `route('tasks.index')` akan mengarahkan ke route yang meng-*handle* daftar tugas, yang didefinisikan oleh `Route::resource('tasks', TaskController::class);`.

3. Route Resource untuk CRUD tasks

- `Route::resource()` secara otomatis membuat 7 route untuk menangani CRUD (*Create, Read, Update, Delete*) dalam `TaskController`.
- Berikut daftar route yang dibuat:

HTTP Method	URI	Controller Method	Keterangan
GET	/tasks	index()	Menampilkan daftar tugas
GET	/tasks/create	create()	Menampilkan form tambah tugas
POST	/tasks	store()	Menyimpan tugas baru
GET	/tasks/{task}	show()	Menampilkan detail tugas
GET	/tasks/{task}/edit	edit()	Menampilkan form edit tugas
PUT/PATCH	/tasks/{task}	update()	Memperbarui tugas
DELETE	/tasks/{task}	destroy()	Menghapus tugas

6. Menampilkan Daftar Tugas (index)

Langkah 1: Definisikan Logika Menampilkan Daftar Tugas di **TaskController**

Buka file **app/Http/Controllers/TaskController.php**, lalu tambahkan *method* **index()** untuk menampilkan daftar tugas:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\Task;
7
8 class TaskController extends Controller {
9     public function index() {
10         $tasks = Task::orderBy('is_completed', 'asc')->get();
11         return view('tasks.index', compact('tasks'));
12     }
13
14     public function create() {
15         //
16     }
17
18     public function store(Request $request) {
19         //
20     }
21
22     public function show(Task $task) {
23         //
24     }
25
26     public function edit(Task $task) {
27         //
28     }
29
30     public function update(Request $request, Task $task) {
31         //
32     }
33
34     public function destroy(Task $task) {
35         //
36     }
37 }
```

Penjelasan:

1. Mengambil Data Tugas dari Database

```
$tasks = Task::orderBy('is_completed', 'asc')->get();
```

- `Task::orderBy('is_completed', 'asc')`
Mengambil semua tugas dari tabel tasks dan mengurutkannya berdasarkan status penyelesaian (`is_completed`).
- Urutan `asc` (*ascending* / naik) berarti tugas yang belum selesai (`is_completed = 0`) akan muncul terlebih dahulu, baru diikuti oleh tugas yang sudah selesai (`is_completed = 1`).
- `get()`
Menjalankan *query* dan mengambil semua data tugas sebagai *collection* Laravel.

2. Mengirim Data ke View

```
return view('tasks.index', compact('tasks'));
```

- `view('tasks.index')`
Mengembalikan tampilan `index.blade.php` yang ada di dalam folder `resources/views/tasks/`.
- `compact('tasks')`
Mengirim variabel `$tasks` ke view, sehingga bisa digunakan dalam tampilan `index.blade.php`.

Langkah 2: Membuat Halaman index.blade.php

Buat file `resources/views/tasks/index.blade.php` untuk menampilkan semua tugas:

```
1 @extends('layouts.app')
2
3 @section('content')
4     <h2>Daftar Tugas</h2>
5
6     <table border="1" cellspacing="0" cellpadding="5">
7         <thead>
8             <tr>
9                 <th>#</th>
10                <th>Nama Tugas</th>
11                <th>Status</th>
12                <th><a href="{{ route('tasks.create') }}">Tambah Tugas</a></th>
13            </tr>
14        </thead>
15        <tbody>
16            @foreach($tasks as $task)
17                <tr>
18                    <td>{{ $loop->index+1 }}</td>
19                    <td>{{ $task->name }}</td>
20                    <td>{{ $task->is_completed ? 'Tandai Belum Selesai' : 'Tandai
Selesai' }}</td>
21                    <td>
22                        <a href="{{ route('tasks.show', $task->id) }}">Detail</a> |
23                        <a href="{{ route('tasks.edit', $task->id) }}">Edit</a> |
24                        <form action="{{ route('tasks.destroy', $task->id) }}"
method="POST" style="display: inline;">
25                            @csrf
26                            @method('DELETE')
27                            <button type="submit" onclick="return confirm('Hapus tugas
ini?')">Hapus</button>
28                        </form>
29                    </td>
30                </tr>
31            @endforeach
32        </tbody>
33    </table>
34 @endsection
```

Penjelasan:

1. Perulangan untuk Menampilkan Daftar Tugas

```
@foreach($tasks as $task)
```

- `$tasks` : Variabel yang berisi daftar tugas yang dikirim dari *controller* (method `index()`).
- `@foreach($tasks as $task)` berarti kita melakukan iterasi (menggunakan `foreach`) terhadap setiap tugas yang ada di dalam `$tasks`.
- Variabel `$task` merepresentasikan satu tugas dalam setiap iterasi.

2. Kolom Nomor Urut

```
<td>{{ $loop->index+1 }}</td>
```

- `$loop->index` adalah fitur bawaan *Blade* yang memberikan indeks perulangan (dimulai dari 0).
- `+1` digunakan agar nomor urut dimulai dari 1.

3. Kolom Nama Tugas

```
<td>{{ $task->name }}</td>
```

- Menampilkan nama tugas dari *database* (`$task->name`).

4. Kolom Status

```
<td>{{ $task->is_completed ? 'Selesai' : 'Belum Selesai' }}</td>
```

Kode ini menggunakan **operator ternary**:

- `$task->is_completed` : Mengambil nilai dari kolom `is_completed` di *database*.
- `?` (Tanda tanya) → Digunakan untuk menentukan kondisi.
- Jika `is_completed == true (1)` : Tampilkan "Selesai".
- Jika `is_completed == false (0)` : Tampilkan "Belum Selesai".

Singkatnya:

- Jika tugas sudah selesai, teks yang ditampilkan "Selesai".
- Jika tugas belum selesai, teks yang ditampilkan "Belum Selesai".

5. Tombol Aksi (Detail, Edit, Hapus)

```
<td>
  <a href="{{ route('tasks.show', $task->id) }}">Detail</a> |
  <a href="{{ route('tasks.edit', $task->id) }}">Edit</a> |
  <form action="{{ route('tasks.destroy', $task->id) }}"
method="POST" style="display: inline;">
    @csrf
    @method('DELETE')
    <button type="submit" onclick="return confirm('Hapus tugas
ini?')">Hapus</button>
  </form>
</td>
```

- Detail (*show*)

```
<a href="{{ route('tasks.show', $task->id) }}">Detail</a>
```

Mengarah ke halaman detail tugas dengan `route('tasks.show', $task->id)`.

- Edit (*edit*)

```
<a href="{{ route('tasks.edit', $task->id) }}">Edit</a>
```

Mengarah ke halaman form edit tugas.

- Hapus (*destroy*)

```
<form action="{{ route('tasks.destroy', $task->id) }}"
method="POST" style="display: inline;">
  @csrf
  @method('DELETE')
  <button type="submit" onclick="return confirm('Hapus tugas
ini?')">Hapus</button>
</form>
```

- Menggunakan metode `DELETE` yang didukung oleh Laravel.
- `@csrf` : Mencegah serangan keamanan.
- Konfirmasi sebelum menghapus (`onclick="return confirm('Hapus tugas ini?')"`).

7. Menambahkan Data (create dan store)

Langkah 1: Menambahkan *Method create* di *Controller*

Fungsi `create()` dan `store()` digunakan untuk menambahkan tugas baru dalam aplikasi To-Do List. Tambahkan method `create()` di `TaskController.php`:

```
14 public function create() {  
15     return view('tasks.create');  
16 }
```

Penjelasan:

- *Function* ini hanya bertugas untuk menampilkan halaman form tambah tugas (`create.blade.php`).
- `return view('tasks.create');`
Mengembalikan tampilan `resources/views/tasks/create.blade.php`, yang berisi form input untuk membuat tugas baru.

Langkah 2: Menambahkan Halaman `create.blade.php`

Buat file `resources/views/tasks/create.blade.php`:

```
1 @extends('layouts.app')  
2  
3 @section('content')  
4     <h2>Tambah Tugas</h2>  
5  
6     <form action="{{ route('tasks.store') }}" method="POST">  
7         @csrf  
8         <label for="name">Nama Tugas:</label>  
9         <input type="text" name="name" required><br><br>  
10  
11         <label for="description">Deskripsi:</label><br>  
12         <textarea name="description"></textarea><br><br>  
13  
14         <button type="submit">Simpan</button>  
15         <a href="{{ route('tasks.index') }}">Batal</a>  
16     </form>  
17 @endsection
```

Langkah 3: Menambahkan Method *store* di *Controller*

Tambahkan method `store()` di `TaskController.php` untuk menyimpan data:

```
18 public function store(Request $request) {
19     $request->validate([
20         'name' => 'required|string|max:255',
21         'description' => 'nullable|string',
22     ]);
23
24     Task::create($request->all());
25     return redirect()->route('tasks.index')->with('success',
    'Tugas berhasil ditambahkan.');
```

Penjelasan:

Fungsi ini digunakan untuk menyimpan data tugas baru ke dalam database setelah user mengisi form.

1. Validasi Input

```
$request->validate([
    'name' => 'required|string|max:255',
    'description' => 'nullable|string',
]);
```

- `$request->validate()`
Memeriksa apakah input yang dikirimkan sesuai aturan berikut:
 - `name` harus diisi (*required*), berupa teks (*string*), dan maksimal 255 karakter.
 - `description` boleh kosong (*nullable*) tapi jika diisi harus berupa teks (*string*).
- Jika validasi gagal, Laravel akan otomatis mengembalikan pengguna ke halaman sebelumnya dengan pesan error.

2. Menyimpan Data ke Database

```
Task::create($request->all());
```

- `Task::create([...])`
Menyimpan tugas baru ke tabel `tasks` menggunakan *mass assignment*.
- `'name' => $request->name`: Mengambil nilai dari form input `name`.
- `'description' => $request->description`: Mengambil nilai dari form input `description`.
 - `'is_completed' => false` : Tugas baru secara *default* dianggap belum selesai (*false* atau 0)
- `$request->all()` mengambil semua data dari form input (kolom `name` dan `description`) yang dikirimkan.
- `Task::create()` menyimpan data ke database dengan *Mass Assignment*, sesuai dengan daftar `$fillable` yang telah didefinisikan di model `Task`.

Bagaimana dengan `is_completed`?

Kolom `is_completed` tidak diisi dalam `store()`, tetapi tetap tersimpan dengan nilai *false*. Ini terjadi karena dalam migration kolom ini telah memiliki nilai default *false*:

```
Schema::create('tasks', function (Blueprint $table) {  
    $table->id();  
    $table->string('name');  
    $table->text('description')->nullable();  
    $table->boolean('is_completed')->default(false);  
    $table->timestamps();  
});
```

- Jika kolom `is_completed` tidak dikirimkan dari form, maka otomatis akan disimpan dengan nilai *false*.
- Ini memastikan bahwa tugas baru selalu dimulai dengan status "Belum Selesai".

3. Redirect ke Halaman Daftar Tugas

```
return redirect()->route('tasks.index')->with('success',  
'Tugas berhasil ditambahkan.');
```

- `redirect()->route('tasks.index')` : Setelah berhasil menyimpan data, pengguna dialihkan ke halaman daftar tugas (`index.blade.php`).
- `with('success', 'Tugas berhasil ditambahkan.')` : Mengirim pesan sukses yang akan ditampilkan di halaman index.

Langkah 4: Update Model Task

File `Task.php` adalah model dalam Laravel yang mewakili tabel `tasks` di *database*. Model ini digunakan untuk berinteraksi dengan data tugas, termasuk mengambil, menyimpan, memperbarui, dan menghapus data dari tabel tersebut. Buka `app/Models/Task.php` dan tambahkan `$fillable` agar *mass assignment* bisa digunakan:

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Task extends Model {
8     protected $fillable = [
9         'name', 'description', 'is_completed',
10    ];
11 }
```

Penjelasan:

Mengaktifkan Mass Assignment dengan `$fillable`

- `$fillable` menentukan kolom mana saja yang boleh diisi secara massal melalui `Task::create([...])` atau `Task::update([...])`.
- `'name'` : Menyimpan nama tugas.
- `'description'` : Menyimpan deskripsi tugas.
- `'is_completed'` : Menyimpan status apakah tugas sudah selesai atau belum (0 untuk belum selesai, 1 untuk selesai).

Menggunakan `$fillable` sangat penting untuk mencegah *Mass Assignment Vulnerability*, yaitu serangan di mana *user* bisa mengubah data yang tidak seharusnya.

8. Menampilkan Detail Tugas (show)

Langkah 1: Menambahkan *Method show* di *Controller*

Buka `app/Http/Controllers/TaskController.php` dan tambahkan *method* berikut:

```
33 public function show(Task $task) {  
34     return view('tasks.show', compact('task'));  
35 }
```

Langkah 2: Membuat Halaman `show.blade.php`

Buat file `resources/views/tasks/show.blade.php` untuk menampilkan detail tugas:

```
1 @extends('layouts.app')  
2  
3 @section('content')  
4     <h2>Detail Tugas</h2>  
5  
6     <p><strong>Judul:</strong> {{ $task->name }}</p>  
7     <p><strong>Deskripsi:</strong> {{ $task->description }}</p>  
8     <p><strong>Status:</strong> {{ $task->is_completed ?  
    'Selesai' : 'Belum Selesai' }}</p>  
9  
10    <a href="{{ route('tasks.index') }}">Kembali</a>  
11 @endsection
```

9. Mengedit Tugas (edit dan update)

Langkah 1: Menambahkan *Method edit* di *Controller*

Tambahkan *method* berikut di **TaskController.php**:

```
37 public function edit(Task $task) {  
38     return view('tasks.edit', compact('task'));  
39 }
```

Penjelasan:

- Parameter **Task \$task** : Laravel otomatis mencari tugas berdasarkan **ID** yang dikirim dalam URL.
- **return view('tasks.edit', compact('task'))**
 - Menampilkan halaman edit (**edit.blade.php**).
 - Mengirimkan data tugas yang dipilih ke tampilan, sehingga form edit dapat terisi otomatis dengan data lama.

Langkah 2: Membuat Halaman **edit.blade.php**

Buat file **resources/views/tasks/edit.blade.php**:

```
1 @extends('layouts.app')  
2  
3 @section('content')  
4     <h2>Edit Tugas</h2>  
5  
6     <form action="{{ route('tasks.update', $task->id) }}" method="POST">  
7         @csrf  
8         @method('PUT')  
9  
10        <label for="name">Nama Tugas:</label>  
11        <input type="text" name="name" value="{{ $task->name }}" required><br><br>  
12  
13        <label for="description">Deskripsi:</label><br>  
14        <textarea name="description">{{ $task->description }}</textarea><br><br>  
15  
16        <button type="submit">Simpan</button>  
17        <a href="{{ route('tasks.index') }}">Batal</a>  
18    </form>  
19 @endsection
```

Langkah 3: Menambahkan Method update di Controller

Tambahkan *method* berikut di **TaskController.php**:

```
41 public function update(Request $request, Task $task) {
42     $request->validate([
43         'name' => 'required|string|max:255',
44         'description' => 'nullable|string',
45     ]);
46
47     $task->update($request->all());
48
49     return redirect()->route('tasks.index')->with('success',
    'Tugas berhasil diperbarui.');
```

Penjelasan:

Fungsi ini digunakan untuk menyimpan perubahan data tugas ke dalam *database* setelah user mengedit form.

1. Validasi Input

```
$request->validate([
    'name' => 'required|string|max:255',
    'description' => 'nullable|string',
]);
```

- **\$request->validate()** : Memeriksa apakah input yang dikirimkan sesuai aturan berikut:
 - **name** harus diisi (**required**), berupa teks (**string**), dan maksimal 255 karakter.
 - **description** boleh kosong (**nullable**) tetapi jika diisi harus berupa teks (**string**).
- Jika validasi gagal, Laravel akan otomatis mengembalikan pengguna ke halaman edit dengan pesan error.

2. Memperbarui Data ke Database

```
$task->update($request->all());
```

- **\$request->all()** : Mengambil semua input dari form yang sudah divalidasi.
- **\$task->update([...])** : Memperbarui tugas yang dipilih dengan data baru.
- Karena kita sudah menggunakan **\$fillable** dalam **Task.php**, kita bisa langsung menggunakan *mass assignment* tanpa harus menulis setiap kolom secara manual.

Pastikan di Model Task.php ada **protected \$fillable** agar *mass assignment* bisa berjalan

3. Redirect ke Halaman Daftar Tugas dengan Pesan Sukses

```
return redirect()->route('tasks.index')->with('success',  
'Tugas berhasil diperbarui.');
```

- `redirect()->route('tasks.index')` : Setelah berhasil memperbarui tugas, *user* diarahkan kembali ke daftar tugas (`index.blade.php`).
- `with('success', 'Tugas berhasil diperbarui.')`
Mengirim pesan sukses ke halaman daftar tugas agar *user* tahu bahwa perubahan telah berhasil.

10. Menghapus Tugas (destroy)

Tambahkan *method* berikut di `TaskController.php`:

```
56 public function destroy(Task $task) {  
57     $task->delete();  
58     return redirect()->route('tasks.index')->with('success',  
    'Tugas berhasil dihapus.');
```

```
59 }
```

Penjelasan:

1. Parameter `Task $task`

```
public function destroy(Task $task)
```

- Laravel otomatis mencari tugas berdasarkan `ID` yang dikirim melalui URL.
- Karena kita menggunakan *Route Model Binding*, Laravel secara otomatis akan mengambil data dari *database* yang sesuai dengan `ID` yang dikirim.

2. Menghapus Data dari Database

```
$task->delete();
```

Menghapus tugas yang telah ditemukan dari *database*. Perintah ini akan menjalankan perintah SQL seperti berikut:

```
DELETE FROM tasks WHERE id = ?
```

Jika tugas berhasil dihapus, data tersebut **tidak bisa dikembalikan**.

3. Redirect ke Halaman Daftar Tugas dengan Pesan Sukses

```
return redirect()->route('tasks.index')->with('success',  
'Tugas berhasil dihapus.');
```

- `redirect()->route('tasks.index')`
Setelah tugas dihapus, pengguna diarahkan kembali ke halaman daftar tugas (`index.blade.php`).
- `with('success', 'Tugas berhasil dihapus.')`
Menyimpan pesan sukses agar bisa ditampilkan di halaman daftar tugas.

11. Membuat Fungsi Tombol Status Selesai/Belum Selesai

Langkah 1: Menambahkan *Route* untuk *Update Status*

Tambahkan route baru di `routes/web.php`:

```
1 <?php  
2  
3 use Illuminate\Support\Facades\Route;  
4 use App\Http\Controllers\TaskController;  
5  
6 Route::get('/', function () {  
7     return redirect()->route('tasks.index');  
8 });  
9  
10 Route::resource('tasks', TaskController::class);  
11 Route::patch('/tasks/{task}/toggle-status',  
12 [TaskController::class, 'index'])->name('tasks.toggle-status');
```

Langkah 2: Perbarui Method `index()` di *Controller*

Sekarang, kita ubah method `index()` di `TaskController.php` agar bisa menangani permintaan perubahan status sekaligus menampilkan daftar tugas:

```
9  public function index(Request $request, Task $task = null) {
10      // Jika request adalah untuk mengubah status
11      if ($request->isMethod('patch') && $task) {
12          $task->update([
13              'is_completed' => !$task->is_completed,
14          ]);
15          return redirect()->route('tasks.index')->with('success',
16              'Status tugas berhasil diperbarui.');
```

Penjelasan penambahan kode pada fungsi `index()`:

1. Parameter Function

```
public function index(Request $request, Task $task = null)
```

- **Request \$request** : Digunakan untuk membaca metode HTTP (GET, PATCH, dll.) dan input dari pengguna.
- **Task \$task = null** : Secara *default*, `$task` diatur sebagai `null`. Jika pengguna mengirimkan permintaan PATCH untuk mengubah status tugas, Laravel akan otomatis mencari tugas berdasarkan ID dari route.

2. Mengubah Status Tugas (PATCH Request)

```
if ($request->isMethod('patch') && $task) {  
    $task->update([  
        'is_completed' => !$task->is_completed,  
    ]);  
    return redirect()->route('tasks.index')-  
>with('success', 'Status tugas berhasil diperbarui.');
```

Cara Kerjanya:

- `$request->isMethod('patch')` : Mengecek apakah request yang diterima menggunakan metode `PATCH`.
- `&& $task` : Memastikan bahwa tugas yang akan diubah statusnya memang tersedia.
- `$task->update([...])`
 - Mengubah nilai `is_completed` menjadi kebalikan dari nilai sebelumnya (`true` → `false` atau `false` → `true`).
 - Jika tugas sebelumnya belum selesai (`false`), maka akan menjadi selesai (`true`), dan sebaliknya.
- `return redirect()->route('tasks.index')->with('success', 'Status tugas berhasil diperbarui.');`
 - Setelah status diperbarui, pengguna akan diarahkan kembali ke halaman daftar tugas dengan pesan sukses.

Fitur ini memungkinkan *user* untuk menandai tugas sebagai selesai atau belum langsung dari halaman index.

Langkah 3: Perbarui Halaman `index.blade.php`

Agar form tetap bekerja dengan *method* `index()`, ubah tombol status di `index.blade.php`:

```
1 @extends('layouts.app')
2
3 @section('content')
4     <h2>Daftar Tugas</h2>
5
6     <table border="1" cellspacing="0" cellpadding="5">
7         <thead>
8             <tr>
9                 <th>#</th>
10                <th>Nama Tugas</th>
11                <th>Status</th>
12                <th><a href="{{ route('tasks.create') }}">Tambah Tugas</a></th>
13            </tr>
14        </thead>
15        <tbody>
16            @foreach($tasks as $task)
17                <tr>
18                    <td>{{ $loop->index+1 }}</td>
19                    <td>{{ $task->name }}</td>
20                    <td>
21                        <form action="{{ route('tasks.toggle-status', $task->id) }}"
method="POST">
22                            @csrf
23                            @method('PATCH')
24                            <button type="submit">
25                                {{ $task->is_completed ? 'Tandai Belum Selesai' :
'Tandai Selesai' }}
26                            </button>
27                        </form>
28                    </td>
29                    <td>
30                        <a href="{{ route('tasks.show', $task->id) }}">Detail</a> |
31                        <a href="{{ route('tasks.edit', $task->id) }}">Edit</a> |
32                        <form action="{{ route('tasks.destroy', $task->id) }}"
method="POST" style="display: inline;">
33                            @csrf
34                            @method('DELETE')
35                            <button type="submit" onclick="return confirm('Hapus tugas
ini?')">Hapus</button>
36                        </form>
37                    </td>
38                </tr>
39            @endforeach
40        </tbody>
41    </table>
42 @endsection
```

Penjelasan:

Tombol untuk Menandai Tugas Selesai/Belum

```
<td>
  <form action="{{ route('tasks.toggle-status', $task->id) }}"
method="POST">
  @csrf
  @method('PATCH')
  <button type="submit">
    {{ $task->is_completed ? 'Tandai Belum Selesai' : 'Tandai Selesai' }}
  </button>
</form>
</td>
```

- Membuat Form untuk Mengubah Status
 - Action: `route('tasks.toggle-status', $task->id)` :
Mengarah ke *route* yang menangani perubahan status tugas.
 - Method: `POST` dengan `@method('PATCH')` :
Laravel menganggap *request* ini sebagai `PATCH` untuk *update* data.
 - `@csrf` : Token keamanan Laravel untuk mencegah `CSRF attack`.
- Tombol Status

```
<button type="submit">
  {{ $task->is_completed ? 'Tandai Belum Selesai' : 'Tandai
Selesai' }}
</button>
```

- Teks tombol berubah secara dinamis berdasarkan status tugas (`is_completed`).
 - Jika tugas sudah selesai (`is_completed == true`), tombol akan menampilkan "`Tandai Belum Selesai`".
 - Jika tugas belum selesai (`is_completed == false`), tombol akan menampilkan "`Tandai Selesai`".
- Saat tombol ditekan, form akan mengirim request `PATCH` ke URL yang sesuai dengan ID tugas, sehingga statusnya berubah.

Kenapa Menggunakan Metode PATCH?

Dalam konteks Laravel dan RESTful API, perbedaan utama antara *method* **PUT** dan **PATCH** adalah bagaimana mereka menangani pembaruan data:

1. PUT (Edit Form - Update Seluruh Data)

- PUT digunakan untuk **mengganti seluruh sumber daya** dengan data baru yang dikirimkan.
- Biasanya digunakan dalam form edit (**edit.blade.php**) untuk memperbarui semua *field* dari suatu model.
- Contohnya, dalam form edit tugas, kita mungkin mengubah **judul tugas** dan **statusnya** sekaligus.

2. PATCH (Toggle Button - Update Sebagian Data)

- **PATCH** digunakan untuk **memperbarui sebagian dari sumber daya**, bukan seluruhnya.
- Cocok untuk fitur seperti tombol *toggle* "**Tandai Selesai/Belum Selesai**" karena hanya mengubah satu atribut (misalnya, **is_completed**).
- Tidak perlu mengirimkan semua *field*, cukup *field* yang berubah saja.

12. Jalankan aplikasi di lokal server

Jalankan perintah `php artisan serve` pada terminal dan akses URI `localhost:8000/` atau `localhost:8000/tasks` pada browser Anda.

To-Do List

localhost:8000/tasks

☆

Incognito

To-Do List

Daftar Tugas

#	Nama Tugas	Status	Tambah Tugas
1	PBO	<div>Tandai Selesai</div>	Detail Edit <div>Hapus</div>
2	Database	<div>Tandai Selesai</div>	Detail Edit <div>Hapus</div>
3	Pemrograman Web	<div>Tandai Belum Selesai</div>	Detail Edit <div>Hapus</div>
4	Matematika	<div>Tandai Belum Selesai</div>	Detail Edit <div>Hapus</div>

To-Do List

localhost:8000/tasks/3

☆

To-Do List

Detail Tugas

Judul: Pemrograman Web

Deskripsi: Belajar Pemrograman Web Laravel

Status: Belum Selesai

[Kembali](#)