TUTORIAL

MEMBANGUN APLIKASI TO-DO LIST DARI NOL

Rifky Najra Adipura
Ode Andi Alamsyah
Rendy Kamaluddin
A.M. Faraziftan
Risyad Ridwansyah
Syafrial Fachri Pane
Muhammad Haitsam Izzuddin Azman

MEMBANGUN APLIKASI TO-DO-LIST DARI NOL

Rifky Najra Adipura
Ode Andi Alamsyah
Risyad Ridwansyah
Rendy Kamaluddin
Nyi Raden Nuraini Siti Fathonah
Syafrial Fachri Pane
Amri Yanuar
Irfan (Guru SMK)

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa, karena atas rahmat-Nya, buku **"Membangun Aplikasi To-Do-List Dari Nol"** ini dapat kami selesaikan. Tujuan utama kami menulis buku ini adalah untuk menyediakan sebuah panduan praktis yang menjembatani antara teori dan praktik dalam dunia pengembangan web. Melalui studi kasus pembuatan aplikasi *to-do-list* dengan *framework* Laravel, kami berharap dapat membantu para pemula untuk merasakan pengalaman membangun proyek nyata dari awal hingga akhir.

Buku ini kami tujukan bagi para penggiat teknologi, para pemula, dan siapa saja yang memiliki semangat untuk belajar *backend development*, bahkan tanpa latar belakang IT yang mendalam. Keunggulan utama buku ini terletak pada pendekatannya yang berbasis proyek, di mana setiap bab akan memandu Anda selangkah demi selangkah dalam membangun sebuah aplikasi yang fungsional.

Akhir kata, kami mengucapkan terima kasih kepada semua pihak yang telah mendukung. Pesan kami untuk para pembaca: jangan pernah takut untuk mencoba dan menghadapi *error*, karena itulah cara terbaik untuk belajar.

Bandung, 25 Agustus 2025

PENULIS

PRAKATA

Kemampuan membangun aplikasi web kini telah menjadi salah satu keahlian esensial di era digital. Namun, banyaknya konsep dan teknologi seringkali menjadi penghalang bagi para pemula. Buku **"Membangun Aplikasi To-Do-List Dari Nol"** yang ditulis oleh kami hadir untuk menjawab tantangan tersebut dengan cara yang lugas dan terstruktur.

Buku ini memiliki keunggulan pada metode pembelajarannya yang berbasis proyek. Dengan mengikuti panduan membangun sebuah aplikasi *to-do-list* menggunakan Laravel, pembaca tidak hanya disuguhkan teori, tetapi diajak langsung untuk mempraktikkannya. Pendekatan ini didukung penuh dengan ketersediaan seluruh kode sumber yang dapat diakses melalui tautan: **https://github.com/ulbithebest/Lara-todolist**.

Saya sangat merekomendasikan buku ini bagi siapa saja yang ingin memulai perjalanannya di dunia pengembangan web. Buku ini adalah panduan yang tepat untuk mengubah rasa penasaran menjadi sebuah keahlian nyata.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENGENALAN LARAVEL

Bab ini akan membahas dasar-dasar *Laravel*, mulai dari sejarah dan perkembangannya, keunggulan yang dimilikinya, hingga konsep arsitektur MVC yang menjadi fondasi *framework* ini. Selain itu, Anda juga akan diperkenalkan dengan fitur-fitur utama *Laravel* yang akan digunakan dalam membangun aplikasi *To-Do-List*. Dengan memahami materi ini, Anda akan memiliki pondasi yang kuat untuk memulai perjalanan pengembangan aplikasi web menggunakan *Laravel*.

1.1 Sejarah dan Perkembangan Laravel

Laravel pertama kali dirilis pada tahun 2011 oleh Taylor Otwell. Taylor menciptakan Laravel untuk menyederhanakan proses pengembangan web dengan menyediakan sintaks yang intuitif dan fitur-fitur siap pakai. Sejak itu, Laravel terus berkembang dengan rilis versi terbaru yang membawa fitur-fitur canggih seperti Laravel Jetstream, Livewire, dan banyak lagi. Saat ini, Laravel menjadi salah satu framework PHP paling banyak digunakan di dunia.

Perkembangan Laravel didorong oleh filosofi *"developer happiness"*, yang bertujuan membuat pengembangan aplikasi web menjadi lebih menyenangkan dan efisien. Dengan dukungan komunitas yang besar dan ekosistem yang terus berkembang, Laravel telah menjadi pilihan utama bagi banyak developer, baik pemula maupun berpengalaman.

1.2 Keunggulan Laravel Dibanding *Framework PHP* Lainnya

Laravel memiliki beberapa keunggulan yang membuatnya menonjol di antara *framework PHP* lainnya:

1. Sintaks yang Elegan dan Mudah Dipahami, Laravel menggunakan sintaks yang bersih dan intuitif, sehingga kode yang ditulis menjadi lebih rapi dan mudah dipelihara. Hal ini membuat developer lebih produktif dan mengurangi kemungkinan kesalahan.
2. Dokumentasi yang Lengkap dan Terstruktur, Laravel menyediakan dokumentasi resmi yang sangat detail dan mudah diikuti. Dokumentasi

ini mencakup semua fitur Laravel, mulai dari instalasi hingga penggunaan fitur-fitur lanjutan.

3. Komunitas yang Besar dan Aktif, Laravel memiliki komunitas global yang sangat aktif. Jika Anda mengalami masalah, kemungkinan besar solusinya sudah tersedia di forum atau situs seperti *Stack Overflow*. Komunitas ini juga sering mengadakan acara seperti Laracon dan *meetup* lokal.
4. Fitur Bawaan yang Lengkap, Laravel menyediakan banyak fitur siap pakai seperti sistem autentikasi, manajemen *database*, dan *templating*, sehingga *developer* tidak perlu membuatnya dari nol. Fitur-fitur ini memungkinkan pengembangan aplikasi *web* menjadi lebih cepat dan efisien.

1.3 Konsep MVC dalam Laravel

MVC *(Model-View-Controller)* adalah pola desain arsitektur yang memisahkan aplikasi menjadi tiga komponen utama: *Model*, *View*, dan *Controller*. Pola ini membantu *developer* mengorganisir kode dengan lebih baik dan memudahkan pengelolaan aplikasi.

1. *Model* bertanggung jawab untuk mengelola data dan logika bisnis aplikasi. Misalnya, dalam aplikasi *To-Do List*, *Model* akan mengatur data tugas seperti judul, deskripsi, dan status tugas. Di Laravel, *Model* disimpan di folder *app/Models/*.
2. *View* bertugas menampilkan data ke pengguna. Contohnya, menampilkan daftar tugas dalam bentuk tabel atau kartu di halaman *web*. Di Laravel, View disimpan di folder *resources/views/*.
3. *Controller* berperan sebagai penghubung antara *Model* dan *View*. *Controller* menerima *input* dari pengguna (misalnya, melalui *form*), memprosesnya, dan mengembalikan *output* yang sesuai. Di Laravel, *Controller* disimpan di folder *app/Http/Controllers/*.

Dengan memisahkan logika aplikasi ke dalam komponen-komponen ini, kode menjadi lebih terstruktur dan mudah dikelola.

1.4 Fitur-Fitur Utama Laravel

Laravel dilengkapi dengan banyak fitur yang memudahkan pengembangan aplikasi web. Berikut adalah beberapa fitur utama yang akan digunakan dalam proyek *To-Do List*:

1. *Routing* digunakan untuk mengatur *URL* dan halaman aplikasi. Misalnya, Anda dapat mengarahkan *URL /tasks* ke halaman yang menampilkan daftar tugas.
2. *Eloquent* adalah *Object-Relational Mapping* (ORM) bawaan Laravel yang memudahkan interaksi dengan *database*. Dengan *Eloquent*, Anda dapat mengelola data tabel *database* seperti mengelola objek *PHP*.
3. *Blade Template Engine* adalah mesin *templating* Laravel yang memungkinkan pembuatan tampilan yang fleksibel dan dinamis. *Blade* menyediakan sintaks sederhana untuk menampilkan data, melakukan perulangan, dan membuat kondisi.
4. *Middleware* digunakan untuk memfilter *request HTTP* sebelum mencapai *controller*. Misalnya, Anda dapat membuat *middleware* untuk memeriksa apakah pengguna sudah *login* sebelum mengakses halaman tertentu.
5. *Migration* & *Seeder***,** *Migration* digunakan untuk mengelola struktur *database* dengan *version control*, sementara *Seeder* digunakan untuk mengisi *database* dengan data awal.

1.5 Mengapa Laravel Cocok untuk Proyek *To-Do-List*?

Laravel adalah pilihan yang tepat untuk membangun aplikasi *To-Do-List* karena beberapa alasan berikut:

1. Struktur yang Terorganisir, Dengan konsep MVC (Model-View-Controller), kode aplikasi *To-Do-List* dapat diatur dengan rapi dan *modular*. Hal ini memudahkan pengelolaan dan pengembangan aplikasi, terutama saat menambahkan fitur baru.
2. Pengembangan Cepat, Laravel menyediakan fitur-fitur siap pakai seperti *migration*, *seeder*, dan *Eloquent ORM*, yang mempercepat proses pengembangan. Misalnya, dengan *Eloquent ORM*, Anda dapat mengelola data tabel *database* seperti mengelola objek *PHP*.
3. Skalabilitas, Laravel memungkinkan aplikasi *To-Do-List* dikembangkan lebih lanjut dengan mudah. Misalnya, Anda dapat menambahkan fitur baru seperti notifikasi, integrasi dengan layanan pihak ketiga, atau bahkan mengubahnya menjadi aplikasi yang lebih kompleks.
4. Keamanan yang Terjamin, Laravel menyediakan fitur keamanan bawaan seperti proteksi terhadap *SQL injection*, *CSRF (Cross-Site Request Forgery)*, dan *XSS (Cross-Site Scripting)*. Hal ini membuat aplikasi *To-Do-List* lebih aman dari serangan siber.

BAB 2
PERSIAPAN LINGKUNGAN PENGEMBANGAN

Sebelum memulai pengembangan aplikasi dengan Laravel, langkah pertama yang harus dilakukan adalah menyiapkan lingkungan pengembangan yang mendukung. Bab ini akan memandu Anda melalui proses instalasi dan konfigurasi alat-alat penting seperti XAMPP, *MySQL*, *Composer*, dan Laravel. Selain itu, Anda juga akan mempelajari struktur folder Laravel dan cara mengonfigurasi *database MySQL* agar siap digunakan dalam proyek *To-Do-List*. Dengan memahami materi ini, Anda akan memiliki fondasi yang kuat untuk memulai pengembangan aplikasi web menggunakan Laravel.

2.1 Instalasi XAMPP dan Menjalankan *MySQL*

Sebelum membangun aplikasi *To-Do-List* dengan Laravel, langkah pertama yang perlu di lakukan adalah menyiapkan lingkungan pengembangan yang mendukung. Salah satu alat yang di gunakan adalah XAMPP, sebuah perangkat lunak yang menggabungkan *Apache* sebagai web *server*, *MySQL* sebagai *database*, dan *PHP* sebagai bahasa pemrograman utama.

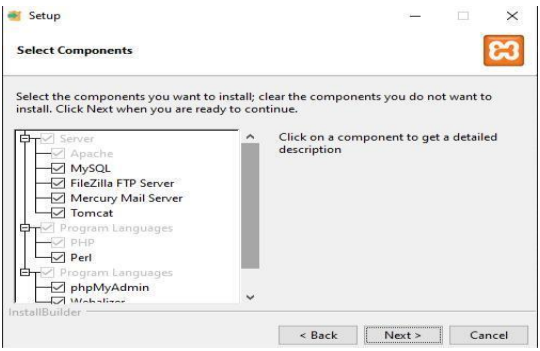
1. Unduh XAMPP dari situs resminya https://www.apachefriends.org.

Gambar 2. 1 Tutorial Install XAMPP 1

2. Pada halaman utama, terdapat opsi mengunduh XAMPP untuk berbagai sistem operasi seperti *Windows*, *Linux*, dan *macOS*. Pilih versi yang sesuai dengan sistem operasi yang digunakan.
3. Lalu, klik tombol unduhan versi yang dipilih, unduhan akan dimulai secara otomatis.
4. Setelah unduhan selesai, temukan *file installer* XAMPP dalam folder sistem.
5. Jalankan *file* instalasi, lalu ikuti petunjuk di layar.
6. Jika sudah masuk ke tampilan *setup* XAMPP seperti gambar dibawah ini, klik ***Next***.

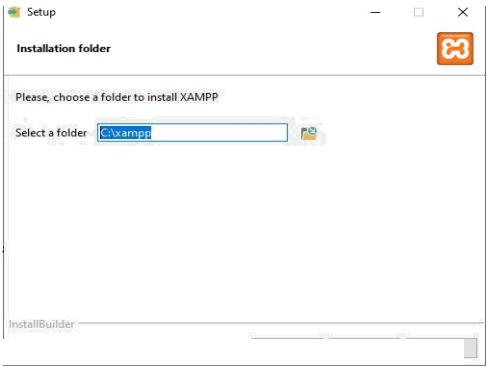
Gambar 2. 2 Tutorial Install XAMPP 2

7. Kemudian akan muncul pilihan mengenai komponen mana dari XAMPP yang ingin di *install* atau tidak, ikuti seperti dibawah ini dan ***Next***.



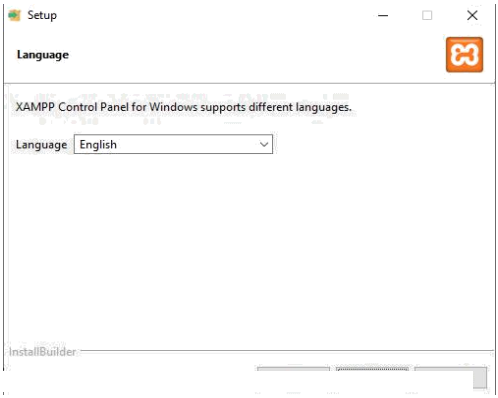
Gambar 2. 3 Totorial Instal XAMPP 3

8. Setelah itu, silahkan pilih folder tujuan dimana XAMPP yang ingin kita *install*. Jika tidak ingin dirubah maka langsung klik ***Next*** saja.



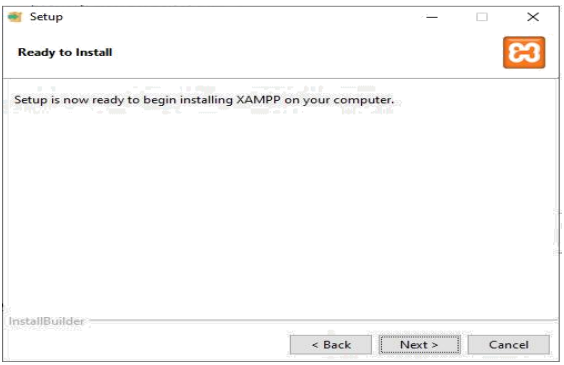
Gambar 2. 4 Tutorial Install XAMPP 4

9. Selanjutnya, kita diminta untuk memilih bahasa yang ingin kita gunakan, jika tidak ingin dirubah maka langsung klik ***Next*** saja.



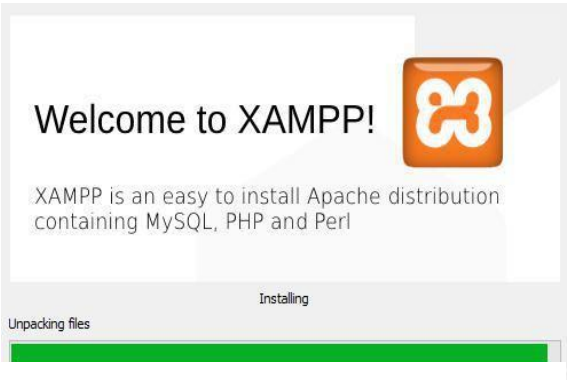
Gambar 2. 5 Tutorial Install XAMPP 5

10. Kemudian, akan muncul tampilan *Ready to Install* seperti gambar dibawah, klik ***Next***.



Gambar 2. 6 Tutorial Install XAMPP 6

11. Tunggu hingga proses instalasi selesai.



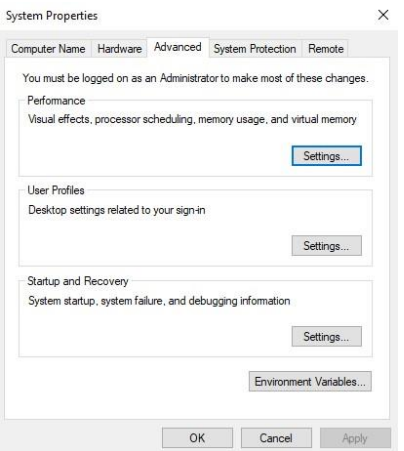
Gambar 2. 7 Tutorial Install XAMPP 7

12. Setelah instalasi selesai, klik ***Finish***.

Gambar 2. 8 Tutorial Install XAMPP 8

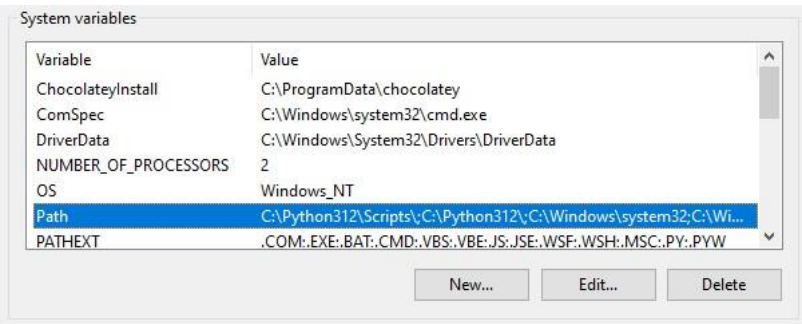
13. Setelah itu cari lokasi folder XAMPP dan salin untuk memasukan *path* ke *environment variables*.
    a. Biasanya: C:\xampp\php (untuk PHP)
    b. Bisa juga: C:\xampp\mysql\bin (untuk MySQL CLI)

14. Buka sistem *environtment variabel*
    a. Klik kanan *This PC* atau *My Computer* → pilih *Properties*
    b. Klik *Advanced system settings*
    c. Di *tab Advanced*, klik *Environment Variables*



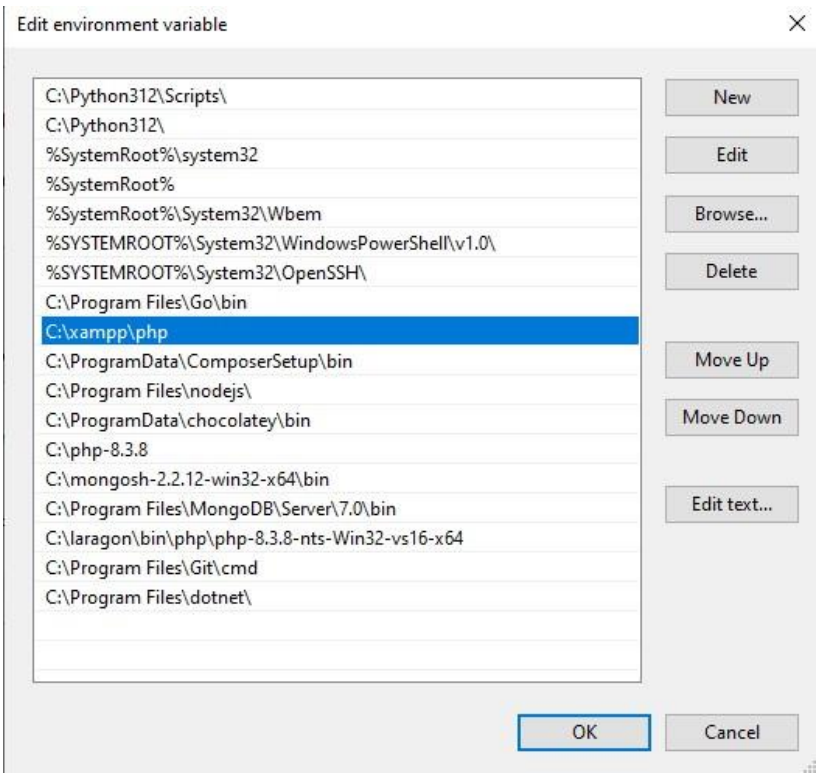
Gambar 2. 9 Tutorial Install XAMPP 9

15. Kemudian di bagian bawah *(System Variables)*, cari dan klik *Path* dan lalu klik Edit



Gambar 2. 10 Tutorial Install XAMPP 10

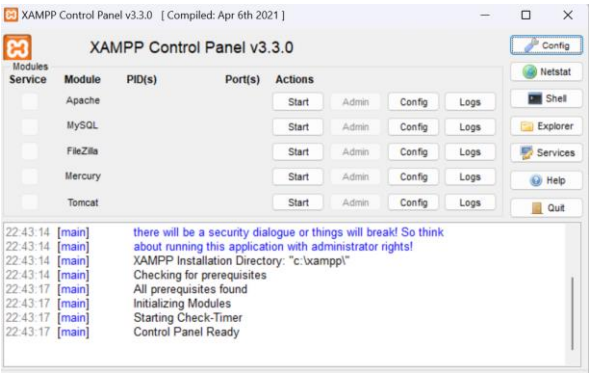
16. Klik *New*, lalu *paste path* XAMPP kamu



Gambar 2. 11 Tutorial Install XAMPP 11

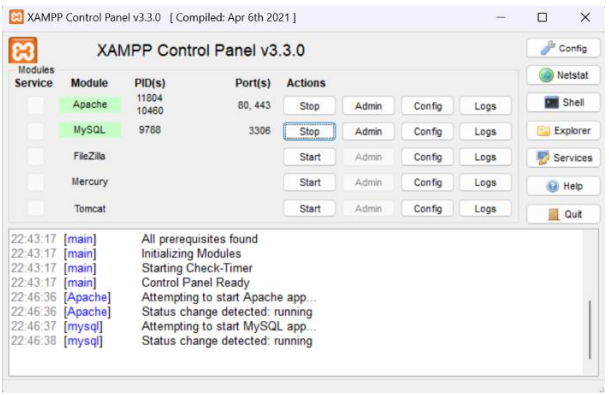
Setelah itu kita akan menajalankan *MySQL* di XAMPP.

1. Buka aplikasi XAMPP yang telah Anda *install*. Akan muncul tampilan seperti ini.



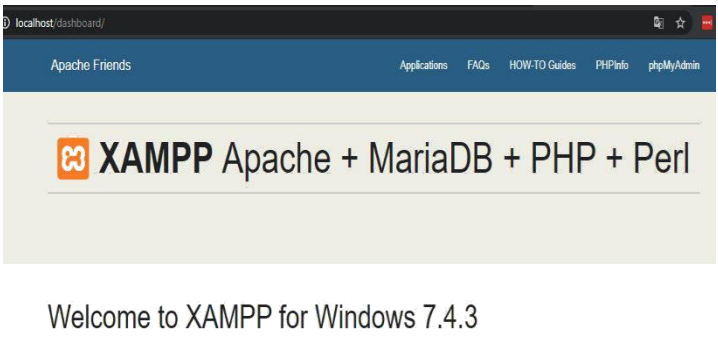
Gambar 2. 12 Tutorial Install XAMPP 12

2. Lalu klik *Start* pada *module Apache* dan *MySQL*



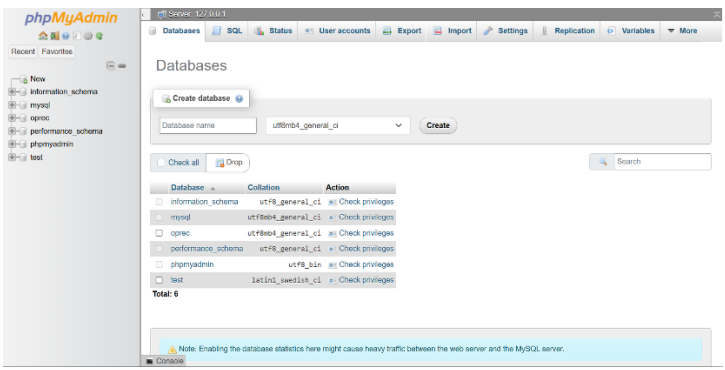
Gambar 2. 13 Tutorial Install XAMPP 13

3. Akses *localhost* menggunakan *link* berikut : http://localhost



Gambar 2. 14 Akses XAMPP

4. Buka *phpMyAdmin* melalui *browser* dengan mengakses http://localhost/phpmyadmin/.



Gambar 2. 15 Tampilan phpMyAdmin

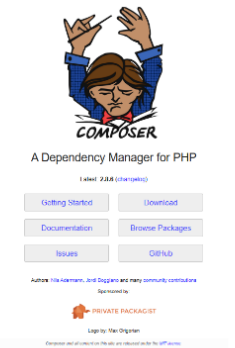
Setelah XAMPP dan *MySQL* berjalan, kita bisa lanjut ke tahap berikutnya, yaitu menginstal *Composer*.

2.2 Instalasi *Composer*

Composer adalah alat yang digunakan untuk mengelola dependensi dalam proyek berbasis *PHP*. Dengan *Composer*, kita bisa meng-*install* dan meng-*update* berbagai pustaka *PHP*, termasuk Laravel.

Langkah-langkah dalam instalasi *Composer*.

1. Unduh *Composer* dari situs resminya https://getcomposer.org/.



Gambar 2. 16 Tampilan Website untuk Download Composer

2. Jalankan *file* instalasi dan ikuti petunjuk yang di berikan. Pastikan *Composer* terhubung dengan *PHP* di XAMPP (biasanya ada di c:\xampp\php\php.exe).
3. Setelah instalasi selesai, buka *Command Prompt* (cmd) dan jalankan berikut untuk memastikan *Composer* sudah terpasang.

Tabel 2. 1 melihat versi composer

```
composer -V
```

Jika instalasi berhasil, akan muncul informasi versi *Composer* yang digunakan.

Dengan *Composer* yang sudah terpasang, kini kita siap untuk menginstal laravel.

2.3 Instalasi Laravel

Laravel adalah *framework PHP* yang membantu dalam membangun aplikasi web secara lebih cepat dan terstruktur. Kita akan menginstalnya menggunakan *Composer*.

1. Buka *Command Prompt* (cmd) dan arahkan ke folder tempat proyek akan disimpan. Misalnya, jika ingin menyimpan proyek di c:\xampp\htdocs, jalankan perintah.

Tabel 2. 2 simpan proyek di folder htdocs

```
cd C:\xampp\htdocs
```

2. Jalankan perintah berikut untuk menginstall laravel dan membuat proyek baru bernama *todo_list*.

Tabel 2. 3 install laravel

```
composer create-project --prefer-dist laravel/laravel
todo_list
```

3. Setelah proses instalasi selesai, masuk ke folder proyek.

Tabel 2. 4 masuk folder proyek

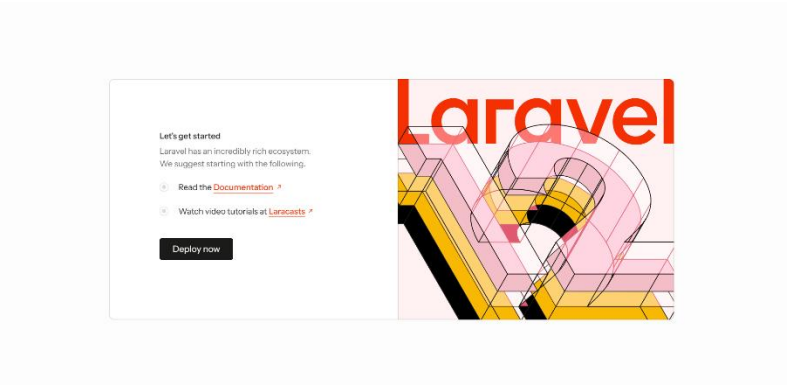
```
cd todo_list
```

4. Jalankan Laravel dengan perintah berikut.

Tabel 2. 5 perintah menjalankan laravel

```
php artisan serve
```

Jika berhasil, Laravel akan berjalan di http://127.0.0.1:8000, dan halaman awalnya bisa diakses melalui browser.



Gambar 2. 17 Tampilan Home Larave

Sampai tahap ini, Laravel sudah siap digunakan. Selanjutnya, kita akan membahas struktur folder dalam Laravel.

2.4 Struktur Folder dalam Laravel

Sebelum mulai mengembangkan aplikasi *To-Do-List*, penting untuk memahami struktur proyek Laravel. Setelah instalasi Laravel, Anda akan memiliki struktur direktori dan *file* seperti berikut:

Tabel 2. 6 struktur folder laravel

```
LARA-TODOLIST
├── app
│   ├── Http
│   │   ├── Controllers
│   │   │   ├── AuthController.php
│   │   │   ├── Controller.php
│   │   │   ├── DashboardController.php
│   │   │   ├── KelolaUserController.php
│   │   │   ├── TaskController.php
│   │   ├── Middleware
│   │   │   ├── CheckRole.php
│   ├── Models
│   │   ├── Role.php
│   │   ├── Task.php
│   │   ├── User.php
```

```
├── bootstrap
├── config
├── database
│   ├── factories
│   │   ├── UserFactory.php
│   ├── migrations
│   │   ├── 0001_01_01_000000_create_users_table.php
│   │   ├── 0001_01_01_000001_create_cache_table.php
│   │   ├── 0001_01_01_000002_create_jobs_table.php
│   │   ├── 2025_02_27_205018_create_tasks_table.php
│   │   ├── 2025_02_27_205449_create_roles_table.php
│   ├── seeders
│   │   ├── DatabaseSeeder.php
│   │   ├── TaskSeeder.php
│   │   ├── UserSeeder.php
├── public
│   ├── assets
│   ├── forms
│   ├── favicon.ico
│   ├── index.php
│   ├── robots.txt
├── resources
│   ├── css
│   │   ├── app.css
│   ├── js
│   │   ├── app.js
│   │   ├── bootstrap.js
│   ├── views
│   │   ├── auth
│   │   │   ├── login.blade.php
│   │   │   ├── register.blade.php
│   │   ├── dashboard
│   │   │   ├── kelolauser
│   │   │   │   ├── create.blade.php
│   │   │   │   ├── edit.blade.php
│   │   │   │   ├── index.blade.php
│   │   ├── layouts
│   │   │   ├── app.blade.php
│   │   │   ├── header.blade.php
```

```
│   │   │   ├── main.blade.php
│   │   │   ├── sidebar.blade.php
│   │   │   ├── task.blade.php
│   │   ├── user
│   │   │   ├── tasks.blade.php
│   │   ├── welcome.blade.php
├── routes
│   ├── console.php
│   ├── web.php
├── storage
├── tests
├── vendor
├── .editorconfig
├── .env
├── .env.example
├── .gitattributes
├── .gitignore
├── artisan
├── composer.json
├── composer.lock
├── package.json
├── phpunit.xml
├── README.md
├── vite.config.js
```

Struktur proyek di atas mencerminkan bagaimana Laravel mengorganisir berbagai komponen utama dalam aplikasi.

1. ***Folder app/*** berisi logika bisnis utama aplikasi, termasuk ***controller***, **middleware**, dan **model**.
2. ***Folder database/*** menyimpan *file* yang berhubungan dengan basis data, seperti migrasi, *factory*, dan *seeder*.
3. ***Folder resources/*** digunakan untuk menyimpan file tampilan seperti ***Blade template***, ***CSS***, dan ***JavaScript***.
4. ***Folder routes/*** berisi file *routing* utama, yaitu web.php dan console.php, yang menentukan bagaimana permintaan *HTTP* akan diproses.
5. **Folder public/** berisi *file* yang dapat diakses langsung oleh pengguna, seperti index.php, favicon, dan aset statis.

Memahami struktur proyek ini akan sangat membantu dalam proses pengembangan, karena setiap *file* dan folder memiliki fungsinya masing-masing.

2.5 Konfigurasi *Database MySQL* di Laravel

Agar laravel bisa berkomunikasi dengan *database MySQL* yang telah dibuat di XAMPP, kita perlu mengkonfigurasi file .env

1. Buka *file* .env yang ada di dalam folder proyek Laravel *(todo_list)*.
2. Cari bagian konfigurasi *database* dan ubah sesuai dengan pengaturan *MySQL* di XAMPP.

Tabel 2. 7 koneksi ke database

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

(Secara default, usernameMySQL di XAMPP adalah root, dan password kosong.)

3. Isi Lengkap File .env
   Berikut adalah isi lengkap file .env yang digunakan dalam proyek ini:

Tabel 2. 8 isi file .env

```
APP_NAME=laravel
APP_ENV=local
APP_KEY=base64:/OduXlcJKHxPywrK6m0wjZbqi/I6HV/bPgk6cllL0bs
=
APP_DEBUG=true
APP_URL=http://localhost
APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US
APP_MAINTENANCE_DRIVER=file
PHP_CLI_SERVER_WORKERS=4
BCRYPT_ROUNDS=12
```

```
LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=lara_todolist
DB_USERNAME=root
DB_PASSWORD=
SESSION_DRIVER=database
SESSION_LIFETIME=120
SESSION_ENCRYPT=false
SESSION_PATH=/
SESSION_DOMAIN=null
BROADCAST_CONNECTION=log
FILESYSTEM_DISK=local
QUEUE_CONNECTION=database
CACHE_STORE=database
MEMCACHED_HOST=127.0.0.1
REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
MAIL_MAILER=log
MAIL_SCHEME=null
MAIL_HOST=127.0.0.1
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false
VITE_APP_NAME="${APP_NAME}"
```

2.6 Menentukan *Routing* dalam Laravel

Laravel menggunakan sistem *routing* untuk menangani setiap permintaan *HTTP* dan menentukan respons yang akan dikirimkan ke pengguna. *Routing* didefinisikan dalam file routes/web.php, yang digunakan untuk menangani rute berbasis *web* (HTTP). Berikut adalah isi file routes/web.php dalam proyek ini:

Tabel 2. 9 kode routes.php

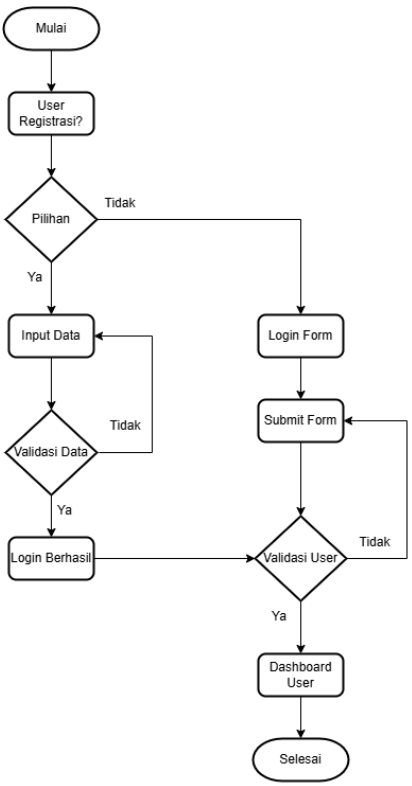
```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TaskController;
use App\Http\Controllers\AuthController;
use App\Http\Controllers\DashboardController;
use App\Http\Controllers\KelolaUserController;
use App\Http\Middleware\CheckRole;
Route::get('/', [AuthController::class, 'showLoginForm'])-
>middleware('guest');
Route::get('/login', [AuthController::class, 'showLoginForm'])->name('login')-
>middleware('guest');
Route::post('/login', [AuthController::class, 'login']);
Route::get('/register', [AuthController::class, 'showRegisterForm'])-
>name('register')->middleware('guest');
Route::post('/register', [AuthController::class, 'register'])-
>name('create.user');
Route::middleware('auth')->group(function () {
   Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
   Route::prefix('dashboard')->group(function () {
      Route::get('/', [DashboardController::class, 'index'])-
>name('dashboard')->middleware(CheckRole::class);
      Route::resource('users', KelolaUserController::class);
   });
   Route::prefix('tasks')->group(function () {
      Route::get('/', [TaskController::class, 'index'])->name('tasks.index')-
>middleware(CheckRole::class);
      Route::post('/store', [TaskController::class, 'store'])-
>name('tasks.store');
      Route::get('/{task}', [TaskController::class, 'show'])-
>name('tasks.show');
```

```
    Route::put('/{task}', [TaskController::class, 'update'])-
>name('tasks.update');
    Route::delete('/{task}', [TaskController::class, 'destroy'])-
>name('tasks.destroy');
    Route::post('/{task}/toggle', [TaskController::class, 'toggleStatus'])-
>name('tasks.toggle');
  });
});
```

2.7 Perancangan Sistem

Bab ini juga membahas perancangan sistem yang digunakan dalam pengembangan aplikasi *To-Do-List* berbasis *web*. Perancangan ini mencakup diagram alur proses yang menggambarkan bagaimana pengguna berinteraksi dengan sistem, baik sebagai pengguna biasa maupun sebagai admin. Dengan adanya diagram ini, pengembang dapat memahami alur kerja sistem sebelum masuk ke tahap implementasi.

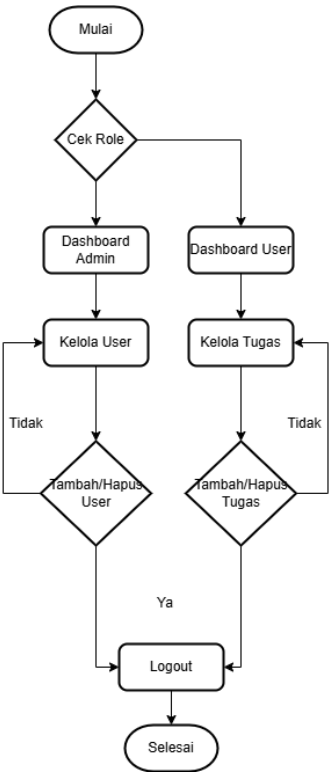
Flowchart Registrasi dan *Login*



Gambar 2. 18 Flowchart Registrasi & Login

Diagram alur ini menjelaskan proses registrasi dan *login* pengguna dalam sistem. Jika pengguna memilih registrasi, mereka harus mengisi data yang diperlukan, lalu sistem akan memvalidasi input tersebut. Jika valid, pengguna bisa langsung *login*, tetapi jika tidak, mereka harus mengisi ulang data yang benar. Jika pengguna langsung menuju *login*, mereka akan memasukkan kredensial yang kemudian divalidasi oleh sistem. Jika benar, mereka akan diarahkan ke *dashboard*, tetapi jika salah, mereka harus mencoba lagi. Diagram ini menekankan pentingnya validasi untuk menjaga keamanan dan kelancaran sistem.

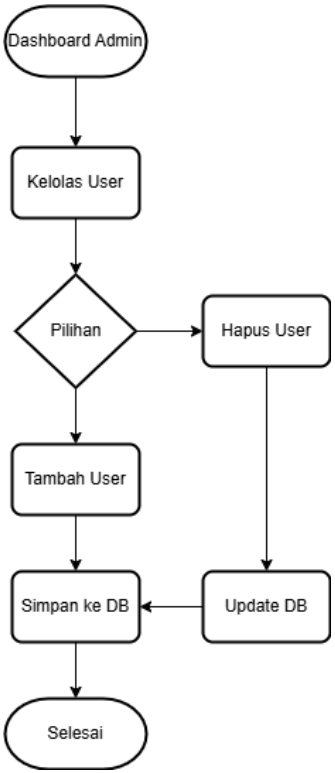
Flowchart Dashboard (Admin & User)



Gambar 2. 19 Flowchart Dashboard

Diagram alur ini menunjukkan bagaimana sistem membagi peran antara admin dan pengguna biasa. Setelah *login*, sistem akan mengecek peran pengguna. Jika sebagai admin, mereka diarahkan ke *dashboard* admin untuk mengelola pengguna lain, sedangkan pengguna biasa masuk ke *dashboard* mereka untuk mengelola tugas. Keduanya bisa menambah atau menghapus pengguna/tugas sesuai hak aksesnya. Setelah selesai, mereka dapat *logout*, dan alur berakhir. Diagram ini menggambarkan bagaimana sistem mengatur hak akses dan fitur berdasarkan peran pengguna.

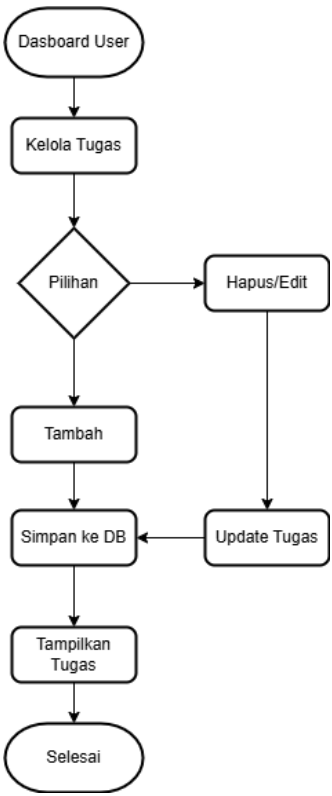
Flowchart Manajemen Pengguna (Admin)



Gambar 2. 20 Flowchart Manajemen Pengguna

Diagram alur ini menjelaskan bagaimana admin mengelola pengguna dalam sistem. Proses dimulai dari *dashboard* admin, di mana mereka bisa memilih untuk menambah atau menghapus pengguna. Jika menambah, admin memasukkan informasi pengguna baru, lalu data tersebut disimpan ke *database*. Jika menghapus, admin memilih pengguna yang ingin dihapus, lalu data diperbarui di *database*. Setelah proses selesai, alur berakhir. Diagram ini menggambarkan bagaimana admin dapat mengelola pengguna dengan mudah melalui dua fungsi utama, yaitu penambahan dan penghapusan data.

Flowchart Manajemen Tugas (User)



Gambar 2. 21 Flowchart Manajemen Tugas

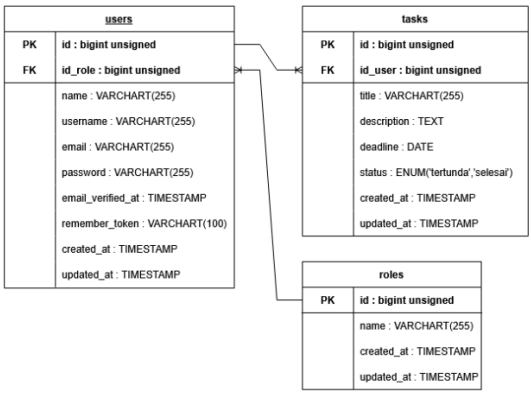
Diagram alur ini menjelaskan bagaimana pengguna mengelola tugas dalam sistem. Proses dimulai dari *dashboard* pengguna, di mana mereka bisa memilih untuk menambah, meng-*edit*, atau menghapus tugas. Jika menambah tugas, pengguna memasukkan informasi tugas baru yang kemudian disimpan ke *database*. Jika meng-*edit* atau menghapus tugas, pengguna memilih tugas yang ingin diubah, lalu sistem memperbarui data di *database*. Setelah perubahan selesai, tugas akan ditampilkan kepada pengguna. Alur berakhir setelah tugas berhasil ditampilkan. Diagram ini menggambarkan bagaimana pengguna dapat mengelola tugas dengan mudah melalui fungsi penambahan, peng-*edit*-an, dan penghapusan.

BAB 3
DATABASE DAN MIGRATION (XAMPP MySQL)

Pada bab ini kita akan membuat dan mengelola database menggunakan Laravel *Migration* dengan XAMPP *MySQL*. Dengan memanfaatkan fitur *migration*, kalian dapat dengan mudah mendefinisikan, mengubah, dan mengelola struktur *database* secara otomatis melalui kode program, tanpa perlu melakukan perubahan manual di *phpMyAdmin*. Selain itu, pembahasan mencakup pembuatan *migration* untuk tabel-tabel penting, menjalankan proses *migration* dalam lingkungan XAMPP, serta mengisi data awal menggunakan *seeder*.

3.1 Desain Database

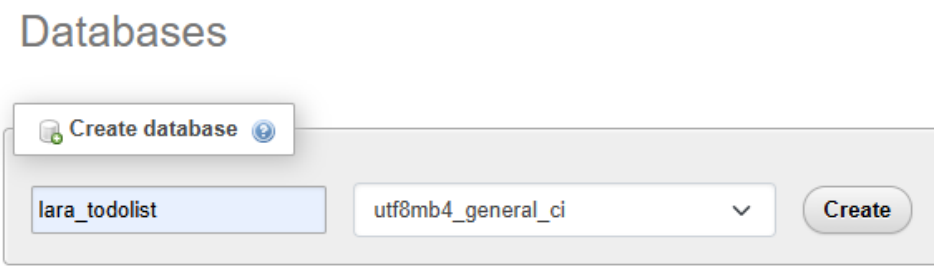
Dalam aplikasi To-Do List ini, kita menggunakan tiga tabel utama untuk mengelola pengguna *(users)*, tugas *(tasks)*, dan peran pengguna *(roles)*. Gambar diagram berikut menunjukkan hubungan antara ketiga tabel tersebut:



Gambar 3. 1 Desain Database

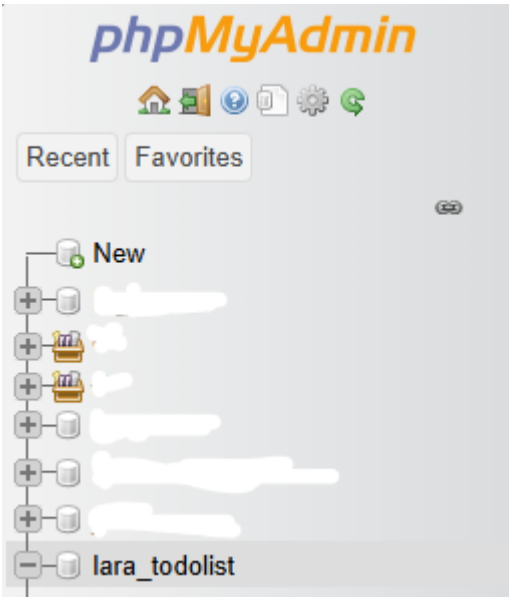
3.2 Membuat *Database* di *phpMyAdmin*

Pertama tama kita akan membuat *database* pada *phpMyAdmin*, klik bagian *"Database"* dan isi nama *database* dengan "lara_todolist" dan klik *create*.



Gambar 3. 2 Membuat Database

Jika berhasil maka akan tampil *database*-nya di samping *phpMyAdmin* seperti gambar berikut



Gambar 3. 3 Tampilan Jika Berhasil Membuat Database

3.3 Membuat *Migration*

Setelah berhasil membuat database kita akan migrasi tabel-tabel penting, yaitu tabel *users*, *roles*, dan *tasks*. Untuk melakukan migrasi, kita akan menjalankan 2 kode perintah berikut pada terminal VS Code.

Migration Tasks:

Tabel 3. 1 perintah membuat tabel database tasks

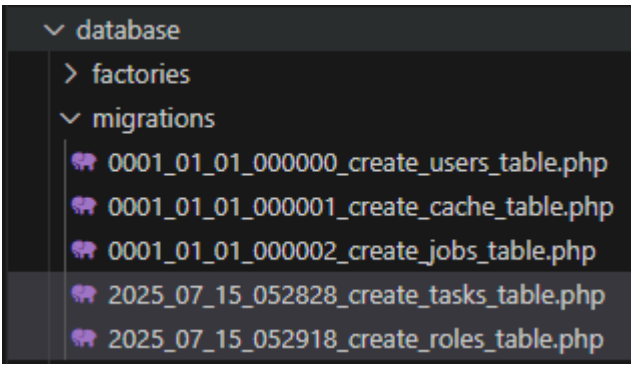
```
php artisan make:migration create_tasks_table
```

Migration Roles:

Tabel 3. 2 perintah membuat tabel database roles

```
php artisan make:migration create_roles_table
```

Jika berhali maka akan muncul *file* seperti berikut pada VSCode kalian.



Gambar 3. 4 Struktur Migrations

Selanjutnya kita akan mengisi codingan *migration* untuk *users, tasks,* dan *roles*.

Code Migration Users:

Tabel 3. 3 kode migrasi users

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
   public function up(): void
   {
      Schema::create('users', function (Blueprint $table) {
         $table->id();
         $table->foreignId('id_role');
         $table->string('nama');
         $table->string('username');
         $table->string('email')->unique();
         $table->string('password');
         $table->timestamp('email_verified_at')->nullable();
         $table->rememberToken();
         $table->timestamps();
      });
      Schema::create('password_reset_tokens', function (Blueprint $table) {
         $table->string('email')->primary();
         $table->string('token');
         $table->timestamp('created_at')->nullable();
      });
      Schema::create('sessions', function (Blueprint $table) {
         $table->string('id')->primary();
         $table->foreignId('user_id')->nullable()->index();
         $table->string('ip_address', 45)->nullable();
         $table->text('user_agent')->nullable();
         $table->longText('payload');
         $table->integer('last_activity')->index();
      });
   }
   public function down(): void
   {
      Schema::dropIfExists('users');
```

```
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

Code Migration Roles:

Tabel 3. 4 kode migrasi roles

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('nama');
            $table->timestamps();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('roles');
    }
};
```

Code Migration Tasks:

Tabel 3. 5 kode migrasi tasks

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
```

```
{
   public function up(): void
   {
      Schema::create('tasks', function (Blueprint $table) {
       $table->id();
       $table->foreignId('id_user');
       $table->string('judul');
       $table->text('deskripsi');
       $table->date('deadline');
       $table->enum('status', ['tertunda', 'selesai'])->default('tertunda');
       $table->timestamps();
      });
   }
   public function down(): void
   {
      Schema::dropIfExists('tugas');
   }
};
```

3.4 Menjalankan *Migration* di XAMPP

Selanjutnya kita akan menjalan *migration* untuk tabel-tabel yang sudah kita buat, dengan menjalankan perintah berikut di terminal VS Code.

Tabel 3. 6 perintah migrasi

```
php artisan migrate
```

Jika terjadi *error "Access Denied"* saat menjalankan perintah diatas, maka sebaiknya kalian cek Kembali *username* atau *password database* di *file ".env".*

3.5 Migrasi Bawaan Laravel

Pada *folder database/migrations*, terdapat beberapa *file* migrasi bawaan dari Laravel, yaitu:

- 0001_01_01_000000_create_users_table.php
- 0001_01_01_000001_create_cache_table.php
- 0001_01_01_000002_create_jobs_table.php

File-file ini otomatis disediakan oleh Laravel saat pertama kali proyek dibuat dan berfungsi untuk menyiapkan tabel-tabel dasar *seperti users, cache,* dan *jobs.*

BAB 4
AUTENTIKASI PENGGUNA
(Login & Register)

Pada bab ini kita akan membuat fitur autentikasi pengguna, tujuannya untuk membuat sistem *login* dan *register* dengan *role-based access*, salah satu fungsinya agar tidak sembarangan pengguna bisa mengakses *dashboard* admin.

4.1 Membuat *Model User* dan *Model Role*

Model dalam Laravel digunakan untuk berinteraksi dengan *database*. *Model User* akan digunakan untuk merepresentasikan tabel *users* dan *model Role* akan digunakan untuk merepresentasikan tabel *roles*.

1. Gunakan *file model User*:

Model User sudah disediakan secara default oleh Laravel dan digunakan untuk merepresentasikan tabel *users*.

Tabel 4. 1 kode didalam model user

```
<?php
namespace App\Models;
// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
// Model User untuk merepresentasikan pengguna sistem
class User extends Authenticatable
{
   /** @use HasFactory<\Database\Factories\UserFactory> */
   use HasFactory, Notifiable;
   // Kolom yang dapat diisi secara massal
   protected $fillable = [
```

```
        'nama',     // Nama lengkap user
        'username',  // Username untuk login
        'email',    // Email user
        'password',  // Password terenkripsi
        'id_role',   // Role/peran user
    ];
    /**
     * Daftar atribut yang harus disembunyikan saat model dikonversi ke
array/JSON
     * Melindungi data sensitif seperti password
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
    /**
     * Daftar atribut yang harus di-cast ke tipe data tertentu
     * Mengkonversi kolom database ke tipe data yang sesuai
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
    /**
     * Relasi ke tabel roles
     */
    public function role()
    {
        return $this->belongsTo(Role::class, 'id_role');
    }
}
```

2. Gunakan perintah *Artisan* berikut untuk membuat *model Role*:

Tabel 4. 2 perintah membuat model role

```
php artisan make:model Role
```

Setelah menjalankan perintah di atas, Laravel akan menghasilkan *file* Role.php yang berisi struktur dasar *model*. Berikut ini adalah kode yang telah merepresentasikan dari tabel *roles* nya:

Tabel 4. 3 kode didalam model role

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
// Model Role untuk merepresentasikan peran/role user
class Role extends Model
{
   // Hanya kolom nama yang bisa diisi massal
   protected $fillable = ['nama'];
   // Relasi one-to-many dengan User
   public function users()
   {
      return $this->hasMany(User::class, 'id_role');
   }
}
```

4.2 Membuat *Controller* Untuk *Login* dan *Register*

Selanjutnya, kita akan membuat file AuthController.php di dalam *folder* app/Http/Controllers. *File* ini akan digunakan untuk menangani logika autentikasi, seperti *form register*, proses *register*, dan nantinya juga bisa dikembangkan untuk *login* dan *logout*.

Untuk membuat *file controller* ini, gunakan perintah *Artisan* berikut:

Tabel 4. 4 perintah membuat AuthController

```
php artisan make:controller AuthController
```

Setelah perintah dijalankan, Laravel akan menghasilkan *file* AuthController.php, dan kita dapat menuliskan fungsi-fungsi seperti berikut:

Tabel 4. 5 kode didalam AuthController

```
<?php
namespace App\Http\Controllers;
use App\Models\User;
use App\Models\Role;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Auth;
class AuthController extends Controller
{
   public function showRegisterForm()
   {
      return view('auth.register');
   }
   public function register(Request $request)
   {
      $validatedData = $request->validate([
         'nama' => 'required|max:255',
         'username' => [
            'required',
            'min:3',
            'max:255',
            'unique:users'
         ],
         'email' => 'required|email|unique:users',
         'password' => 'required|min:6|max:255|confirmed'
      ], [
         // Pesan error kustom untuk setiap validasi
         'nama.required' => 'Nama harus diisi',
         'username.required' => 'Username harus diisi',
         'username.unique' => 'Username sudah digunakan',
         'email.required' => 'Email harus diisi',
         'email.email' => 'Format email tidak valid',
         'email.unique' => 'Email sudah terdaftar',
         'password.required' => 'Password harus diisi',
         'password.min' => 'Password minimal 6 karakter',
```

```
            'password.confirmed' => 'Konfirmasi password tidak cocok'
        ]);
        $userRole = Role::where('nama', 'user')->first();
        if (!$userRole) {
            Role::create(['nama' => 'admin']);
            $userRole = Role::create(['nama' => 'user']);
        }
        $validatedData['id_role'] = $userRole->id;
        $validatedData['password'] = Hash::make($validatedData['password']);
        User::create($validatedData);
            return redirect('/login')->with('success', 'Registrasi Berhasil! Silahkan
Login');
    }
    public function showLoginForm()
    {
        return view('auth.login');
    }
    public function login(Request $request)
    {
        $credentials = $request->validate([
            'login' => 'required|string', // Bisa email atau username
            'password' => 'required|string'
        ], [
            'login.required' => 'Email atau username harus diisi',
            'password.required' => 'Password harus diisi'
        ]);
        $loginType = filter_var($credentials['login'], FILTER_VALIDATE_EMAIL) ?
'email' : 'username';
        $authCredentials = [
            $loginType => $credentials['login'],
            'password' => $credentials['password']
        ];
        if (Auth::attempt($authCredentials)) {
            $request->session()->regenerate();
            return Auth::user()->role->nama === 'admin'
                ? redirect()->route('dashboard')->with('success', 'Selamat datang
Admin!')
                : redirect()->route('tasks.index')->with('success', 'Login berhasil!');
        }
```

```
        return back()
            ->withInput($request->only('login'))
            ->withErrors([
                'login' => 'Email/Username atau password salah'
            ]);
    }
    public function logout(Request $request)
    {
        Auth::logout(); // Logout user
        $request->session()->invalidate(); // Invalidate session
        $request->session()->regenerateToken(); // Regenerate CSRF token
        return redirect('login')->with('success', 'Anda telah berhasil logout');
    }
}
```

4.3 Membuat *Layout* Utama (app.blade.php)

Agar tampilan halaman dalam aplikasi lebih konsisten dan mudah dikelola, kita akan membuat layout utama yang akan digunakan oleh berbagai halaman, termasuk halaman *login* dan *register*. Dengan menggunakan Blade *Template* Laravel, kita dapat membuat satu *template* utama yang bisa digunakan ulang, sehingga tidak perlu menulis ulang struktur *HTML* di setiap halaman.

1. Membuat *File* Layout Utama
   Buka *terminal* di dalam direktori proyek Laravel, lalu jalankan perintah berikut untuk membuat *file layout* utama:

   Tabel 4. 6 perintah membuat folder layouts

   ```
   mkdir -p resources/views/layouts
   ```

   Selanjutnya tulis perintah ini:

   Tabel 4. 7 perintah membuat file app.blade.php

   ```
   touch resources/views/layouts/app.blade.php
   ```

Perintah ini akan membuat file app.blade.php di dalam *folder* resources/views/layouts/. File ini akan menjadi *template* utama yang akan digunakan oleh halaman lainnya.

2. Menambahkan Kode *Layout* Utama
   Buka file resources/views/layouts/app.blade.php yang telah dibuat, lalu tambahkan kode berikut:

Tabel 4. 8 kode didalam app.blade.php

```
<!DOCTYPE html>
<html lang="en">
        <head>
          <meta charset="UTF-8">
          <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
          <title>{{ config('app.name', 'Todo List App') }}</title>
          <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/cs
        s/bootstrap.min.css" rel="stylesheet">
          <link rel="stylesheet"
        href="https://cdnjs.cloudflare.com/ajax/libs/font-
        awesome/6.5.1/css/all.min.css">
          <link rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap-
        icons@1.11.3/font/bootstrap-icons.min.css">
          <style>
            .todo{
              color: #6f42c1;
            }
            .btn-purple {
              position: relative;
              overflow: hidden;
              background-color: transparent;
              border: 2px solid #6f42c1;
              color: #6f42c1;
              transition: color 0.3s ease-in-out, border-color 0.3s
        ease-in-out;
            }
            .btn-purple::before {
```

```
    content: "";
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background-color: #6f42c1;
    transition: left 0.3s ease-in-out;
    z-index: 0;
  }
  .btn-purple:hover {
    color: white;
    border-color: #6f42c1;
  }
  .btn-purple:hover::before {
    left: 0;
  }
  .btn-purple span {
    position: relative;
    z-index: 1;
  }
  .btn-outline-purple {
    color: #6f42c1;
    border-color: #6f42c1;
  }
  .btn-outline-purple:hover {
    background-color: #6f42c1;
    color: white;
  }
  .form-control:focus {
    border-color: #6f42c1;
    box-shadow: 0 0 0 0.25rem rgb(255, 255, 255);
  }
  .link-purple {
    color:#6f42c1;
    text-decoration: none;
    position: relative;
  }
  .link-purple:after {
```

```
    content: '';
    position: absolute;
    width: 100%;
    height: 2px;
    bottom: -2px;
    left: 0;
    background-color: #6f42c1;
    transform: scaleX(0);
    transform-origin: right;
    transition: transform 0.3s ease;
}
.link-purple:hover {
    color: #6f42c1;
}
.link-purple:hover:after {
    transform: scaleX(1);
    transform-origin: left;
}
.feature-icon {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    background-color: #e9ecef;
    display: flex;
    align-items: center;
    justify-content: center;
    color: #6f42c1;
}
.auth-card {
    border: 1px solid rgba(111, 66, 193, 0.1);
    border-radius: 1rem;
}
.auth-icon {
    width: 64px;
    height: 64px;
    background-color: rgba(111, 66, 193, 0.1);
    border-radius: 50%;
    display: flex;
    align-items: center;
```

```
                justify-content: center;
                color: #6f42c1;
                margin: 0 auto 1.5rem;
            }
        </style>
    </head>
    <body class="bg-light">
        <div id="app">
            @yield('content')
        </div>
        <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/
    bootstrap.bundle.min.js"></script>
    </body>
</html>
```

3. Menggunakan *Layout* di Halaman Lain
   Setelah *layout* utama dibuat, halaman lain seperti *login* dan *register* dapat menggunakannya dengan cukup menambahkan:

Tabel 4. 9 kode pewarisan

```
@extends('layouts.app')
    @section('content')
    <!-- Isi halaman -->
@endsection
```

Dengan cara ini, setiap halaman hanya perlu menulis bagian kontennya saja, tanpa harus mengulang kembali kode *HTML* yang sama.

4.4 Membuat *Form Register*

Selanjutnya kita akan membuat *Form Register*, yang dimana *form* ini akan digunakan untuk proses registrasi pengguna, jika pengguna belum memiliki akun.

Buka *terminal* di dalam direktori proyek Laravel, lalu jalankan perintah berikut untuk membuat folder *auth*:

Tabel 4. 10 perintah membuat folder auth

```
mkdir -p resources/views/auth
```

lalu jalankan perintah berikut untuk membuat *file* register.blade.php:

Tabel 4. 11 perintah membuat register.blade.php

```
touch resources/views/auth/register.blade.php
```

Buka *file* resources/views/auth/register.blade.php yang telah dibuat, lalu tambahkan kode berikut:

Tabel 4. 12 kode didalam register.blade.php

```
@extends('layouts.app')
@section('content')
  <div class="min-vh-100 d-flex align-items-center justify-content-center py-
5">
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-12 col-md-6 col-lg-5">
          <div class="card auth-card shadow-sm">
            <div class="card-body p-4 p-md-5">
              <div class="text-center mb-4">

                <h2 class="todo fw-bold mb-4">To-Do-List</h2>
                <p class="text-muted">Buat akun dan mulai membuat
tugas baru</p>
              </div>

              <form method="POST" action="{{ route('create.user') }}">
                @csrf

                <div class="mb-3">
                  <div class="input-group">
                    <span class="input-group-text bg-light border-end-0">
                      <i class="bi bi-person"></i>
                    </span>
```

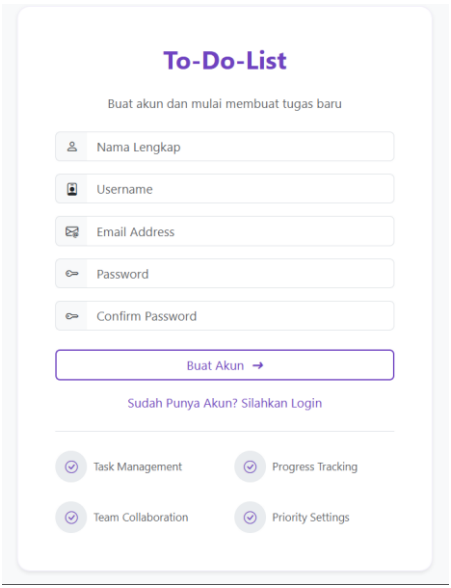
```
                        <input type="text" name="nama"
                            class="form-control border-start-0 @error('nama')
is-invalid @enderror"
                            placeholder="Nama Lengkap" value="{{ old('nama')
}}" required autocomplete="nama"
                            autofocus>
                        </div>
                        @error('nama')
                            <div class="invalid-feedback d-block">
                                {{ $message }}
                            </div>
                        @enderror
                    </div>

                    <div class="mb-3">
                        <div class="input-group">
                            <span class="input-group-text bg-light border-end-0">
                                <i class="bi bi-person-badge"></i>
                            </span>
                            <input type="text" name="username"
                                class="form-control border-start-0
@error('username') is-invalid @enderror"
                                placeholder="Username" value="{{ old('username')
}}" required autocomplete="username">
                        </div>
                        @error('username')
                            <div class="invalid-feedback d-block">
                                {{ $message }}
                            </div>
                        @enderror
                    </div>

                    <div class="mb-3">
                        <div class="input-group">
                            <span class="input-group-text bg-light border-end-0">
                                <i class="bi bi-envelope-at"></i>
                            </span>
                            <input type="email" name="email"
```

```
                            class="form-control border-start-0 @error('email')
is-invalid @enderror"
                            placeholder="Email Address" value="{{ old('email')
}}" required
                            autocomplete="email">
                        </div>
                        @error('email')
                            <div class="invalid-feedback d-block">
                                {{ $message }}
                            </div>
                        @enderror
                    </div>

                    <div class="mb-3">
                        <div class="input-group">
                            <span class="input-group-text bg-light border-end-0">
                                <i class="bi bi-key"></i>
                            </span>
                            <input type="password" name="password"
                                class="form-control border-start-0
@error('password') is-invalid @enderror"
                                placeholder="Password" required
autocomplete="new-password">
                        </div>
                        @error('password')
                            <div class="invalid-feedback d-block">
                                {{ $message }}
                            </div>
                        @enderror
                    </div>

                    <div class="mb-4">
                        <div class="input-group">
                            <span class="input-group-text bg-light border-end-0">
                                <i class="bi bi-key"></i>
                            </span>
                            <input type="password"
name="password_confirmation"
```

```
                        class="form-control border-start-0"
placeholder="Confirm Password" required
                        autocomplete="new-password">
                    </div>
                </div>

                <button type="submit" class="btn btn-purple w-100 mb-3
position-relative overflow-hidden">
                    <span class="d-flex align-items-center justify-content-
center">
                        Buat Akun
                        <i class="ms-2">➜</i>
                    </span>
                </button>

                <div class="text-center">
                    <a href="{{ route('login') }}" class="link-purple">
                        Sudah Punya Akun? Silahkan Login
                    </a>
                </div>

                <div class="mt-4 pt-4 border-top">
                    <div class="row g-4">
                        <div class="col-6">
                            <div class="d-flex align-items-center">
                                <div class="feature-icon me-2"><i class="bi bi-
check-circle"></i></div>
                                <span class="small text-muted">Task
Management</span>
                            </div>
                        </div>
                        <div class="col-6">
                            <div class="d-flex align-items-center">
                                <div class="feature-icon me-2"><i class="bi bi-
check-circle"></i></div>
                                <span class="small text-muted">Progress
Tracking</span>
                            </div>
                        </div>
```

```
                        <div class="col-6">
                          <div class="d-flex align-items-center">
                            <div class="feature-icon me-2"><i class="bi bi-
check-circle"></i></div>
                            <span class="small text-muted">Team
Collaboration</span>
                          </div>
                        </div>
                        <div class="col-6">
                          <div class="d-flex align-items-center">
                            <div class="feature-icon me-2"><i class="bi bi-
check-circle"></i></div>
                            <span class="small text-muted">Priority
Settings</span>
                          </div>
                        </div>
                      </div>
                    </div>
                  </form>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
@endsection
```

Setelah *form* register selesai dibuat dan diakses melalui *URL* http://127.0.0.1:8000/register, maka akan ditampilkan halaman seperti pada Gambar 4.1. Halaman ini berisi *form* registrasi yang memungkinkan pengguna untuk membuat akun baru dengan mengisi data seperti nama lengkap, *username, email address, password,* dan *konfirmasi password*.



Gambar 4. 1 Tampilan Form Register

4.5 Membuat *Form Login*

Berikutnya kita akan membuat *Form Login*, yang dimana *form* ini akan digunakan untuk proses masuk pengguna, jika pengguna sudah memiliki akun.

lalu jalankan perintah berikut untuk membuat file login.blade.php:

Tabel 4. 13 perintah membuat login.blade.php

```
touch resources/views/auth/login.blade.php
```

Buka file resources/views/auth/login.blade.php yang telah dibuat, lalu tambahkan kode berikut:

Tabel 4. 14 kode didalam login.blade.php

```
@extends('layouts.app')
@section('content')
  <div class="min-vh-100 d-flex align-items-center justify-content-center py-5">
    <div class="container">
```

```
        <div class="row justify-content-center">
          <div class="col-12 col-md-6 col-lg-5">
            <div class="card auth-card shadow-sm">
              <div class="card-body p-4 p-md-5">
                @if(session('success'))
                  <div class="alert alert-success alert-dismissible fade show"
role="alert">
                    {{ session('success') }}
                    <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>
                  </div>
                @endif

                <div class="text-center mb-4">
                  <h2 class="todo fw-bold mb-4">To-Do-List</h2>
                  <p class="text-muted">Masuk ke akun Anda</p>
                </div>


                <form method="POST" action="{{ route('login') }}">

                  @csrf

                  <div class="mb-3">
                    <div class="input-group">
                      <span class="input-group-text bg-light border-end-0">
                        <i class="bi bi-person"></i>
                      </span>
                      <input type="text" name="login"
                        class="form-control border-start-0 @error('login') is-
invalid @enderror"
                        placeholder="Email atau Username" value="{{
old('login') }}" required
                        autofocus>
                    </div>
                    @error('login')
                      <div class="invalid-feedback d-block">
                        {{ $message }}
                      </div>
```

```
                    @enderror
                </div>
                <div class="mb-4">
                    <div class="input-group">
                        <span class="input-group-text bg-light border-end-0">
                            <i class="bi bi-key"></i>
                        </span>
                        <input type="password" name="password"
                            class="form-control                    border-start-0
@error('password') is-invalid @enderror"
                            placeholder="Password" required>
                    </div>
                    @error('password')
                        <div class="invalid-feedback d-block">
                            {{ $message }}
                        </div>
                    @enderror
                </div>

                <button type="submit" class="btn btn-purple w-100 mb-3
position-relative overflow-hidden">
                    <span class="d-flex align-items-center justify-content-
center">
                        Masuk
                        <i class="ms-2">➜</i>
                    </span>
                </button>

                <div class="text-center">

                    <a href="{{ route('register') }}" class="link-purple">
                        Belum Punya Akun? Daftar Sekarang
                    </a>
                </div>

                <div class="mt-4 pt-4 border-top">
                    <div class="row g-4">
                        <div class="col-6">
                            <div class="d-flex align-items-center">
```

```
                        <div class="feature-icon me-2"><i class="bi bi-check-circle"></i></div>
                        <span class="small text-muted">Task Management</span>
                      </div>
                    </div>
                    <div class="col-6">
                      <div class="d-flex align-items-center">
                        <div class="feature-icon me-2"><i class="bi bi-check-circle"></i></div>
                        <span class="small text-muted">Progress Tracking</span>
                      </div>
                    </div>
                    <div class="col-6">
                      <div class="d-flex align-items-center">
                        <div class="feature-icon me-2"><i class="bi bi-check-circle"></i></div>
                        <span class="small text-muted">Team Collaboration</span>
                      </div>
                    </div>
                    <div class="col-6">
                      <div class="d-flex align-items-center">
                        <div class="feature-icon me-2"><i class="bi bi-check-circle"></i></div>
                        <span class="small text-muted">Priority Settings</span>
                      </div>
                    </div>
                  </div>
                </div>
              </form>
            </div>
          </div>
        </div>
      </div>
    </div>
```

```
@endsection
```

Form login digunakan agar pengguna yang sudah memiliki akun dapat masuk ke dalam aplikasi. Pada halaman *login*, pengguna diminta untuk mengisi *email* atau *username* serta *password*, kemudian menekan tombol Masuk untuk memulai sesi. Tampilan *form login* ditunjukkan pada Gambar 4.2.

Halaman *login* ini dapat diakses melalui dua *URL*, yaitu:

- http://127.0.0.1:8000/
- http://127.0.0.1:8000/login

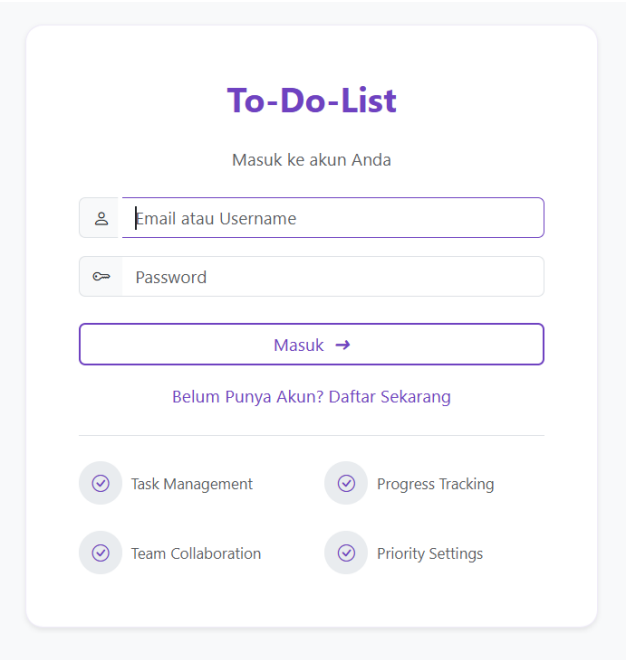
Kedua *URL* tersebut mengarah ke fungsi showLoginForm() pada AuthController, dan hanya dapat diakses oleh pengguna yang belum login karena dilindungi oleh *middleware guest*. Route ini memungkinkan pengguna diarahkan langsung ke halaman login saat pertama kali membuka aplikasi.

Saat *form* dikirimkan, data *login* akan dikirim melalui metode *POST* ke *URL*:

- http://127.0.0.1:8000/login

URL ini akan memproses autentikasi melalui fungsi login() pada AuthController.

Dengan alur ini, pengguna yang berhasil *login* akan diarahkan ke halaman *dashboard* atau halaman utama yang sesuai, sedangkan pengguna yang gagal *login* akan mendapatkan pesan error yang sesuai.



Gambar 4. 2 Tampilan Form Login

4.6 *Middleware* untuk *Role-Based Access*

Pada bagian ini, kita akan membuat sebuah *middleware* yang digunakan untuk membatasi akses pengguna berdasarkan *role* tertentu, misalnya hanya admin yang bisa mengakses halaman tertentu.

Untuk membuat *middleware* ini, kita akan menggunakan perintah Artisan berikut:

Tabel 4. 15 perintah membuat middleware checkrole

php artisan make:middleware CheckRole

Perintah ini akan menghasilkan *file* baru bernama CheckRole.php yang secara otomatis disimpan di dalam folder app/Http/Middleware/.

Middleware ini nantinya akan digunakan untuk mengecek role pengguna yang sedang login, dan menentukan apakah ia diizinkan mengakses halaman yang

diminta. *Middleware* ini juga dapat diregistrasikan di *file* Kernel.php dan digunakan pada *route* atau *controller* yang membutuhkan proteksi *role-based access.*

Berikut ini contoh penempatan *file* nya.

Tabel 4. 16 alamat folder checkrole.php

```
├── app/
    ├── http/
        ├── controller/
        ├── Middleware/
            ├── CheckRole.php
    ├── model/
    ├── profider/
```

Berikut ini kode untuk konfigurasi pada CheckRole.php.

Tabel 4. 17 kode didalam checkrole.php

```
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Http\Request;
class CheckRole
{
   public function handle(Request $request, Closure $next)
   {
      if (!$request->user()) {
         return redirect('login');
      }
      $userRole = $request->user()->role->nama;
      $intendedRoute = null;
      if ($userRole === 'admin') {
         $intendedRoute = 'dashboard';
      } elseif ($userRole === 'user') {
         $intendedRoute = 'tasks.index';
      }
      if ($intendedRoute && !$request->routeIs($intendedRoute)) {
         return redirect()->route($intendedRoute);
      }
```

```
        return $next($request);
    }
}
```

Setelah konfigurasi autentikasi selesai, aplikasi ini siap untuk pengembangan aplikasi berikutnya.

4.7 Membuat *Seeders* untuk *User* Awal

Agar mempermudah dalam proses pengujian fitur *login*, kita dapat menambahkan data *user* awal secara otomatis dengan menggunakan *Seeder*. *Seeder* ini akan membuat akun dengan peran admin dan *user* pada saat dijalankan.

1. Membuat File UserSeeder

   Untuk membuat seeders, kita akan menggunakan perintah Artisan berikut:

   Tabel 4. 18 peintah membuat user seeder

   ```
   php artisan make:seeder UserSeeder
   ```

   Perintah ini akan menghasilkan file baru bernama UserSeeder.php yang secara otomatis disimpan di dalam folder database/seeders. Setelah itu tambahkan kode berikut:

   Tabel 4. 19 kode didalam user seeders

   ```
   <?php
   namespace Database\Seeders;
   use App\Models\Role;
   use App\Models\User;
   use Illuminate\Database\Seeder;
   use Illuminate\Support\Facades\Hash;
   class UserSeeder extends Seeder
   {
      public function run(): void
   ```

```
    {
        // Buat role jika belum ada
        $adminRole = Role::firstOrCreate(['nama' => 'admin']);
        $userRole = Role::firstOrCreate(['nama' => 'user']);
        // Buat akun admin default
        User::create([
            'nama' => 'Administrator',
            'username' => 'admin',
            'email' => 'admin@gmail.com',
            'password' => Hash::make('password'),
            'id_role' => $adminRole->id
        ]);
        // Buat akun user default
        User::create([
            'nama' => 'Regular User',
            'username' => 'user',
            'email' => 'user@gmail.com',
            'password' => Hash::make('password'),
            'id_role' => $userRole->id
        ]);
        User::create([
            'nama' => 'ucup',
            'username' => 'ucup',
            'email' => 'ucup@gmail.com',
            'password' => Hash::make('password'),
            'id_role' => $userRole->id
        ]);
    }
}
```

2. Menjalankan *Seeder*

Setelah membuat UserSeeder, selanjutnya kita harus memanggilnya ke dalam *file* DatabaseSeeder agar fungsinya dapat dijalankan. Berikut kode yang ditambahkan:

Tabel 4. 20 kode didalam seeders

```
<?php
namespace Database\Seeders;
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
   public function run(): void
   {
      $this->call([
         UserSeeder::class,
         // TaskSeeder::class
      ]);
   }
}
```

Kemudian jalankan perintah berikut:

Tabel 4. 21 perintah menjalankan database seeders

```
php artisan db:seed
```

Setelah mengikuti seluruh tahapan di atas, maka akan dibuat 2 akun bawaan secara otomatis yaitu admin dan user.

d	id_role	nama	username	email	password
1	1	Administrator	admin	admin@gmail.com	$2y$12$ryv3c7g/K2feionUv/HgyuoB6DB6Nb0YuHCfV1MVkWb...
2	2	Regular User	user	user@gmail.com	$2y$12$cJVz342Hm0NQ49W5pg3CUev8WfyqMctMpe6w9UIDD0W...

Gambar 4. 3 User Hasil Seeders

4.8 Tahapan Proses Register dan Login

Untuk mempermudah pengguna agar memahami cara menggunakan fitur autentikasi yang telah dibuat, berikut adalah tahapan dalam mengisi form register dan login:

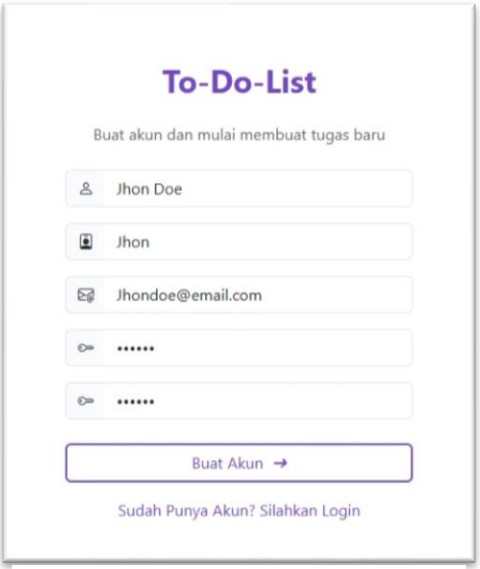
4.8.1 Tahapan Mengisi Form Register

1. Akses halaman registrasi melalui alamat:

Tabel 4. 22 url halaman register

http://127.0.0.1:8000/register

2. Isi data *form* registrasi.



Gambar 4. 4 Form Register

3. Klik tombol buat akun.
4. Jika data valid, pengguna akan diarahkan ke halaman *login* dengan notifikasi:

Tabel 4. 23 notifikasi setelah registrasi berhasil

Registrasi Berhasil! Silahkan Login

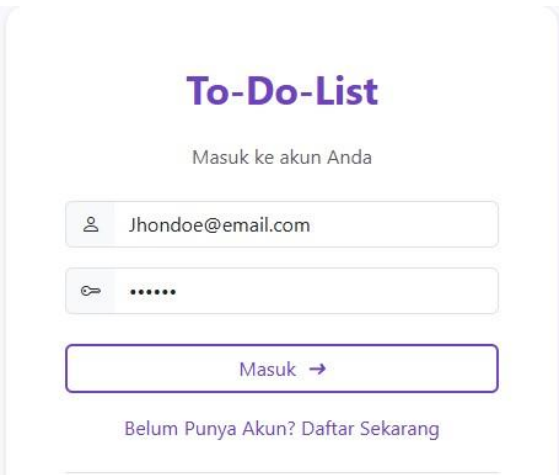
4.8.2 Tahapan Mengisi Form Login

1. Akses halaman *login* melalui:

Tabel 4. 24 url halaman login

http://127.0.0.1:8000/login atau http://127.0.0.1:8000/

2. Isi data pada *form login*:



Gambar 4. 5 Form Login

3. Klik tombol masuk.
4. Jika data benar, *user* dengan *role* admin akan di arahkan ke halaman /dashboard dan user dengan *role user* akan diarahkan ke halaman /task.
5. Jika gagal *login* maka akan muncul pesan *error* "Email/Username atau password salah".

Proses autentikasi merupakan tahap penting yang harus dilakukan setiap pengguna sebelum mengakses aplikasi *To-Do-List*. Melalui tahapan pengisian

form register dan *login* yang telah dijelaskan, sistem memastikan bahwa setiap pengguna yang masuk telah melalui proses validasi data dan diarahkan sesuai dengan peran (role) yang dimilikinya, baik sebagai admin maupun pengguna biasa.

BAB 5
HALAMAN *USER*
(To-Do-List / Tasks)

Bab ini akan membahas pembuatan halaman *To-Do-List* untuk pengguna, termasuk *routing*, *controller*, tampilan daftar tugas, fitur *CRUD (Create, Read, Update, Delete)*, serta filter dan pencarian tugas. Dengan menyelesaikan bab ini, pengguna akan memiliki halaman untuk mengelola tugas mereka dengan fitur yang lengkap dan mudah digunakan.

5.1 Membuat Model *Task*

Model dalam Laravel digunakan untuk berinteraksi dengan *database*. Model *Task* akan digunakan untuk merepresentasikan tabel *tasks* dan mengelola data tugas yang dibuat oleh pengguna. Gunakan perintah *Artisan* berikut untuk membuat model Task:

Tabel 5. 1 Membuat Model Task

```
php artisan make:model Task
```

Setelah menjalankan perintah di atas, Laravel akan menghasilkan *file* Task.php yang berisi struktur dasar model. File ini secara *default* hanya berisi kode berikut:

Tabel 5. 2 Kode Model Task

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Task extends Model
{
   use HasFactory;
}
```

Namun, agar model *Task* dapat berfungsi dengan baik dalam aplikasi *To-Do-List*, kita perlu menambahkan atribut yang dapat diisi (*fillable*), relasi ke *User*, serta metode untuk pencarian dan filter tugas. Buka *file* app/Models/Task.php dan tambahkan kode berikut:

Tabel 5. 3 Kode Model Task 2

```
<?php

namespace App\Models;

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

class Task extends Model
{
   use HasFactory;

   protected $fillable = [
      'judul',
      'deskripsi',
      'deadline',
      'status',
      'id_user'
   ];

   protected $casts = [
```

```
        'deadline' => 'datetime'
    ];

    protected $attributes = [
        'status' => 'tertunda'
    ];

    public function user()
    {
        return $this->belongsTo(User::class, 'id_user');
    }

    public function scopeSearch($query, $filters)
    {
        return $query
            ->when($filters['search'] ?? false, function($query, $search) {
                $query->where('judul', 'like', '%' . $search . '%');
            })
            ->when($filters['status'] ?? null, fn($query, $status) => $query-
>where('status', $status))
            ->where('id_user', auth()->id());
    }

    public function scopeCountByStatus($query, $status = null)
    {
        return $query->where('id_user', auth()->id())
            ->when($status, fn($query) => $query->where('status', $status))
            ->count();
    }
}
```

5.2 Membuat *Controller TaskController*

Controller dalam Laravel digunakan untuk menangani logika aplikasi dan menghubungkan antara model (Task.php) dan *view* (tasks.blade.php). *Controller* ***T****askController* akan berisi metode untuk mengelola tugas, seperti menampilkan daftar tugas, menambah tugas baru, meng-edit tugas, dan menghapus tugas. Gunakan perintah *Artisan* berikut untuk membuat *TaskController*:

Tabel 5. 4 Membuat Controller Task

```
php artisan make:controller TaskController
```

Perintah ini akan membuat *file* TaskController.php di dalam folder app/Http/Controllers/. Setelah menjalankan perintah di atas, Laravel akan menghasilkan *file* TaskController.php dengan struktur dasar berikut:

Tabel 5. 5 Kode Controller Task

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class TaskController extends Controller
{
   //
}
```

Namun, agar *controller* ini dapat menangani CRUD (Create, Read, Update, Delete) tugas, kita perlu menambahkan metode yang diperlukan. Buka *file* app/Http/Controllers/TaskController.php dan tambahkan kode berikut:

Tabel 5. 6 Kode Controller Task 2

```
<?php

namespace App\Http\Controllers;

use App\Models\Task;
use Illuminate\Http\Request;
use App\Http\Requests\TaskRequest;

class TaskController extends Controller
{
   public function index(Request $request)
   {
      $filters = $request->only('search', 'status');

      $tasks = Task::search($filters)
         ->orderBy('created_at', 'desc')
```

```
            ->get();

        $counters = [
            'all' => Task::countByStatus(),
            'tertunda' => Task::countByStatus('tertunda'),
            'selesai' => Task::countByStatus('selesai'),
        ];

        return view('user.tasks', compact('tasks', 'counters'));
    }

    public function create()
    {
        //
    }

    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'judul' => 'required|max:255',
            'deskripsi' => 'required',
            'deadline' => 'required|date|after_or_equal:today',
            'status' => 'required|in:tertunda,selesai'
        ], [
            'judul.required' => 'Judul tugas harus diisi',
            'judul.max' => 'Judul tugas maksimal 255 karakter',
            'deskripsi.required' => 'Deskripsi tugas harus diisi',
            'deadline.required' => 'Tenggat waktu harus diisi',
            'deadline.after_or_equal' => 'Tenggat waktu tidak boleh kurang dari
hari ini',
            'status.required' => 'Status harus dipilih',
            'status.in' => 'Status tidak valid'
        ]);

        $validatedData['id_user'] = auth()->id();

        Task::create($validatedData);
```

```
        return redirect('/tasks')->with('success', 'Tugas baru berhasil
ditambahkan!');
    }

    public function show(string $id)
    {
        //
    }

    public function edit(string $id)
    {
        //
    }

    public function update(Request $request, $id)
    {
        $task = Task::where('id_user', auth()->id())->findOrFail($id);

        $validatedData = $request->validate([
            'judul' => 'required|max:255',
            'deskripsi' => 'required',
            'deadline' => 'required|date',
            'status' => 'required|in:tertunda,selesai'
        ], [
            'judul.required' => 'Judul tugas harus diisi',
            'judul.max' => 'Judul tugas maksimal 255 karakter',
            'deskripsi.required' => 'Deskripsi tugas harus diisi',
            'deadline.required' => 'Tenggat waktu harus diisi',
            'status.required' => 'Status harus dipilih',
            'status.in' => 'Status tidak valid'
        ]);

        $task->update($validatedData);

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

```
    public function destroy(string $id)
    {
        $task = Task::where('id_user', auth()->id())
                    ->where('id', $id)
                    ->firstOrFail();

        $task->delete();

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

    public function toggleStatus($id)
    {
        $task = Task::where('id_user', auth()->id())->findOrFail($id);

        $task->update([
            'status' => $task->status === 'selesai' ? 'tertunda' : 'selesai',
        ]);

        return redirect()->back()->with('success', 'Status tugas berhasil diubah');
    }
}
```

5.3 Membuat Tampilan *To-Do-List* (*View*)

Setelah membuat Model *Task* dan *Controller TaskController*, langkah berikutnya adalah membuat tampilan *(view)* untuk halaman daftar tugas. *View* ini bertanggung jawab untuk menampilkan daftar tugas, *form input* tugas, serta tombol untuk meng-edit dan menghapus tugas. Dalam Laravel, tampilan dibuat menggunakan *Blade Template Engine* yang memungkinkan kita untuk menyusun *HTML* dengan fitur tambahan seperti *inheritance* (@extends), *section* (@section), dan *loops* (@foreach).

5.3.1 *Layout* Halaman *To-Do-List* (layouts/task.blade.php)

Layout adalah *template* utama yang digunakan sebagai kerangka untuk halaman lain. Dengan *layout*, kita dapat menjaga konsistensi tampilan tanpa harus menulis ulang kode HTML yang sama di setiap halaman. File layouts/task.blade.php akan digunakan sebagai *layout* utama untuk halaman daftar tugas. *Layout* ini mencakup *navbar*, struktur dasar halaman, serta tempat untuk menyisipkan konten dinamis menggunakan @yield('content'). Buat file task.blade.php di dalam folder resources/views/layouts/ dengan perintah berikut:

Tabel 5. 7 Membuat Tampilan Task

```
touch resources/views/layouts/task.blade.php
```

Setelah menjalankan perintah di atas, sebuah file kosong bernama task.blade.php akan dibuat di dalam folder resources/views/layouts/. *File* ini akan berfungsi sebagai *template* utama yang dapat digunakan oleh halaman lain dengan menggunakan @extends('layouts.task'). Agar *layout* ini dapat digunakan secara optimal, kita akan menambahkan struktur *HTML* dasar dan elemen pendukung seperti *navbar* dan *Bootstrap*. Buka file resources/views/layouts/task.blade.php dan tambahkan kode berikut:

Tabel 5. 8 Kode Dari Tampilan Task

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>{{ config('app.name', 'Task Manager') }}</title>

  <!-- CSRF Token -->
```

```
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <!-- Fonts -->
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400
;500;600;700&display=swap" rel="stylesheet">

    <!-- Styles -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bo
otstrap.min.css" rel="stylesheet">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">

    <style>
        :root {
            --purple-primary: #6B46C1;
            --purple-dark: #553C9A;
            --purple-light: #9F7AEA;
            --purple-lighter: #E9D8FD;
            --purple-lightest: #FAF5FF;
        }

        body {
            font-family: 'Inter', sans-serif;
            background-color: var(--purple-lightest);
            padding-top: 75px;
        }

        .navbar {
            background-color: rgba(107, 70, 193, 0.3);
            transition: background 0.3s ease;
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            box-shadow: 0 10px 25px rgba(107, 70, 193, 0.1);
            z-index: 1000;
```

```
}

.navbar-brand {
  color: var(--purple-dark);
  font-weight: 700;
  text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.1);
  font-size: 1.25rem;
}

.navbar-profile {
  display: flex;
  align-items: center;
  gap: 1rem;
  background: rgba(255, 255, 255, 0.5);
  border-radius: 50px;
  padding: 0.3rem 0.5rem;
  transition: background 0.3s ease;
}

.navbar-profile:hover {
  background: rgba(255, 255, 255, 0.7);
}

.profile-info {
  text-align: right;
  line-height: 1.2;
}

.profile-name {
  font-weight: 600;
  font-size: 0.9rem;
}

.profile-email {
  font-size: 0.8rem;
  opacity: 0.8;
}

.profile-avatar {
```

```
        width: 36px;
        height: 36px;
        background: linear-gradient(135deg, var(--purple-light),
var(--purple-primary));
        border-radius: 50%;
        display: flex;
        align-items: center;
        justify-content: center;
        color: white;
    }

    .btn-primary {
        background-color: var(--purple-primary);
        border-color: var(--purple-primary);
    }

    .btn-primary:hover {
        background-color: var(--purple-dark);
        border-color: var(--purple-dark);
    }

    .nav-tabs .nav-link.active {
        color: var(--purple-primary);
        border-color: var(--purple-primary);
    }

    .form-check-input:checked {
        background-color: var(--purple-primary);
        border-color: var(--purple-primary);
    }

    .card {
        border-radius: 1rem;
        box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);
    }

    .search-input {
        background-color: var(--purple-dark);
        border: none;
```

```
        color: white;
        border-radius: 0.75rem;
      }

      @media (max-width: 768px) {
        .container {
          padding-left: 1rem;
          padding-right: 1rem;
        }

        .nav-tabs {
          overflow-x: auto;
          flex-wrap: nowrap;
          scrollbar-width: none;
          width: 100%;
        }

        .profile-info {
          display: none;
        }
      }
    </style>
</head>
<body>
    <div id="app">
      <nav class="navbar navbar-expand-lg navbar-light">
        <div class="container-fluid">
          <a class="navbar-brand ms-2" href="#">To-Do-List</a>
          <div class="navbar-profile dropdown ms-auto">
            <div class="d-flex align-items-center gap-3"
role="button" data-bs-toggle="dropdown">
                <div class="profile-info d-none d-md-block">
                  <div class="profile-name">{{ Auth::user()->name
}}</div>
                  <div class="profile-email">{{ Auth::user()->email
}}</div>
                </div>
                <div class="profile-avatar">
                  <i class="fas fa-user"></i>
```

```
                </div>
              </div>
              <ul class="dropdown-menu dropdown-menu-end">
                <li>
                  <form action="{{ route('logout') }}"
method="POST">
                    @csrf
                    <button type="submit" class="dropdown-item
text-danger">
                      <i class="fas fa-sign-out-alt me-2"></i>Keluar
                    </button>
                  </form>
                </li>
              </ul>
            </div>
          </div>
        </nav>
      </div>

      <div>
        @yield('content')
      </div>

      <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/boots
trap.bundle.min.js"></script>
      <script>

document.getElementById('searchTask')?.addEventListener('keyu
p', function(e) {
          const searchText = e.target.value.toLowerCase();
          const tasks = document.querySelectorAll('.list-group-
item');
          tasks.forEach(task => {
            const title =
task.querySelector('h6').textContent.toLowerCase();
            task.style.display = title.includes(searchText) ? '' : 'none';
          });
        });
```

```
    document.querySelectorAll('.nav-link').forEach(link => {
        link.addEventListener('click', function(e) {
            if (this.dataset.filter) {
                e.preventDefault();
                const filter = this.dataset.filter;
                const tasks = document.querySelectorAll('.list-group-
item');

                document.querySelectorAll('.nav-link').forEach(l =>
l.classList.remove('active'));
                this.classList.add('active');

                tasks.forEach(task => {
                    const isCompleted = task.querySelector('.form-
check-input').checked;
                    task.style.display =
                        filter === 'all' ||
                        (filter === 'active' && !isCompleted) ||
                        (filter === 'completed' && isCompleted)
                            ? '' : 'none';
                });
            }
        });
    });
  </script>
  @stack('scripts')
</body>
</html>
```

Setelah membuat *layout* task.blade.php, kini aplikasi *To-Do-List* sudah memiliki kerangka halaman dengan *navbar* dan area konten dinamis. Proses *login* juga sudah berhasil, sehingga pengguna dapat mengakses halaman dengan aman.

5.3.2 Tampilan Daftar Tugas (user/tasks.blade.php)

Setelah membuat *layout* utama, sekarang kita akan membuat halaman daftar tugas yang digunakan untuk menampilkan semua tugas milik pengguna. Halaman ini akan menggunakan *layout* yang telah dibuat dengan @extends('layouts.task') dan mengisi bagian @section('content'). Halaman ini akan memiliki:

1. Daftar tugas berdasarkan status (tertunda/selesai)
2. *Filter* pencarian tugas
3. Tombol untuk menambah, meng-edit, dan menghapus tugas

Selanjutnya membuat folder resources/views/user/ dengan perintah berikut:

Tabel 5. 9 Membuat Tampilan User

```
mkdir resources/views/user
```

Setelah folder tersedia, jalankan perintah berikut untuk membuat file tasks.blade.php:

Tabel 5. 10 Membuat Tampilan User 2

```
touch resources/views/user/tasks.blade.php
```

Setelah menjalankan perintah di atas, Laravel akan membuat file kosong bernama tasks.blade.php di dalam folder resources/views/user/. File ini akan digunakan sebagai tampilan utama untuk menampilkan daftar tugas pengguna. Buka file resources/views/user/tasks.blade.php dan tambahkan kode berikut:

Tabel 5. 11 Kode Dari Tampilan User

```
@extends('layouts.task')

@section('content')
```

```
   <div class="container py-4">
      @if (session('success') || session('error'))
         <div class="alert alert-{{ session('success') ? 'success' :
'danger' }} alert-dismissible fade show" role="alert">
            {{ session('success') ?? session('error') }}
            <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>
         </div>
      @endif

      <!-- Page Header with Search -->
      <div class="row g-3 align-items-center mb-4">
         <div class="col-md-4">
            <div class="d-flex align-items-center">
               <i class="fas fa-tasks fa-3x me-3"></i>
               <div>
                  <h1 class="h3 mb-0">Daftar Tugas</h1>
                  <p class="text-muted small mb-0">
                     <i class="fas fa-clipboard-list me-1"></i>Kelola
tugas-tugas Anda
                  </p>
               </div>
            </div>
         </div>
         <div class="col-md-5">
            <form action="{{ route('tasks.index') }}"
method="GET">
               <div class="input-group">
                  <input type="text" class="form-control"
name="search" placeholder="Cari tugas..."
                     value="{{ request('search') }}" id="searchInput">
                  <button type="submit" class="btn btn-secondary">
                     <i class="fas fa-search"></i>
                  </button>
               </div>
            </form>
         </div>
         <div class="col-md-3">
```

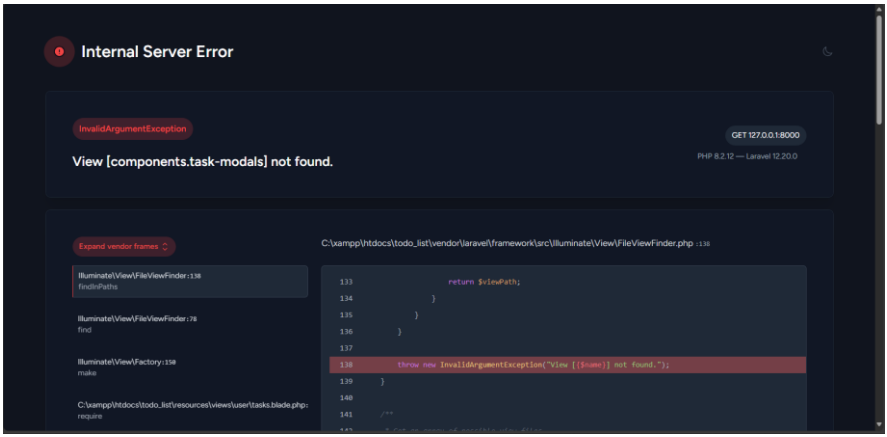
```
            <button class="btn btn-primary w-100" data-bs-
toggle="modal" data-bs-target="#addTaskModal">
                <i class="fas fa-plus-circle me-1"></i> Tambah Tugas
            </button>
          </div>
      </div>

      <!-- Task List Card -->
      <div class="card border-0 shadow-sm">
        <div class="card-header bg-white py-3">
          <ul class="nav nav-tabs card-header-tabs">
            <li class="nav-item flex-grow-1 text-center">
              <a class="nav-link {{ !request('status') ? 'active' : ''
}}"
                 href="{{ route('tasks.index') }}">
                 <i class="fas fa-list me-2"></i>Semua
                 <span class="badge bg-secondary ms-2">{{
$tasks->count() }}</span>
              </a>
            </li>
            <li class="nav-item flex-grow-1 text-center">
              <a class="nav-link {{ request('status') === 'tertunda'
? 'active' : '' }}"
                 href="{{ route('tasks.index', ['status' =>
'tertunda']) }}">
                 <i class="fas fa-clock me-2"></i>Aktif
                 <span class="badge bg-primary ms-2">{{ $tasks-
>where('status', 'tertunda')->count() }}</span>
              </a>
            </li>
            <li class="nav-item flex-grow-1 text-center">
              <a class="nav-link {{ request('status') === 'selesai' ?
'active' : '' }}"
                 href="{{ route('tasks.index', ['status' => 'selesai'])
}}">
                 <i class="fas fa-check-circle me-2"></i>Selesai
                 <span class="badge bg-success ms-2">{{ $tasks-
>where('status', 'selesai')->count() }}</span>
              </a>
```

```
            </li>
          </ul>
        </div>

        <div class="card-body p-0">
          @if ($tasks->isEmpty())
            <div class="text-center py-5">
              <i class="fas fa-tasks fa-4x text-muted mb-3"></i>
              <h4>Belum Ada Tugas</h4>
              <p class="text-muted">Mulai dengan
menambahkan tugas baru</p>
            </div>
          @else
            <div class="list-group list-group-flush">
              @foreach ($tasks as $task)
                <div class="list-group-item p-3 {{ $task->status
== 'selesai' ? 'bg-light' : '' }}">
                  <div class="row align-items-center">
                    <div class="col-auto">
                      <form action="{{ route('tasks.toggle',
$task->id) }}" method="POST">
                        @csrf
                        <input type="checkbox" class="form-
check-input"
                          onchange="this.form.submit()"
                          {{ $task->status == 'selesai' ?
'checked' : '' }}>
                      </form>
                    </div>
                    <div class="col">
                      <h6 class="{{ $task->status == 'selesai' ?
'text-decoration-line-through text-muted' : '' }}">
                        {{ $task->judul }}
                      </h6>
                      @if ($task->deskripsi)
                        <p class="text-muted small mb-1">{{
Str::limit($task->deskripsi, 100) }}</p>
                      @endif
                      <div class="d-flex gap-2">
```

```
                                @if ($task->deadline)
                                    <span class="badge {{
strtotime($task->deadline) < time() ? 'bg-danger' : 'bg-info' }}">
                                        <i class="far fa-calendar-alt me-
1"></i>
                                        {{ $task->deadline->format('d M Y')
}}
                                    </span>
                                @endif
                                <span class="badge {{ $task->status ==
'selesai' ? 'bg-success' : 'bg-warning' }}">
                                    <i class="fas fa-{{ $task->status ==
'selesai' ? 'check' : 'clock' }} me-1"></i>
                                    {{ ucfirst($task->status) }}
                                </span>
                            </div>
                        </div>
                        <div class="col-auto">
                            <button class="btn btn-sm btn-light" data-
bs-toggle="modal"
                                data-bs-target="#editTaskModal{{
$task->id }}">
                                <i class="fas fa-edit text-primary"></i>
                            </button>
                            <button class="btn btn-sm btn-light" data-
bs-toggle="modal"
                                data-bs-target="#deleteTaskModal{{
$task->id }}">
                                <i class="fas fa-trash-alt text-
danger"></i>
                            </button>
                        </div>
                    </div>
                </div>
            @endforeach
        </div>
    @endif
</div>
</div>
```

```
  </div>

  @include('components.task-modals')
@endsection

@push('scripts')
  <script>
    (() => {
      'use strict'
      document.querySelectorAll('.needs-
validation').forEach(form => {
        form.addEventListener('submit', event => {
          if (!form.checkValidity()) {
            event.preventDefault()
            event.stopPropagation()
          }
          form.classList.add('was-validated')
        })
      })
    })()
  </script>
@endpush
```

Setelah membuat halaman tasks.blade.php dan menjalankannya, akan muncul error “View [components.task-modals] not found.” seperti pada gambar. Error ini terjadi karena file komponen modal yang berfungsi untuk menambah, meng-edit, dan menghapus tugas belum dibuat.



Gambar 5. 1 Error Halaman Task

Untuk mengatasinya, langkah selanjutnya adalah membuat file task-modals.blade.php di folder resources/views/components/. File ini akan berisi struktur modal Bootstrap yang dibutuhkan agar tombol *Tambah*, *Edit*, dan *Hapus* pada halaman daftar tugas dapat berfungsi dengan baik.

5.3.3 Komponen Modal Tugas (components/task-modals.blade.php)

Agar proses menambah dan mengedit tugas dapat dilakukan dengan tampilan yang lebih interaktif dan user-friendly, kita akan membuat sebuah file komponen Blade bernama task-modals.blade.php. File ini akan berisi elemen-elemen modal Bootstrap yang digunakan untuk menampilkan form tambah atau edit tugas dalam bentuk popup, tanpa harus berpindah halaman.

Dengan memisahkan modal ke dalam komponen terpisah, kita dapat menjaga struktur kode agar tetap rapi dan lebih mudah dikelola. Komponen ini nantinya akan digunakan di halaman tasks.blade.php menggunakan sintaks @include.

Sebelum membuat file tersebut, pastikan folder components sudah tersedia. Jika belum ada, buat terlebih dahulu dengan perintah:

Tabel 5. 12 Membuat Komponen Untuk Views

mkdir resources/views/components

Kemudian, buat file task-modals.blade.php menggunakan perintah berikut:

Tabel 5. 13 Membuat Task Modals

```
touch resources/views/components/task-modals.blade.php
```

Setelah menjalankan perintah di atas, Laravel akan membuat sebuah file kosong bernama task-modals.blade.php. Buka file tersebut dan tambahkan kode HTML untuk dua buah modal, yaitu:

1. Modal Tambah Tugas: digunakan untuk menambahkan tugas baru.
2. Modal Edit Tugas: digunakan untuk memperbarui informasi tugas yang sudah ada.

Kode ini akan menggunakan Bootstrap untuk styling dan memanfaatkan elemen *form* yang sesuai agar dapat berinteraksi dengan *controller*.

Selanjutnya, buka *file* resources/views/components/task-modals.blade.php dan tambahkan kode berikut:

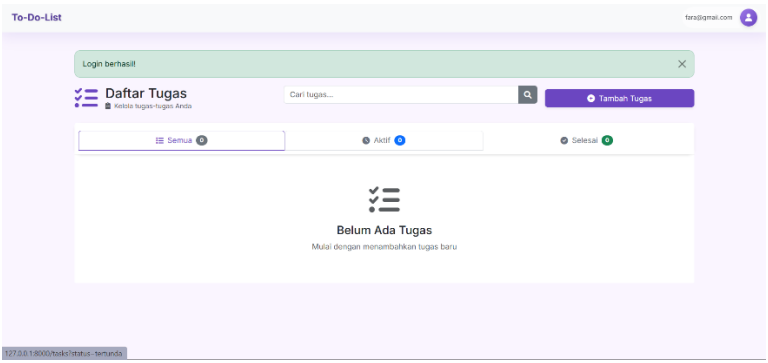
Tabel 5. 14 Kode Dari Task-modals

```
<div class="modal fade" id="addTaskModal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Tambah Tugas Baru</h5>
        <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
      </div>
      <form action="{{ route('tasks.store') }}" method="POST" class="needs-
validation" novalidate>
        @csrf
        <div class="modal-body">
          <div class="mb-3">
            <label class="form-label">Judul Tugas</label>
            <input type="text" class="form-control" name="judul"
required>
            <div class="invalid-feedback">Judul tugas harus diisi</div>
```

```
            </div>
            <div class="mb-3">
              <label class="form-label">Deskripsi</label>
              <textarea class="form-control" name="deskripsi" rows="3"
required></textarea>
              <div class="invalid-feedback">Deskripsi tugas harus diisi</div>
            </div>
            <div class="mb-3">
              <label class="form-label">Deadline</label>
              <input type="date" class="form-control" name="deadline"
required>
              <div class="invalid-feedback">Deadline harus diisi</div>
            </div>
            <div class="mb-3">
              <label class="form-label">Status</label>
              <select class="form-select" name="status" required>
                <option value="tertunda">Tertunda</option>
                <option value="selesai">Selesai</option>
              </select>
              <div class="invalid-feedback">Status harus dipilih</div>
            </div>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Batal</button>
            <button type="submit" class="btn btn-
primary">Simpan</button>
          </div>
        </form>
      </div>
    </div>
</div>

<!-- Edit Task Modals -->
@foreach ($tasks as $task)
<div class="modal fade" id="editTaskModal{{ $task->id }}" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
```

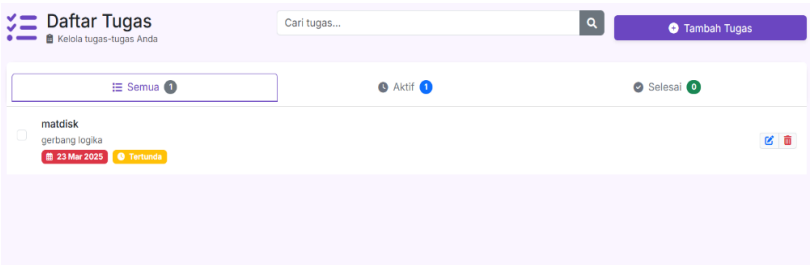
```
            <h5 class="modal-title">Edit Tugas</h5>
            <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
          </div>
          <form action="{{ route('tasks.update', $task->id) }}" method="POST"
class="needs-validation" novalidate>
            @csrf
            @method('PUT')
            <div class="modal-body">
              <div class="mb-3">
                <label class="form-label">Judul Tugas</label>
                <input type="text" class="form-control" name="judul"
value="{{ $task->judul }}" required>
                <div class="invalid-feedback">Judul tugas harus diisi</div>
              </div>
              <div class="mb-3">
                <label class="form-label">Deskripsi</label>
                <textarea class="form-control" name="deskripsi" rows="3"
required>{{ $task->deskripsi }}</textarea>
                <div class="invalid-feedback">Deskripsi tugas harus diisi</div>
              </div>
              <div class="mb-3">
                <label class="form-label">Deadline</label>
                <input type="date" class="form-control" name="deadline"
                  value="{{ $task->deadline ? $task->deadline->format('Y-m-d')
: '' }}" required>
                <div class="invalid-feedback">Deadline harus diisi</div>
              </div>
              <div class="mb-3">
                <label class="form-label">Status</label>
                <select class="form-select" name="status" required>
                  <option value="tertunda" {{ $task->status == 'tertunda' ?
'selected' : '' }}>Tertunda</option>
                  <option value="selesai" {{ $task->status == 'selesai' ?
'selected' : '' }}>Selesai</option>
                </select>
                <div class="invalid-feedback">Status harus dipilih</div>
              </div>
            </div>
```

```
            <div class="modal-footer">
              <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Batal</button>
              <button type="submit" class="btn btn-primary">Simpan
Perubahan</button>
            </div>
          </form>
        </div>
      </div>
    </div>

<!-- Delete Task Modal -->
<div class="modal fade" id="deleteTaskModal{{ $task->id }}" tabindex="-1">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title">Hapus Tugas</h5>
          <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
        </div>
        <div class="modal-body">
          <p>Apakah Anda yakin ingin menghapus tugas ini?</p>
          <p class="text-muted mb-0">{{ $task->judul }}</p>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Batal</button>
          <form action="{{ route('tasks.destroy', $task->id) }}"
method="POST" class="d-inline">
            @csrf
            @method('DELETE')
            <button type="submit" class="btn btn-danger">Hapus</button>
          </form>
        </div>
      </div>
    </div>
</div>
@endforeach
```



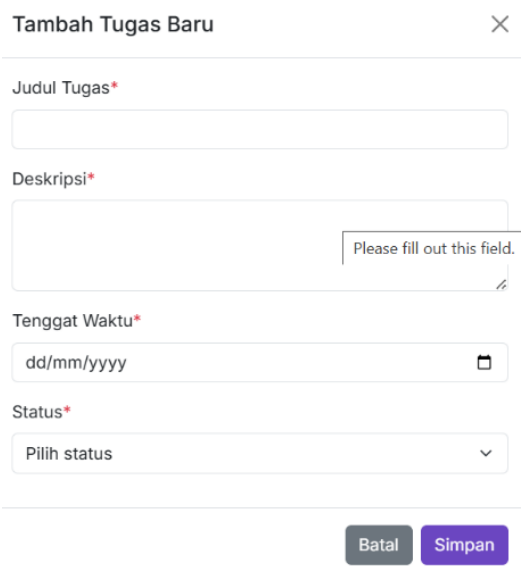
Gambar 5. 2 Tampilan To-Do List

Pada gambar 5.2 menampilkan halaman utama, dimana daftar seluruh tugas pengguna akan di tampilkan. Disini kita dapat melihat tombol tambah tugas, tombol search, daftar tugas dan sebagainya.



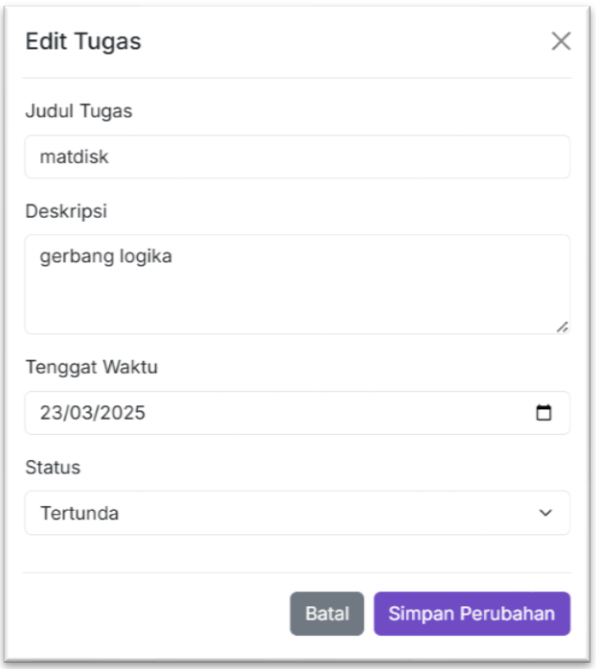
Gambar 5. 3 Tampilan Daftar Tugas

Pada gambar 5.3 menampilkan tampilan daftar tugas setelah dibuat. Setiap tugas ditampilkan dengan informasi penting seperti judul, deskripsi, dan status sehingga memudahkan pengguna dalam mengelola pekerjaan mereka.



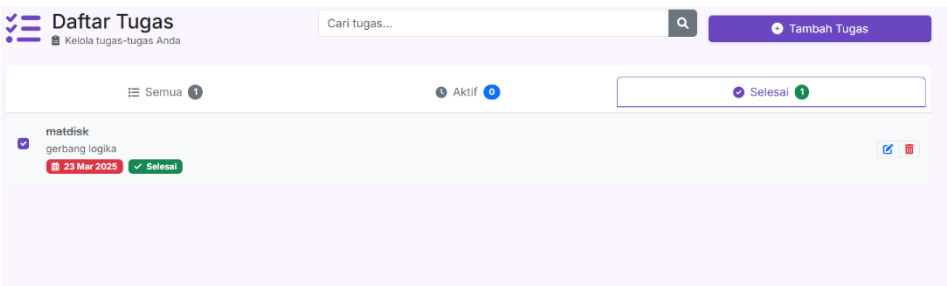
Gambar 5. 4 Tampilan Daftar Tugas Baru

Pada gambar 5.4 menunjukkan tampilan form yang digunakan untuk menambahkan tugas baru ke dalam daftar. Pengguna dapat memasukkan judul tugas, deskripsi, dan status serta batas waktu tugas sebelum disimpan agar dapat segera muncul pada daftar tugas utama.



Gambar 5. 5 Tampilan Edit Tugas

Pada gambar 5.5 menunjukkan tampilan ketika pengguna melakukan proses edita tugas yang sudah ada. Fitur ini memungkinkan pengguna memperbarui informasi seperti judul tugas, deskripsi, tenggat waktu dan status sesuai dengan kebutuhan yang berubah.



Gambar 5. 6 Tampilan Daftar Tugas Selesai

Pada gambar ini terlihat daftar tugas yang telah ditandai selesai oleh pengguna. Tampilan ini berfungsi untuk membedakan antara tugas yang masih aktif dengan yang sudah selesai, sekaligus membantu pengguna memantau progres pekerjaan mereka.

BAB 6
HALAMAN ADMIN
(*Dashboard* & Kelola *User*)

Bab ini akan membahas bagaimana membangun halaman Admin dalam aplikasi *To-Do-List* menggunakan Laravel. Halaman Admin berfungsi sebagai pusat kontrol bagi pengelola sistem untuk mengawasi aktivitas pengguna, mengelola akun pengguna, serta menampilkan statistik terkait tugas yang telah dibuat. Dengan memahami materi ini, Anda akan dapat membangun fitur Admin yang efektif dan terstruktur dalam proyek Laravel.

6.1 Membuat *Controller* DashboardController

Controller dalam Laravel bertugas menangani logika aplikasi dan menjembatani antara model serta *view*. Pada bagian ini, kita akan membuat *controller* bernama DashboardController yang digunakan untuk menangani halaman *dashboard* yaitu halaman awal yang ditampilkan setelah pengguna *login*. Halaman ini dapat berisi ringkasan informasi seperti jumlah tugas yang belum selesai, tugas yang telah selesai, atau statistik lainnya.

Gunakan perintah Artisan berikut untuk membuat *controller*:

Tabel 6. 1 Membuat Controller Dashboard

```
php artisan make:controller DashboardController
```

Perintah ini akan menghasilkan file DashboardController.php di dalam folder app/Http/Controllers/. Setelah perintah dijalankan, Laravel akan membuat *file controller* dengan struktur dasar seperti berikut:

Tabel 6. 2 Kode Dari Dashboard Controller

```
<?php

namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;

class DashboardController extends Controller
{
   //
}
```

Namun, agar *controller* ini dapat menangani logika tampilan *dashboard*, kita perlu menambahkan metode yang diperlukan, misalnya index() untuk menampilkan halaman *dashboard*. Kita juga dapat menambahkan logika untuk mengambil data statistik dari *database* dan mengirimkannya ke *view*.

Buka file app/Http/Controllers/DashboardController.php dan tambahkan kode berikut:

Tabel 6. 3 Membuat Controller Dashboard 2

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use App\Models\Role;
use Illuminate\Http\Request;

class DashboardController extends Controller
{
   public function index()
   {
      $totalUsers = User::count();

      $usersByRole = Role::withCount('users')->get();

      $latestUsers = User::with('role')
         ->orderBy('created_at', 'desc')
         ->limit(5)
         ->get();
```

```
        return view('dashboard.index', [
            'totalUsers' => $totalUsers,
            'usersByRole' => $usersByRole,
            'latestUsers' => $latestUsers
        ]);
    }
}
```

6.2 Tampilan Dashboard (dashboard/index.blade.php)

Setelah membuat DashboardController dan mendefinisikan logika untuk mengirimkan data statistik pengguna ke tampilan, langkah berikutnya adalah membuat halaman *dashboard* pada sisi *frontend* menggunakan *Blade*. *File* ini akan digunakan untuk menampilkan ringkasan statistik dan informasi terkini seputar pengguna dalam sistem.

Untuk itu, buat *file* baru bernama index.blade.php di dalam folder resources/views/dashboard/. Sebelum membuat *file*, pastikan folder *dashboard* sudah tersedia. Jika belum ada, jalankan perintah berikut untuk membuatnya:

Tabel 6. 4 Membuat View Dashboard

```
mkdir resources/views/dashboard
```

Setelah folder tersedia, buat file index.blade.php dengan perintah:

Tabel 6. 5 Membuat View Dashboard 2

```
touch resources/views/dashboard/index.blade.php
```

Perintah di atas akan menghasilkan file kosong index.blade.php. Selanjutnya, kita akan mengisinya dengan struktur HTML untuk menampilkan informasi statistik. Buka file resources/views/dashboard/index.blade.php dan tambahkan kode berikut:

Tabel 6. 6 Kode Dari Dashboard Main

```
@extends('dashboard.layouts.main')

@section('container')
<div class="container-fluid mt-4">
  <!-- Welcome Section -->
  <div class="row mb-4">
    <div class="col-12">
      <div class="d-sm-flex align-items-center justify-content-between">
        <div>
          <h1 class="h3 mb-0 text-gray-800 d-flex align-items-center">
            <i class="fas fa-tachometer-alt me-3 text-primary"></i>
            <span>Selamat Datang, {{ Auth::user()->nama }}!</span>
          </h1>

        </div>
        <div class="d-none d-sm-inline-block">
          <div class="text-end text-muted">
            <div class="small">{{ now()->format('l') }}</div>
            <div class="h5 mb-0">{{ now()->format('d F Y') }}</div>
          </div>
        </div>
      </div>
    </div>
  </div>

  <!-- Stats Cards -->
  <div class="row">
    <!-- Total Users Card -->
    <div class="col-xl-3 col-md-6 mb-4">
      <div class="card border-0 shadow-sm h-100 position-relative
overflow-hidden">
        <div class="position-absolute top-0 start-0 w-1 h-100 bg-
primary"></div>
        <div class="card-body">
          <div class="d-flex justify-content-between align-items-center mt-
3">
            <div class="text-xs text-uppercase fw-bold text-primary">Total
Pengguna</div>
```

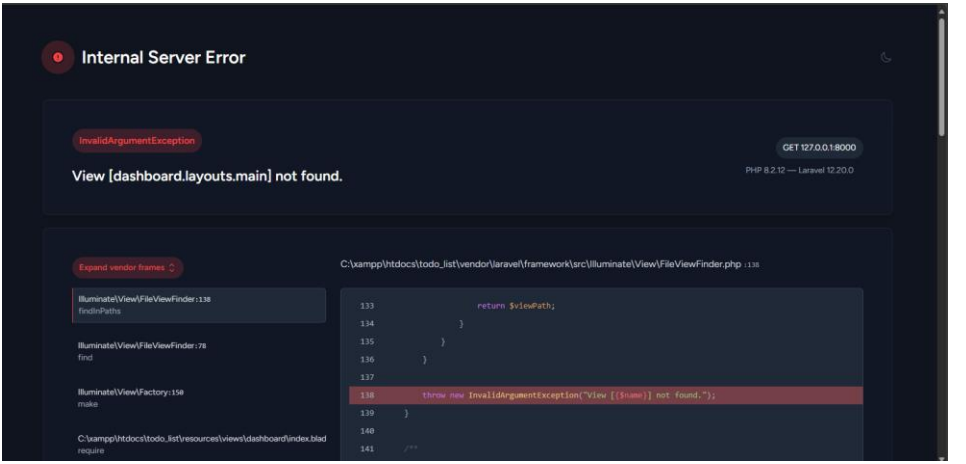
```
                <div class="rounded-circle bg-primary bg-opacity-10 p-2">
                  <i class="fas fa-users fa-fw text-primary"></i>
                </div>
              </div>
              <div class="h2 mb-0 fw-bold mt-2 justify-center text-center">{{
number_format($totalUsers) }}</div>
              <div class="text-muted small mt-2 justify-center text-center">
                <i class="fas fa-chart-line me-1"></i>
                Pengguna Aktif
              </div>
            </div>
          </div>
        </div>

        <!-- Users by Role Card -->
        <div class="col-xl-3 col-md-6 mb-4">
          <div class="card border-0 shadow-sm h-100 position-relative
overflow-hidden">
            <div class="position-absolute top-0 start-0 w-1 h-100 bg-
success"></div>
            <div class="card-body">
              <div class="d-flex justify-content-between align-items-center mt-
3">
                <div class="text-xs text-uppercase fw-bold text-
success">Pengguna per Role</div>
                <div class="rounded-circle bg-success bg-opacity-10 p-2">
                  <i class="fas fa-user-tag fa-fw text-success"></i>
                </div>
              </div>
              @foreach($usersByRole as $role)
              <div class="d-flex justify-content-between align-items-center mt-
2">
                <span class="text-muted">{{ $role->nama }}</span>
                <div class="d-flex align-items-center">
                  <span class="fw-bold me-2">{{ $role->users_count }}</span>
                  <div class="progress" style="width: 50px; height: 4px;">
                    <div class="progress-bar bg-success" style="width: {{
($role->users_count / $totalUsers) * 100 }}%"></div>
                  </div>
```

```
                </div>
              </div>
              @endforeach
            </div>
          </div>
        </div>

        <!-- Latest Users Card -->
        <div class="col-xl-6 col-md-12 mb-4">
          <div class="card border-0 shadow-sm h-100">
            <div class="card-header bg-white py-3 border-0">
              <div class="d-flex align-items-center justify-content-between">
                <h6 class="m-0 fw-bold text-primary d-flex align-items-center">
                  <i class="fas fa-user-clock me-2"></i>
                  Pengguna Terbaru
                </h6>
              </div>
            </div>
            <div class="card-body p-0">
              <div class="table-responsive">
                <table class="table table-hover align-middle mb-0">
                  <thead class="bg-light">
                    <tr>
                      <th class="border-0 ps-4">Nama</th>
                      <th class="border-0">Email</th>
                      <th class="border-0">Role</th>
                      <th class="border-0 text-center">Status</th>
                      <th class="border-0">Bergabung</th>
                    </tr>
                  </thead>
                  <tbody>
                    @foreach($latestUsers as $user)
                    <tr>
                      <td class="ps-4">
                        <div class="d-flex align-items-center">
                          <div class="rounded-circle bg-primary bg-opacity-10
p-2 me-3">
                            {{ strtoupper(substr($user->nama, 0, 1)) }}
                          </div>
```

```
                        <div>
                          <div class="fw-bold text-dark">{{ $user->nama }}</div>
                          <div class="text-muted small">ID: #{{ $user->id }}</div>
                        </div>
                      </div>
                    </td>
                    <td>{{ $user->email }}</td>
                    <td>
                      <span class="badge rounded-pill bg-primary bg-opacity-10 text-primary px-3 py-2">
                        {{ $user->role->nama ?? 'Tidak ada role' }}
                      </span>
                    </td>
                    <td class="text-center">
                      <span class="badge rounded-pill bg-success">Aktif</span>
                    </td>
                    <td>
                      <div class="text-muted">
                        <i class="far fa-clock me-1"></i>
                        {{ $user->created_at->diffForHumans() }}
                      </div>
                    </td>
                  </tr>
                  @endforeach
                </tbody>
              </table>
            </div>
          </div>
        </div>
      </div>
    </div>

    <!-- Activity Timeline -->
    <div class="row">
      <div class="col-12">
        <div class="card border-0 shadow-sm">
```

```
            <div class="card-header bg-white py-3 border-0">
              <div class="d-flex align-items-center justify-content-between">
                <h6 class="m-0 fw-bold text-primary">
                  <i class="fas fa-history me-2"></i>
                  Aktivitas Terakhir
                </h6>
              </div>
            </div>
            <div class="card-body">
              <div class="timeline">
                @foreach($latestUsers->take(5) as $user)
                <div class="timeline-item pb-4 position-relative ms-4">
                  <div class="timeline-marker bg-primary position-absolute top-0 start-0 translate-middle-x rounded-circle" style="width: 12px; height: 12px;"></div>
                  <div class="timeline-content ps-4 border-start">
                    <div class="d-flex justify-content-between align-items-center mb-1">
                      <div class="fw-bold">{{ $user->nama }} bergabung</div>
                      <div class="text-muted small">{{ $user->created_at->format('d M Y H:i') }}</div>
                    </div>
                    <p class="text-muted mb-0">Pengguna baru dengan role {{ $user->role->nama ?? 'Tidak ada role' }} telah bergabung ke sistem.</p>
                  </div>
                </div>
                @endforeach
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>


@endsection
```

Setelah membuat *file* index.blade.php untuk halaman *Dashboard*, langkah berikutnya adalah *login* terlebih dahulu sebagai admin agar dapat mengakses halaman *Dashboard*. Pastikan id_role pada data pengguna di *database* diubah secara manual menjadi 1 untuk memberikan hak akses admin.



Gambar 6. 1 Tampilan Error Dasboard

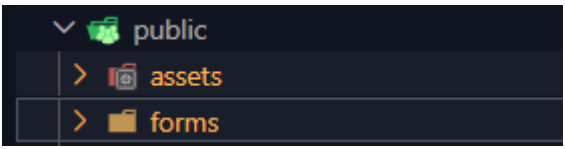
Setelah berhasil *login* sebagai admin dan membuka halaman *Dashboard*, akan muncul *error* "*View* [dashboard.layouts.main] *not found*." seperti pada gambar. Ini menunjukkan bahwa proses sudah benar, tetapi folder *layouts* dan *file* main.blade.php di dalam resources/views/dashboard/ belum dibuat. Langkah berikutnya adalah membuat folder *layouts* dan menambahkan *file* main.blade.php, header.blade.php dan sidebar.blade.php sebagai *layout* utama untuk halaman *Dashboard*.

6.3 Masukan *template dashboard* di fiolder *public*

Langkah selanjutnya adalah mengunduh *template dashboard* yang akan digunakan sebagai tampilan dasar halaman admin. Silakan *download file template dashboard* melalui *link* berikut:

https://drive.google.com/drive/folders/1kmDpkC9ZG_XYeRX09C4-mBBXAsqlThBm?usp=drive_link

Setelah *file ZIP* berhasil diunduh, ekstrak *file* tersebut, kemudian pindahkan folder hasil ekstrak ke dalam folder *public* pada proyek Laravel Anda. Dengan cara ini, semua *file* aset seperti *CSS*, *JavaScript*, gambar, dan *font* dari *template* dapat diakses langsung melalui *URL* aplikasi.



Gambar 6. 2 Struktur Folder CSS Dan JS

Setelah template tersimpan di folder *public*, Anda dapat mulai memanggil *file CSS* dan *JS*-nya melalui *tag* <*link*> dan <*script*> di *file* layout main.blade.php yang akan dibuat.

6.4 *Layout Template* (resources/views/dashboard/layouts/)

Untuk memudahkan pengelolaan tampilan antarmuka secara konsisten di seluruh halaman, dibuatlah struktur *layout* yang terdiri dari tiga *file* utama, yaitu header.blade.php, sidebar.blade.php, dan main.blade.php. Ketiganya diletakkan di dalam folder resources/views/dashboard/layouts/.

1. Buat *file* header.blade.php didalamnya
   File ini berfungsi untuk menampilkan bagian atas halaman (*navbar* atau *topbar*) yang biasanya memuat informasi akun, tombol *logout*, atau judul halaman.
   Sebelum membuat *file*, pastikan folder *layouts* sudah tersedia di dalam resources/views/dashboard/. Jika belum ada, jalankan perintah berikut:

   Tabel 6. 7 Membuat Tampilan Dashboard

   ```
   mkdir -p resources/views/dashboard/layouts
   ```

   Setelah itu, buat *file* header.blade.php dengan perintah:

   Tabel 6. 8 Membuat Tampilan Dashboard 2

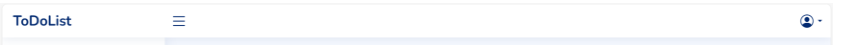
   ```
   touch resources/views/dashboard/layouts/header.blade.php
   ```

Setelah *file* berhasil dibuat, buka *file* tersebut dan masukkan potongan kode berikut ke dalamnya:

Tabel 6. 9 Kode Dari Tampilan Dashboard

```
<header id="header" class="header fixed-top d-flex align-
items-center">

  <div class="d-flex align-items-center justify-content-
between">
    <a href="/dashboard" class="logo d-flex align-items-
center">
      <span class="d-none d-lg-block">ToDoList</span>
    </a>
    <i class="bi bi-list toggle-sidebar-btn"></i>
  </div>

  <nav class="header-nav ms-auto">
    <ul class="d-flex align-items-center">
      <li class="nav-item dropdown pe-3">
        <a class="nav-link nav-profile d-flex align-items-center
pe-0" href="#" data-bs-toggle="dropdown">
          <i class="bi bi-person-circle fs-4"></i>
          <span class="d-none d-md-block dropdown-toggle
ps-2">{{ auth()->user()->name }}</span>
        </a>

        <ul class="dropdown-menu dropdown-menu-end
dropdown-menu-arrow profile">
          <li class="dropdown-header">
            <h6>{{ auth()->user()->name }}</h6>
            <span>{{ auth()->user()->email }}</span>
          </li>

          <li>
            <hr class="dropdown-divider">
          </li>
```

```
                <li>
                  <form action="{{ route('logout') }}"
          method="POST">
                    @csrf
                    <button type="submit" class="dropdown-item d-
          flex align-items-center">
                      <i class="bi bi-box-arrow-right"></i>
                      <span>Sign Out</span>
                    </button>
                  </form>
                </li>

              </ul>
            </li>
          </ul>
        </nav>

</header>
```



Gambar 6.3 Tampilan Side Bar

2. Buat file sidebar.blade.php didalamnya

   File ini berfungsi sebagai menu navigasi di sisi kiri halaman. Sidebar ini akan memuat tautan ke berbagai fitur utama dari dashboard. Kemudian, buat *file* sidebar.blade.php dengan perintah:

Tabel 6. 10 Membuat Tampilan Sidebar

```
touch resources/views/dashboard/layouts/sidebar.blade.php
```

Setelah *file* berhasil dibuat, buka *file* tersebut dan masukkan potongan kode berikut:

Tabel 6. 11 Kode Dari Tampilan Sidebar

```
        <aside id="sidebar" class="sidebar">

          <ul class="sidebar-nav" id="sidebar-nav">

            <li class="nav-item">
              <a class="nav-link {{ Request::is('dashboard') ? '' :
        'collapsed' }}" href="/dashboard">
                <i class="bi bi-grid"></i>
                <span>Dashboard</span>
              </a>
            </li>
            <li class="nav-item">
              <a class="nav-link {{ Request::is('dashboard/users*') ? ''
        : 'collapsed' }}" href="{{ route('users.index') }}">
                <i class="bi bi-people"></i>
                <span>Kelola User</span>
              </a>
            </li>

            <!-- Additional Nav Items -->

          </ul>

</aside>
```

Gambar 6. 3 Tampilan Side Bar

3. Buat *file* main.blade.php didalamnya

File main.blade.php adalah *layout* utama yang akan menyatukan *header*, *sidebar*, dan konten halaman (@yield('container')), serta mengatur kerangka HTML dasar.
Kemudian buat *file* main.blade.php dengan perintah:

Tabel 6. 12 Membuat Dashboard Main

```
touch resources/views/dashboard/layouts/main.blade.php
```

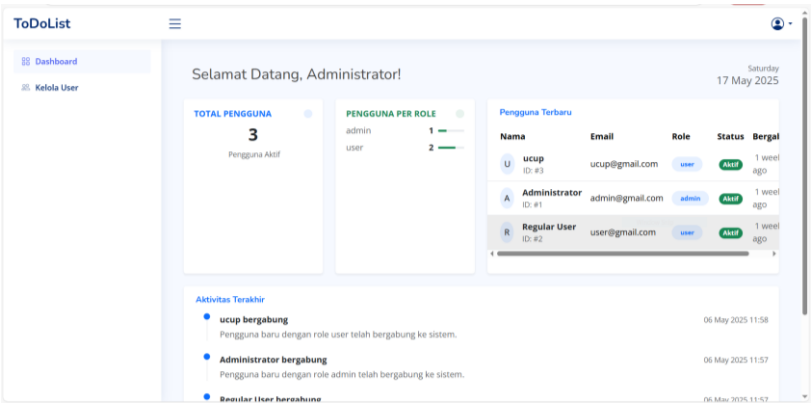
Kemudian buka *file* tersebut dan masukkan kode HTML dasar berikut:

Tabel 6. 13 Kode Dari Tampilan Dashboard Main

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">

  <title>Dashboard - ToDoList</title>
  <meta content="" name="description">
  <meta content="" name="keywords">

  <!-- Favicons -->
  <link href="{{ asset('assets/img/favicon.png') }}" rel="icon">
  <link href="{{ asset('assets/img/apple-touch-icon.png') }}" rel="apple-touch-icon">

  <!-- Google Fonts -->
  <link href="https://fonts.gstatic.com" rel="preconnect">
  <link href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Nunito:300,300i,400,400i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,700i" rel="stylesheet">

  <!-- Vendor CSS Files -->
```

```
 <link href="{{ asset('assets/vendor/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/bootstrap-icons/bootstrap-icons.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/boxicons/css/boxicons.min.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/quill/quill.snow.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/quill/quill.bubble.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/remixicon/remixicon.css') }}" rel="stylesheet">
 <link href="{{ asset('assets/vendor/simple-datatables/style.css') }}" rel="stylesheet">

 <!-- Template Main CSS File -->
 <link href="{{ asset('assets/css/style.css') }}" rel="stylesheet">
</head>

<body>

 @include('dashboard.layouts.header')

 @include('dashboard.layouts.sidebar')

 <main id="main" class="main">
  @yield('container')
 </main>

 <footer id="footer" class="footer">
  <div class="copyright">
   &copy; Copyright <strong><span>ToDoList</span></strong>. All Rights Reserved
  </div>
```

```
        </footer>

        <a href="#" class="back-to-top d-flex align-items-center
        justify-content-center"><i class="bi bi-arrow-up-
        short"></i></a>

        <!-- Vendor JS Files -->
        <script src="{{
        asset('assets/vendor/apexcharts/apexcharts.min.js')
        }}"></script>
        <script src="{{
        asset('assets/vendor/bootstrap/js/bootstrap.bundle.min.js')
        }}"></script>
        <script src="{{ asset('assets/vendor/chart.js/chart.umd.js')
        }}"></script>
        <script src="{{ asset('assets/vendor/echarts/echarts.min.js')
        }}"></script>
        <script src="{{ asset('assets/vendor/quill/quill.min.js')
        }}"></script>
        <script src="{{ asset('assets/vendor/simple-
        datatables/simple-datatables.js') }}"></script>
        <script src="{{
        asset('assets/vendor/tinymce/tinymce.min.js') }}"></script>
        <script src="{{ asset('assets/vendor/php-email-
        form/validate.js') }}"></script>

        <!-- Template Main JS File -->
        <script src="{{ asset('assets/js/main.js') }}"></script>

    </body>

</html>
```



Gambar 6. 4 Tampilan Halaman Admin

6.5 Membuat *Controller* KelolaUserController

Dalam bagian ini, kita akan membuat *controller* bernama KelolaUserController, yang bertanggung jawab untuk mengelola data pengguna. Fungsionalitas yang dapat dikembangkan melalui *controller* ini antara lain: menampilkan daftar pengguna, menambahkan pengguna baru, mengedit data pengguna, dan menghapus pengguna.

Untuk membuat *controller* ini, gunakan perintah Artisan berikut pada terminal:

Tabel 6. 14 Membuat Controller Kelola User

```
php artisan make:controller KelolaUserController
```

Setelah membuat file KelolaUserController.php, langkah selanjutnya adalah menambahkan logika untuk mengelola data pengguna. Berikut adalah kode lengkap yang dimasukkan ke dalam KelolaUserController:

Tabel 6. 15 Kode Dari Controller Kelola User

```
<?php

namespace App\Http\Controllers;

use App\Models\User;
use App\Models\Role;
```

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Rules\Password;

class KelolaUserController extends Controller
{
    public function index()
    {
        $users = User::with('role')->get();
        return view('dashboard.kelolauser.index', compact('users'));
    }

    public function create()
    {
        $roles = Role::all();
        return view('dashboard.kelolauser.create', compact('roles'));
    }

    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'nama' => 'required|max:255',
            'username' => 'required|min:3|max:255|unique:users',
            'email' => 'required|email:dns|unique:users',
            'password' => ['required', 'confirmed', Password::min(6)],
            'id_role' => 'required|exists:roles,id'
        ], [
            'nama.required' => 'Nama harus diisi',
            'username.required' => 'Username harus diisi',
            'username.min' => 'Username minimal 3 karakter',
            'username.unique' => 'Username sudah digunakan',
            'email.required' => 'Email harus diisi',
            'email.email' => 'Format email tidak valid',
            'email.unique' => 'Email sudah digunakan',
            'password.confirmed' => 'Konfirmasi password tidak cocok',
            'id_role.exists' => 'Role tidak valid'
        ]);

        try {
```

```
        $validatedData['password'] = Hash::make($validatedData['password']);
        User::create($validatedData);
        return redirect()->route('users.index')->with('success', 'User berhasil ditambahkan!');
      } catch (\Exception $e) {
        return back()->with('error', 'Terjadi kesalahan saat menambahkan user!')->withInput();
      }
    }

    public function edit(string $id)
    {
      $user = User::findOrFail($id);
      $roles = Role::all();
      return view('dashboard.kelolauser.edit', compact('user', 'roles'));
    }

    public function update(Request $request, $id)
    {
      $user = User::findOrFail($id);

      $validatedData = $request->validate([
        'nama' => 'required|max:255',
        'username' => 'required|min:3|max:255|unique:users,username,'.$id,
        'email' => 'required|email:dns|unique:users,email,'.$id,
        'id_role' => 'required|exists:roles,id',
        'password' => 'nullable|required_with:password_confirmation|confirmed|min:6'
      ]);

      if ($request->filled('password')) {
        $validatedData['password'] = Hash::make($validatedData['password']);
      } else {
        unset($validatedData['password']);
      }
```

```
        $user->update($validatedData);
        return redirect()->route('users.index')->with('success', 'User berhasil
diupdate!');
    }

    public function destroy(string $id)
    {
        User::findOrFail($id)->delete();
        return redirect()->route('users.index')->with('success', 'User berhasil
dihapus!');
    }
}
```

6.6 Tampilan Kelola User (resources/views/dashboard/kelolauser/)

Setelah membuat KelolaUserController, langkah selanjutnya adalah menyiapkan tampilan (*view*) yang digunakan untuk menampilkan data *user*, menambahkan *user* baru, dan mengedit data *user*. dibuat tiga *file* tampilan utama dalam folder resources/views/dashboard/kelolauser/, yaitu:

1. Buat *file* index.blade.php didalamnya
   File ini digunakan untuk menampilkan daftar pengguna dalam bentuk tabel, lengkap dengan informasi nama, *username*, *email*, dan peran (*role*) masing-masing pengguna. Juga tersedia tombol aksi seperti Edit dan Hapus pada setiap baris data.
   Sebelum membuat *file*, pastikan folder kelolauser sudah tersedia. Jika belum, jalankan perintah berikut:

   Tabel 6. 16 membuat Tampilan Kelola User

   ```
   mkdir -p resources/views/dashboard/kelolauser
   ```

   Lalu buat *file* index.blade.php dengan perintah:

   Tabel 6. 17 Membuat Tampilan Kelola User 2

   ```
   touch resources/views/dashboard/kelolauser/index.blade.php
   ```

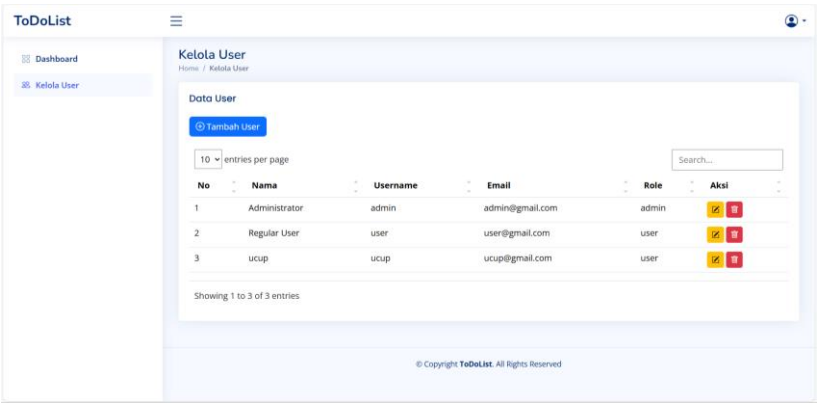
Selanjutnya, buka *file* tersebut dan masukkan sintaks *Blade* awal seperti berikut:

Tabel 6. 18 Kode Dari Tampilan Kelola User

```
@extends('dashboard.layouts.main')

@section('container')
<div class="pagetitle">
 <h1>Kelola User</h1>
 <nav>
  <ol class="breadcrumb">
   <li class="breadcrumb-item"><a
href="/dashboard">Home</a></li>
   <li class="breadcrumb-item active">Kelola User</li>
  </ol>
 </nav>
</div>

<section class="section">
 <div class="row">
  <div class="col-lg-12">
   <div class="card">
    <div class="card-body">
     <h5 class="card-title">Data User</h5>

     @if(session()->has('success'))
     <div class="alert alert-success alert-dismissible fade
show" role="alert">
      {{ session('success') }}
      <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>
     </div>
     @endif

     <a href="{{ route('users.create') }}" class="btn btn-
primary mb-3"><i class="bi bi-plus-circle"></i> Tambah
User</a>

     <!-- Table with stripped rows -->
```

```
            <table class="table datatable">
              <thead>
                <tr>
                  <th>No</th>
                  <th>Nama</th>
                  <th>Username</th>
                  <th>Email</th>
                  <th>Role</th>
                  <th>Aksi</th>
                </tr>
              </thead>
              <tbody>
                @foreach($users as $user)
                <tr>
                  <td>{{ $loop->iteration }}</td>
                  <td>{{ $user->nama }}</td>
                  <td>{{ $user->username }}</td>
                  <td>{{ $user->email }}</td>
                  <td>{{ $user->role->nama }}</td>
                  <td>
                    <a href="{{ route('users.edit', $user->id) }}"
class="btn btn-sm btn-warning">
                      <i class="bi bi-pencil-square"></i>
                    </a>
                    <button type="button" class="btn btn-sm btn-
danger" data-bs-toggle="modal"
                        data-bs-target="#deleteModal{{ $user->id }}">
                      <i class="bi bi-trash"></i>
                    </button>
                  </td>
                </tr>

                <!-- Delete Modal -->
                <div class="modal fade" id="deleteModal{{ $user->id
}}" tabindex="-1">
                  <div class="modal-dialog">
                    <div class="modal-content">
                      <div class="modal-header">
                        <h5 class="modal-title">Konfirmasi Hapus</h5>
```

```
                    <button type="button" class="btn-close" data-
        bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                 Apakah Anda yakin ingin menghapus user
        <strong>{{ $user->nama }}</strong>?
                </div>
                <div class="modal-footer">
                 <button type="button" class="btn btn-
        secondary" data-bs-dismiss="modal">Batal</button>
                 <form action="{{ route('users.destroy', $user-
        >id) }}" method="POST" class="d-inline">
                  @method('delete')
                  @csrf
                  <button type="submit" class="btn btn-
        danger">Hapus</button>
                 </form>
                </div>
               </div>
              </div>
              @endforeach
             </tbody>
            </table>
           </div>
          </div>
         </div>
        </section>
@endsection
```



Gambar 6. 5 Tampilan Utama Kelola User

2. Buat *file* create.blade.php didalamnya
   File ini digunakan untuk menampilkan *form* pembuatan *user* baru. Pengguna akan diminta mengisi data seperti nama, *username, email, password, konfirmasi password,* dan *role.*
   Untuk membuat *file* ini, jalankan perintah berikut:

Tabel 6. 19 Membuat Tampilan Tambah User

```
touch resources/views/dashboard/kelolauser/create.blade.php
```

Setelah *file* berhasil dibuat, tambahkan kode berikut untuk menampilkan struktur dasar halaman *form*:

Tabel 6. 20 Kode Dari Tampilan Tambah User

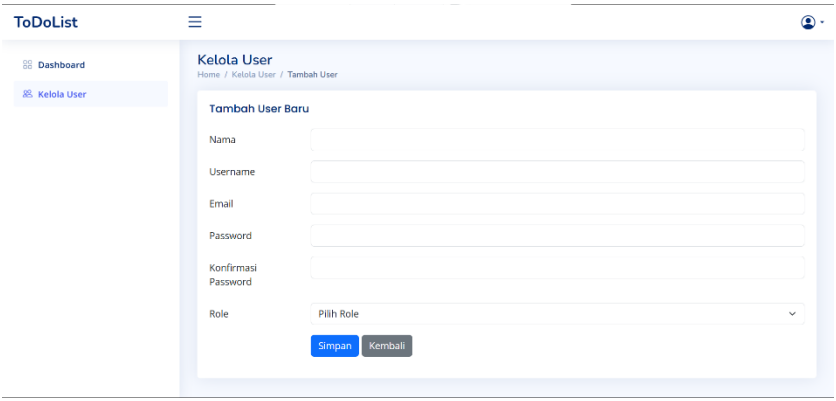
```
@extends('dashboard.layouts.main')

@section('container')
<div class="pagetitle">
 <h1>Kelola User</h1>
 <nav>
  <ol class="breadcrumb">
   <li class="breadcrumb-item"><a
href="/dashboard">Home</a></li>
   <li class="breadcrumb-item"><a href="{{
route('users.index') }}">Kelola User</a></li>
```

```
            <li class="breadcrumb-item active">Tambah User</li>
          </ol>
        </nav>
      </div>

      <section class="section">
        <div class="row">
          <div class="col-lg-12">
            @if(session()->has('error'))
              <div class="alert alert-danger alert-dismissible fade show" role="alert">
                {{ session('error') }}
                <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
              </div>
            @endif

            <div class="card">
              <div class="card-body">
                <h5 class="card-title">Tambah User Baru</h5>

                <!-- Form Tambah User -->
                <form action="{{ route('users.store') }}" method="POST">
                  @csrf
                  <div class="row mb-3">
                    <label for="nama" class="col-sm-2 col-form-label">Nama</label>
                    <div class="col-sm-10">
                      <input type="text" class="form-control @error('nama') is-invalid @enderror"
                        id="nama" name="nama" value="{{ old('nama') }}" required maxlength="255">
                      @error('nama')
                      <div class="invalid-feedback">{{ $message }}</div>
                      @enderror
                    </div>
                  </div>
```

```
            <div class="row mb-3">
              <label for="username" class="col-sm-2 col-form-
label">Username</label>
              <div class="col-sm-10">
                <input type="text" class="form-control
@error('username') is-invalid @enderror"
                    id="username" name="username" value="{{
old('username') }}" required minlength="3">
                @error('username')
                <div class="invalid-feedback">{{ $message
}}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="email" class="col-sm-2 col-form-
label">Email</label>
              <div class="col-sm-10">
                <input type="email" class="form-control
@error('email') is-invalid @enderror"
                    id="email" name="email" value="{{
old('email') }}" required>
                @error('email')
                <div class="invalid-feedback">{{ $message
}}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="password" class="col-sm-2 col-form-
label">Password</label>
              <div class="col-sm-10">
                <input type="password" class="form-control
@error('password') is-invalid @enderror"
                    id="password" name="password" required
minlength="6">
```

```
                @error('password')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="password_confirmation" class="col-sm-2 col-form-label">Konfirmasi Password</label>
              <div class="col-sm-10">
                <input type="password" class="form-control"
                    id="password_confirmation"
name="password_confirmation" required>
              </div>
            </div>

            <div class="row mb-3">
              <label for="id_role" class="col-sm-2 col-form-label">Role</label>
              <div class="col-sm-10">
                <select class="form-select @error('id_role') is-invalid @enderror"
                    id="id_role" name="id_role" required>
                  <option value="">Pilih Role</option>
                  @foreach($roles as $role)
                    <option value="{{ $role->id }}" {{ old('id_role') == $role->id ? 'selected' : '' }}>
                      {{ $role->nama }}
                    </option>
                  @endforeach
                </select>
                @error('id_role')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
              </div>
            </div>
```

```
                    <div class="row mb-3">
                      <div class="col-sm-10 offset-sm-2">
                        <button type="submit" class="btn btn-
primary">Simpan</button>
                        <a href="{{ route('users.index') }}" class="btn btn-
secondary">Kembali</a>
                      </div>
                    </div>

                  </form><!-- End Form Tambah User -->

                </div>
              </div>
            </div>
          </div>
        </section>
@endsection
```



Gambar 6. 6 Tampilan kelola User (Tambah)

3. Buat *file* edit.blade.php didalamnya

   File ini digunakan untuk menampilkan *form* pengeditan *user*. Formulir ini mirip dengan *form* tambah *user*, namun data sudah terisi (*prefilled*). Untuk membuat *file* ini, jalankan perintah berikut:

Tabel 6. 21 Membuat Edit User

```
touch resources/views/dashboard/kelolauser/edit.blade.php
```

Setelah *file* dibuat, tambahkan struktur awal tampilan berikut:

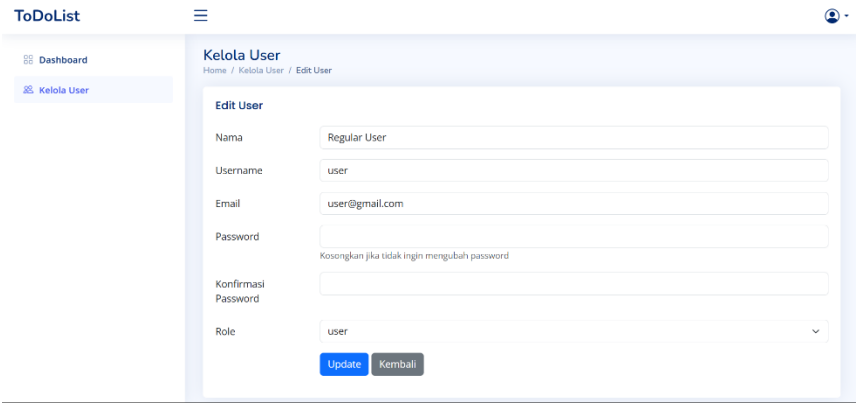
Tabel 6. 22 Kode Dari Edit User

```
@extends('dashboard.layouts.main')

@section('container')
<div class="pagetitle">
 <h1>Kelola User</h1>
 <nav>
  <ol class="breadcrumb">
   <li class="breadcrumb-item"><a
href="/dashboard">Home</a></li>
   <li class="breadcrumb-item"><a href="{{
route('users.index') }}">Kelola User</a></li>
   <li class="breadcrumb-item active">Edit User</li>
  </ol>
 </nav>
</div>

<section class="section">
 <div class="row">
  <div class="col-lg-12">
   @if(session()->has('error'))
    <div class="alert alert-danger alert-dismissible fade
show" role="alert">
     {{ session('error') }}
     <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>
    </div>
   @endif

   <div class="card">
    <div class="card-body">
     <h5 class="card-title">Edit User</h5>

     <form action="{{ route('users.update', $user->id) }}"
method="POST">
      @method('put')
```

```
            @csrf
            <div class="row mb-3">
              <label for="nama" class="col-sm-2 col-form-
label">Nama</label>
              <div class="col-sm-10">
                <input type="text" class="form-control
@error('nama') is-invalid @enderror"
                  id="nama" name="nama" value="{{ old('nama',
$user->nama) }}" required>
                @error('nama')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="username" class="col-sm-2 col-form-
label">Username</label>
              <div class="col-sm-10">
                <input type="text" class="form-control
@error('username') is-invalid @enderror"
                  id="username" name="username" value="{{
old('username', $user->username) }}" required>
                @error('username')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="email" class="col-sm-2 col-form-
label">Email</label>
              <div class="col-sm-10">
                <input type="email" class="form-control
@error('email') is-invalid @enderror"
                  id="email" name="email" value="{{ old('email',
$user->email) }}" required>
                @error('email')
                <div class="invalid-feedback">{{ $message }}</div>
```

```
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="password" class="col-sm-2 col-form-label">Password</label>
              <div class="col-sm-10">
                <input type="password" class="form-control @error('password') is-invalid @enderror"
                    id="password" name="password">
                <small class="text-muted">Kosongkan jika tidak ingin mengubah password</small>
                @error('password')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
              </div>
            </div>

            <div class="row mb-3">
              <label for="password_confirmation" class="col-sm-2 col-form-label">Konfirmasi Password</label>
              <div class="col-sm-10">
                <input type="password" class="form-control"
                    id="password_confirmation"
name="password_confirmation">
              </div>
            </div>

            <div class="row mb-3">
              <label for="id_role" class="col-sm-2 col-form-label">Role</label>
              <div class="col-sm-10">
                <select class="form-select @error('id_role') is-invalid @enderror"
                    id="id_role" name="id_role" required>
                  <option value="">Pilih Role</option>
                  @foreach($roles as $role)
                  <option value="{{ $role->id }}"
```

```
                    {{ old('id_role', $user->id_role) == $role->id ?
        'selected' : '' }}>
                    {{ $role->nama }}
                  </option>
                 @endforeach
                </select>
                @error('id_role')
                <div class="invalid-feedback">{{ $message }}</div>
                @enderror
               </div>
              </div>

              <div class="row mb-3">
               <div class="col-sm-10 offset-sm-2">
                <button type="submit" class="btn btn-
        primary">Update</button>
                 <a href="{{ route('users.index') }}" class="btn btn-
        secondary">Kembali</a>
               </div>
              </div>

             </form>

            </div>
           </div>
          </div>
         </div>
        </section>
@endsection
```

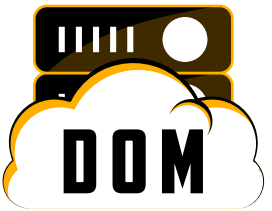


Gambar 6. 7 Tampilan Kelola User (Edit)

Setelah itu lanjut ke proses deployment ke hosting online selamat menikmati guys!!.

BAB 7
DEPLOYMENT KE *HOSTING ONLINE* MENGGUNAKAN *DOM CLOUD*

7.1 Pengenalan *Dom Cloud*



Gambar 7. 1 Dom Cloud

Dom Cloud adalah layanan *hosting* berbasis *cloud* yang memungkinkan pengguna untuk mengelola dan meng-*hosting website* dengan mudah. Layanan ini mendukung berbagai *framework* seperti Laravel, *PHP*, *Node.js*, dan lainnya.

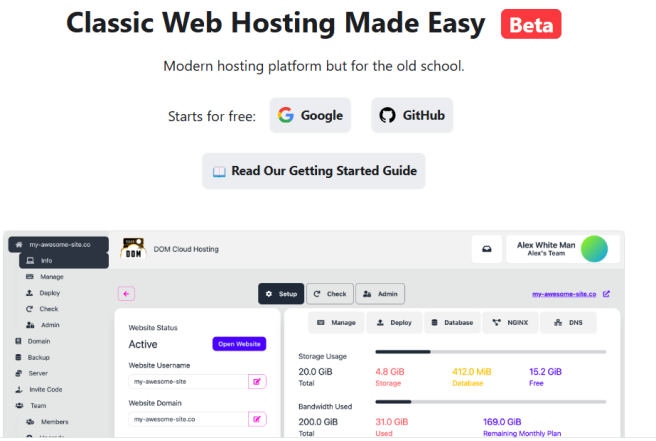
Untuk memulai *deployment* aplikasi *Laravel*, langkah pertama yang harus dilakukan adalah membuat akun di *Dom Cloud* melalui situs resmi *https://domcloud.co*. Setelah berhasil mendaftar, pengguna dapat masuk ke akun dan diarahkan ke halaman *Dashboard*. Jika belum *login*, sistem akan secara otomatis mengarahkan ke halaman *login* sebelum dapat mengakses fitur pembuatan proyek.

Keunggulan *Hosting* di *Dom Cloud*:

1. Gratis dan mudah digunakan, *Dom Cloud* menyediakan opsi gratis untuk pengembangan proyek.
2. Dukungan berbagai framework, mendukung Laravel, *PHP*, *Node.js*, dan lainnya.
3. Integrasi dengan *GitHub,* memudahkan pengelolaan kode sumber dan *deployment* otomatis.
4. Pengelolaan *domain* kustom, memungkinkan penggunaan *domain* sendiri atau *default* *.domcloud.dev.

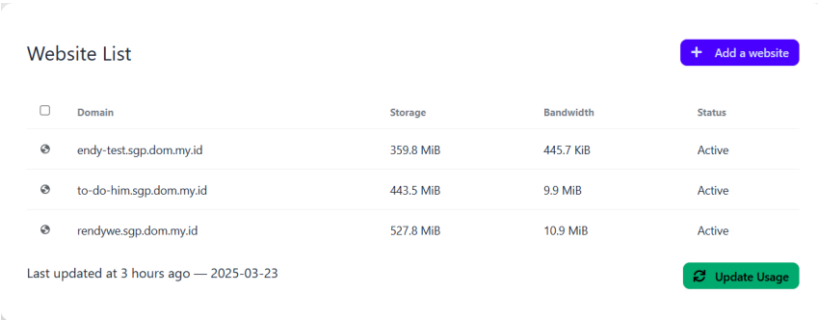
7.2 Persiapan *Deployment Website*

Untuk memulai *deployment* aplikasi Laravel, langkah pertama yang harus dilakukan adalah membuat akun di *Dom Cloud* melalui situs resmi *https://domcloud.co*. Setelah berhasil mendaftar, pengguna dapat masuk ke akun dan diarahkan ke halaman *Dashboard*. Jika belum *login*, sistem akan secara otomatis mengarahkan ke halaman *login* sebelum dapat mengakses fitur pembuatan proyek. Anda bisa *Login* menggunakan *google* atau *github*



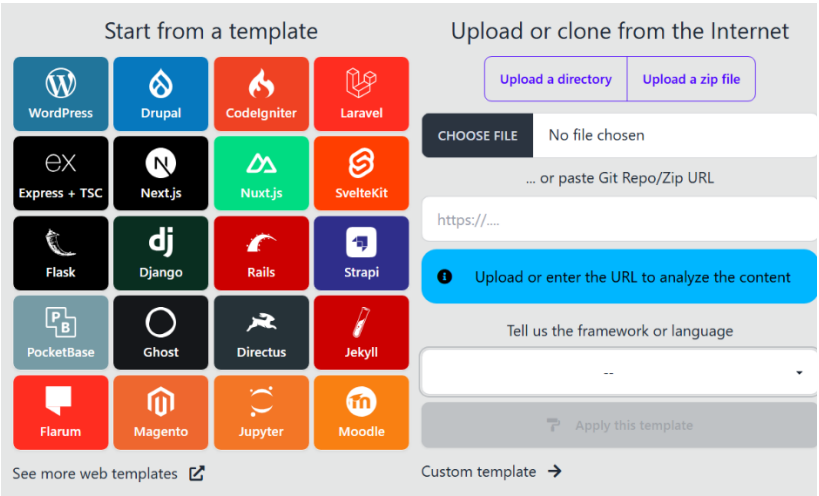
Gambar 7. 2 Halaman Login Dom Cloud

Setelah masuk ke *Dashboard***,** pengguna dapat memulai proyek baru dengan mengklik tombol *"New Project"***.** Pada halaman ini, terdapat beberapa pilihan *framework*, dan untuk proyek ini, pilih Laravel sebagai *framework* yang digunakan.



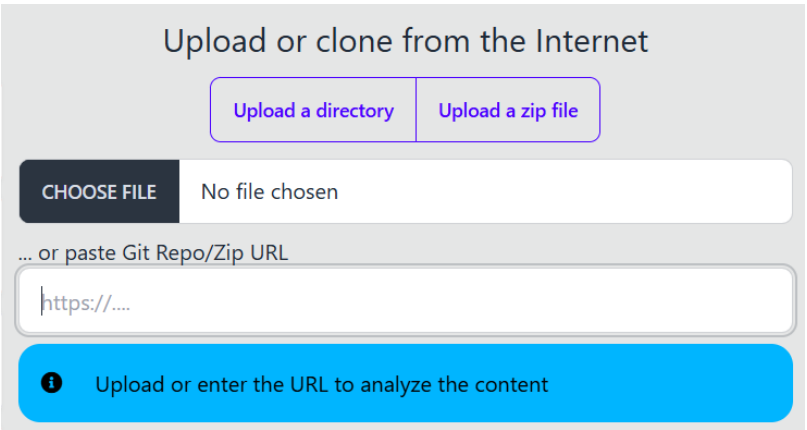
Gambar 7. 3 Halaman Dashboard Utama

Langkah selanjutnya adalah memasukkan nama proyek sesuai keinginan. Pastikan nama proyek yang dimasukkan mudah dikenali dan relevan dengan aplikasi yang sedang dikembangkan. Setelah itu, klik tombol *"Create"* untuk memulai proses *deployment*.



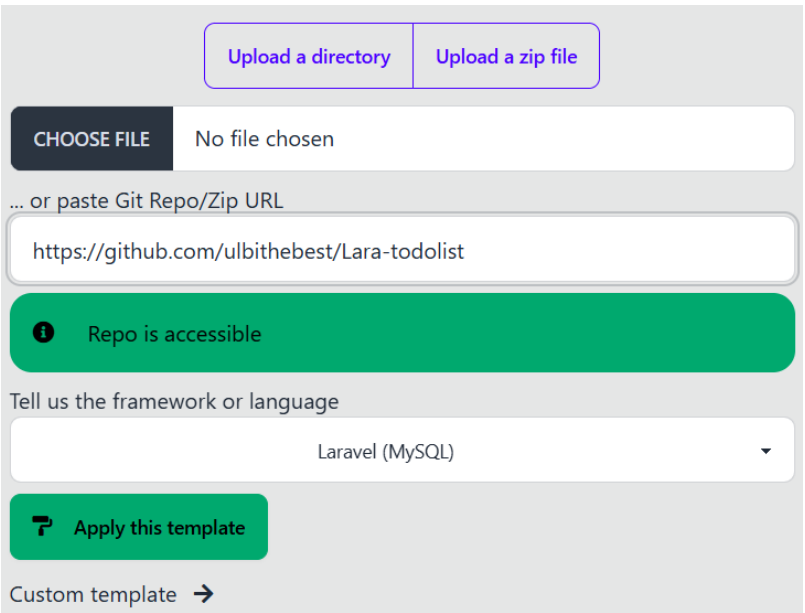
Gambar 7. 4 Tampilan Membuat Website

Jika Anda memiliki proyek yang sudah ada di *GitHub* atau di komputer lokal, Anda dapat menggunakan opsi ini. Caranya cukup mudah, Anda bisa langsung mengunggah folder proyek atau menyalin *URL repository* dari *GitHub*. Setelah itu, pilih *framework* atau bahasa pemrograman yang sesuai dengan proyek Anda. Pilih salah satu contoh *github* yang bisa anda coba *https://github.com/ulbithebest/Lara-todolist*.



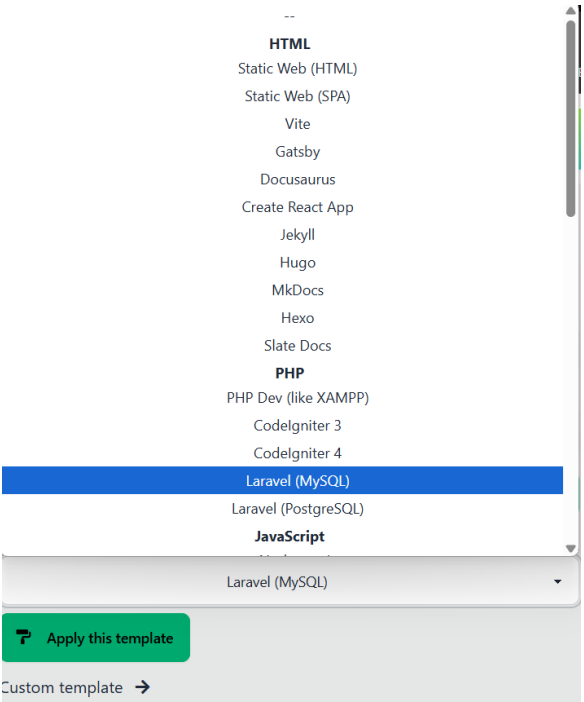
Gambar 7. 5 Tampilan Membuat Website 2

Jika *repositori GitHub* bersifat *private*, sistem akan meminta Anda untuk menginstal kunci publik *SSH* agar dapat mengkloning proyek dari *server*. Selain itu, *GitHub* juga memungkinkan integrasi dengan *webhook* untuk menjaga situs tetap sinkron dengan perubahan terbaru dari *repository* Anda.



Gambar 7. 6 Tampilan Membuat Website 3

Setelah sudah memasukan *URL github* atau folder *website* anda selanjutnya klik "*analyze the content*". Jika berwarna hijau berarti *URL github* anda terbaca disini sebagai contoh menggunakan *https://github.com/ulbithebest/Lara-todolist*. Dan terakhir pilih *Framework* atau Bahasa, jika anda menggunakan *link* contoh pilih Laravel dan *mysql* sebagai *database*-nya. Lalu klik "*apply this templete*"



Gambar 7. 7 Tampilan Membuat Website 4

7.3 Proses *Deployment Website*

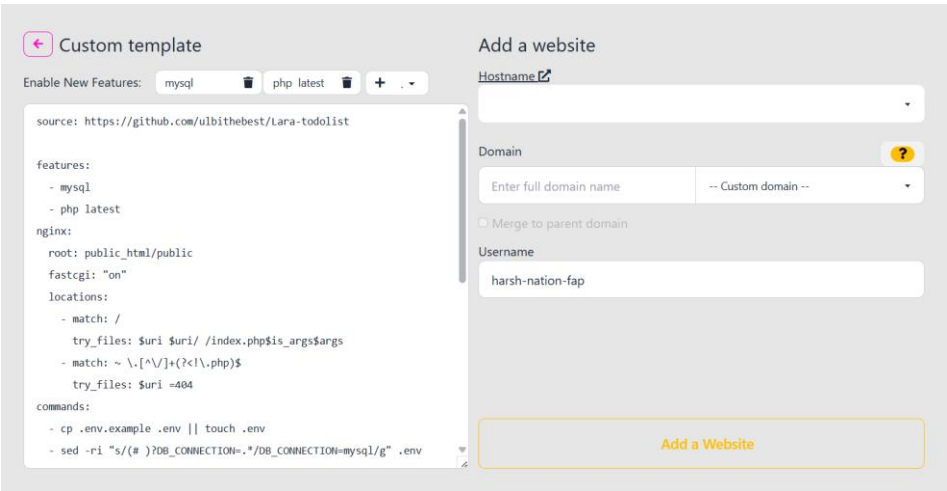
Ketika persiapan *website* sudah siap selanjutnya eksekusi karena sebelumnya sudah memilih *template framework* atau bahasa maka *dom cloud* otomatis membuat *command* sesuai dengan yang anda pilih.

```
commands:
  - cp .env.example .env || touch .env
  - sed -ri "s/(# )?DB_CONNECTION=.*/DB_CONNECTION=mysql/g" .env
  - sed -ri "s/(# )?DB_HOST=.*/DB_HOST=localhost/g" .env
  - sed -ri "s/(# )?DB_PORT=.*/DB_PORT=3306/g" .env
  - sed -ri "s/(# )?DB_DATABASE=.*/DB_DATABASE=${DATABASE}/g" .env
  - sed -ri "s/(# )?DB_USERNAME=.*/DB_USERNAME=${USERNAME}/g" .env
  - sed -ri "s/(# )?DB_PASSWORD=.*/DB_PASSWORD=${MYPASSWD}/g" .env
  - sed -ri "s/APP_URL=.*/APP_URL=https:\/\/${DOMAIN}/g" .env
  - sed -ri "s/LOG_STACK=single/LOG_STACK=daily,errorlog/g" .env
  - composer install
  - php artisan migrate:fresh --seed || true
  - php artisan key:generate
  - php artisan storage:link
  - npm install
  - npm run build
```

Gambar 7. 8 Config Deployment

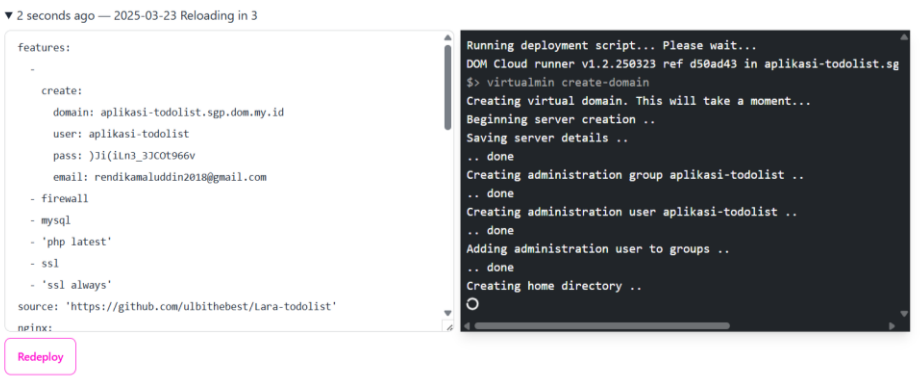
Selanjutnya isikan *hostname*, *domain*, dan *username*. *Hostname* menunjukkan lokasi *server* tempat *website* akan di-*deploy*. Informasi ini mencakup spesifikasi *server*, *latensi* dari lokasi Anda, serta alamat *IP* yang nantinya dapat digunakan.

Domain Di sini, Anda dapat menentukan nama *domain* untuk situs *web* Anda. Bisa menggunakan *domain* khusus yang sudah Anda miliki atau memilih *subdomain *.domcloud.dev* jika menggunakan paket berbayar.



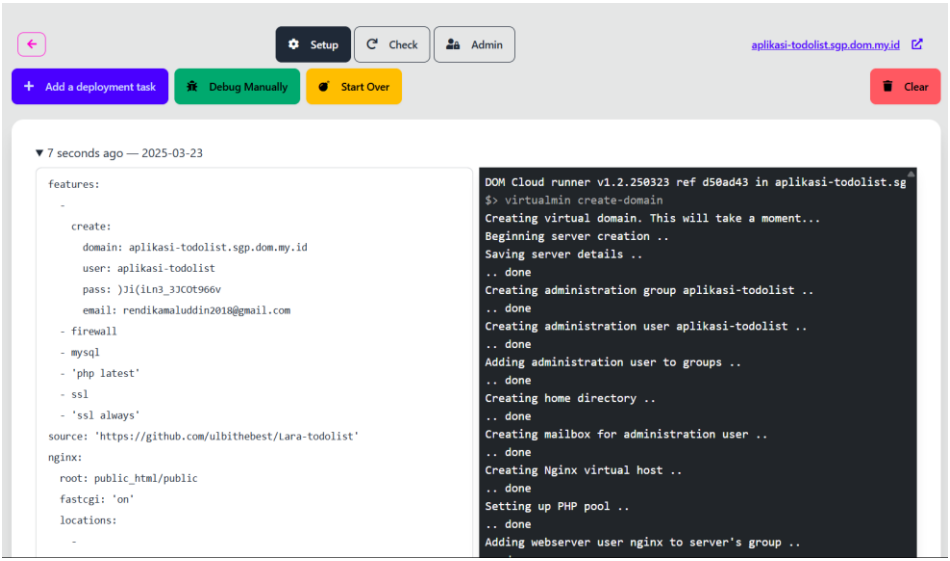
Gambar 7. 9 Tahap Deployment

Setelah semua pengaturan selesai, klik "*Add a Website*" untuk membuat proyek. Anda akan diarahkan ke halaman pemrosesan *deployment*, di mana Anda dapat melihat progres instalasi.



Gambar 7. 10 Proses Deployment

Anda hanya perlu menunggu hingga proses selesai. Setelah berhasil, akan muncul pesan "*Your website is ready!*", beserta tombol untuk langsung membuka *website* yang telah dibuat.



Gambar 7. 11 Tampilan Website Sudah Bisa Diakses

DAFTAR PUSTAKA

Az, M., Pane, S. F., & Awangga, R. M. (2021). Cryptography: Perancangan Middleware Web Service Encryptor menggunakan Triple Key MD5. Base64, dan AES. *Jurnal Tekno Insentif*, *15*(2), 65–75. https://doi.org/10.36787/jti.v15i1.497

Feblia, K. N., Anggarah, R., Villareal, Y., Amanda, K. R., Mardiyah, Q. A., Amanda, F. C., Mursyidah, A. S., & Aritonang, M. D. V. (2025). Studi Perbandingan Penerapan Pola Model-View-Controller (MVC) dalam Lima Framework Web Populer: Laravel, Django, Ruby on Rails, Asp. net MVC, Spring MVC. *Indonesian Journal of Computer Science and Engineering*, *2*(01), 16–22.

Holop, R. (2022). *NCF TO-DO ' ER : A TO-DO LIST WEB APPLICATION THAT ASSISTS*.

Louis, L., & Goodrum Kershner, J. (2019). *Compartments: A To-Do List App for Busy People*.

Mahendra, F. W. P., & Beeh, Y. R. (2025). WEBSITE MANAJEMEN RESTORAN AYAM GORENG SAM-SAM MENGGUNAKAN PHP DAN FRAMEWORK LARAVEL. *JIPI (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *10*(3), 1859–1868.

Pane, S. F., Sari, W. K., & Wicaksono, Z. A. (2020). *Membuat Aplikasi Pengolahan Data Administrasi Barang Menggunakan Aplikasi Apex Online*. Kreatif. https://books.google.co.id/books?id=3s3XDwAAQBAJ

Subecz, Z. (2021). Web-development with Laravel framework. *Gradus*, *8*(1), 211–218. https://doi.org/10.47833/2021.1.csc.006

GLOSARIUM

A

***Admin*:** Pengguna dengan hak akses penuh untuk mengelola sistem, termasuk mengelola pengguna lain dan memantau seluruh data aplikasi.

Autentikasi: Proses verifikasi identitas pengguna agar dapat mengakses aplikasi dengan aman, biasanya melalui *login* dan *password*.

B

***Blade*:** *Template engine* bawaan Laravel yang digunakan untuk membangun tampilan aplikasi dengan sintaks sederhana dan dinamis.

C

***Controller*:** Komponen dalam arsitektur *MVC* yang bertugas menghubungkan *Model* dan *View*, serta menangani logika aplikasi.

***CRUD* (*Create*, *Read*, *Update*, *Delete*):** Empat operasi dasar dalam pengelolaan data di aplikasi. Dalam aplikasi To-Do-List, operasi ini digunakan untuk mengelola daftar tugas.

D

***Dashboard*:** Halaman utama untuk admin atau user setelah *login*, berfungsi sebagai pusat kendali aplikasi.

***Database*:** Kumpulan data yang disimpan secara terstruktur dan dapat diakses oleh aplikasi, misalnya data pengguna dan tugas.

E

***Eloquent ORM*:** Fitur bawaan Laravel untuk mempermudah interaksi dengan *database* menggunakan sintaks PHP berbasis objek.

F

G

H

***Hosting / Deployment*:** Proses menempatkan aplikasi Laravel ke server agar bisa diakses secara online.

I

J

K

L

Laravel: *Framework* PHP berbasis *MVC* yang digunakan untuk membangun aplikasi web dengan cepat, aman, dan terstruktur.

M

***Migration*:** Fitur Laravel untuk mengatur struktur *database* dengan kode, sehingga perubahan database lebih mudah dilacak dan dikelola.

***Middleware*:** Lapisan filter dalam Laravel yang memproses request sebelum sampai ke *controller*, misalnya untuk membatasi akses berdasarkan role.

***Model*:** Komponen dalam arsitektur *MVC* yang merepresentasikan tabel di database serta mengatur logika bisnis terkait data.

***MVC* (*Model-View-Controller*):** Pola arsitektur aplikasi yang memisahkan data (*Model*), tampilan (*View*), dan logika (*Controller*) agar kode lebih rapi dan terstruktur.

N

O

P

Q

R

***Role-Based Access*:** Mekanisme pembatasan akses berdasarkan peran pengguna, misalnya admin atau user biasa.

S

***Seeder*:** Fitur Laravel untuk mengisi *database* dengan data awal secara otomatis.

T

***Task* (Tugas):** Objek utama dalam aplikasi To-Do-List yang berisi informasi pekerjaan, seperti judul, deskripsi, *deadline*, dan status.

U

***User* (Pengguna):** Orang yang menggunakan aplikasi, baik sebagai admin maupun *user* biasa yang dapat membuat dan mengelola tugas.

V

***View*:** Bagian dari *MVC* yang bertanggung jawab menampilkan data ke pengguna dalam bentuk halaman *web*.

W

X

XAMPP: Paket perangkat lunak yang menyediakan Apache, MySQL, PHP, dan Perl untuk membangun dan menjalankan aplikasi *web* di komputer lokal.

Y

KREDIT GAMBAR

https://domcloud.co

INDEKS

TENTANG PENULIS

Rifky Najra Adipura, lahir di Kota Cimahi, Jawa Barat pada tanggal 03 Juli 2005. Ia adalah seorang penulis dan pengembang perangkat lunak dengan fokus pada pengembangan aplikasi web menggunakan Laravel. Melalui buku ini, ia menyalurkan semangatnya untuk berbagi ilmu dengan menyajikan materi teknis yang kompleks menjadi langkah-langkah yang praktis dan mudah diikuti.

Ode Andi Alamsyah, lahir di Kabupaten Buton, Sulawesi Tenggara, pada 02 Maret 2005. Ia menempuh pendidikan di SD Negeri 3 Laompo, SMP Negeri 1 Batauga, dan SMK Negeri 2 BAUBAU. Saat ini, Ode melanjutkan pendidikan di Bandung di Universitas Logistik dan Bisnis Internasional (ULBI), jurusan D4 Teknik Informatika.

Risyad Ridwansyah

Rendy Kamaluddin, lahir di kota Bandung, Jawa Barat pada tanggal 24 November 2004. Menempuh Pendidikan Tingkat dasar hingga menengah atas di kota Bandung, Jawa Barat. Saat buku ini ditulis, penulis sedang menempuh Pendidikan perguruan tinggi jurusan DIV Teknik Informatika di Universitas Logistik & Bisnis Internasional. ia menyalurkan semangatnya untuk berbagi ilmu dengan menyajikan materi teknis yang kompleks menjadi langkah-langkah yang praktis dan mudah diikuti.

Nyi Raden Nuraini Siti Fathonah

Syafrial Fachri Pane lahir di Medan, Sumatera Utara pada April 1988. Ia memperoleh gelar sarjana informatika dari Universitas Pasundan dan magister informatika dari Universitas Bina Nusantara, Bandung, pada tahun 2019 dan 2021, masing-masing. Saat ini, ia sedang menempuh program doktoral di Universitas Telkom, Bandung. Ia terlibat dalam penelitian di bidang ilmu data dan pembelajaran mesin. Minat penelitiannya meliputi analisis data dan pembelajaran mesin. Disertasi penelitiannya berfokus pada Pembelajaran Mesin Metaheuristik Multi-objektif Hibrida untuk Modelling Pandemi.

Amri Yanuar

Irfan (Guru SMK)

Buku ini adalah panduan lengkap bagi pemula maupun pengembang yang ingin memahami proses pembuatan aplikasi To-Do List dari nol.
Dengan pendekatan praktis dan bertahap, buku ini membahas mulai dari dasar-dasar perancangan antarmuka, struktur database, hingga implementasi fitur utama seperti menambah, mengedit, dan menghapus tugas. Setiap langkah dijelaskan dengan kode yang mudah dipahami, serta disertai dengan penjelasan logis agar pembaca dapat menerapkannya dalam proyek mereka sendiri. Tidak hanya itu, buku ini juga mencakup tips optimasi, penggunaan teknologi modern, dan best practice dalam pengembangan aplikasi. Cocok untuk mahasiswa, pengembang pemula, atau siapa saja yang ingin memperdalam keterampilan pemrograman mereka.
Bangun aplikasi pertama Anda sekarang dan kuasai dasar-dasar pengembangan perangkat lunak!

bukuPedia