

BAB V

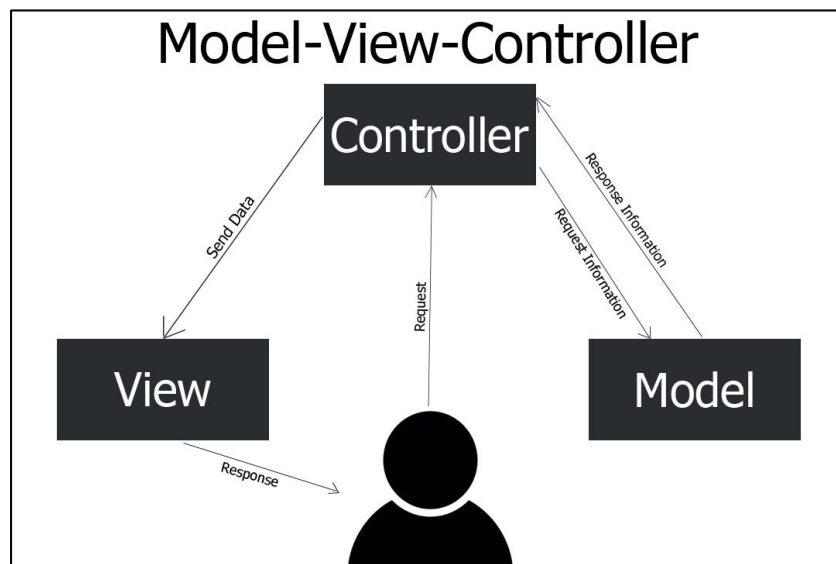
LARAVEL MVC, ROUTE DAN BLADE FRAGMENT

1. MVC

MVC adalah sebuah arsitektur perancangan kode program. Tujuannya untuk memecah kode program utama menjadi 3 komponen terpisah dengan tugas yang spesifik. Ketiga komponen tersebut adalah:

- Pengaksesan database, disebut sebagai **Model**.
- Tampilan design (user interface), disebut sebagai **View**.
- Alur logika program, disebut sebagai **Controller**.

Ide awal dari perlunya konsep MVC adalah agar aplikasi yang dibuat bisa mudah dikelola dan dikembangkan, terutama untuk aplikasi besar. Arsitektur MVC ini ditawarkan oleh Bahasa pemrograman PHP sebagai pengganti bentuk Object Oriented Programming. Karena PHP tidak dapat mengimplementasikan OOP secara baik.



Gambar 1. Arsitektur MVC pada Laravel

Ini adalah contoh alur yang terjadi dalam sebuah aplikasi yang menerapkan konsep MVC. Kita berangkat dari user yang sedang membuka sebuah web browser.

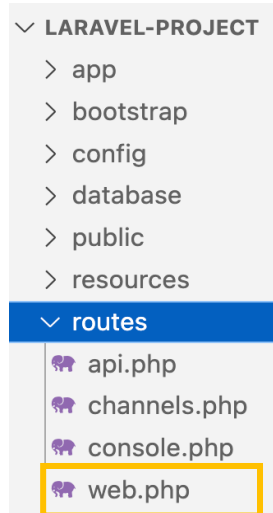
Setiap interaksi yang dilakukan user akan ditangani oleh **Controller**. Misalnya ketika user mengetik alamat situs *www.kis.ibik.ac.id*, maka sebuah controller di server *KIS* akan menangkap “request” tersebut. Atau ketika user selesai mengisi form register dan men-klik tombol submit, file controller akan menerima sebuah proses.

Controller pada dasarnya berisi logika program. Seandainya perlu mengambil data dari database, maka controller akan memanggil **Model**. Model inilah yang bertanggung jawab mengakses database lalu mengembalikan hasilnya kembali ke controller.

Setelah data dari model diterima kembali, controller kemudian meneruskan data tersebut ke dalam **View**. Data ini kemudian diproses sebagai kode HTML dan CSS di dalam view. Inilah yang dilihat oleh user di dalam web browser.

2. Laravel Route

Route berperan sebagai penghubung antara user dengan keseluruhan framework. Dalam Laravel, setiap alamat web yang kita ketik di web browser akan melewati route terlebih dahulu. Route-lah yang menentukan ke mana proses akan dibawa, apakah ke Controller atau ke View. Untuk mengakses file route masuklah kedalam project anda carilah folder routes dan buka file web.php.



Gambar 2. Structure folder Route

2.1. Basic Route

Berikut ini beberapa contoh penggunaan Route secara basic untuk menampilkan sebuah text berdasarkan alamat website:

Web.php

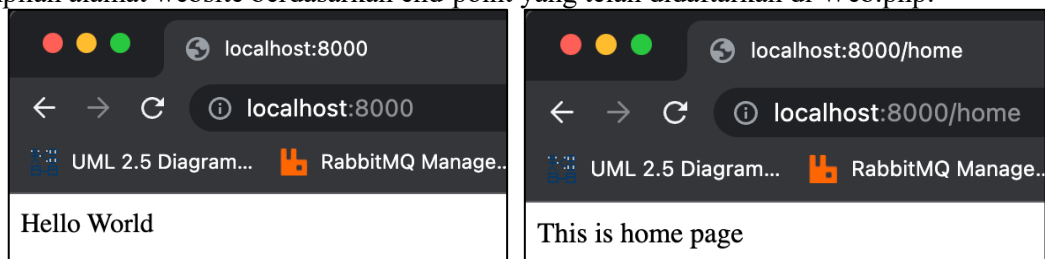
```
Route::get('/', function () {
    return "Hello World";
});
```

```
Route::get('/home', function () {
    return "This is home page";
});
```

Inilah kode program yang dipanggil untuk memproses alamat `http://localhost:8000`. Secara sederhana, penulisan route Laravel mengikuti format berikut:

```
Route::<jenis method>(<alamat URL>,<proses yang dijalankan>)
```

Tampilan alamat website berdasarkan end-point yang telah didaftarkan di Web.php:



Gambar 2.1. Contoh dasar inisialisasi route

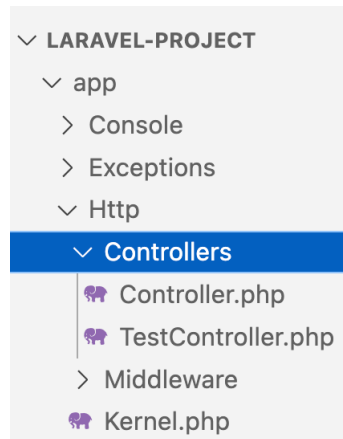
Pertama, kita harus menulis perintah `'Route::'`. Di dalam konsep MVC PHP, tanda `::` merupakan perintah untuk mengakses sebuah static method (atau bisa juga static property) kepunyaan sebuah class, yang dalam contoh ini adalah milik class Route.

Selanjutnya diikuti dengan penulisan `<jenis method>`. Jenis method adalah *'cara sebuah URL diakses'*. `'get'` merupakan salah satu jenis method yang akan dijalankan ketika alamat URL diakses secara normal.

Maksud "normal" di sini adalah dengan mengetik langsung alamat web di address bar web browser atau ketika kita men-klik sebuah link. Nantinya terdapat jenis method lain seperti *post*, *put*, dan *delete*.

2.2. Route dengan basic Controller

Pada penggunaan route kali ini akan memanfaatkan Controller sebagai wadah untuk menampung logika program. Controller merupakan suatu bagian penting dari pemrograman MVC yang berfungsi sebagai penghubung antara View dan model. Didalam controller akan terdapat banyak logika-logika pemrograman untuk menyusun fungsi tertentu. Berbagai pemrosesan juga pada umumnya dilakukan di dalam controller. Secara bawaan file controller akan disimpan pada folder `app/Http/Controllers`.



Gambar 2.2.1. Lokasi directori Controller

Untuk membuat sebuah controller kita dapat membuatnya dengan cara memasukan syntax dibawah ini pada terminal/cmd project Laravel anda:

```
php artisan make:controller [namafilename]
```

```
febryfairuz@Febrys-MacBook-Air laravel-project % php artisan make:controller ProfileController
```

```
INFO Controller [app/Http/Controllers/ProfileController.php] created successfully.
```

Pada contoh syntax diatas, akan membuat sebuah controller bernama ProfileController. Jika berhasil seperti pada contoh diatas maka akan membuat file seperti pada direktori gambar 2.2.1 bernama *ProfileController*. Pada file ProfileController silakan anda membuat sebuah function dengan nama *index*, *identity*, dan *family*.

ProfileController.php

```
<?php

namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;

class ProfileController extends Controller
{
    public function index(){
        return "Welcome to Profile page";
    }

    public function indentity(){
        return "Welcome to Profile page sub menu identity";
    }

    public function family(){
        return "Welcome to Profile page sub menu family";
    }
}
```

Selanjutnya definisikanlah 3 buah route bernama profile yang memanggil controller ProfileController:

```
use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;

Route::get('/profile', [ProfileController::class, 'index']);
Route::get('/profile/identity', [ProfileController::class, 'indentity']);
Route::get('/profile/family', [ProfileController::class, 'family']);
```

Setiap memanggil sebuah controller pada file route anda perlu memanggil file tersebut. Contoh pada script diatas memanggil sebuah file bernama ProfileController untuk ketiga buah end-point. File controller tersebut wajib dipanggil dengan menambahkan:

```
use App\Http\Controllers\ProfileController;
```

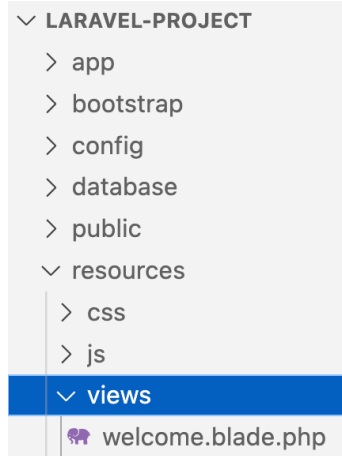
Berikan output dari halaman ini, dan simpan sebagai bentuk **Latihan soal nomor 1**.

2.3. Route dengan Controller dan View

Laravel sudah memiliki 1 view bawaan yang terlihat saat mengakses halaman homepage atau halaman root Laravel.

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

Kode ini bisa dibaca: “Jika halaman root ‘/’ diakses, tampilkan view bernama welcome”. Di dalam Laravel, nama sebuah view mewakili nama suatu file, artinya harus terdapat file bernama ‘welcome’ di dalam folder instalasi Laravel. Letak direktori penyimpanan view di dalam folder resources\views.



Gambar 2.3. Direktori file views laravel

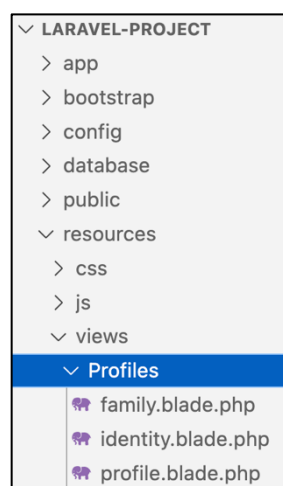
Dalam Laravel, setiap view harus ditulis dengan format berikut:

<nama_file>.blade.php

Sehingga jika kita ingin membuat view baru untuk *profile*, maka nama filenya adalah *profile.blade.php*.

Blade merupakan sebuah templating engine bawaan Laravel. Bahasan lebih lanjut tentang blade akan dibahas dalam section setelah ini. Perlu dipahami bahwa semua file view Laravel **harus** diakhiri dengan **.blade.php**. Selain itu file view juga harus berada di dalam folder `resources\views`.

Melanjutkan contoh kasus route pada ProfileController, bukalah file tersebut dan tambahkan function view untuk memanggil sebuah file dari masing-masing end-point yang telah didaftarkan.



Buatlah 3 buah file view blade yang berada didalam folder Profiles:

- *profile.blade.php*
- *identity.blade.php*, dan
- *family.blade.php*

Setelah membuat ketiga file view blade silakan anda isi file tersebut dengan syntax HTML untuk menampilkan sebuah informasi secara berbeda-beda.

Pada file ProfileController, silakan anda ubah script return output pada setiap function dengan memanggil file view blade:

ProfileController.php

<?php

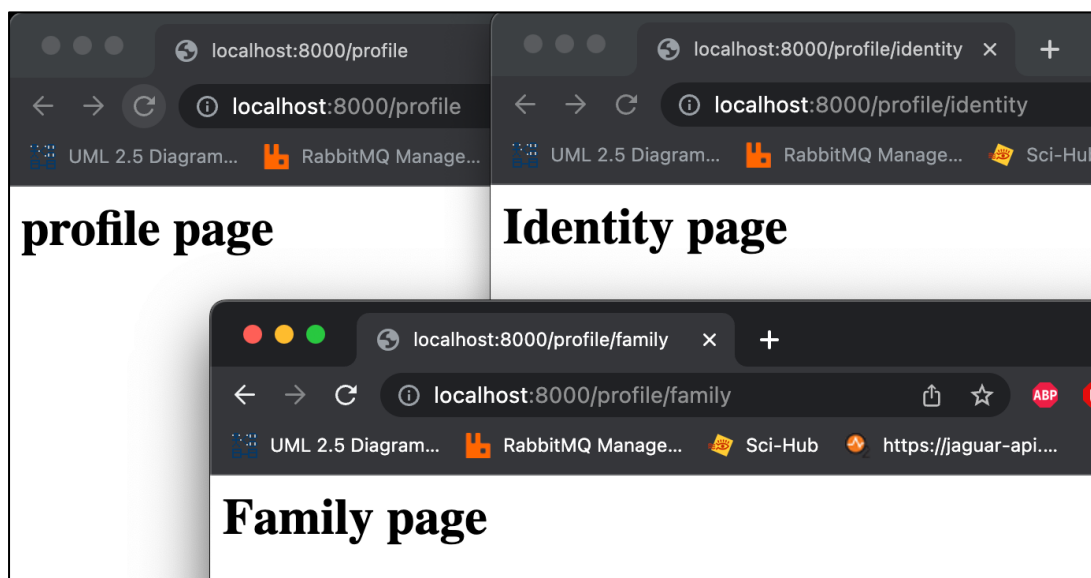
```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProfileController extends Controller
{
    public function index(){
        return view('Profiles.profile');
    }

    public function identity(){
        return view('Profiles.identity');
    }

    public function family(){
        return view('Profiles.family');
    }
}
```



Gambar 2.3.1. Tampilan route dengan controller dan view

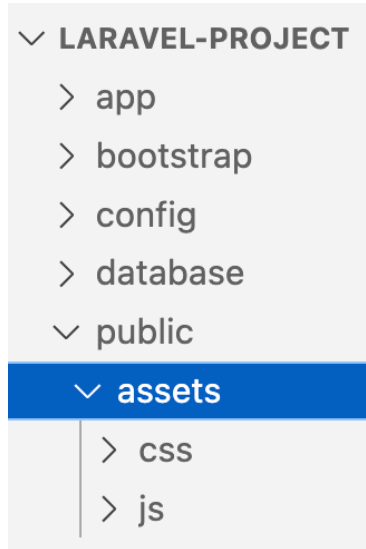
3. Laravel Blade

3.1. Blade Layout

Jika pada module 1-3 anda telah mendesain sebuah website dengan bentuk web DYNAMIC, pada bagian ini kita akan membuat sebuah website dengan bentuk STATIC. Disini akan menggunakan Framework CSS – Bootstrap v.5 untuk mempermudah mengatur layout pada website.

Silakan anda download bundle package bootstrap di: <https://getbootstrap.com/docs/5.3/getting-started/download/>

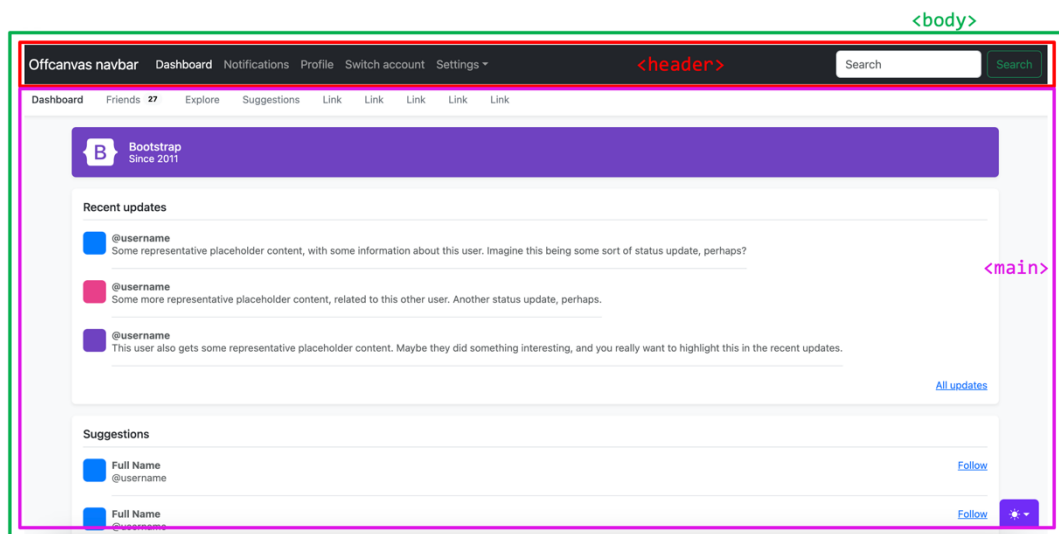
Setelah berhasil anda download extraklah file tersebut dan pindahkan kedalam project Laravel anda di folder **public** dan **rename** folder '**bootstrap-5.3.0-alpha2-dist**' menjadi '**assets**'. Jika anda memiliki file media lainnya seperti gambar, video ataupun memiliki file custome CSS atau JS anda bisa menyimpannya didalam folder assets.



Gambar 3.1. Direktori folder media assets

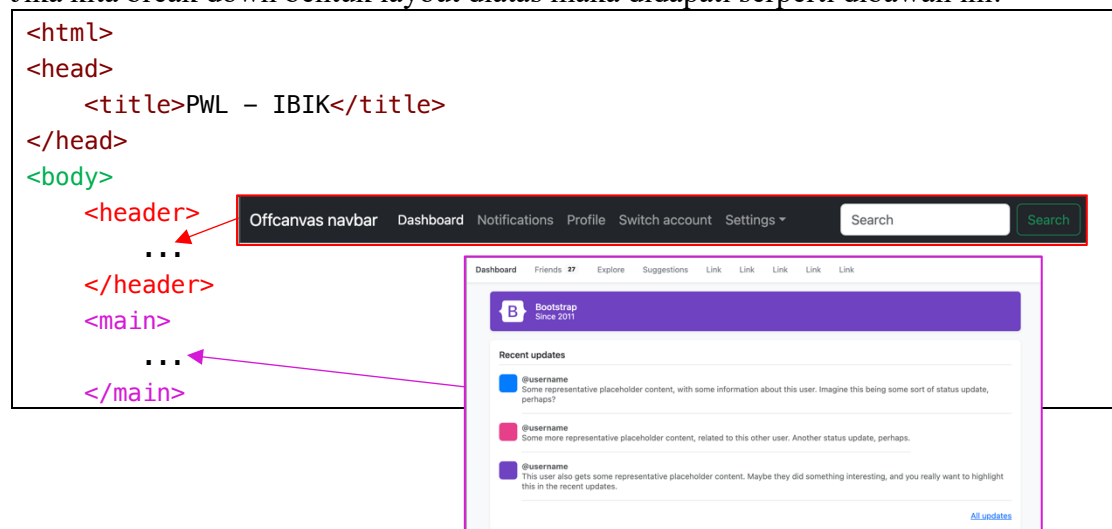
A. Mengatur Blade Layout

Pada contoh kasus ini akan mengambil sebuah bentuk template website milik bootstrap yang dapat dilihat pada: <https://getbootstrap.com/docs/5.3/examples/offcanvas-navbar/>



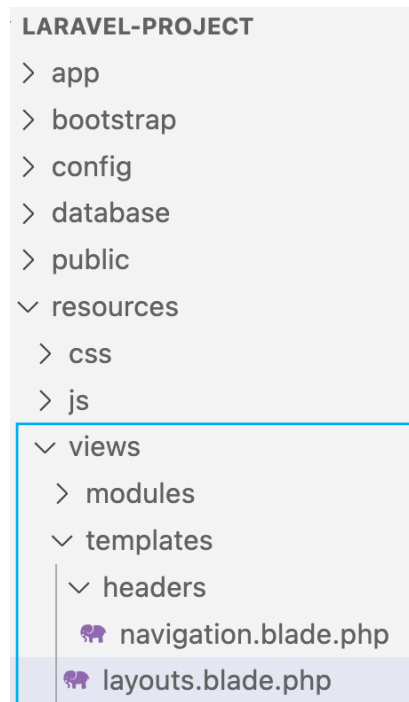
Gambar 3.1.1. Layout template bootstrap

Jika kita break down bentuk layout diatas maka didapatkan seperti dibawah ini:



```
<footer>
    .
</footer>
</body>
</html>
```

Pada project Laravel anda, buatlah structure folder pada view seperti gambar dibawah ini:



Gambar 3.1.2 Structure folder blade layout

Pada gambar diatas folder templates diperuntukan bagi file-file yang berhubungan dengan template UI. Sedangkan folder modules berisi file-file mentah yang akan dijadikan komponen main web static.

Pada file layouts.blade.php akan berisi structure html seperti yang telah dibreak down sebelumnya.

Layouts.blade.php

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>PW-IBIK</title>
    <link rel="stylesheet" href="{{ url('assets/css/bootstrap.min.css') }}">
    <link rel="stylesheet" href="{{ url('assets/bootstrap-
icons/font/bootstrap-icons.css') }}">
```



```

</head>

<body class="bg-body-tertiary">
  <header>
    @extends('templates.headers.navigation')
  </header>

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

  <footer class="container mt-5">
    <p class="fs-7">Copyright &copy; 2023</p>
  </footer>

  <script src="{{ url('./assets/js/bootstrap.bundle.min.js')
}}"></script>
</body>

</html>

```

Note:

`@extends` → penanda bahwa akan memanggil file bernama navigation.blade.php yang berada didalam folder templates/headers

`@yield` → menampilkan tag html yang telah ditandai sebagai bentuk komponen main static

`{{ url('') }}` → `url('')` menandakan bentuk dari base url (exp: localhost:8000)

navigation.blade.php

```

<nav class="navbar navbar-expand-lg fixed-top navbar-dark bg-dark" aria-
label="Main navigation">
  <div class="container-fluid">
    <a class="navbar-brand" href="{{ url('/') }}">PWL</a>
    <button class="navbar-toggler p-0 border-0" type="button"
id="navbarSideCollapse" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="navbar-collapse offcanvas-collapse"
id="navbarsExampleDefault">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page"
href="{{ url('/home') }}">Home</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" data-bs-
toggle="dropdown"
aria-expanded="false">Profile</a>

```

```

        <ul class="dropdown-menu">
            <li><a class="dropdown-item" href="{{
url('/profile/identity') }}">Identity</a></li>
            <li><a class="dropdown-item" href="{{
url('/family') }}">Family</a></li>
        </ul>
    </li>
</ul>
<form class="d-flex" role="search">
    <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search">
    <button class="btn btn-outline-success"
type="submit">Search</button>
</form>
</div>
</div>
</nav>

```

Untuk source code navigasi anda dapat melihat macam bentuknya di:

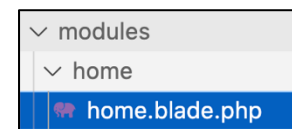
<https://getbootstrap.com/docs/5.3/components/navbar/#nav>

setelah berhasil membuat blade layout, selanjutnya akan mengatur isian komponen static yang telah ditandai oleh @yield

B. Mengatur file main blade view sebagai module

Disini kita akan mengarahkan sebuah alamat website dengan route /home sebagai bagian dari bentuk blade layout. Sehingga website yang kita bangun akan menjadi bentuk web static.

Pada gambar 3.1.2. terdapat folder modules, buatlah folder baru untuk route view home. Dan isi dari file home.blade.php anda dapat mengisinya dengan desain HTML, CSS, dan JS secara bebas.



home.blade.php

```

@extends('templates.layouts')

@section('content')

    <div class="d-flex align-items-center p-3 my-3 text-white bg-purple
rounded shadow-sm">
        
        <div class="lh-1">
            <h1 class="h6 mb-0 text-white lh-1">Bootstrap</h1>
            <small>Since 2011</small>
        </div>
    </div>

    <div class="card">
        <div class="card-header">
            <h3 class="card-title">This is a HOME page</h3>
        </div>
    </div>

```

```
<div class="card-body">
    Welcome folks..
</div>
</div>
```

@endsection

Notes:

@extends('templates.layouts') → menandakan bahwa file [home.blade.php](#) ini memanggil file blade layouts

@section('content') → menandakan bahwa pada file blade layout isi component HTML dibawah fragment section dengan nama content ini akan melakukan 'append' di dalam @yield('content').

Selanjutnya yang perlu anda lakukan adalah menambahkan route baru dengan nama home dan route ini memanggil file controller dan mengembalikan bentuk file UI home.blade.php.

Web.php

```
use App\Http\Controllers\HomeController;
use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;

Route::get('/', [HomeController::class, 'index']);
Route::get('/home', [HomeController::class, 'index']);

Route::get('/profile', [ProfileController::class, 'index']);
Route::get('/profile/identity', [ProfileController::class, 'identity']);
Route::get('/profile/family', [ProfileController::class, 'family']);
```

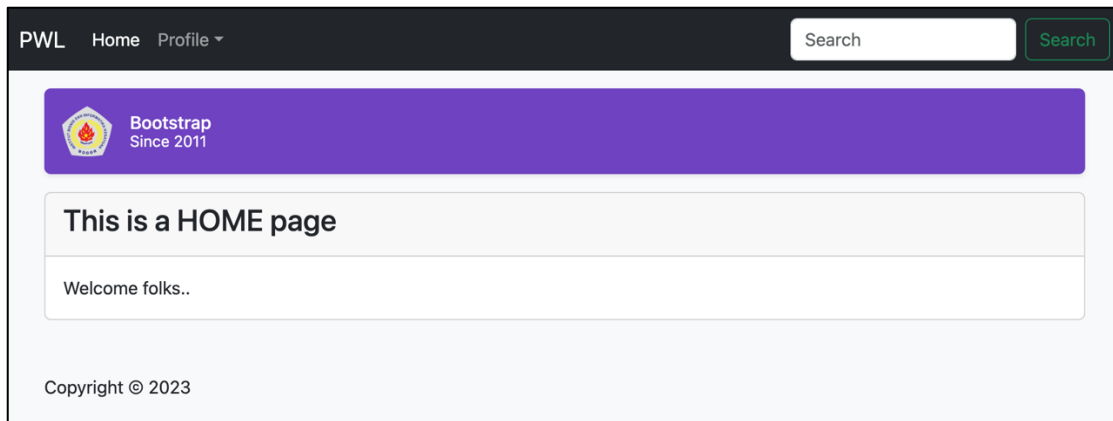
HomeController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
class HomeController extends Controller
{
    public function index(){
        return view('modules.home.home');
    }
}
```

Maka output yang didapatkan akan seperti ini:



Gambar 3.1.3. Tampilan halaman Home dengan blade layout

3.2. Blade Fragment

Ada beberapa blade fragment yang akan dibahas pada bagian ini, dari mulai cara merender sebuah data dan melakukan operation statement.

A. Rendering View

Blade merupakan pengaturan tampilan dengan menggunakan HTML markup, dengan penambahan beberapa directive dari Laravel.

Pada contoh sebelumnya terlihat directive pada bagian `@section` dan `@yield`. Berikut adalah beberapa markup HTML dalam bentuk blade view:

- Directive `@section` mendefinisikan sebuah bagian (section) dari isi halaman web
- Directive `@yield` digunakan untuk menampilkan isi dari bagian tersebut.

B. Rendering variable

Jika didalam controller ingin mengirimkan sebuah variable dan ditampilkan kedalam view blade maka anda dapat menggunakan cara sebagai berikut:

HomeController.php

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

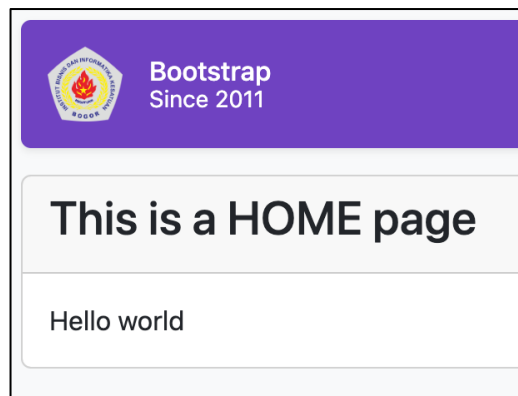
class HomeController extends Controller
{
    public function index(){
        return view('modules.home.home',["title"=>"Hello world"]);
    }
}
```

Pada method view terdapat `["title"=>"Hello world"]` ini adalah contoh pengiriman sebuah variable dari controller ke dalam blade view. Setiap data yang ingin dikirimkan harus disimpan dalam bentuk 'key'. Contoh pada script diatas key title memiliki sebuah value dalam bentuk STRING yang berisi tulisan "Hello World". Sehingga untuk menampilkan key title pada blade view cukup memanggilnya dengan cara: `{{ $title }}`

Contoh pada file home.blade.php:

```
<div class="card">
  <div class="card-header">
    <h3 class="card-title">This is a HOME page</h3>
  </div>
  <div class="card-body">
    {{ $title }}
  </div>
</div>
```

Maka hasil yang didapatkannya yaitu:



Atau anda juga dapat menggunakan method with() untuk mengirimkan sebuah data kedalam blade view.

```
return view('modules.home.home')->with("title","Hello world");
```

Lalu jika data yang anda ingin kirimkan dalam bentuk multiple atau lebih dari satu bagaimana mengirimkannya ?

Anda dapat memanfaatkan bentuk array sebagai bagian dari pengiriman data secara multiple dan merendernya dengan menggunakan blade directive.

C. Directive

Selain konsep pewarisan template (blade layout) dan menampilkan data, Blade juga menyediakan struktur kontrol umum PHP, seperti pernyataan bersyarat dan pengulangan.

- Selection: IF


HomeController

```
class HomeController extends Controller
{
    2 references | 0 overrides
    public function index(){
        $data = array();
        $data['title'] = "Sample IF ELSE";
        $data['npm'] = 212310029;
        return view('modules.home.home')->with("data",$data);
    }
}
```

home.blade.php

```
<div class="card">
    <div class="card-header">
        <h3 class="card-title">{{ $data['title'] }}</h3>
    </div>
    <div class="card-body">
        <p>NPM {{ $data['npm'] }} termasuk bilangan
        @if ($data['npm'] % 2 === 0)
            <span class="text-primary">GENAP</span>
        @else
            <span class="text-info">GANJIL</span>
        @endif
    </p>
    </div>
</div>
```

Output

 Bootstrap
Since 2011

Sample IF ELSE

NPM 212310029 termasuk bilangan GANJIL

- Selection: Switch case

```
<p>NPM {{ $data['npm'] }} termasuk bilangan
@switch($data['npm'] % 2)
    @case(0)
        <span class="text-primary">GENAP</span>
        @break
    @default
        <span class="text-info">GANJIL</span>
        @break
@endswitch
</p>
```

- Repetition: FOR

```
<div class="skills">
    <h4 class="text-uppercase">Skill:</h4>
    <div class="d-flex flex-row justify-content-between">
        <p class="text-dark">PHP</p>
        <div>
            @for ($i = 0; $i < 5; $i++)
                <span class="me-1 text-warning">
                    <i class="bi bi-star-fill"></i>
                </span>
            @endfor
        </div>
    </div>

    <div class="d-flex flex-row justify-content-between">
        <p class="text-dark">JAVA</p>
        <div>
            @for ($i = 0; $i < 5; $i++)
                <span class="me-1 text-warning">
                    <i class="bi bi-star{{ ($i > 2) ? '-fill' : '' }}"></i>
                </span>
            @endfor
        </div>
    </div>
</div>
```

Maka output dari script diatas sebagai berikut:

Blade Directive

NPM 212310029 termasuk bilangan **GANJIL**

NPM 212310029 termasuk bilangan **GANJIL**

SKILL:

PHP ★★★★★

JAVA ☆☆☆★★

- Repetition: Foreach
Pengulangan dengan menggunakan FOREACH biasanya hanya diperuntukan jika memiliki nilai dalam bentuk multiple Array.

HomeController.php

```
public function index(){
    $data = array();
    $data['title'] = "Blade Directive";
    $data['npm'] = 212310029;
    $students[] = array("npm"=>212310004, "name"=>"Muhamad Agus Setiawan");
    $students[] = array("npm"=>212310029, "name"=>"Muhamad Ilham Fachririzal");
    $students[] = array("npm"=>212310044, "name"=>"Hana Yulia Rahmah");
    $students[] = array("npm"=>212310027, "name"=>"MUHAMMAD ASKAH");
    $students[] = array("npm"=>212310005, "name"=>"ADJIE SYERAFFI RAHMAT");
    $data['students'] = $students;
    return view('modules.home.home')->with("data",$data);
}
```

home.blade.php

```
<div class="card-body">
    <table class="table">
        <thead>
            <tr>
                <th>No</th>
                <th>NPM</th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach ($data['students'] as $index => $result)
                <tr>
                    <td>{{ $index+1 }}</td>
                    <td>{{ $result['npm'] }}</td>
                    <td>{{ $result['name'] }}</td>
                </tr>
            @endforeach
        </tbody>
    </table>
</div>
```

Maka output dari script diatas sebagai berikut:

List of Student

| No | NPM | Name |
|----|-----------|---------------------------|
| 1 | 212310004 | Muhamad Agus Setiawan |
| 2 | 212310029 | Muhamad Ilham Fachririzal |
| 3 | 212310044 | Hana Yulia Rahmah |
| 4 | 212310027 | MUHAMMAD ASKAH |
| 5 | 212310005 | ADJIE SYERAFFI RAHMAT |

4. Latihan Praktikum

1. Jawablah contoh kasus pada section 2.2
2. Jika pada backend memiliki sebuah data array dalam bentuk table dibawah ini, bagaimana mengubah bentuk table skil akademik menjadi bentuk multiple array ?

| Course | Type | Rate |
|-------------|----------------|------|
| Matematika | Diskrit | 4 |
| Matematika | Aljabar Linear | 3 |
| Basis Data | DDL | 2 |
| Bhs Inggris | Speaker | 1 |

3. Dari soal nomor 2, buatlah modifikasi pada contoh kasus pada gambar 2.3.1. Dimana Ketika mengakses route Profile maka akan menampilkan informasi soal nomor 2 (Route→Controller→View), dimana bentuk array tersebut dikirimkan dari Controller dan ditampilkan ke blade profile. Gunakan blade layout pada contoh kasus ini.
4. Dari hasil code nomor 3, silakan anda buat tampilan hasil rendernya menjadi seperti gambar dibawah ini:

| Course | Type | Rate |
|-------------|----------------|-------|
| Matematika | Diskrit | ★★★★★ |
| | Aljabar Linear | ★★★★★ |
| Basis Data | DDL | ☆☆★★★ |
| Bhs Inggris | Speaker | ☆☆☆☆★ |

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PW-TI-21-[PA/KA]-NPM. File source code disimpan sesuai nama package-praktikum dan masukan kedalam repositori tersebut. Buatlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository anda kepada:

@FebryFairuz dan (@IrvanRizkyAriansyah atau @thesyamarcella)