MODUL 11. Laravel : Introduction

Tujuan Praktikum
1. Mahasiswa mampu memahami konsep dan implementasi MVC menggunakan *web* framework Laravel.

11.1 Framework & MVC

Framework atau dalam Bahasa Indonesia dapat diartikan sebagai "kerangka kerja" merupakan kumpulan dari fungsi-fungsi/prosedur-prosedur dan *class-class* untuk tujuan tertentu yang sudah siap digunakan sehingga bisa lebih mempermudah dan mempercepat pekerjaan seorang *programmer*, tanpa harus membuat fungsi atau *class* dari awal.
Alasan mengapa menggunakan *Framework*

- Mempercepat dan mempermudah pembangunan sebuah aplikasi *web*.
- Relatif memudahkan dalam proses *maintenance* karena sudah ada pola tertentu dalam sebuah *framework* (dengan syarat *programmer* mengikuti pola standar yang ada).
- Umumnya *framework* menyediakan fasilitas-fasilitas yang umum dipakai sehingga kita tidak perlu membangun dari awal (misalnya validasi, ORM, *pagination*, *multiple database*, *scaffolding*, pengaturan *session*, *error handling*, dll).
- Lebih bebas dalam pengembangan jika dibandingkan CMS.

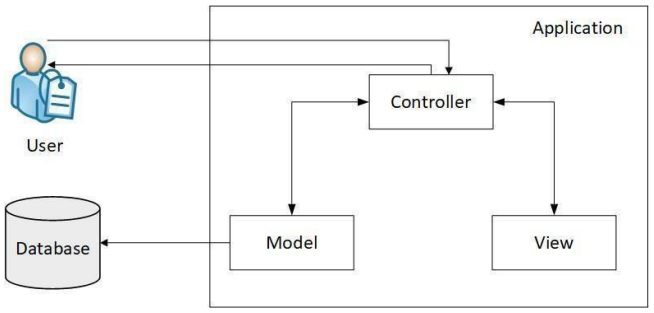
Framework biasanya memakai pola arsitektur tertentu sebagai fondasi.
Salah satu pola yang sering dipakai adalah MVC.

Model-View-Controller (MVC) merupakan suatu konsep yang cukup popular dalam pembangunan aplikasi *web*. MVC memisahkan pengembangan aplikasi berdasarkan komponen utama yang membangun sebuah aplikasi seperti manipulasi data, *user interface*, dan bagian yang menjadi kontrol aplikasi. Terdapat 3 jenis komponen yang membangun suatu MVC *pattern* dalam suatu aplikasi yaitu:

- *View*, merupakan bagian yang menangani *presentation logic*. Pada suatu aplikasi *web* bagian ini biasanya berupa *file template* HTML, yang diatur oleh *controller*. *View* berfungsi untuk menerima dan merepresentasikan data kepada *user*. Bagian ini tidak memiliki akses langsung terhadap bagian *model*.
- *Model*, biasanya berhubungan langsung dengan *database* untuk memanipulasi data (*insert*, *update*, *delete*, *search*), menangani validasi dari bagian *controller*, namun tidak dapat berhubungan langsung dengan bagian *view*.
- *Controller*, merupakan bagian yang mengatur hubungan antara bagian *model* dan bagian *view*, *controller* berfungsi untuk menerima *request* dan data dari *user* kemudian menentukan apa yang akan diproses oleh aplikasi.

Singkat kata ***Model*** untuk mengatur alur *database*, ***View*** untuk menampilkan *web*, sedangkan ***Controller*** untuk mengatur alur kerja antara *Model* dan *View*.

Jadi misalnya Anda akan membuat akun *e-mail*. Pertama anda akan melihat tampilan *sign-up / register*, itulah yang disebut dengan *View*. Kemudian Anda mengisi *form username*, *password*, dan lain-lain dan Anda klik tombol *Register*, maka disinilah *View* akan memanggil *Controller* dan *Controller* memanggil *Model*. Adapun tugas *Model* disini untuk mengecek apakah Anda sudah mengisi sesuai dengan kriteria dan akan dihubungkan dengan *database*. Kemudian *Model* akan mengembalikan ke *Controller* dan *Controller* akan mengembalikan ke *View*. Berikut adalah gambaran konsep MVC yang diterapkan pada framework.



Gambar 11.1 Cara kerja MVC

11.2 Pengenalan Laravel

Laravel adalah sebuah web application framework yang bersifat open-source yang digunakan untuk membangun aplikasi php dinamis. Laravel menjadi sebuah framework PHP dengan model MVC (Model, View, Controller) yang dapat mempercepat pengembang untuk membuat sebuah aplikasi web. Selain ringan dan cepat, Laravel juga memiliki dokumentasi yang lengkap disertai dengan contoh implementasi kodenya.

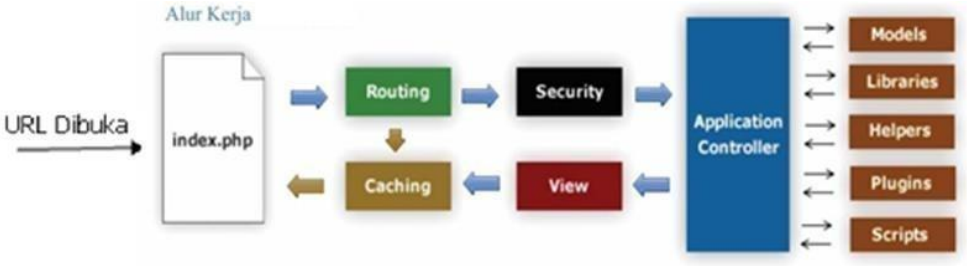
Laravel pertama kali dikembangkan pada tahun 2011 oleh Taylor Otwell. Saat ini Laravel sudah mencapai versi 9 yang dirilis pada tanggal 8 Februari 2022. Sebagai web framework populer yang menggunakan bahasa pemrograman PHP, Laravel mempunyai beberapa keunggulan yaitu:

1. *Free*, karena berada di bawah lisensi open source, kita dapat melakukan apa pun.
2. Menggunakan kaidah MVC, dengan menggunakan *Model-View-Controller*, kita dapat memisahkan bagian *logic* dan *presentation* dari aplikasi yang kita bangun.
3. Menghasilkan URL yang bersih. URL yang dihasilkan oleh Codeigniter bersih dan ramah terhadap *search engine*. Codeigniter menggunakan pendekatan *segment-based* dibandingkan dengan *query string* yang biasa digunakan oleh programmer yang tidak menggunakan *web framework*.
4. *Packs a Punch*, Laravel hadir dengan berbagai *library* yang akan membantu tugas-tugas di pengembangan web yang sudah umum dan sering dilakukan, seperti mengakses *database*, mengirim email, validasi data dari form, mengelola *session*, memanipulasi file, dan masih banyak lagi.
5. *Extensible*, kita dapat menambahkan *library* atau *helper* yang kita ciptakan sendiri ke dalam Laravel. Selain itu, kita dapat juga menambahkan fitur lewat *class extension*.
6. *Throughly Documented*, hampir semua fitur, *library*, dan *helper* yang ada di Laravel telah terdokumentasi dengan lengkap dan tersusun dengan baik. Dokumentasi cara penggunaannya dapat dilihat di https://laravel.com/docs/9.x.

Berikut adalah istilah yang sering ditemui di Laravel.

1. *Model*, *class* PHP yang dirancang untuk bekerja dengan informasi dari *database*.
2. *Controller*, inti aplikasi yang menentukan penanganan logic dari aplikasi web
3. *Route*, bagian yang menangani HTTP *request*.
4. *View*, halaman web seperti *header*, *footer*, *sidebar*, dan sebagainya yang ditanamkan di halaman *web*. *View* tidak pernah dipanggil secara langsung, tetapi harus dipanggil dari *controller*.
5. *Library*, *class* yang berisi fungsi-fungsi untuk penyelesaian kasus tertentu.

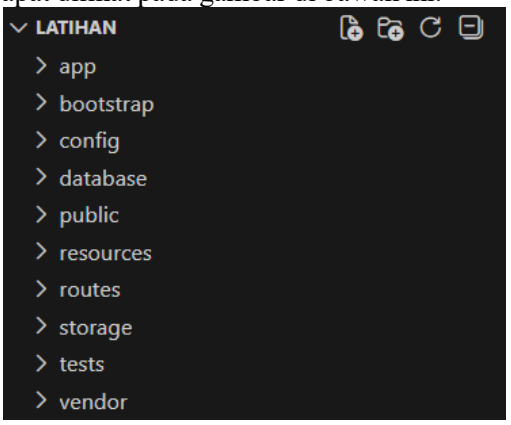
11.3 Cara Kerja Laravel



Gambar 11.2 Alur kerja Laravel

Ketika suatu URL dibuka, maka file index.php akan dibaca. Di dalam file ini, Laravel memeriksa apakah URL tersebut terdaftar pada Routing yang sudah dibuat. Jika ada pemetaannya dan memenuhi Security (jika diharuskan, autentikasi misalnya) maka Laravel akan memanggil Controller sesuai yang dipetakan oleh Routing. Di dalam Controller semua logika back-end dieksekusi, lalu dapat mengembalikan View (tampilan) ke browser. Tampilan ini disimpan dalam Caching agar pemrosesan setelahnya lebih cepat.

Sedangkan struktur folder Laravel dapat dilihat pada gambar di bawah ini:



Gambar 11.3 Struktur Folder Laravel

- app

  Folder ini berisi Controller, Model, Middleware, dan Provider

  Di sini file-file bekerja untuk mengoperasikan proses bisnis, seperti hitung total belanja atau tambah produk ke keranjang.

  Contoh: **ProductController.php** untuk mengatur alur tampilan list produk.

- bootstrap

  Folder ini berisi Cache untuk mempercepat pemrosesan

- config

  Folder ini berisi semua file konfigurasi untuk aplikasi

- database

  Folder ini berisi file-file migrasi dari/ke database

- public

  Folder ini berisi file yang dapat diakses langsung, biasanya file asset (gambar, CSS, dan JS) atau file konten (file untuk di-download)

- resources

  Folder ini berisi Routing untuk pemetaan URL ke aplikasi

  Berisi file mentah seperti tampilan sebelum dikompilasi.

  Biasanya dipakai untuk menampilkan halaman yang dilihat user.

  Contoh: **views/products/index.blade.php** untuk menampilkan daftar produk.
- storage

  Folder ini berisi hasil kompilasi View dan log dari aplikasi

  Tempat Laravel menyimpan file yang dihasilkan aplikasi.

  Digunakan untuk menyimpan log error dan cache tampilan.
- tests

  Folder ini berisi file-file untuk unit testing.

  Contoh: **Feature/ProductTest.php** untuk ngetes apakah halaman produk muncul.
- vendor

  Folder ini berisi file-file library yang dibutuhkan aplikasi

MODUL II
ROUTE / ROUTING

1.1 TUJUAN

1. Memahami fungsi routing pada framework Laravel
2. Memahami jenis-jenis route pada framework laravel
3. Menerapkan route pada framework Laravel

1.2 DASAR TEORI

Route atau **Routing** 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.**

Terdapat beberapa jenis route pada Laravel:

- ✓ Route parameter
- ✓ Route dengan optional parameter
- ✓ Route parameter dengan regular expression
- ✓ Route redirect
- ✓ Route Group
- ✓ Route Fallback
- ✓ Route priority

1.3 PEMBAHASAN

1. Route bawaan Laravel

Secara default, Laravel sudah menyertakan 1 route bawaan yang berada di dalam folder **routes/web.php**. Dan secara default sudah berisi beberapa kode program:

```php
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

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

?>
```

Ketika program pertama kali dijalankan, file yang di akses adalah welcome.blade.php yang berada pada folder recources/view

2. **Membuat route**
Untuk membuat route, cukup tulis pemanggilan static method Route baru ke dalam file routes/web.php. Sebagai contoh, tulis kode program berikut di bawah route bawaan Laravel:

```
Route::get('/beranda', function () {
    return 'Halaman Beranda';
});
```

3. **Route parameter**
Terkadang saat membuat sebuah URI, kita perlu mengirimkan sebuah parameter yang merupakan bagian dari segmen URI dalam route kita untuk kemudian di proses. Oleh karena itu, router Laravel juga mengizinkan kita untuk mengirimkan parameter sehingga kita bisa mengolah parameter ini nantinya.

Parameter route selalu terbungkus dalam kurung kurawal "{}" dan harus terdiri dari karakter alfabet.

```
Route::get('/kendaraan/{jenis}', function ($jenis) {
    return "Tampilkan data kendaraan dengan jenis $jenis";
});
```

4. **Route dengan optional parameter**
Apabila kita membuat route dengan dua parameter, dan salah satu parameter tidak terpenuhi, maka akan muncul keterangan 'tampil halaman 404'. Dengan optional parameter kita bisa membuat salah satu atau beberapa parameter menjadi opsi.

```
Route::get('/kendaraan/{jenis?}/{merek?}',
      function ($a = 'motor',$b = 'honda') {
      return "Cek harga kendaraan $a $b";
});
```

5. **Route parameter dengan regular expression**
Route parameter juga bisa dibatasi dengan regular expression. Ini dipakai jika kita ingin penulisan alamat URL harus memenuhi pola tertentu. Sebagai contoh perhatikan route berikut:

```
Route::get('/product/{id}', function ($id) {
     return "Tampilkan product dengan id = $id";
});
```

Route ini bisa diakses dari alamat localhost:8000/product/10, localhost:8000/product/AB02 maupun localhost:8000/product/lima. Artinya, apapun nilai segmen terakhir tetap bisa diprosesoleh Laravel.
Namun biasanya, id terdiri dari angka saja. Id yang diinput dengan huruf seperti 'lima', seharusnya tidak bisa diakses. Untuk pembatasan seperti ini, kita bisa menggunakan regular expression dengan contoh berikut:

```
Route::get('/product/{id}', function ($id) {
     return "Tampilkan product dengan id = $id";
})->where('id', '[0-9]+');
```

Perhatikan perintah ->where('id', '[0-9]+'). Kode ini artinya agar parameter 'id' hanya bisa diakses jika memenuhi pola regular expression '[0-9]+'. Pola ini berarti segmen 'id' hanya bisa diisi dengan angka 0 – 9 sebanyak 1 karakter atau lebih. Selain itu, route tidak bisa diakses.

6. **Route redirect**

Laravel juga menyediakan fitur untuk proses redirect antar route. Caranya, gunakan method Route::redirect seperti contoh berikut:

```
Route::get('/hubungi-kami', function () {
     return '<h1>Hubungi Kami</h1>';
 });

 Route::redirect('/contact-us', '/hubungi-kami');
```

Di sini terdapat 2 route. Route di baris 1 – 3 merupakan route normal untuk alamat localhost:8000/hubungi-kami. Di baris 5 terdapat kode untuk membuat redirect. Sehingga ketika diakses alamat localhost:8000/contact-us, maka halaman langsung dialihkan ke localhost:8000/hubungi-kami.

7. **Route group**

Jika kita memiliki beberapa route yang ingin di proses secara kelompok, bisa menggunakan route group. Dengan menggunakan route group, beberapa route bisa di proses sebagai satu kesatuan. Sebagai contoh:

```
Route::prefix('/admin')->group(function() {

    Route::get('/dashboard', function() {
        return 'Tampilkan dashboard aplikasi';
    });

    Route::get('/datapegawai', function() {
        return 'Tampilkan data pegawai';
    });

    Route::get('/datamahasiswa', function() {
        return 'Tampilkan data mahasiswa';
    });

});
```

Ketiga route tersebut dapat di akses dari alamat:

localhost:8000/admin/mahasiswa
localhost:8000/admin/dosen
localhost:8000/admin/karyawan

Method ini akan menambah awalan (prefix) '/admin' ke semua route yang ada di dalam group admin.

8. **Route fallback**

Laravel menyediakan route khusus yang akan dijalankan jika tidak ditemukan route untuk sebuah alamat URL. Secara default, laravel akan menampilkan halaman 404 | Not Found, namun kita bisa menimpanya menggunakan method Route::fallback()

```
Route::fallback(function () {
     return "Maaf, alamat tidak ditemukan";
 });
```

9. **Route priority**

Urutan penulisan route memiliki efek prioritas yang berbeda tergantung jenis route. Jika kita menggunakan route biasa (tanpa parameter), maka apabila terdapat lebih dari 1 route dengan alamat yang sama, route paling akhir yang akan dijalankan:

```
Route::get('/baju/1', function () {
    return "Baju ke-1";
 });

 Route::get('/baju/1', function () {
    return "Baju saya ke-1";
 });

 Route::get('/baju/1', function () {
    return "Baju kita ke-1";
 });
```

Ketika alamat http://localhost:8000/baju/1 diakses, yang akan tampil di web browser adalah "Baju kita ke-1", yakni hasil dari route ke tiga.

Namun ketika menggunakan route parameter, hasilnya jadi berbeda:

```
Route::get('/baju/{a}', function ($a) {
 return "Baju ke-$a";
 });

 Route::get('/baju/{b}', function ($b) {
 return "Baju saya ke-$b";
 });

 Route::get('/baju/{c}', function ($c) {
 return "Baju kita ke-$c";
 });
```

Ketika mengakses http://localhost:8000/baju/1, yang tampil adalah "Baju ke-1", yakni hasil dari route pertama.

MODUL III
VIEW

2.1 TUJUAN

1. Memahami fungsi view pada framework Laravel
2. Memahami cara kerja view
3. Mampu menggunakan view

2.2 DASAR TEORI

View adalah komponen MVC yang menangani tampilan. Di dalam viewlah kode HTML, CSS dan juga JavaScript berada.

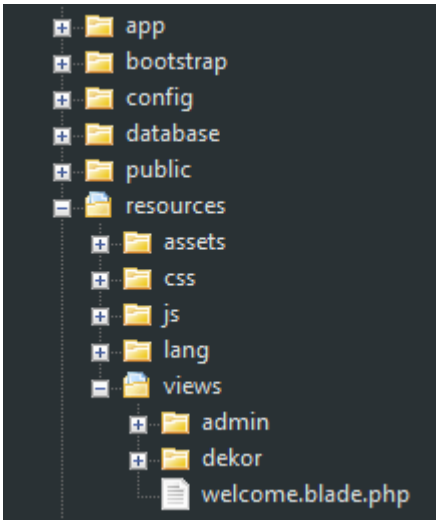
2.3 PEMBAHASAN & LATIHAN

1. View bawaan Laravel

Laravel memiliki 1 view bawaan yang terlihat saat kita mengakses halaman homepage atau halaman root Laravel. Pemanggilan view ini berasal dari kode berikut:
Lokasi : routes/web.php

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

Lokasi view bawaaan laravel : resources/views/welcome.blade.php



Secara default file tampilan pada Laravel menggunakan template engine, yaitu **Blade**. Sehingga, setiap file tampilan yang kita buat harus mengikuti penamaan
namaview.blade.php
diletakkan pada folder **resources/views** agar dapat di-load oleh fungsi **view().**

2. Membuat View

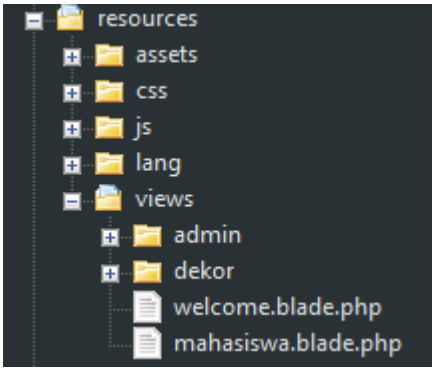
cara membuat view:

1. Buat route yang akan mengembalikan view.
2. Buat file view di folder resources\views dengan format <nama_file>.blade.php

Untuk langkah pertama, cukup tulis perintah return view('nama_view') ke dalam anonymous function di route.
Sebagai contoh, ketika mengakses URL http://localhost:8000/mahasiswa akan ditampilkan view mahasiswa. Maka kode route-nya adalah sebagai berikut:

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

Kemudian buat file mahasiswa.blade.php didalam folder resources/views



Tulis script berikut didalam file mahasiswa.blade.php

```
<!DOCTYPE html>
<html>
    <head>
       <title>Belajar Laravel</title>
    </head>
    <body>
       <h2> Data Mahasiswa </h2>
       <ol>
            <li>Mahasiswa 1</li>
            <li>Mahasiswa 2</li>
            <li>Mahasiswa 3</li>
            <li>Mahasiswa 4</li>
       </ol>
    </body>
</html>
```

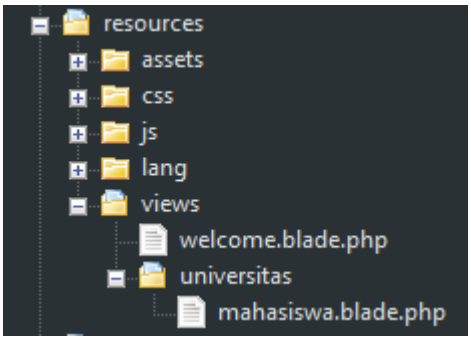
Jalankan URL localhost:8000/mahasiswa maka akan tampil halaman seperti berikut

Catatan:
Hal yang harus diperhatikan yaitu jika ingin menggunakan file eksternal CSS / Javascript, maka harus menggunakan fungsi **{{ asset('path file di folder public') }},**

Contoh:

```
<script src="{{ asset('js/jquery.min.js') }}"></script>
<img src="{{ asset('images/produk1.png') }}" alt="Produk 1">
```

3. **Membuat struktur folder view**
   Pada pembuatan view diatas file mahasiswa.blade.php berada langsung di bawah folder resources\views. Ini sebenarnya tidak masalah, tapi untuk aplikasi yang kompleks kita butuh sebuah manajemen file agar view lebih terstruktur.
   Misalnya untuk membuat aplikasi sistem informasi universitas, halaman view yang dibutuhkan bisa terdiri dari puluhan file. Semua view ini perlu diatur ke dalam folder mahasiswa, folder dosen, folder karyawan, dst.
   Sebagai contoh praktek, pindahkan file mahasiswa.blade.php ke dalam folder universitas. Sehingga alamat lengkap file ini ada di resources\views\universitas\ mahasiswa.blade.php.



   Maka route view jg perlu dirubah menjadi

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

4. **Mengirim data ke view**
   Kita bisa mengirim data dari route untuk ditampilkan secara dinamis oleh view dengan du acara:
   1. Menulis data sebagai argument kedua dari function view().
   2. Menggunakan method with().

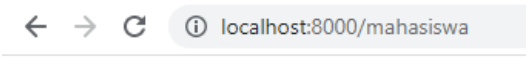
   A. Mengirim Data ke View Sebagai Argumen
   Cara pertama untuk mengirim data dari route ke view adalah dengan menulis data tersebut sebagai argumen kedua pada saat pemanggilan function view().
   contoh : buat route mahasiswa menjadi seperti berikut

```
Route::get('/mahasiswa', function () {
   return view('universitas.mahasiswa',["mhs1"=>"Abdul Hafidz"]);
 });
```

   Ubah script pada file mahasiswa.blade.php menjadi seperti berikut:

```
<!DOCTYPE html>
<html>
    <head>
       <title>Belajar Laravel</title>
    </head>
    <body>
       <h2> Data Mahasiswa </h2>
       <ol>
            <li><?php echo $mhs1; ?></li>
            <li>Mahasiswa 2</li>
            <li>Mahasiswa 3</li>
            <li>Mahasiswa 4</li>
       </ol>
    </body>
</html>
```

Ketika URL dijalankan maka tampilanya menjadi seperti berikut:



Data Mahasiswa

1. Abdul hafidz
2. Mahasiswa 2
3. Mahasiswa 3
4. Mahasiswa 4

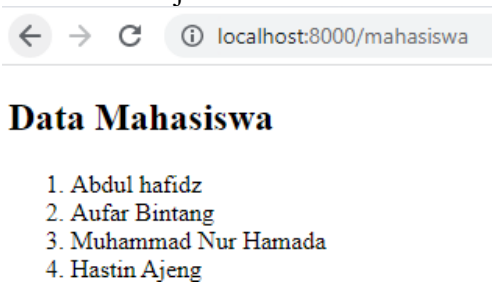
Kemudian apabila ingin mengirim lebih banyak data, tambah element baru ke dalam array yang akan dikirim, ubah route menjadi:

```
Route::get('/mahasiswa', function () {
    return view('universitas.mahasiswa',
    [
    "mhs1" => "Abdul hafidz",
    "mhs2" => "Aufar Bintang",
    "mhs3" => "Muhammad Nur Hamada",
    "mhs4" => "Hastin Ajeng"
    ]);
 });
```

Ubah script pada file mahasiswa.blade.php menjadi seperti berikut:

```
<ol>
    <li><?php echo $mhs1; ?></li>
    <li><?php echo $mhs2; ?></li>
    <li><?php echo $mhs3; ?></li>
    <li><?php echo $mhs4; ?></li>
</ol>
```

Ketika URL dijalankan maka akan muncul tampilan seperti:



Selain cara tersebut, agar lebih rapi, kode program di bagian route bisa ditulis sebagai berikut:

```
Route::get('/mahasiswa', function () {
    $arrMhs = [
    "mhs1" => "Abdul hafidz",
    "mhs2" => "Aufar Bintang",
    "mhs3" => "Muhammad Nur Hamada",
    "mhs4" => "Hastin Ajeng"
    ];
    return view('universitas.mahasiswa', $arrMhs);
 });
```

Proses menampilkan data seperti ini tidak salah, tapi masih kurang efisien. Di dalam view kita harus men-echo satu per satu variabel $mhs1, $mhs2, dst. Karena ini merupakan data yang berulang dan sama (berisi nama-nama mahasiswa), maka seharusnya bisa diproses menggunakan perulangan foreach PHP.
Namun struktur array yang dikirim juga harus diubah. Agar bisa diakses menggunakan perulangan foreach, kita harus rancang supaya data yang sampai di view masih berbentuk array, bukan variabel yang sudah dipecah. Solusinya, kirim data dalam bentuk nested array seperti contoh berikut:

```
Route::get('/mahasiswa', function () {
    $arrMhs = ["Abdul hafidz","Aufar Bintang",
               "Muhammad Nur Hamada","Hastin Ajeng"];
    return view('universitas.mahasiswa',['mahasiswa' => $arrMhs]);
});
```

Ubah script pada file mahasiswa.blade.php menjadi seperti berikut:

```
<ol>
    <li><?php echo $mahasiswa[0]; ?></li>
    <li><?php echo $mahasiswa[1]; ?></li>
    <li><?php echo $mahasiswa[2]; ?></li>
    <li><?php echo $mahasiswa[3]; ?></li>
</ol>
```

Cara ini boleh dilakukan, akan tetapi jika datanya banyak dan memiliki karakter yang sama sebenarnya bisa dibuat lebih efisien dengan loop menggunakan foreach:

```
<ol>
 <?php foreach ($mahasiswa as $nama) {
    echo "<li> $nama </li>";
 } ?>
 </ol>
```

B. Mengirim Data ke View menggunakan method with

Cara kedua yang bisa dipakai untuk mengirim data dari route ke view adalah menggunakan method with().

Contoh:

```
Route::get('/mahasiswa', function () {
    $arrMhs = ["Abdul hafidz","Aufar Bintang",
               "Muhammad Nur Hamada","Hastin Ajeng"];

    return view('universitas.mahasiswa')->with('mahasiswa',
$arrMhs);
});
```

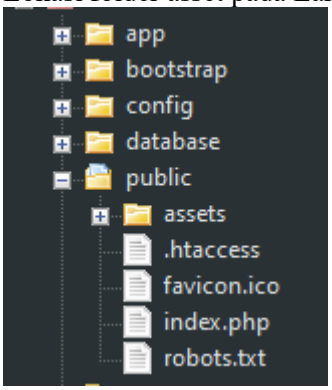
Tampilkan data menggunakan loop foreach, hasilnya sama dengan pengiriman data menggunakan argument.

5. Pengelolaan Assets

Dalam web programming terdapat istilah assets, atau web assets. Ini sebutan untuk file external seperti file CSS, file JavaScript, gambar, dan icon. Untuk project besar, kita butuh sebuah struktur pengelolaan assets agar semua file tersimpan rapi.

Cara sederhana yang biasa dilakukan adalah dengan membuat folder css, folder js, dan folder img untuk menyimpan file assets sesuai jenis filenya.

Lokasi folder asset pada Laravel berada pada folder public.



Secara default, dalam laravel terdapat 4 file dalam folder public yaitu:

- **.htaccess** dan web.config: Kedua file ini merupakan file konfigurasi untuk web server. File .htaccess ditujukan untuk web server Apache serta beberapa web server berbasis Linux lain seperti Nginx. Sedangkan file web.config ditujukan untuk web server Microsoft IIS. Untuk saat ini kedua file tidak perlu kita utak-atik.
- **favicon.ico**: File gambar untuk tampilan favicon, yakni gambar kecil di sudut kiri atas web browser. Bawaan laravel, file favicon.ico hanya gambar icon dummy, file ini berukuran 0 byte yang artinya tidak ada gambar sama sekali. Nantinya kita bisa ganti dengan gambar favicon lain.
- **robots.txt**: File khusus yang ditujukan untuk mesin pencari seperti Google. Di dalam file ini kita bisa menulis daftar halaman web yang tidak ingin di index oleh Google.
- **index.php**: Inilah file yang diakses Laravel pertama kali pada setiap pemrosesan halaman. Misalnya ketika kita mengetik alamat URL http://localhost:8000/ mahasiswa, maka file index.php ini akan di proses pertama kali. Di dalamnya terdapat kode program khusus yang akan memanggil berbagai file Laravel. Kita juga tidak perlu mengutak-atik file ini, biarkan saja apa-adanya.

MODUL IV
BLADE TEMPLATE ENGINE

3.1 TUJUAN

1. Memahami fungsi dan cara kerja blade pada framework laravel
2. Mampu menerapkan blade

3.2 DASAR TEORI

Blade adalah template engine bawaan Laravel. Secara sederhana, template engine berisi perintah tambahan untuk mempermudah pembuatan template. Template sendiri bisa disebut sebagai kerangka dasar tampilan.

Secara garis besar, terdapat 2 fungsi utama blade di dalam Laravel:

1. Mempersingkat penulisan perintah PHP.
2. Pemecahan file template (proses pembuatan layout).

3.3 PEMBAHASAN

1. Menampilkan Data

Kegunaan paling dasar dari blade adalah untuk mempermudah proses menampilkan data di dalam view. Biasanya untuk menampilkan variabel di dalam view kita menggunakan perintah berikut:

```
<?php echo $produk; ?>
```

Dengan blade, bisa ditulis menjadi:

```
{{ $produk }}
```

Berikut ini penggunaan blade directive

- Sintaks Blade untuk echo data: **{{ $xxx }}.**
- If/Else versi Blade: **@if, @else.**
- Loop Blade: **@foreach, @for, @while**.
- Komentar Blade: **{{-- ... --}}.**
- Layouting dengan Blade: **@extends, @yield, @include**.

Blade directive = perintah, sintaks, atau tanda khusus yang disediakan Blade untuk mempermudah penulisan view di Laravel.

- **@extends** = halaman ini numpang layout mana
- **@yield** = titik tempat konten halaman turunan diberikan
- **@include** = sisipkan komponen lain (navbar, footer, card, dsb.)

Contoh Menampilkan data:
Buat route:

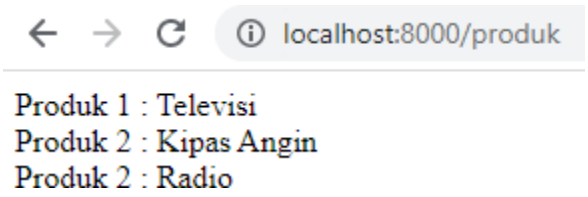
```
Route::get('/produk', function () {

    $arrProduk  = [
    "prod1" => "Televisi",
    "prod2" => "Kipas Angin",
    "prod3" => "Radio"
    ];
    return view('produk', $arrProduk);
});
```

Buat view produk.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Data Produk</title>
</head>
<body>
    Produk 1 : {{ $prod1; }}
    <br>
    Produk 2 : {{ $prod2; }}
    <br>
    Produk 2 : {{ $prod3; }}
</body>
</html>
```

Tampilan pada browser



Contoh membuat kondisi If Else
Menggunakan PHP biasa

```php
<?php

    if (($nilai >= 0) and ($nilai < 50)){
        echo "Tidak lulus";
    } else if (($nilai >= 50) and ($nilai <= 100)) {
        echo "Lulus";
    } else {
    echo  "Tidak Teridentifikasi";
    }
?>
```

Menggunakan blade

```
@if (($nilai >= 0) and ($nilai < 50))
   Tidak lulus
@elseif (($nilai >= 50) and ($nilai <= 100))
   Lulus
@else
   Tidak Teridentifikasi
@endif
```

Dengan format blade, Kita tidak perlu menulis tag pembuka dan penutup php <?php ?>

2. Baris komentar dan PHP mode

Untuk membuat baris komentar, blade menyediakan cara penulisan khusus, yakni dengan tanda pembuka {{-- dan penutup --}}.

3. Merancang Layout

Umumnya pada halaman web terdapat banyak element yang terus berulang di setiap file, seperti bagian header, menu navigasi, sidebar dan footer.
Teknik yang umum dipakai adalah memecah bagian tersebut menjadi file terpisah, lalu disatukan kembali menggunakan fungsi include() bawaan PHP.

Untuk mempermudah proses pembuatan layout atau template adalah memecah bagian yang berulang menjadi file terpisah, kemudian di satukan kembali di view yang membutuhkan. Blade menyediakan perintah @include untuk keperluan ini, cara penggunaannya mirip seperti fungsi include() bawaan PHP.

Contoh:
Buat 2 view yang berisi:
- master.blade.php untuk master layout
- produk.blade.php yang berisi produk-produk yang akan ditampilkan

Master.blade.php

```
<!DOCTYPE html>
 <html lang="en">
 <head>
 <title>@yield('title')</title>
 </head>
 <body>
    @yield('content')
 </body>
 </html>
```

Perintah @yield dipakai untuk menandai bagian yang bisa ditimpa oleh view turunan nantinya. Teks di dalam tanda kurung merupakan nama atau id dari yield tersebut. Perintah @yield('title') artinya terdapat yield dengan id 'title', serta perintah @yield('content') akan membuat yield dengan id 'content'.

produk.blade.php

```
@extends('master')
@section('title','Data Produk')

@section('content')

    Produk 1 : {{ $prod1; }}
    <br>
    Produk 2 : {{ $prod2; }}
    <br>
    Produk 2 : {{ $prod3; }}

@endsection
```

Perintah @extends('master') merupakan perintah yang memberitahu laravel bahwa view ini merupakan turunan dari file master.blade.php

Master.blade.php merupakan file yang bersisi element berulang, apabila kita ingin membuat menu lain dengan elemen mengulang yang sama, kita cukup memanggil masternya saja.

3.4 LATIHAN

Membuat switch case dengan 4 kondisi:
Jika $nilai bernilai 0, tampilkan teks "Sangat Buruk"
Jika $ nilai bernilai 75, tampilkan teks "Baik"
Jika $ nilai bernilai 100, tampilkan teks "Sempurna"
Jika $ nilai bernilai selain 3 angka di atas, tampilkan teks "Nilai tidak valid"

```
@switch($nilai)

@case(0)
   Sangat Buruk
@break
@case(75)
   Baik
@break
@case(100)
   Sempurna
@break
@default
   Nilai tidak valid
@endswitch
```

MODUL V
CONTROLLERS

4.5 TUJUAN

1. Memahami fungsi dan cara kerja controllers pada framework laravel
2. Mampu menghubungkan controllers dan views

4.6 DASAR TEORI

Controller adalah bagian kode program yang mengatur logika serta lalu lintas data. Controller berada di pusat MVC. Pada saat user mengetik sebuah alamat di web browser, route akan membaca alamat tersebut dan (idealnya) memanggil controller yang sesuai.

Controller adalah salah satu komponen inti dari MVC yang berfungsi sebagai penghubung antara request user (View) ke model yang nantinya akan di kembalikan lagi ke View dalam bentuk response. Controller ini akan banyak berisi logika – logika dalam menyusun suatu fungsi tertentu. Contohnya adalah aktivitas CRUD (Create, Read, Update, Delete) yang prosesnya berjalan di dalam Controller.

Di dalam controller inilah logika program kita tulis. Apabila butuh mengambil data dari database, controller akan mengakses model. Hasil dari model dikembalikan ke controller yang bisa diolah lebih lanjut untuk kemudian dikirim ke view. Sesampainya di view, data tinggal ditampilkan ke web browser.

Secara garis besar, terdapat 5 fungsi utama controllers di dalam Laravel:

1. Menangani Permintaan HTTP
   Controller dalam Laravel bertanggung jawab untuk menangani permintaan HTTP dari pengguna. Saat pengguna mengakses suatu rute (route) dalam aplikasi web kita, controller yang terkait akan dipanggil. Controller kemudian akan memproses permintaan tersebut, melakukan tugas yang diperlukan, dan menghasilkan respons yang sesuai.

2. Mengelola Logika Bisnis
   Salah satu fungsi utama dari controller adalah mengelola logika bisnis dalam aplikasi web. Ketika sebuah permintaan masuk, controller akan memproses data yang diterima, memvalidasi input, memanggil model untuk mengakses atau memodifikasi data, dan menjalankan logika bisnis khusus. Dengan menyimpan logika bisnis dalam controller, kita dapat dengan mudah memahami alur program dan memisahkan peran-peran yang berbeda dalam MVC.

3. Memanipulasi Data
   Controller juga bertanggung jawab untuk memanipulasi data. Ketika pengguna mengirimkan data melalui permintaan, controller akan menerima data tersebut dan memvalidasi nilainya. Selanjutnya, controller dapat menggunakan model untuk mengakses database, melakukan operasi CRUD (Create, Read, Update, Delete), dan memanipulasi data sesuai kebutuhan. Dengan menggunakan controller, kita dapat dengan mudah mengatur logika manipulasi data dan menjaga integritas data dalam aplikasi.

4. Memberikan Respons ke Tampilan (View)
   Setelah controller selesai memproses permintaan, tugas selanjutnya adalah memberikan respons ke tampilan (view). Controller dapat mengirimkan data yang diolah ke tampilan agar dapat ditampilkan kepada pengguna. Controller juga dapat mengatur bagaimana respons akan diformat, seperti menghasilkan tampilan HTML, JSON, atau respons kustom lainnya. Dengan menggunakan controller, kita dapat dengan mudah mengatur respons yang dihasilkan dan memisahkan logika presentasi dari bisnis.

5. Mengelola Penggunaan Rute (Routing)
   Controller berperan penting dalam pengelolaan rute (*routing*) dalam Laravel. Rute adalah aturan yang digunakan untuk mengarahkan permintaan pengguna ke controller yang tepat. Dalam controller, kita dapat mendefinisikan metode-metode yang akan dipanggil ketika suatu rute tertentu diakses. Dengan menggunakan controller, kita dapat dengan mudah mengatur dan memisahkan logika rute dalam aplikasi web kita.

4.7 PEMBAHASAN

4.7.1 Cara Mengakses Controller

Dalam Laravel, route adalah 'penerima tamu' dari setiap alamat yang kita ketik di web browser. Route-lah yang menentukan proses mana yang akan menangani URL tersebut, apakah langsung ke View, atau ke Controller.

Di Laravel 8, untuk memanggil Controller ditulis dengan format berikut:

```
Route::get('<url>',[App\Http\Controllers\Nama_Controller::class,'nama_method']);
```

Sebagai contoh, untuk memanggil method index() di controller bernama PageController, penulisan route-nya adalah:

```
Route::get('/', [App\Http\Controllers\PageController::class,'index']);
```

Dengan penulisan ini, maka ketika alamat http://localhost:8000 di akses, route akan menjalankan method index() milik PageController.

Contoh lain, apabila kita ingin alamat URL http://localhost:8000/mahasiswa akan menjalankan method tampil() milik PageController, penulisan routenya adalah sebagai berikut:

```
Route::get('/mahasiswa', [App\Http\Controllers\PageController::class,'tampil']);
```

Di dalam folder Controllers sudah ada 1 file bernama Controller.php. File Controller.php merupakan file induk dari semua controller yang akan di buat seperti berikut:

app > Http > Controllers > Controller.php

```
1   <?php
2
3   namespace App\Http\Controllers;
4
5   use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
6   use Illuminate\Foundation\Bus\DispatchesJobs;
7   use Illuminate\Foundation\Validation\ValidatesRequests;
8   use Illuminate\Routing\Controller as BaseController;
9
10  class Controller extends BaseController
11  {
12      use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
13  }
```

Di baris 3 terdapat deklarasi namespace, yang artinya file Controller.php berada di dalam namespace App\Http\Controllers. Ini bersesuaian dengan struktur folder tempat file Controller.php berada, yakni di dalam folder app\Http\Controllers\.

Di baris 5 – 8 berisi pemanggilan 4 class menggunakan keyword use. Pengertian sederhana,keyword use dipakai untuk meng-import suatu class ke dalam file saat ini. Artinya terdapat 4 class baru yang di import, yakni AuthorizesRequests, DispatchesJobs, ValidatesRequests danBaseController.

Lanjut ke baris 10 – 13, terdapat kode untuk membuat class Controller yang di extends dari class BaseController. Isi dari class hanya 1 baris yang dipakai untuk mendeklarasikan penggunaan class AuthorizesRequests, DispatchesJobs, dan ValidatesRequests.

4.7.2 Membuat Controller Secara Manual

Terdapat 2 cara pembuatan controller, yakni secara manual dari text editor, atau menggunakan perintah php artisan.
Silahkan buat file baru dengan nama PageController.php, lalu simpan di folder app\Http\Controllers. Sehingga alamat file ada di: app\Http\Controllers\PageController.php.
Isi file ini dengan kode berikut:

```
<?php

namespace App\Http\Controllers;

class PageController extends Controller
 {
    public function index()
    {
    return "Halaman Home";
    }
    public function tampil()
    {
    return "Data Mahasiswa";
    }
 }
```

Di baris 3 terdapat pendeklarasian namespace App\Http\Controllers. Ini sama seperti yang ada di file Controller.php. Karena memiliki namespace yang sama, kita bisa langsung mengakses semua kode yang ada di Controller.php.
Struktur namespace ini juga bersesuaian dengan alamat path, yakni folder app\Http\ Controllers\. Jika nantinya file controller di pindah ke folder lain, penulisan namespace ini juga harus disesuaikan.

Kemudian coba akses dengan menambah dua routes berikut ke dalam file routes/web.php:

```
Route::get('/', [App\Http\Controllers\PageController::class,'index']);
Route::get('/mahasiswa',[App\Http\Controllers\PageController::class,'tampil']);
```

4.7.3 Cara Penulisan Route Untuk Controller

Salah satu perubahan pada Laravel 8 dibandingkan versi sebelumnya ada di cara penulisan alamat route yang dipakai untuk mengakses controller. Di Laravel 7 ke bawah, format penulis route bisa langsung ditulis dalam bentuk string:

```
Route::get('<alamat_url>','<nama_controller>@<nama_method>');
```

Misalnya untuk mengakses method index() milik PageController, routenya adalah:

```
Route::get('/', 'PageController@index');
```

Sedangkan di Laravel 8, kita harus menulis lengkap namespace file controller, yakni "App\Http\Controllers\", kemudian baru menulis nama class controller beserta nama methodnya:

```
Route::get('/', [App\Http\Controllers\PageController::class,'index']);
```

Alternatif penulisan lain, namespace "App\Http\Controllers\PageController" bisa di pindah ke bagian atas file routes\web.php. Sebagai contoh, dalam praktek sebelumnya file routes\web.php sudah berisi 2 route berikut:

```
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', [App\Http\Controllers\PageController::class,'index']);
Route::get('/mahasiswa',[App\Http\Controllers\PageController::class,'tampil']);
```

Dengan memindahkan namespace "App\Http\Controllers\PageController" ke bagian atas, file route bisa ditulis seperti berikut:

```
<?php

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

Route::get('/', [PageController::class,'index']);
Route::get('/mahasiswa',[PageController::class,'tampil']);
```

4.7.4 Mengakses View dari Controller

Sebuah controller pada dasarnya terdiri dari kumpulan method. Di dalam method inilah kita bisa melakukan banyak hal, misalnya langsung menampilkan teks dengan perintah return seperti contoh kita sebelumnya.

Namun yang paling ideal adalah, mengembalikan sebuah View. Caranya juga sangat mudah karena sama persis seperti yang sering kita lakukan di route. Sebagai contoh, modifikasi kembali PageController.php menjadi seperti berikut:

```
<?php

namespace App\Http\Controllers;

class PageController extends Controller
 {
    public function index()
    {
    return view('welcome');
    }
```

```
    public function tampil()
    {
    $arrMahasiswa = ["Kholifahdina","Rahmat Taufik","Nita
Fitrotunimah","Defrin Anggun Saputri"];

    return view('mahasiswa')->with('mahasiswa', $arrMahasiswa);
    }
 }
```

Kemudian buat views dengan nama mahasiswa.php seperti berikut:

```
<!DOCTYPE html>
 <html lang="en">
 <head>
 <meta charset="UTF-8">
 <title>Data Mahasiswa</title>
 </head>
 <body>
 <div>
    <h1>Data Mahasiswa</h1>
    <div>
        <div>
        <ol>
        @forelse ($mahasiswa as $val)
        <li>{{$val}}</li>
        @empty
        <div>Tidak ada data...</div>
        @endforelse
        </ol>
        </div>
    </div>
 </div>

 </body>
 </html>
```

Method HTTP	URL Pola	Fungsi di Controller
GET	/resource	index() menampilkan semua data
GET	/resource/create	create() menampilkan form tambah data
POST	/resource	store() menyimpan data baru
GET	/resource/{id}	show() menampilkan satu data berdasarkan ID
GET	/resource/{id}/edit	edit() menampilkan form edit
PUT/PATCH	/resource/{id}	update() mengubah data yang ada
DELETE	/resource/{id}	destroy() menghapus data

TUGAS - UNGUIDED

1.1 TUGAS - Router

1. Buat minimal 5 route tanpa parameter
2. Buat minimal 3 route dengan parameter
3. Buat minimal 3 route dengan optional parameter

Route yang dibuat nantinya akan diimplementasikan pada tugas besar di mata kuliah teori.

2.1 TUGAS - View

Pada pembelajaran PHP dasar kita pasti sudah mempelajari pengelolaan asset. Pada tugas kali ini terapkan pengelolaan asset pada framework Laravel.

1. Buat folder asset yang didalamnya birisi folder css, js, img.
2. Buat view yang menampilkan gambar yang terletak di folder img
3. Buat view yang mengakses css dan javascript dari folder css dan js.

3.1 TUGAS - BLADE TEMPLATE ENGINE

Kerjakan tugas berikut menggunakan blade

1. Buat perulangan for untuk menampilkan bilangan 1 s.d 10
2. Buat perulangan while untuk menampilkan bilangan 1 s.d 10
3. Buat perulangan foreach untuk menampilkan nilai pada route berikut

```
Route::get('/mahasiswa', function () {
 $nilai = [80,64,30,76,95];
    return view('mahasiswa',$nilai);
 });
```

4.1 TUGAS - Controller

1. Buat root yang mengarah ke controller
2. Tampilkan template pada view menggunakan controller