View

Pendahuluan

Pada praktikum kali ini anda akan mempelajari tentang view serta template engine pada framework Laravel.

Learning Objectives

1. Mahasiswa memahami konsep view dalam Web Framework
2. Mahasiswa menerapkan template engine pada view Web Framework
3. Mahasiswa melakukan pe-layout-an pada view Web Framework

Alat dan Bahan

1. PC atau Laptop
2. Text Editor/IDE (rekomendasi PHPStorm atau VSCode)
3. Web Browser
4. PHP

Intro

View merupakan tempat bagi kita untuk meletakkan kode-kode HTML. Kita tidak akan menggunakan lagi file .html ya. Tapi kita tidak hanya menggunakan HTML karena kita perlu menghandle tampilan dengan lebih canggih. Menampilkan data yang diberikan oleh controller. Untuk itu kita akan menggunakan templating engine, yaitu Blade.

Blade merupakan templating engine bawaan Laravel. Berguna untuk mempermudah dalam menulis kode tampilan. Dan juga memberikan fitur tambahan untuk memanipulasi data di view yang dilempar dari controller. Blade juga memungkinkan penggunaan plain PHP pada kode View. Karena Laravel menggunakan *templating engine* bawaan Blade, maka setiap *file* View diakhiri dengan .blade.php. Misal: index.blade.php, home.blade.php, product.blade.php. Contoh sederhana dari view dengan nama *file* hello.blade.php adalah sebagai berikut.

```
<!-- View pada resources/views/hello.blade.php -->
<html>
 <body>
      <h1>Hello, {{ $name }}</h1>
 </body>
</html>
```

View tersebut dapat dijalankan melalui Routing, dimana *route* akan memanggil View sesuai dengan nama *file* tanpa ‘blade.php’.

```
Route::get('/hello', function () {
 return view('hello', ['name' => 'Andi']);
});
```

Setiap kita membuat view, kita mungkin akan menampilkannya melalui router atau controller menggunakan global view helper. Selain menggunakan global view helper, kita juga dapat menggunakan view facade untuk menampilkan view.

```
Use Illuminate\Support\Facades\View;


return View::make('hello', ['name' => 'Andi']);
```

Seperti yang kita lihat, argumen pertama yang diteruskan ke view helper sesuai dengan nama file view di direktori resources / views. Argumen kedua adalah larik data yang harus tersedia untuk tampilan. Dalam hal ini, laravel meneruskan variabel nama yang ditampilkan dalam tampilan menggunakan sintaks Blade.

View di dalam direktori

Jika di dalam direktori `resources/views` terdapat direktori lagi untuk menyimpan *file* view, sebagai contoh hello.blade.php ada di dalam direktori blog, maka kita bisa menggunakan “dot” notation untuk mereferensikan direktori, sehingga syntax dalam route akan seperti berikut:

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

 return view('blog.hello', ['name' => 'Andi']);

});
```

Menampilkan view dari controller

View dapat dipanggil melalui Controller. Sehingga Routing akan memanggil Controller terlebih dahulu, dan Controller akan me-*return* view yang dimaksud.

```
Route::get(‘/hello,[WelcomeController::class, ‘hello’]);
```

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class WelcomeController extends Controller

{

      public function hello() {

      return view('blog.hello', ['name' => 'Andi']);

       }

}
```

Meneruskan data ke view

Seperti yang anda lihat di contoh sebelumnya, anda dapat meneruskan data array ke view agar data tersebut tersedia untuk view:

```
return view('hello', ['name' => 'Andi']);
```

Saat meneruskan informasi dengan cara ini, data harus berupa array dengan pasangan kunci / nilai. Setelah memberikan data ke view, anda kemudian dapat mengakses setiap nilai dalam view menggunakan kunci data seperti: `<?php echo $name; ?>` atau `{{$name}}`.

Sebagai alternatif untuk meneruskan array data lengkap ke fungsi view helper, anda dapat menggunakan metode **with** untuk menambahkan bagian data individual ke view. Metode with mengembalikan instance view objek sehingga anda dapat melanjutkan rangkaian metode sebelum mengembalikan tampilan:

```
return view('hello')
             ->with('name','Andi')
             ->with('occupation','Astronaut');
```

Berbagi data dengan semua view

Terkadang, anda mungkin perlu berbagi data dengan semua tampilan yang dirender oleh aplikasi anda. anda dapat melakukannya dengan menggunakan metode berbagi tampilan fasad. Biasanya, anda harus melakukan panggilan ke share methodi dalam metode boot penyedia layanan. anda bebas menambahkannya ke kelas `App\Providers\AppServiceProvider` atau membuat penyedia layanan terpisah untuk menampungnya:

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
```

```
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        View::share('key', 'value');
    }
}
```

Blade Layout, Section, dan Component

Dalam membuat suatu tampilan, seringkali ada beberapa bagian yang sama dan selalu ditampilkan di setiap halaman view. Bagian-bagian ini dapat dibuat *template* sehingga tidak perlu dibuat berulang kali di setiap halaman view.

Layout dan Section

Pada laravel, layout digunakan untuk membuat master view yang akan selalu ditampilkan oleh view-view child yang menggunakannya. Dalam sebuah layout kita bisa memberikan tempat-tempat yang bisa digunakan oleh child view. Tempat-tempat tersebut adalah section. Misalnya, dalam layout utama, kita definisikan section sidebar, main_content, dan footer. Selanjutnya, setiap child view yang menggunakan layout utama dapat menempatkan kode view di masing-

masing section yang tersedia di layout utama. Pada layout, setiap kode html akan digunakan oleh child view. Sehingga child view tidak perlu mendefinisikan tag html, head, title, dll pada tiap-tiap view. Terdapat beberapa istilah yang digunakan untuk menerapkan layout.

@yield

@yield("nama_section") digunakan untuk mendefinisikan bagian dari layout yang akan digunakan dan diisi oleh child view.

@section

@section digunakan selain untuk mendefinisikan sebuah section, juga bisa untuk mengisi section yang diharapkan oleh parent view / layout melalui @yield. Diakhiri dengan @endsction.

@parent

Dalam child view kita bisa menampilkan juga konten yang ada pada parent dalam section tertentu, hal tersebut dilakukan dengan @parent.

@extends

Extends digunakan pada setiap child view yang ingin menggunakan sebuah view sebagai parent / layout.

Berikut adalah contoh dari sebuah layout dari master/parent view.

```
<!-- Disimpan di resources/views/layouts/app.blade.php -->
 <html>
 <head>
      <title> Halaman @yield('title')</title>
 </head>
 <body>
      @section('sidebar')
            Ini adalah master sidebar.
      @show
      <div class="container">
            @yield('content')
      </div>
 </body>
</html>
```

Keterangan:

Kode yang merupakan layout global, mendefinisikan title, sidebar, dan content yang dapat diisi oleh child view yang menggunakannya.

Berikut adalah contoh child view yang menggunakan layout app.blade.php.

```
<!—Disimpan di resources/views/child.blade.php -->

 @extends('layouts.app')

 @section('title', 'Profil')

 @section('sidebar')

 @parent

      <p>Sidebar halaman Profil.</p>

 @endsection

 @section('content')

     <p>Ini adalah bagian konten. NIM - Nama</p>

 @endsection
```

Keterangan: @extends(“layouts.app”) digunakan untuk menjadikan file view app.blade.php. sebagai master view. Kemudian kita isi section title dengan “Profil” yang akan direnderer sebagai <title> Halaman Profil </title>. Kemudian kita juga mengisi section sidebar menggunakan directive @parent. Sehingga ditampilkan konten sidebar parent/master serta menampilkan kalimat “Sidebar halaman Profil”. Setelah itu kita juga isi section “content” dengan tulisan “Ini adalah bagian konten. NIM - Nama”.

Component

Component berfungsi untuk membuat view yang dapat kita gunakan berulang kali. Berbeda dengan layout yang bertindak sebagai master, component dapat dianggap sebagai child view yang bisa kita pakai di view lain yang membutuhkannya.

Misalnya dalam pengembangan sebuah aplikasi kita akan membutuhkan view untuk alert yang memberikan notifikasi kepada pengguna aplikasi terkait informasi, peringatan ataupun pesan error. Alert ini akan digunakan berulang kali di aplikasi. Oleh karena itu kita bisa membuatnya sebagai component yang bisa digunakan di view lainnya.

Berikut adalah contoh component untuk membuat alert.

```
<!-- Disimpan di resources/views/components/alert.blade.php -->

 <div class="alert alert-danger">

{{ $slot }}

</div>
```

Di bawah ini adalah contoh view yang menerapkan component alert.

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

// kode…

@component("alert")

<b>Tulisan ini akan mengisi variabel $slot</b>

@endcomponent
```

Bootstrap pada Laravel

Laravel menyediakan titik awal dasar menggunakan Bootstrap. Secara default, Laravel menggunakan NPM untuk menginstal paket frontend ini. Untuk menggunakannya, ikuti langkah-langkah berikut:

1. Lakukan instalasi Node.js. Installer dapat diperoleh dari https://nodejs.org/en/. Sesuaikan dengan sistem operasi yang digunakan.
2. Untuk memastikan instalasi Node.js dan NPM berjalan lancar, Anda dapat memeriksanya dengan menjalankan dua perintah berikut melalui Command Prompt:

   node -v

   npm -v

   Setelah Anda mengetikkan kedua perintah tersebut, Command Prompt akan menunjukkan versi Node.js dan NPM yang ter-install di komputer.

3. Kemudian melalui command prompt, ubah direktori ke dalam direktori project laravel yang telah dibuat sebelumnya.
4. Scaffolding Bootstrap yang disediakan oleh Laravel terletak di dalam paket Composer, yang dapat diinstal menggunakan Composer:laravel/ui.

   ```
   composer require laravel/ui
   ```

5. Setelah paket diinstal, kita dapat menginstal scaffolding frontend menggunakan perintah Artisan:laravel/ui

   ```
   php artisan ui bootstrap

   php artisan ui bootstrap --auth
   ```

6. Sebelum mengompilasi CSS, install dependensi frontend proyek Anda menggunakan Node package manager (NPM) :

   ```
   npm install
   ```

7. Setelah paket diinstal, Anda dapat menggunakan perintah berikut untuk mengompilasi aset Anda .

   ```
   npm run dev
   ```

Menggunakan Template Bootstrap di Laravel

Sebenarnya pada project laravel, sudah ada file css bootstrap secara default pada pertama kali kita menginstall laravel. Letaknya ada pada file app.css dalam folder css. Anda dapat langsung menggunakannya dengan menghubungkan file app.css tersebut dengan file view anda.
Untuk mengubungkan file css ke laravel anda dapat menggunakan syntax berikut:

```
<link rel="stylesheet" type="text/css" href="/css/style.css">
```

Atau

```
<link rel="stylesheet" type="text/css" href="{{
asset('/css/app.css') }}">
```

Dan file JSnya:

```
<script type="text/javascript" src="/js/app.js"></script>
```

Atau

```
<script type="text/javascript" src="{{ asset('/js/app.js')
}}"></script>
```

Secara default linknya dimulai dari folder public. Jadi anda bisa meletakan file css dan js didalam folder public.

Praktikum

Praktikum 1 – Membuat View Web Framework Laravel

1. Buatlah project laravel baru degan nama 03_praktikum_web_lanjut_satu dengan menginstall depedency laravel ui dan ui bootstrap.
2. Buatlah beberapa view dengan ektensi .blade.php untuk halaman pada contoh kasus praktikum sebelumnya.

1	Halaman Home Menampilkan halaman awal website	home.blade.php

2	Halaman Products Menampilkan daftar product (route prefix)	product.blade.php
3	Halaman News Menampilkan Daftar berita (route param)	news.blade.php
4	Halaman Program Menampilkan Daftar Program (route prefix)	program.blade.php
5	Halaman About Us Menampilkan About Us (route biasa)	about-us.blade.php
6	Halaman Contact Us Menampilkan Contact Us (route resource only)	contact-us.blade.php

3. Sambungkan View tersebut dengan route yang sudah dibuat sebelumnya.

Praktikum 2 - Menghubungkan Template Bootstrap dengan Laravel

1. Buatlah kelompok maksimal beranggotakan 2 orang.
2. Silahkan unduh template bootstrap yang sudah disiapkan pada link berikut:

   https://drive.google.com/drive/folders/1hwsYrbJ9Q4W2FOMskvrnm5zeOM5j_pUG?usp=sharing

3. Modifikasi praktikum 1 diatas dan gunakan template bootstrap pada project laravel tersebut.
4. Buatlah sebuah master view (layout) yang terdiri dari Header, Sidebar, Content, dan Footer. Master View ini akan diterapkan untuk view view pada praktikum satu.
5. Simpan praktikum ini dalam project anda menggunakan Git publish ke github dengan nama 03_praktikum_web_lanjut_dua ke repository github anda dan lakukan commit pada setiap perubahan yang anda lakukan.