TRAINING 7 JURUS MENJADI PROGRAMMER

STUDI KASUS PROGRAMMING LARAVEL

**Tahap Persiapan:**

1. **Tahap instalasi**

Tahap persiapan pembuatan server web untuk laravel:

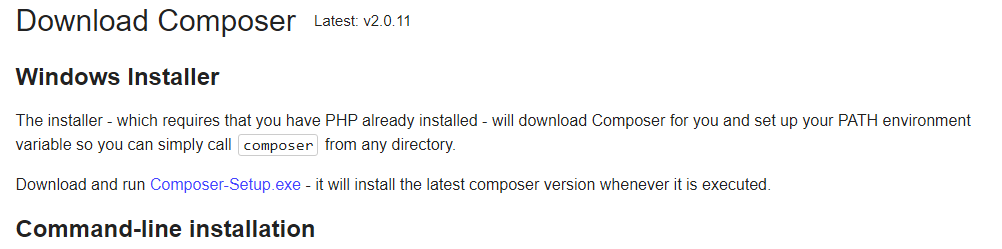
1. DOWNLOAD XAMPP SERVER

Klink link: <https://www.apachefriends.org/download.html>

Pilih 

1. Download composer

Klink link: <https://getcomposer.org/download/>

klik download link composer-setup.exe

1. Install xampp
2. Install Composer

**Jurus 1:**

**Kuasai Algoritma**

Apa yang dimaksud dengan algoritma (*algorithm*)? Secara umum, pengertian algoritma adalah suatu urutan dari beberapa langkah logis dan sistematis yang digunakan untuk menyelesaikan masalah tertentu.

Pendapat lain mengatakan definisi algoritma adalah proses atau serangkaian aturan yang harus diikuti dalam perhitungan atau operasi pemecahan masalah lainnya, terutama oleh komputer.

Dalam algoritma terdapat rangkaian terbatas dari beberapa intruksi untuk menghitung suatu fungsi yang jika dieksekusi dan diproses akan menghasilkan output, lalu berhenti pada kondisi akhir yang sudah ditentukan.

Berikut ini tiga bentuk dasar algoritma:

* Algoritma Sekuensial (*Sequence Algorithm*); yaitu sebuah perintah yang dapat tersusun dengan sistematis dan berurutan yang nantinya muncul beberapa instruksi.
* Algoritma Perulangan (*Looping Algorithm*); yaitu sebuah perintah yang dapat digunakan untuk mengulang beberapa banyak perintah dengan memperhitungkan syarat-syarat tertentu.
* Algoritma Percabangan atau Bersyarat (*Conditional Algorithm*); yaitu suatu perintah yang dapat digunakan untuk memilih salah satu dari beberapa pilihan yang ada.

Algoritma memiliki lima ciri utama yang saling berhubungan satu dengan lainnya. Menurut [Donald E. Knuth](https://en.wikipedia.org/wiki/Donald_Knuth" \t "_blank), adapun ciri-ciri algoritma adalah sebagai berikut:

1. **Ada Input**, yaitu permasalahan yang dihadapi dan akan dicarikan solusinya. Algoritma memiliki nilai nol atau lebih input (masukan).
2. **Ada Proses**, yaitu langkah-langkah yang harus dilakukan untuk mencapai tujuan akhir.
3. **Ada Output**, yaitu solusi atau tampilan akhir yang didapatkan dari suatu algoritma. Algoritma memiliki minimal satu output.
4. **Ada intruksi-intruksi yang jelas dan tidak ambigu**, yaitu instrukti yang jelas dalam algoritma sehingga tidak terjadi kesalahan dalam menghasilkan output.
5. **Ada tujuan akhir yang dicapai**, yaitu akhir dari program dimana program akan berhenti ketika tujuan akhir telah tercapai.

## **Contoh Algoritma**

Pada dasarnya algoritma tidak hanya dalam ilmu komputer dan matematika saja, tetapi dapat juga diimplementasikan dalam berbagai kegiatan manusia sehari-hari.

Berikut ini adalah beberapa contoh algoritma:

### 1. Menentukan Bilangan Ganjil atau Genap

Contoh algoritma menentukan bilangan ganjil atau genap:

* Masukkan sebuah bilangan bulat.
* Bagi bilangan tersebut dengan angka 2.
* Jika bilangan tersebut habis dibagi dengan 2 maka bilangan tersebut genap.
* Jika bilangan tersebut tidak habis dibagi dengan 2 (terdapat sisa 1) maka bilangan tersebut ganjil.

2. Masak Mie Instan

Contoh algoritma memasak mie instan:

* Siapkan 1 bungkus mie instan, 400 ml air (2 gelas), panci, mangkuk, sendok dan garpu.
* Masukkan air ke dalam panci.
* Masak air di atas kompor dengan api sedang.
* Biarkan hingga air mendidih.
* Masukkan mie instan ke dalam air mendidih.
* Aduk perlahan hingga 3 menit.
* Jika mie sudah matang masukkan bumbu.
* Aduk mie hingga bumbu merata.
* Sajikan mie instan.

**EXAM:**

**1. Buat algoritma menukar isi 2 gelas**

**2. Buat algoritma menghitung Gaji**

**3. Buat algoritma menghitung luas persegi panjang**

**Jurus 2:**

**Kuasai Struktur Data**

#### **Pengertian Struktur Data**

**Dalam istilah ilmu komputer**, **struktur data adalah** cara penyimpanan , pengorganisasian , dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien.   
  
**Dalam teknik pemrograman**, struktur data berarti tata letak data yang berisi kolom-kolom data,baik itu kolom yang tampak oleh pengguna (user) ataupunkolom yang hanya digunakan untuk keperluan pemrograman yang tidak tampak oleh pengguna. Setiap baris dari kumpulan kolom-kolom tersebut dinamakan catatan (record). Lebar kolom untuk data dapat berubah dan bervariasi. Ada kolom yang lebarnya berubah secara dinamis sesuai masukan dari pengguna dan juga ada kolom yang lebarnya tetap.   
  
Dengan sifatnya ini, sebuah struktur data dapat diterapkan untuk pengolahan database, misalnya untuk keperluan data keuangan, atau untuk pengolah kata (word processor) yang kolomnya berubah secara dinamis. Contoh struktur data dapat dilihat  pada file-file spreadsheet, database, pengolahan kata, gambar yang dikompres, dan pemampatan file (kompres) dengan teknik tertentu yang memanfaatkan struktur data.

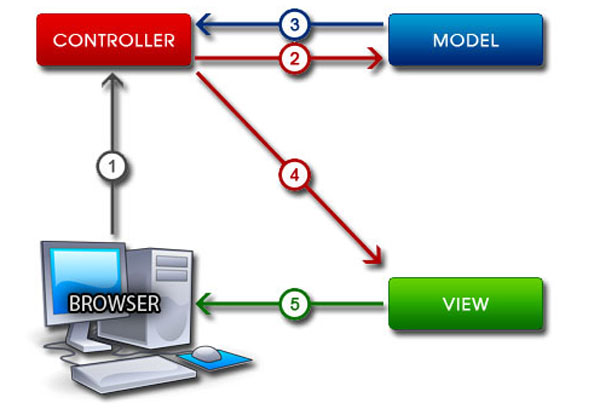
#### **Klasifikasi Data**

Pada garis besarnya, data dapat dikategorikan menjadi:  
  
A.**Tipe Data Sederhana**, yang terdiri dari:  
  
– Data Sederhana Tunggal, misalnya Integer, Real, Boolean, dan Karakter.  
– Data Sederhana Majemuk, misalnya String.  
  
B. **Struktur Data**  
  
– Struktur Data Sederhana, misalnya Array dan Record.  
– Struktur Data majemuk, terdiri dari:  
      o Linier, misalnya: Stack, Queue, dan Linier Linked List.  
      o Nonlinier, misalnya Binary Tree, Binary Search Tree, Graph, dll.  
  
Pemakaian struktur data yang tepat di dalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

**Jurus 3:**

**Kuasai Konsep Pemrograman MVC,**





Sebelum belajar lebih jauh mengenai laravel, kita mulai terlebih dahulu dari konsep dasar yang digunakan, yaitu konsep MVC (Model, View, Controller). Laravel merupakan salah satu framework yang memisahkan antara data (**Model**) dari tampilan (**View**) dan cara bagaimana memprosesnya (**Controller**).

Apa itu Model?  **Model** adalah sebagai penghubung antara Controller dengan database yang berguna untuk mengambil data. Sementara  **Controller**, dapat diartikan sebagai kendali atau pengendali diambil dari kata controll bisa juga diartikan jembatan antara view dan model. Controller memiliki tugas mengendalikan ataupun menjembatani antara apa yang di-*request*yang selanjutnya Controller akan memprosesnya. Pada Laravel file Controller berada di app/Http/Controllers. Adapun **View** digunakan untuk menerima dan merepresentasikan data kepada user. Letak file View berada pada folder Resources/views.

**Struktur laravel**

Berikut adalah beberapa struktur dari laravel :

* **App**Berisi kumpulan logika dan alur sistem yang akan dibuat.
* **Bootstrap**Direktori ini berisi beberapa file kerangka *framework* laravel termasuk *autoload*yang befungsi untuk mengoptimasi kinerja sistem yang dihasilkan.
* **Config**Mencakup seluruh konfigurasi *framework* mulai dari *database, app, mail*, dan lain sebagainya.
* **Database**Sebagai folder penampung file *migrations* dan *seeds* yang berhubungan langsung ke pengolahan data dalam database.
* **Public**Sebagai folder yang akan diakses oleh *public/users* nantinya. Folder ini juga berisi file-file assets (css/js/images/dll)
* **Resource**Folder ini berisi semua *resource* untuk bagian *frontend.*
* **Routes**Folder ini digunakan untuk menentukan format url yang digunakan untuk mengakses halaman yang dibuat
* **Storage**Berisi *compiled blade templates, session, cache, logs* dan file lainnya yang di-*generate*otomatis oleh *framework*.
* **Tests**Berisi semua *test* yang kita buat untuk aplikasi.
* **Vendor**Berisi seluruh library-library yang digunakan dalam *framework* laravel maupun yang diinstall melalui composer.

**Menguasai Route**

**Jurus 4:**

**Kuasai Penyimpanan (storage) Database**

**Membuat Model Database**

Pada [tutorial sebelumnya](https://tianrosandhy.com/blog/tutorial-dasar-penggunaan-model-view-dan-controller-di-laravel-part-2" \t "_blank) kita sudah belajar membuat Controller, View, dan Model untuk menampilkan data. Dalam sebuah model, kita harus mendefinisikan nama tabel, nama primary key, beserta dengan list kolom-kolomnya dengan tepat. Jika sudah, maka untuk memanggil data pada tabel yang bersangkutan, kita cukup memanggil melalui nama modelnya saja dengan method static ::get(). Lalu bagaimana caranya menampilkan data pada 2 buah tabel yang memiliki relasi? Hal tersebut bisa diselesaikan dengan sangat mudah di Laravel. Sebelum lanjut ke penggunaannya, kita perlu mengenal model relasi database yang paling umum digunakan.

1. Relasi one to one. Relasi dimana 1 data pada sebuah tabel hanya memiliki relasi ke 1 data pada tabel yang lain. Misalnya, 1 data tabel tb\_User memiliki relasi 1 nomor telepon di tabel tb\_Contact. Di laravel, kita menggunakan nama method **hasOne(...)** untuk mendefinisikan hal ini.
2. Relasi one to many. Relasi dimana 1 data pada sebuah tabel memiliki relasi ke beberapa data pada tabel yang lain. Misalnya, 1 data tabel tb\_Category memiliki relasi banyak data barang di tb\_Inventory. Atau dengan kata lain, 1 kategori memiliki banyak data inventory. Di laravel, kita menggunakan nama method **hasMany(...)** untuk mendefinisikan hal ini.
3. Relasi many to one (One to many Inverse). Relasi ini merupakan kebalikannya dari relasi one to many. Misalnya kita ingin mengetahui data barang di tb\_Inventory memiliki kategori apa, maka relasi ini yang akan digunakan. Di laravel, kita akan sering menggunakannya dengan nama method **belongsTo(...)**
4. Relasi many to many. Relasi dimana banyak data pada sebuah tabel memiliki relasi ke banyak data juga pada tabel yang lainnya. Relasi tersebut terbentuk melalui sebuah tabel bantu. Misalnya, banyak data pada tabel tb\_Siswa memiliki relasi peminjaman ke banyak data pada tabel tb\_Buku. Relasi tersebut terbentuk dengan tabel bantu bernama tb\_Transaksi. Relasi tersebut dapat dilakukan di Laravel dengan method **belongsToMany(...).**

Untuk mendefinisikan relasi pada sebuah model di Laravel, pertama-tama kita pilih dulu model relasi yang ingin kita buat. Misalnya pada [contoh tabel di tutorial sebelumnya](https://tianrosandhy.com/blog/tutorial-dasar-penggunaan-model-view-dan-controller-di-laravel-part-2" \t "_blank) ada 2 buah tabel yaitu tabel tb\_Inventory dan tb\_Inventory\_Category. Apa relasi antar kedua tabel tersebut? Relasinya adalah one to many pada tb\_Inventory\_Category ke tb\_Inventory (karena 1 data kategori bisa memiliki banyak data barang), dan relasi inverse one to many pada tb\_Inventory ke tb\_Inventory\_Category (karena banyak data kategori masing-masing hanya memiliki 1 data kategori). Jika relasi sudah diketahui, sekarang model sudah bisa dibuat. Kita buka kedua file model tersebut, dan kita buat definisinya seperti ini :

<?php

#InventoryModel.php

namespace App\Model;

use Illuminate\Database\Eloquent\Model;

class InventoryModel extends Model

{

protected $table = 'tb\_inventory';

protected $primaryKey = 'id';

protected $fillable = [

'nama\_barang',

'jumlah',

'kategori',

'deskripsi',

'created\_at',

'updated\_at'

];

//relasi many to one (Saya adalah anggota dari model ......)

public function get\_kategori(){

return $this->belongsTo('App\\Model\\InventoryCategoryModel', 'kategori', 'id');

}

}

<?php

#InventoryCategoryModel.php

namespace App\Model;

use Illuminate\Database\Eloquent\Model;

class InventoryCategoryModel extends Model

{

protected $table = 'tb\_inventory\_category';

protected $fillable = [

'nama\_kategori',

'deskripsi',

'created\_at',

'updated\_at'

];

//relasi one to many (Saya memiliki banyak anggota di model .....)

public function get\_inventory(){

return $this->hasMany('App\\Model\\InventoryModel', 'kategori', 'id');

}

}

Pendefinisian relasi di model dibuat dengan membuat sebuah method dengan nama bebas. Nama tersebut nantinya yang akan dipanggil ketika relasi ingin dijalankan. Isinya method tersebut hanya return $this-> dengan nama method relasi yang sesuai dengan kebutuhan. **hasOne()** untuk relasi one to one, **hasMany()** untuk relasi one to many, atau **belongsTo()** untuk relasi inverse one to many. Parameter pertama adalah alamat lengkap class model dimana relasi akan diberikan, parameter kedua adalah nama kolom relasi trigger, dan parameter terakhir (opsional) adalah parameter primary key target relasi dijalankan. Kalau bingung dengan parameternya, pastikan saja dulu alamat model relasinya sudah benar, sisanya kalau error tinggal dituker-tuker aja (pengalaman banget sering ketuker-tuker).

Dari contoh script diatas, Relasi **get\_kategori** di model InventoryModel akan memanggil data kategori yang sesuai dengan kolom kategori di model InventoryCategoryModel, dan relasi **get\_inventory** akan memanggil data inventory apa saja yang ada di kategori bersangkutan. Cara memanggilnya adalah dengan menggunakan method **with('nama\_relasi')** saat pemanggilan model. Contoh pemanggilannya dapat kita buat di file HomeController.

//sebelumnya :

$inventory = InventoryModel::get();

//diganti menjadi :

$inventory = InventoryModel::with('get\_kategori')->get();

//nama get\_kategori diambil dari nama relasi yang dibuat pada model InventoryModel

Apabila cara pemanggilannya sudah benar, maka variabel $inventory sudah berisi data inventory beserta dengan relasi yang bernama 'get\_kategori'. Untuk menampilkannya, kita harus menyebutkan nama relasinya dulu, kemudian diikuti dengan nama kolom relasi yang ingin ditampilkan. Sebagai contoh, saya ingin menampilkan data "nama\_kategori" di tabel tb\_Inventory\_Category. Karena relasi sudah terbentuk dengan nama "get\_kategori", maka pemanggilan kategori di view home.blade.php berubah menjadi seperti ini :

<!--sebelumnya-->

@foreach ($data as $row)

<tr>

<td>{{ $row->nama\_barang }}</td>

<td>{{ $row->jumlah }}</td>

<td>{{ $row-kategori }}</td>

<td>{{ $row->deskripsi }}</td>

<td></td>

</tr>

@endforeach

<!--sekarang-->

@foreach ($data as $row)

<tr>

<td>{{ $row->nama\_barang }}</td>

<td>{{ $row->jumlah }}</td>

<td>{{ $row->get\_kategori->nama\_kategori }}</td>

<td>{{ $row->deskripsi }}</td>

<td></td>

</tr>

@endforeach

Apabila pembuatan relasi di model tidak ada error, script pemanggilan di controller juga tidak error, dan nama relasi yang disebutkan pada view juga tidak error, hasilnya adalah seperti ini :

Konsep relasi di Laravel ini sangat layak untuk dipelajari dan terus didalami. Kalau sudah dipahami, kita bisa membuat query database menjadi lebih rapi.

**Jurus 5:**

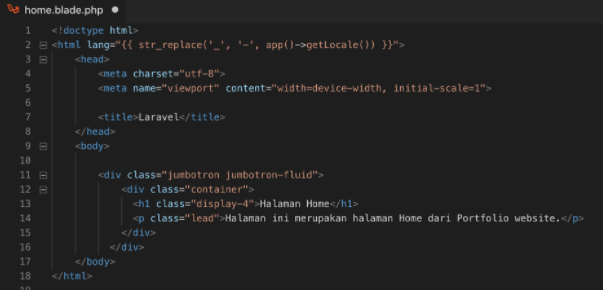
**Kuasai Tampilan (view) : CSS dan HTML**

**Membuat View (tampilan)**

**Tutorial Halaman Web dengan Laravel View dan Blade**

**Pada tutorial ini akan dibuat tampilan dari halaman web dengan memanfaatkan CSS dan framework CSS Boostrap agar menghasilkan tampilan yang lebih professional. Menggunakan Template di View Buat sebuah file home.blade.php dan letakkan dalam folder view. Pada file ini akan dibuat sebuah halaman sederhana, dengan memanfaatkan bootstrap. Misalnya dengan menggunakan Jumbotron pada contoh ini:**

[**https://getbootstrap.com/docs/4.2/components/jumbotron/**](https://getbootstrap.com/docs/4.2/components/jumbotron/)

****

**Jombotron Code**

**Manfaat Parameter Config**

**Ikuti langkah berrikut untuk melakukan konfigurasi sistem pada laravel :**

**• Buka.env file**

**• Ganti APP\_NAME dengan Protfolio**

**• Akess APP\_NAME dari Home dengan menggunakan perintah :**

**1. {{config(app.name)}}**

**Dengan parameter ke -2 sebagai default value :**

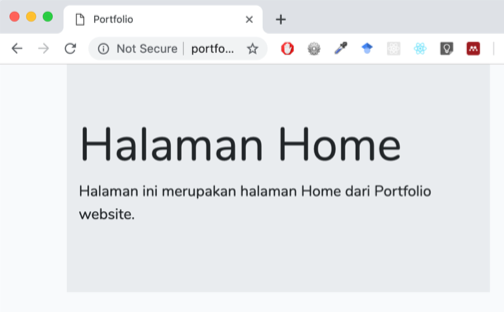
**2. {{config(‘app.name’, ‘Protfolio’)}}**

**Code yang dibuat akan terlihat seperti gambar berikut ini :**

****

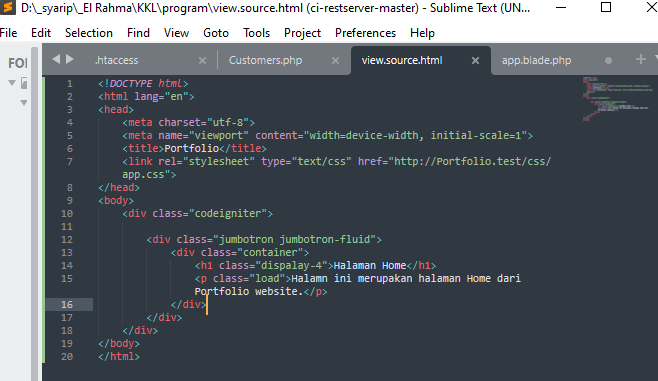
**Pengenalan Bootsrap**

* Laravel menggabungkan Bootstrap css dalam project yang di-generate.
* Bootstrap css tersedia dalam file /public/css/app.css
* Untuk menambahkan Bootstrap dalam web yang dibuat, pada halaman html bagian dapat ditambahkan**:**

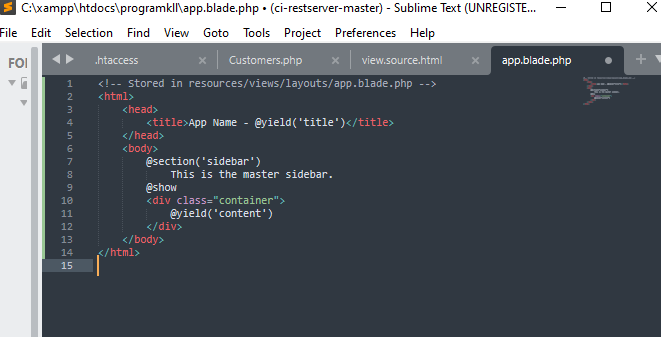
**<link rel="stylesheet" href="{{asset('css/app.css')}}">**

Tampilan Home dengan Boottsrap CSS

Apabila dicermati, Maka kode sumber dari halaman ini akan tampil sebagai berikut :

****

**Mendefinisikan Blade**

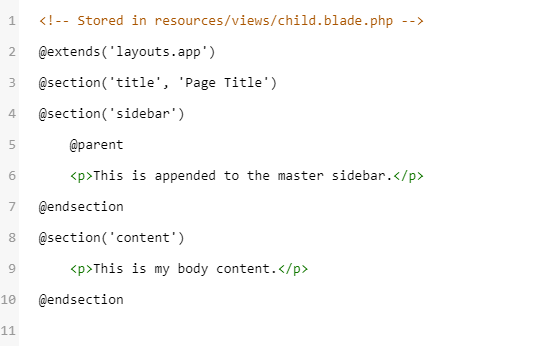
Potongan program berikut dalam file app.blade.php yang disimpan dalam sebuah folder baru dalam folders Views yang diberi nama Layouts.

Berikut adalah penjelasan dari potongan program tersebut:

* Blade merupakan pengaturan tampilan dengan menggunakan HTML markup, dengan penambahan beberapa directive dari Laravel.
* Pada contoh sebelumnya terlihat directive pada bagian @section dan @yield
* Directive @section mendefinisikan sebuah bagian (section) dari isi halaman web
* Directive @yield digunakan untuk menampilkan isi dari bagian tersebut.

**Menggunakan Blade**

Berikut adalah contoh penggunaan Blade Template



Berikut adalah penjelasan dari potongan program tersebut:

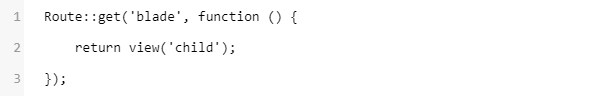
* View yang extend sebuah Blade layout akan mengganti setiap bagian (section) dari layout. Content yang dituliskan pada layout dapat disertakan dengan menggunakan directive @parent dalam section.
* Pada penggunaan @yield, default value dapat diatur atau dikirimkan dengan menuliskannya di argument kedua, contohnya**: @yield(‘section’, ‘Default Content’)**

Jadi manakah yang sebaiknya dipergunakan, Yield atau Section?

* Pada contoh sebelumnya terlihat penggunaan @yield dan @section yang serupa.
* Intinya untuk hal sederhana gunakan @yield, sedangkan untuk kebutuhan yang lebih advanced bisa digunakan @section… @show.
* Karena semua yang bisa dilakukan dengan @yield bisa dilakukan dengan @section… @show, namun tidak sebaliknya.

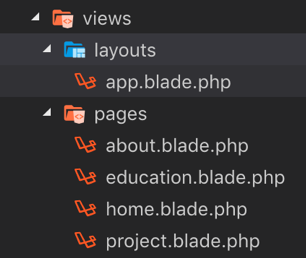
**Menampilkan Blade**

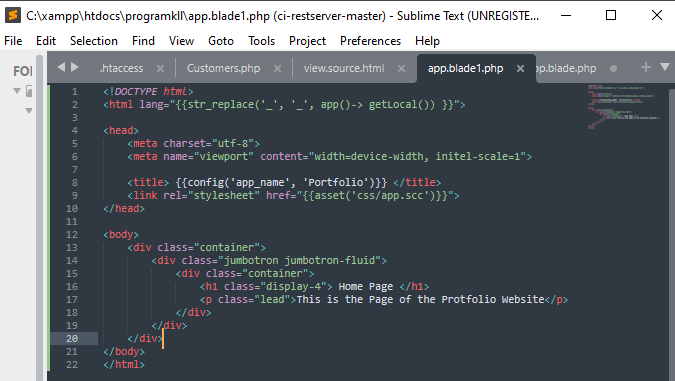
Blade views yang telah dibuat dapat ditampilkan dengan perintah view yang dapat diakses secara global.



**Layout**

* Buat sebuah folder layouts di dalam folder views
* Buat sebuah file bernama app.blade.php dalam folder tersebut
* Copy paste struktur utama dari halaman home.blade.php yang dibuat ke file tersebut.



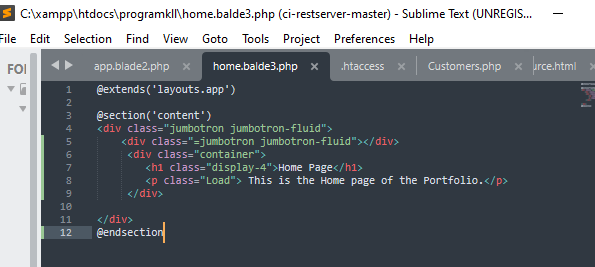


Isi File app.blade.php

**Silahkan mengganti bagian isi dari container dengan code menggunakan @yield seperti tampak pada gambar berikut:**

****

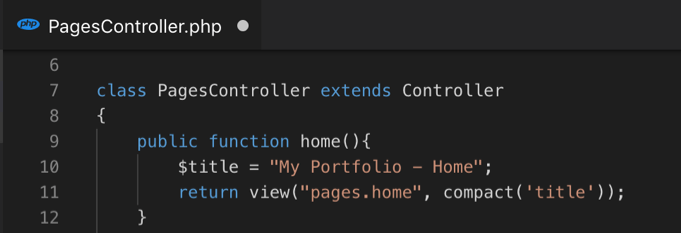
Silahkan mengubah isi dari **home.blade.php** dengan kode seperti terlihat pada gambar :



**Mengirim Value ke Blade**

Berikut adalah beberapa cara untuk mengirimkan Value ke Blade:

* Menggunakan compact(‘namavariabel’)



Blade Mneggunakan Compact

* Menggunakan ->width(‘namavar’, ‘vardikirim’). Parameter pertama adalah nama variabel untuk menerima di dalam blade, parameter kedua adalah variable dari value yang akan dikirimkan.



Balade menggunakan With

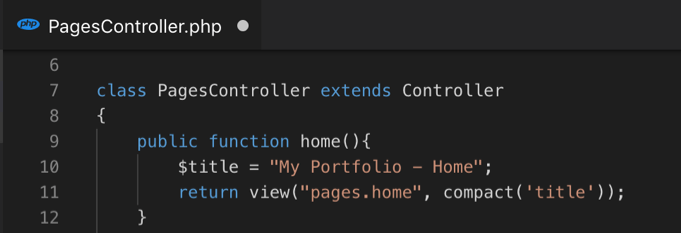
* Menggunakan ->width(‘namavar’, ‘var\_array\_dikirim’). Parameter pertama adalah nama variabel untuk menerima di dalam blade, parameter kedua adalah variable dari value yang akan dikirimkan berupa array.



Blade Menggunakan With dengan 2 Parameter

**Menerima multi Value dari Blade**

Berikut adalah potongan program yang dipergunakan untuk mengambil muti value yang dikirimkan menggunakan Blade.



Mendapatkan Multi Value dengan Blade

**Jurus 6:**

**Membuat Controller**

Mengenal konsep MVC (Model View Controller) yang digunakan di Larave

sekarang kita akan memanfaatkan tabel yang sudah ada tersebut untuk menampilkan data melalui controller dan view juga.

Pertama-tama, masih tetap dengan jendela command, jalankan perintah

**php artisan make:controller HomeController**

Perintah tersebut akan membuat sebuah ile controller baru yang bernama HomeController. Jika sudah berhasil, maka file controller tersebut dapat kita lihat di direktori **app/Http/Controllers/**. Jika dibuka, maka kita akan menemukan sebuah class kosong yang dapat kita isi dengan sembarang method.

Sebagai contoh, saya membuat sebuah method *index* dengan isi echo untuk menampilkan teks.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

//

public function index(){

echo 'Hai haiii.. Saya sedang belajar Laravel';

}

}

Setelah membuat controller tersebut, di laravel kita WAJIB mendefinisikan route untuk menentukan alamat pemanggilan dan baris kode apa yang akan dipanggil. Route dapat kita buat di  /routes/web.php. Isi route default bisa kita hapus, dan diganti menjadi seperti ini :

<?php

#Cara baca : memanggil root (/) = memanggil method index di HomeController

Route::get('/', 'HomeController@index');

Kalau sudah, maka hasilnya adalah seperti berikut ini

Untuk menampilkan sebuah tampilan/view, kita dapat mengaturnya dengan membuat sebuah view template. Tempat melatakkan view ada di direktori resource/views/. Sekarang kita akan membuat sebuah file bernama home.blade.php. "blade" itu sendiri merupakan ekstensi khusus untuk menjalankan template engine bawaan Laravel sehingga kita dapat membuat tampilan dengan sangat mudah. Tapi topik ini akan kita bahas lebih detail di lain waktu~

File home.blade.php tersebut bisa kita isi dengan teks apa saja, seperti misalnya dibawah ini :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contoh File Template</title>

</head>

<body>

<h1>Halaman Home</h1>

</body>

</html>

Kalau view sudah dibuat, kita akan memanggil view tersebut melalui controller. Laravel menyediakan fungsi view() untuk menampilkannya melalui controller. Sekarang kita kembali ke HomeController, dan hapus baris yang berisi echo dan diganti menjadi

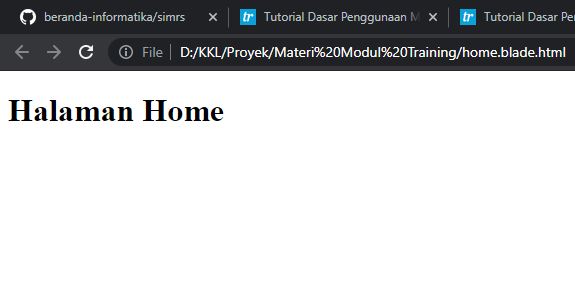
...

//memanggil view yang bernama home.blade.php

return view('home');

...

Hasilnya adalah seperti ini :



Seperti framework lainnya, kita bisa mengirim parameter tambahan ke view tersebut. Cara mengirim parameter adalah dengan menggunakan method tambahan ->with([...]) yang berisi array parameter yang ingin dikirim. Sebagai contoh, return view('home') tadi dapat kita lengkapi dengan parameter seperti ini :

....

return view('home')->with([

'title' => 'Contoh Judul Halaman',

'paragraf' => 'Lorem ipsum dolor sit amet, consectetur adipisicing elit. Id, dignissimos!'

]);

....

Contoh diatas akan mengirimkan parameter bernama title dan paragraf ke view "home", dan parameter tersebut dapat diakses dengan nama variabel *$title*dan *$paragraf*. Untuk melihat hasilnya, sekarang file View home.blade.php bisa kita update menjadi :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contoh File Template</title>

</head>

<body>

<h1>{{ $title }}</h1>

<p>

{!! $paragraf !!}

</p>

</body>

</html>

Kita melihat ada kurung yang modelnya sedikit berbeda diatas. ada kurung {{...}}, dan juga kurung {!! ... !!}. Pada dasarnya kedua kurung tersebut gunanya sama-sama menampilkan variabel, akan tetapi bedanya kurung {{ ... }} akan mengescape karakter yang dimunculkan, sedangkan kurung {!! ... !!} menampilkan data mentah-mentah tanpa escaping. Setelah disimpan, hasilnya adalah seperti ini :

**Jurus 7:**

**Project Sistem Informasi Keuangan**