PEMROGRAMAN WEB DAN PERANGKAT BERGERAK

XII
REKAYASA PERANGKAT LUNAK
SEMESTER GANJIL

SMK NEGERI 2 TRENGGALEK

KEGIATAN BELAJAR 2

- MVC (*MODEL VIEW CONTROLLER*) PART 1 -

A. KOMPETENSI DASAR

3.23 Menerapkan aplikasi *web* dengan model *view* controler (mvc)

4.23 Membuat aplikasi *web* menggunakan model *view* controler (mvc)

B. TUJUAN

Setelah proses belajar, berdiskusi, dan menggali informasi, peserta didik diharapkan mampu :

1. Memahami konsep Model *View* Control
2. Menerapkan konsep MVC pada pembuatan *web*

C. ALOKSI WAKTU

13 x 45 menit

D. DASAR TEORI

1. MVC (*MODEL VIEW CONTROLLER*)

a. Konsep MVC

MVC atau *Model View Controller* adalah sebuah pola desain arsitektur dalam sistem pengembangan *website* terdiri dari tiga bagian, yaitu:

- ***Model***- *Model* adalah bagian yang berhubungan langsung dengan database, model bertugas untuk memanipulasi data (select, insert, update, delete) serta menangani validasi dari bagian *Controller*, namun tidak dapat berhubungan langsung dengan bagian *View*, serta model digunakan sebagai *logic* bisnis suatu program.
- ***View***- *View* adalah bagian yang menangani *Presentation Logic*. Pada suatu aplikasi *web* bagian ini merupakan *template* yang berupa *file* HTML, *View* ini diatur oleh bagian *Controller*. Bagian *View* berfungsi untuk menerima dan mempresentasikan data kepada user, atau *View* ini bisa dibilang sebagai interface aplikasi. Bagian ini tidak memiliki akses langsung terhadap database atau bagian Model.
- ***Controller***- *Controller* adalah bagian yang mengatur hubungan antara bagian Model dan bagian *View*. *Controller* berfungsi untuk menerima sebuah request data dari user,

kemudian menentukan apa yang akan diproses oleh aplikasi tersebut. Selain itu, bagian *Controller* juga mengatur *routing* URL yang akan digunakan.

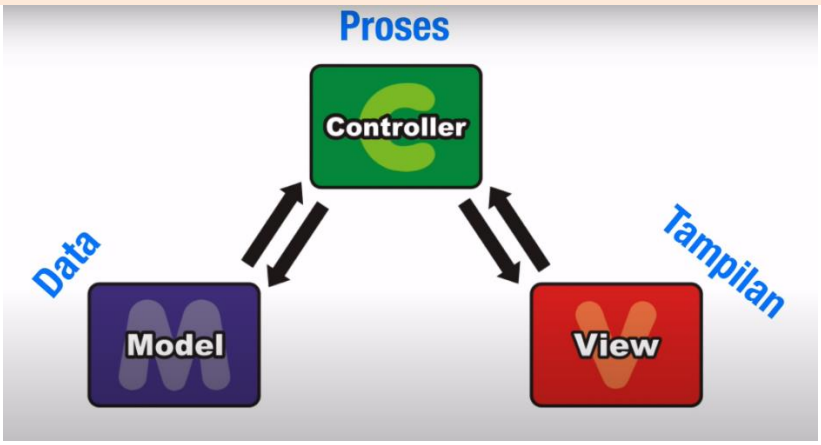
Tujuan penggunaan MVC

- Memisahkan antara tampilan, data dan proses.

Keuntungan penggunaan MVC

- Organisasi dan Struktur kode yang baik.
- Pemisahan logic dan tampilan
- Perawatan kode yang mudah Ketika ingin mengebangkan program
- Implementasi dari konsep OOP pada *web*
- MVC digunakan oleh banyak *Web* Application *Framework*

b. Alur Kerja MVC pada Sistem Pengembangan *Website*



Gambar 1. Konsep MVC

Keterangan:

1) Bagian *view* akan merequest informasi untuk bisa ditampilkan kepada pengguna.
2) Request tersebut kemudian diambil oleh *controller* dan diserahkan bagian model untuk diproses;
3) Model akan mengolah dan mencari data informasi tersebut di dalam database;
4) Model memberikan kembali pada *controller* untuk ditampilkan hasilnya di *view*;
5) *Controller* mengambil hasil olahan yang dilakukan di bagian model dan menatanya di bagian *view*.

Alur kerja MVC dalam sistem *website* sebenarnya cukup sederhana seperti ditunjukkan pada bagan di atas. Analogi yang lebih mudah sebagai berikut:

Gambar 2. Analogi alur kerja MVC

Ibaratnya ada seseorang yang sedang berada di sebuah restoran (pembeli). Dalam konsep MVC ini, pembeli adalah ***view***, pelayan adalah ***controller***, dan chef adalah **model**. Ketika Pembeli memesan salah satu menu, pelayan akan mencatat pesanan Pembeli dan memberikannya pada chef. Setelah itu, chef akan mencari bahan yang diperlukan di kulkas (database) dan mulai memasaknya untuk Pembeli. Setelah selesai dimasak, chef akan memberikan pada pelayan untuk diantarkan pada Pembeli.

c. Manfaat MVC

1) Proses Pengembangan *Website* Lebih Efisien

Konsep MVC bisa membuat proses pengembangan *website* lebih cepat. Sebab, MVC membagi *website* menjadi tiga bagian terpisah. Bagian model dan *controller* bisa dikerjakan oleh back end developer sementara *view* bisa dilakukan oleh front end developer dan UI UX tim.

Sebagai contoh, setelah tim UI UX menyelesaikan rancangan desain halaman depan, tim back end dan front end bisa mulai membuat kode pemrograman untuk desain itu. Tim UI UX bisa beralih merancang desain halaman lain misalnya halaman produk.

2) Testing Jadi lebih Mudah

MVC memungkinkan proses testing bisa dilakukan per bagian yang telah siap, tanpa menunggu keseluruhan *website* jadi. Selain itu, pembuatan dokumentasi dari setiap fitur bisa lebih efisien dan rapi karena proses testing bisa dilakukan per bagian.

3) Error atau Bug Lebih Cepat dan Mudah Ditangani

Pembagian *website* oleh MVC membuat developer bisa lebih fokus pada bagian pengembangannya masing-masing. Jadi, mereka bisa lebih cepat menemukan bug dan memperbaikinya.

4) Pemeliharaan atau Maintenance Lebih Mudah

Konsep MVC memungkinkan penggunaan *script* yang lebih terstruktur dan rapi. Hal ini dapat memudahkan tim developer dalam proses pengembangan dan maintenance *website*.

d. *Framework*

Dalam pemrograman *web*, mvc diaplikasikan sepenuhnya *dalam Web Application Framework*. ***Framework*** sendiri adalah sebuah kerangka kerja yang sudah siap digunakan sehingga dapat mempermudah dan mempercepat pekerjaan seseorang. Dengan menggunakan *Framework* seorang *developer* tidak perlu menulis *script* dari 0 (nol), jadi Anda cukup membuat *class* dan *function* sesuai dengan kebutuhan. Di dalam *Framework* sudah tertanam core *script* yang bertugas untuk melakukan *mapping class* dan *function* yang sudah di buat.

Framework sangat berbeda dengan CMS (*Content Management System*), meskipun berfungsi sama yaitu memudahkan dalam pembuatan *website* dan aplikasi. JIka menggunakan CMS, tidak perlu pusing *script*. Karena CMS dibuat secara *fix* dan hanya perlu mengatur konten dan *interface*-nya. Namun tidak demikian dengan *Framework*, membangun sebuah *website* ataupun aplikasi dengan *Framework*, tetap diharuskan menuliskan *script* sesuai dengan kebutuhan dan ruang lingkup yang disediakan oleh *Framework* itu sendiri.

e. *MVC Framework*

Contoh beberapa framework yang menggunakan konsep MVC:

Framework yang Menggunakan Konsep MVC	
Framework PHP	• Laravel • Symfony • CakePHP • Zend

	• Codeigniter • Yii
framework Python	• Django • Turbogears2 • Watson-*Framework*
framework Nodejs	• Express • Adonis • Sails.js • Total.js • Mean.js • Mojito

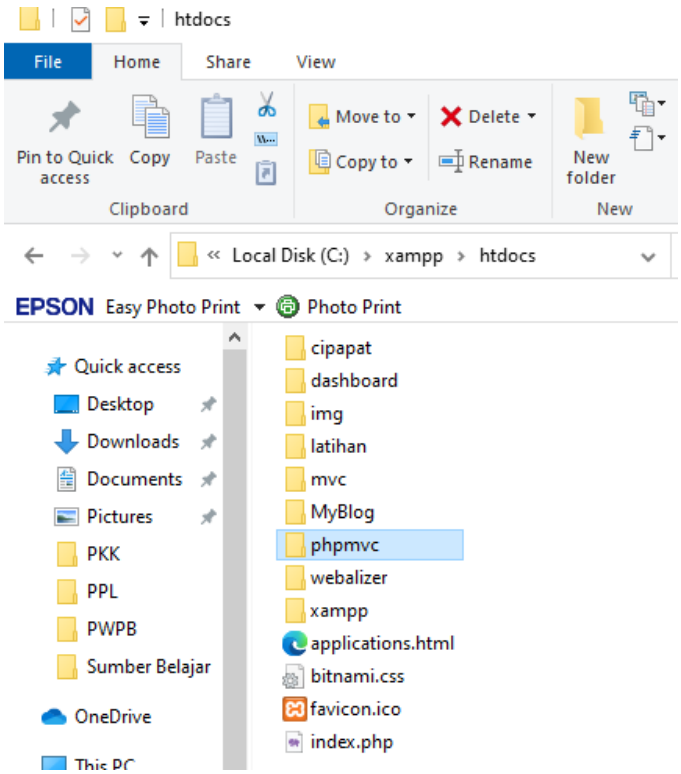
2. PENERAPAN KONSEP MVC DALAM PEMBUATAN *WEB*

a. Persiapan

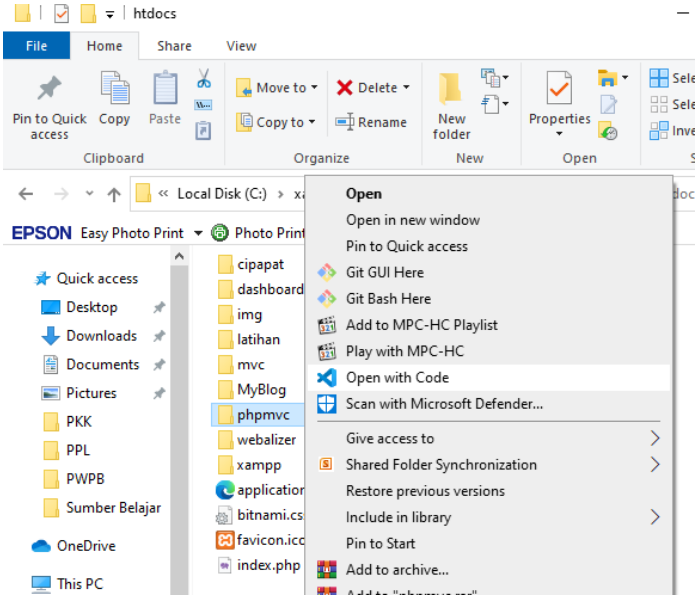
Untuk membuat lingkungan *web* sesuai dengan gaya MVC yaitu

1) Jalankan terlebih dahulu aplikasi **xampp** dan aktifkan Module **Apache (*Web* Server)**.

2) Kemudian **buka *folder* htdocs** yang tedapat di dalam *folder* xampp yang terinstall di laptop. Kemudian buat *folder* baru dengan nama **phpmvc.**



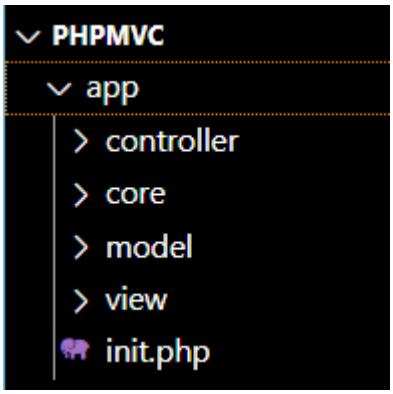
3) Klik kanan pada *folder* phpmvc, dan pilih aplikasi teks editor untuk mengedit semua coding kita nanti, bisa sublime, vs *code*, notepad++ dan sebagainya.



4) **Buat *folder* public**, kemudian juga didalamnya buat beberapa *folder* (**css, img dan js**) dan jangan lupa sisipkan juga *file* **index.php.**

Folder public ini adalah *folder* yang bisa diakses oleh user dimana yang isi di dalamnya adalah segala sesuatu yang user boleh mengetahuinya, jangan sertakan informasi apapun yang sifatnya *privacy* dari *web* kita di *folder* ini.

5) **Buat *folder* app** dan beberapa *folder* di dalamnya (***controller*s, core, models, *view*s)**, juga tambahkan *file* **init.php**



Di *folder* app ini, user tidak akan bisa mengakses karena di *folder* app ini berisi inti dari *web* kita, sebenarnya kalau lebih kompleks di dalam *folder* app ini juga bisa diisi dengan *library* atau *helper*. Keterangan *folder-folder* yang terdapat di dalam *folder* App:

- **Core**, *folder* ini berisi *file* yang menjadi inti dari mvc, bisa dibilang "beranda" dari mvc.
- **Models, *View*, dan *Controller***, berisi *file-file* yang sesuai dengan namanya
- **Init.php**, satu *file* yang dinggunakan untuk teknik *bootstraping* dimana ketika memanggil *file* ini otomatis juga akan memanggil "semua" *file* yang kita butuhkan

6) Masuk ke public→index.php dan app→Init.php. Lalu, tambahkan kode berikut untuk menginisialisasi halaman awal dari *web*.

index.php

```
public > index.php > ...
1   <?php
2
3   require_once '../app/init.php';
4
5   $app = new App;
```

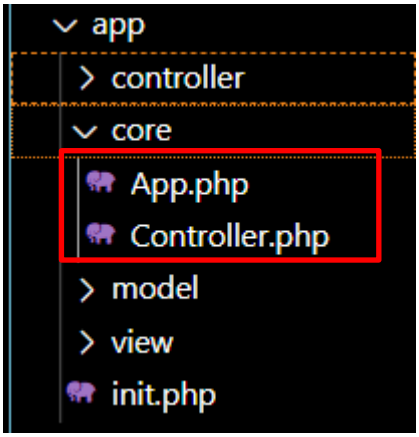
Makna codingan di **index.php** menandakan bahwa kita sedang menginisialisasi kelas App yang mana merupakan *homepage* daripada *website* kita.

init.php

```
app > init.php
1   <?php
2
3   require_once 'core/App.php';
4   require_once 'core/Controller.php';
```

Penulisan huruf kapital pada nama *file* menandakan kalau itu adalah sebuah **Class. Class App dan *Controller* merupakan inti dari mvc pada *web* kita**. Penjelasan sederhananya **App adalah class utama dari aplikasi** kita, dan class *Controller* yang dimaksud ini berbeda dengan *controller* yang akan kita isi pada *folder controller*, **class *Controller* ini nantinya akan menjadi parent class yang akan diextends oleh *controller-controller* yang ada.**

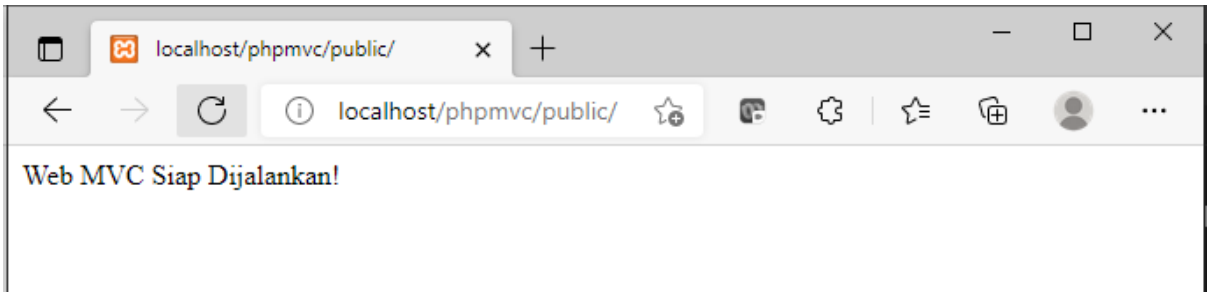
7) Selanjutnya buka ***folder* app → *folder* core** dan tambahkan 2 *file* php yaitu **App.php dan *Controller*.php**.



Beri kode berikut pada **App.php**

```php
app > core > App.php > ...
<?php

class App
{
    public function __construct()
    {
        echo "Web MVC Siap Dijalankan!";
    }
}
```

Jalankan *web* dengan cara ketikkan "localhost/phpmvc/public/" pada *address bar web* browser kita.



b. *Routing*

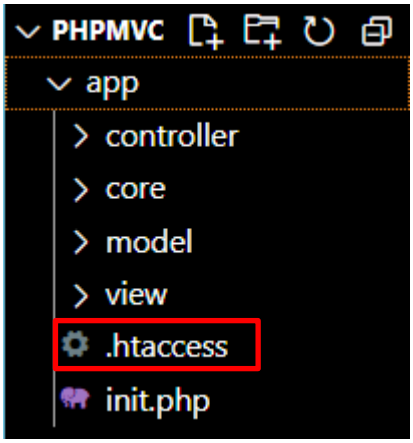
Routing ini merupakan proses "diantara" *user* dengan *controller*. Ketika user melakukan sebuah aksi pada aplikasi kita maka aplikasi akan "merutekan, menunjukkan, mengarahkan" ke *controller* yang sesuai dengan yang diminta ke *user*.

.htaccess

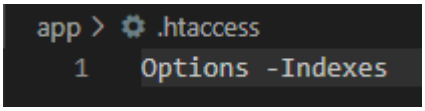
File **.httaccess** ini memungkinkan kita untuk mengkonfigurasi server apache kita dalam mengelola akses user terhadap *website* kita. Namun, pada materi ini kita hanya akan fokus pada penerapan .htaccess dalam url *web* kita.

.htaccess di *folder* app

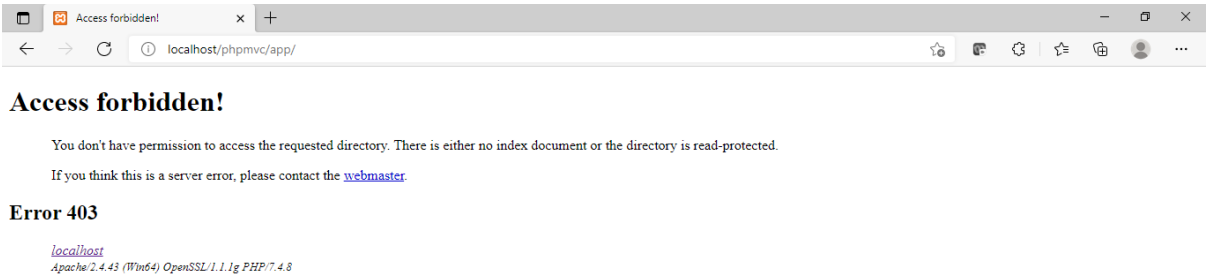
1) Buat *file* **.htaccess** di *folder* app lewat *text editor*.



2) Masukkan kode berikut di dalamnya.



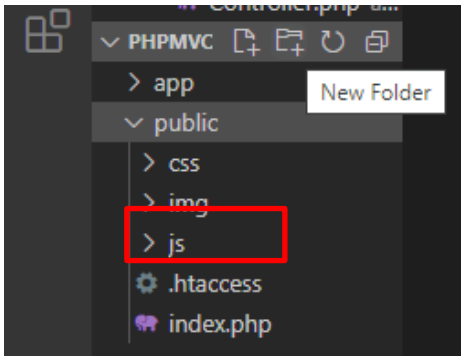
Maksud dari kode tersebut adalah, selama di dalam *folder* itu tidak ada *file* index, baik itu index.php maupun index.html ***jangan tampilkan isi foldernya.*** Jika dijalan dan kita link ke *folder* app maka yang muncul akan seperti berikut :



.htaccess di *folder* public

.htaccess di *folder* public berfungsi sebagai manajemen dari url kita yang akan kita buat "cantik" seperti penjabaran materi di lembar sebelumnya.

1) Buat *file* **.htaccess** di *folder* public lewat *text editor*.



2) Masukkan kode berikut di dalamnya.

public > .htaccess

```
1   Options -Multiviews
2   RewriteEngine On
3   RewriteCond %{REQUEST_FILENAME} !-d
4   RewriteCond %{REQUEST_FILENAME} !-f
5   RewriteRule ^(.*)$ index.php?url=$1 [L]
```

Penjelasan:

Options –Multi*views*: Kode ini untuk menghindari ambigu ketika kita memanggil *file/folder* pada aplikasi kita, karena bisa saja nama *folder* dan *file* yang kita miliki sama dengan nama *controller* yang akan kita buat

RewriteEngine On: Untuk mulai menjalankan proses Rewrite-nya

RewriteCond %{REQUEST_*FILE*NAME} !-d

RewriteCond %{REQUEST_*FILE*NAME} !-f

- d untuk directory dan f untuk file, kode ini mengartikan bahwa untuk mengabaikan apabila ada nama file atau folder yang sama dengan controller dan *method* kita nantinya

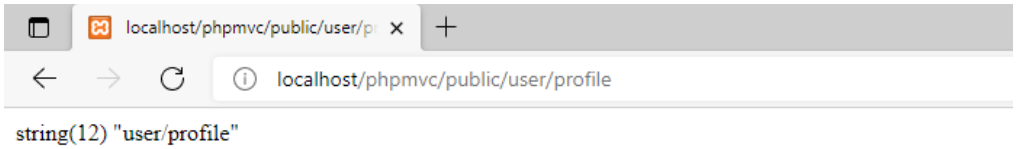
**RewriteRule ^(.*)$ index.php?url=$1 [L]** : Kode ^(.*)$ itu berarti ambil semua string setelah public/ lalu simpan isinya kedalam $1 dan untuk [L] itu berarti apabila ada rule yang sudah terjalankan maka jangan jalankan rule lain, hal ini yang membuat kita mengantisipasi dari orang-orang yang akan melalukan sesuatu yang jahat pada *web* kita

Untuk uji coba **.httaccess** jalan atau tidak, silakan edit ***method* construct** pada App.php lalu kita **var_dump url-nya.**

app > core > App.php > App

```
4   {
5       public function __construct()
6       {
7           $url = $this->parseURL();
8           var_dump($url);
9       }
10
11      public function parseURL()
12      {
13          if (isset($_GET['url'])) {
14              $url = $_GET['url'];
15              return $url;
16          }
17      }
18  }
```

Untuk mengecek hasil, ketikkan "localhost/phpmvc/public/user/pro*file*" pada address bar *web* browser kita.



Setelah itu, pada App.php tambahkan function baru dengan nama parseURL(), lalu isi dengan kode seperti berikut:

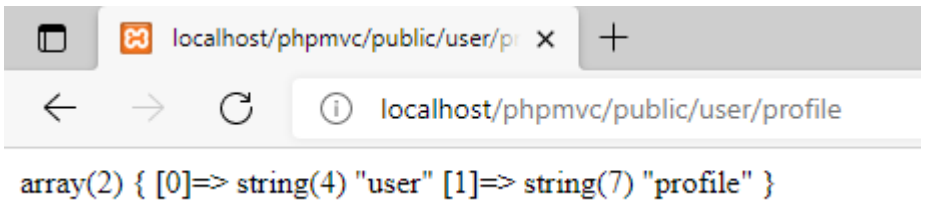
```php
app > core > App.php > ...
<?php

class App
{
    public function __construct()
    {
        $url = $this->parseURL();
        var_dump($url);
    }

    public function parseURL()
    {
        if (isset($_GET['url'])) {
            $url = rtrim($_GET['url'], "/");
            $url = filter_var($url, FILTER_SANITIZE_URL);
            $url = explode('/', $url);
            return $url;
        }
    }
}
```

Jadi ketika ada url yang kita kirimkan maka pertama, kita hilangkan dulu slash paling akhir dari sebuah url misal:

localhost/phpmvc/home/page/1/2/

- Untuk menghilangkan (/) itu kita menggunakan *method* rtrim().
- Setelah itu **filter_var() →** digunakan untuk memfilter url dari karakter — karakter asing yang memungkinkan *website* kita dihack.
- Dan **explode() →** untuk memecah string yang kita punya menjadi bagian-bagian dalam bentuk array, yang memecah menjadi *controller*, *method*, dan data

Untuk mengecek hasil, ketikkan "localhost/phpmvc/public/user/profile" pada address bar *web* browser kita.



c. *Controller*

Controller ini tidak berbeda jauh dengan bahasan *routing* pada bab sebelumnya. Dalam *Routing* kita belajar untuk mengambil data dari url dan mengubahnya ke dalam

bentuk array, selanjutnya kita akan kirimkan data-data tersebut untuk dijalankan sesuai fungsinya masing- masing.

```
(array) [4 elements]
    0: (string) "user"
    1: (string) "profile"
    2: (string) "karindra"
    3: (string) "pelajar"
```

Dari data di atas maka url nya sebagai berikut:

localhost/phpmvc/public/user/pro*file*/karindra/pelajar

Keterangan :

- User = *controller*
- Pro*file* = *method*
- Karindra, pelajar = data1, data2

Dari penjabaran tersebut, kita akan membuat ***controller default*** yang dapat dipanggil secara "manual" atau "otomatis" ketika *controller* yang dituliskan pada url tidak terdaftar pada aplikasi itu. Juga kita akan buat ***method default*** di setiap *controller* sebagai fungsi *default* ketika *controller* itu tidak memanggil *method* lain atau *method* yang dituliskan pada url juga tidak terdaftar pada aplikasi kita.

Langkah selanjutnya yaitu kita akan mengidentifikasi satu-persatu isi dari elemennya hasil parsing url, langkahnya sebagai berikut:

Core → App.php

1) Tambahkan property

Sebelum kita mulai mengidentifikasi tiap bagian dari url kita, perlu kita buat terlebih dahulu variabel untuk menyimpan tiap - tiap nilainya.

app > core > App.php > App > __construct

```php
<?php
class App
{
    protected $controller = "Home";
    protected $method = "index";
    protected $params = [];
```

Pada tiap - tiap variabel kita beri nilai *default* untuk mengantisipasi bila di url yang diberikan tidak ditemukan *controller* atau *method* tersebut.

2) Setup *Controller*

```
7     public function __construct()
8     {
9         $url = $this->parseURL();
10        //setup controller
11        if(file_exists('../app/controller/'. $url
          [0].'.php')) {
12            $this->controller = $url[0];
13            unset($url[0]);
14        }
15        require_once '../app/controller/'.
          $this->controller.'.php';
16            $this->controller = new
              $this->controller;
```

Di kode tersebut kita cek dulu apakah *controller* yang diminta user tersedia atau tidak (*file*nya) pada aplikasi kita dengan *method* ***file*_exists()**, kalau ada maka simpan nilainya pada *variable*. Lalu gunakan **unset()** untuk menghapus elemen *controller* pada array. Terlihat disini bahwa elemen *controller* terletak pada index ke-0 akan tetapi bila elemen ini dihapus dengan **unset()** maka elemen lain tidak ikut berubah indexnya (tetap). Setelah itu kita inisialisasi *class* dari *controller* tersebut.

3) Setup *Method*.

```
18        //setup method
19        if(isset($url[1]))
20        {
21            if(method_exists($this->controller,
              $url[1])) {
22            $this->method= $url[1];
23            unset($url[1]);
24            }
25        }
```

***method*_exists()** kita gunakan untuk mengecek apakah aplikasi itu memlilki *method* sesuai dengan data yang kita dapat dari url, jika ada set nilainya dan juga hapus elemennya dengan menggunakan **unset().**

4) Setup Data/Params.

Alasan dibalik kenapa kita harus menghapus elemen-elemen *controller* dan *method* adalah untuk mengecek apakah di url yang ditulis oleh user itu mengandung

data atau tidak karena bisa jadi user hanya mengirimkan *controller* dan *method* saja (Alternatif cara untuk mengecek data). Jika misal ternyata url mengandung data, maka data tersebut dimasukkan ke variabel params dengan menggunakan *method* **array_values()**.

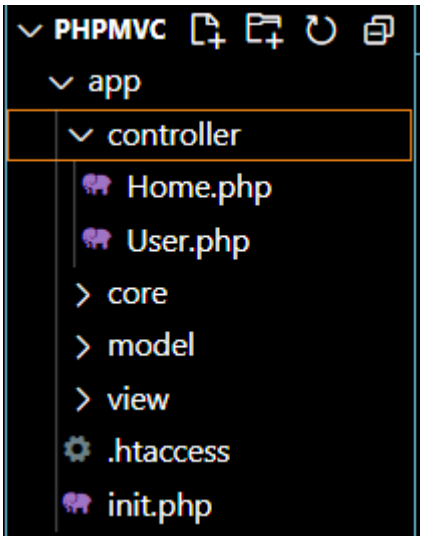
```
27          //setup Params/Data
28          if(!empty($url)) {
29              $this->params = array_values($url);
30          }
```

5) Menjalankan *Controller*, *Method* dan mengirim data

Untuk menjalankan *controller* kita akan menggunakan fungsi dari **call_user_func_array()** kemudian kita masukkan parameter berupa variabel-variabel yang sebelumnya sudah melewati proses validasi.

```
32          //Jalankan Controller dan Method, serta
            kirimkan data
33          call_user_func_array(
34              [$this->controller, $this->method],
                $this->params);
35      }
```

6) Membuat *file* Home.php dan User.php di*folder controller*.



Setelah membuat *file* Home.php maka kita akan isikan script dibawah ini.

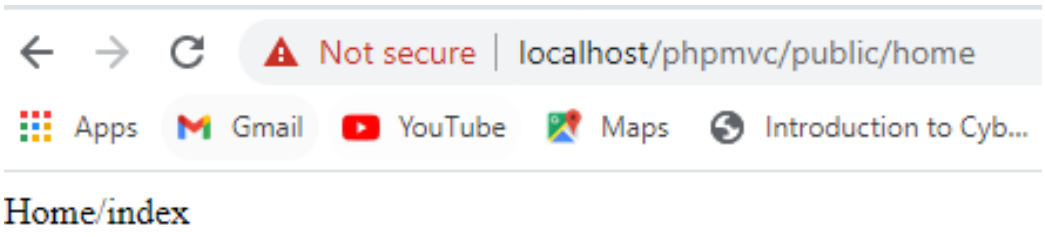
Home.php

```
app > controller > Home.php > ...
1   <?php
2   class Home {
3       public function index() {
4           echo "Home/index";
5       }
6   }
```

index() dalam echo menandakan semua *method default* yang akan kita pasang di setiap *controller*. sekarang kita cek dengan jalankan *code*nya terlebih dahulu, masukkan url seperti berikut :

localhost/phpmvc/public/home/

Jika berhasil, seharusnya halaman sudah diteruskan menuju *controller* home dan walaupun kita tidak menuliskan *method*nya, secara otomatis program akan mengarahkan ke *method default* yaitu **index()**.



Selanjutnya mengisi *file* User.php dengan script seperti dibawah ini.

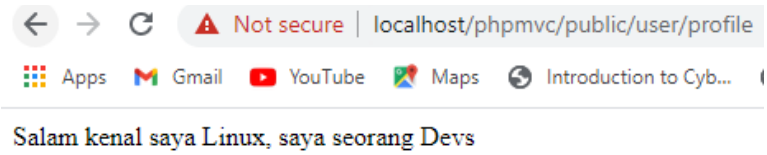
User.php

```
app > controller > User.php > ...
1   <?php
2   class User {
3       public function index() {
4           echo "User/index";
5       }
6       public function profile($nama = "Linux",
        $pekerjaan = "Devs") {
7           echo "Salam kenal saya $nama, saya seorang
            $pekerjaan";
8       }
9   }
```

Dalam *controller* **User** ini kita memiliki 2 *method*, yaitu **index** sebagai *method default* dan **pro*file*** sebagai *method* optional. Nah, pada *method* **pro*file*()** kita menambahkan parameter *$nama* dan *$pekerjaan* yang akan diisi oleh data params yang sudah kita kirim bersamaan dengan menjalankan *controller* ini. Jalankan *website*nya dengan mengetikkan url berikut ini:

localhost/phpmvc/public/user/pro*file*

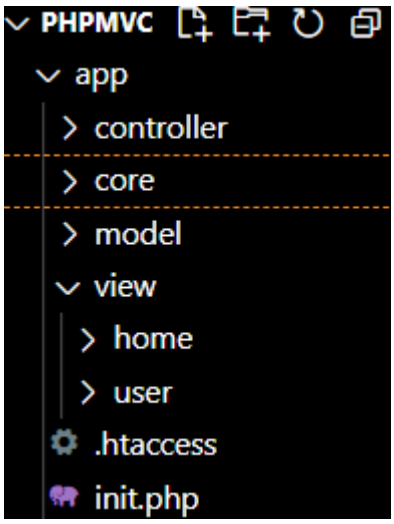
Jika berhasil, maka seharusnya tampilannya akan seperti ini:



d. *View*

Setelah membuat *controller*, kita akan membuat *view*. Sekarang kita akan melakukan perubahan — perubahan kecil pada output kita yang selama ini hanya sebatas "echo" saja sekarang kita akan menyambungkannya dengan ***View*** yang berlandaskan html.

1) Membuat *folder* home dan user pada *Folder View*



2) Membuat ***file* index.php pada *folder* home** dan ***file* index.php, user.php serta pro*file*.php pada *folder* user.** *File-file* tersebut yang nantinya akan berisi laman html yang bertindak sebagai *view* dari *website* kita.
3) Edit **Controller.php** pada **folder app/core**

app > core > Controller.php > ...

```php
<?php
class Controller {
    public function view($view, $data = []) {
        require_once "../app/view/" . $view . '.php';
    }
}
```

method view() ini yang akan mengkoneksikan *controller* dengan *view*. **method view()** memiliki 2 parameter yaitu **view** dan **data** yang akan dikirim ke *view* (jika ada). Class *Controller* ini akan dijadikan sebagai **parent** dari *controller-controller* yang ada.

4) **Edit Home.php** yang terdapat dalam **folder app/controller**

app > controller > Home.php > ...

```php
<?php
class Home extends Controller {
    public function index() {
        $data['judul'] = "Home";
        $this->view('home/index');
    }
}
```

sekarang kita ganti isi dari *method* index menjadi pemanggilan *method view* yang akan diisi oleh parameter alamat dari *view* itu, yang berada pada *folder* **home** dengan nama *file* **index**. Alasan kenapa kita bisa memanggil *folder view* pada class ini adalah karena kita sudah meng-extend kan nya dengan class *Controller* yang mengandung *method view*(), secara otomatis class Home juga ikut mewarisi *method view*() tersebut.

5) Edit **User.php** yang terdapat dalam **folder app/controller**

app > controller > User.php > ...

```php
<?php
class User extends Controller {
    public function index() {
        $data['judul'] = "User";
        $this->view("user/index");
    }

    public function profile($nama = "Linux", $pekerjaan = "Devs") {
        $data['judul'] = "User";
        $data['nama'] = $nama;
        $data['pekerjaan'] = $pekerjaan;
        $this->view("user/profile", $data);
    }
}
```

Pada *Controller* User yang sudah kita extend dengan class *Controller method* index() dan pro*file*() dalam tiap-tiapnya akan kita koneksikan dengan *view*nya . Khusus pada *method* pro*file*() karena *method* ini mempunyai params/data yang dikirimkan maka pada pemanggilan *method view*()-nya juga kita kirimkan data sesuai yang tertulis di url

6) Membuat halaman masing – masing *method*

app/view/Home/index.php

app > view > home > index.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>Halaman <?= $data['judul']; ?></title>
  </head>
  <body>
    <h1>Selamat Datang!</h1>
  </body>
</html>
```

app/view/User/index.php

app > view > user > index.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>Halaman <?= $data['judul']; ?></title>
    </head>

    <body>
      <h1>Halaman User</h1>
    </body>

</html>
```

app/view User/profile.php

app > view > user > profile.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>Halaman <?= $data['judul']; ?></title>
  </head>

  <body>
    <h1>Halaman Profile</h1>
    <p>Halo, nama saya <?= $data['nama']; ?>, saya seorang <?= $data['pekerjaan'];?></p>
  </body>

</html>
```

Pada dasarnya *controller* yang sudah kita buat itu akan mengirimkan data ke *file* **.php** *view* kita, itulah yang menjadi alasan kenapa kita bisa memang-gil **$data** padahal variable itu tidak terdeklarasi atau terdefinisi di *file view* kita. Dan kita implementasikan **$data** tersebut sesuai fungsinya masing-masing.

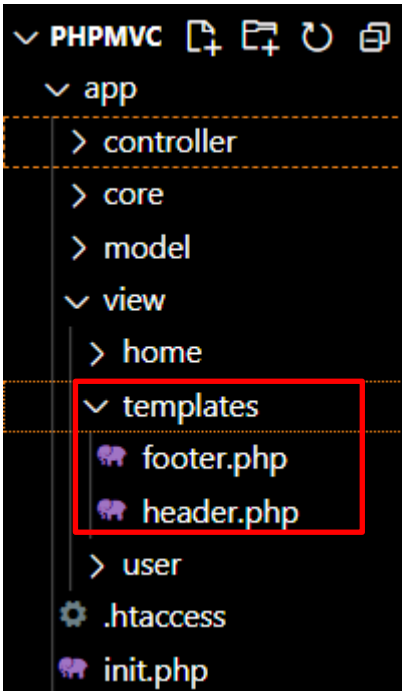
Catatan:

Halaman user.php tidak perlu di isi karena jika dipanggil dengan url http://localhost/phpmvc/public/user/user maka otomatis akan menampilakan **index.php** (sesuai dengan rute atau alur pada controller User di **method atau function index**)

Mempersimple Pengolalaan *View* dengan Template

Semua *view* tersebut dibangun atas asas html, dimana dalam setiap html, kita bisa bagi menjadi 3 bagian yaitu **header, content, footer.** Kita akan buat template dari masing-masing bagian itu, yang dengan template itu dapat mempermudah kita mambuat view baru lagi nantinya supaya kita tidak mengulangi menulis ulang kode yang sama. Langkah-langkahnya sebagai berikut:

1. **Membuat *folder* templates di folder app/view dan buat *file* header.php serta footer.php**



2. Edit header.php dan footer.php dengan cara mengcopy bagian header dari **home/index.php, user/index.php dan user/user.php** seperti berikut ini.

header.php

app > view > templates > header.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-  scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Halaman <?= $data['judul']; ?></title>
8    </head>
9    <body>
```

footer.php

app > view > templates > footer.php

```
1  </body>
2  </html>
```

Sehingga isi dari file-file **home/index.php, user/index.php dan user/user.php** menjadi

home/index.php

app > view > home > index.php

```
1  <h1>Selamat Datang!</h1>
```

user/index.php

app > view > user > index.php

```
1  <h1>Halaman User</h1>
```

user/profile.php

app > view > user > profile.php

```
1  <h1>Halaman Profile</h1>
2  <p>Halo, nama saya <?= $data['nama']; ?>, saya seorang <?= $data['pekerjaan'];?></p>
```

3. Selanjutnya menyambungkan header dan footer dengan fil-file di folder *view* dengan controller.

controller/Home.php

app > controller > Home.php > ...

```php
<?php
class Home extends Controller {
    public function index() {
        $data['judul'] = "Home";
        $this->view('templates/header', $data);
        $this->view('home/index');
        $this->view('templates/footer');
    }
}
```

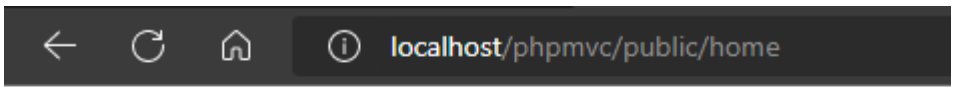
controller/User.php

app > controller > User.php > ...

```php
<?php
class User extends Controller {
    public function index() {
        $data['judul'] = "User";
        $this->view("templates/header", $data);
        $this->view("user/index");
        $this->view("templates/footer");
    }
    public function profile($nama = "Linux", $pekerjaan = "Devs") {
        $data['judul'] = "User";
        $data['nama'] = $nama;
        $data['pekerjaan'] = $pekerjaan;
        $this->view('templates/header', $data);
        $this->view('user/profile', $data);
        $this->view('templates/footer');
    }
}
```

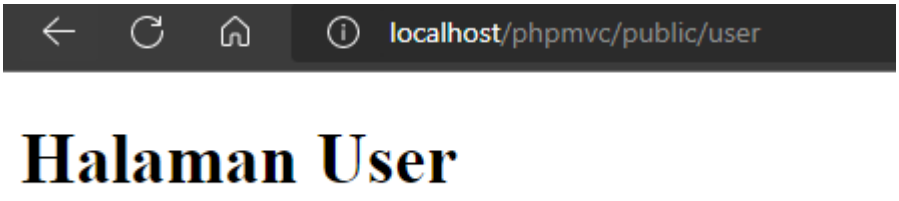
sekarang coba jalankan dulu *web*nya dan cek hasilnya, pastikan bila *view* dan data nya sudah tampil dengan benar.

Halaman home



Selamat Datang!

Halaman user

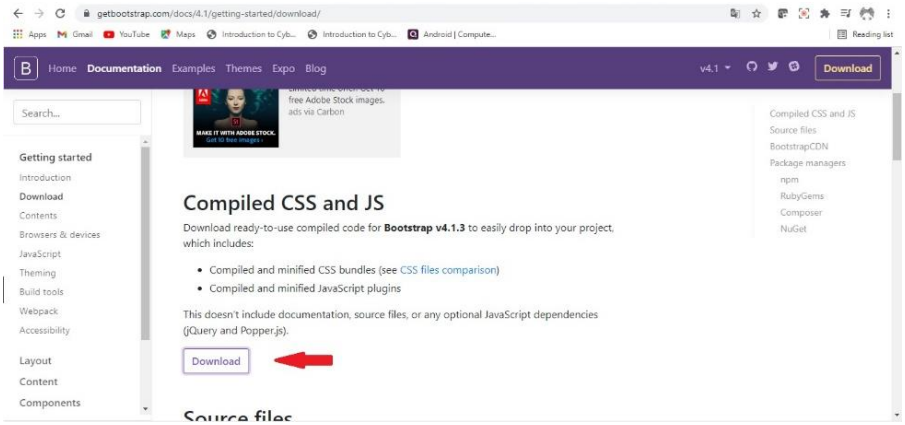


Halaman profile

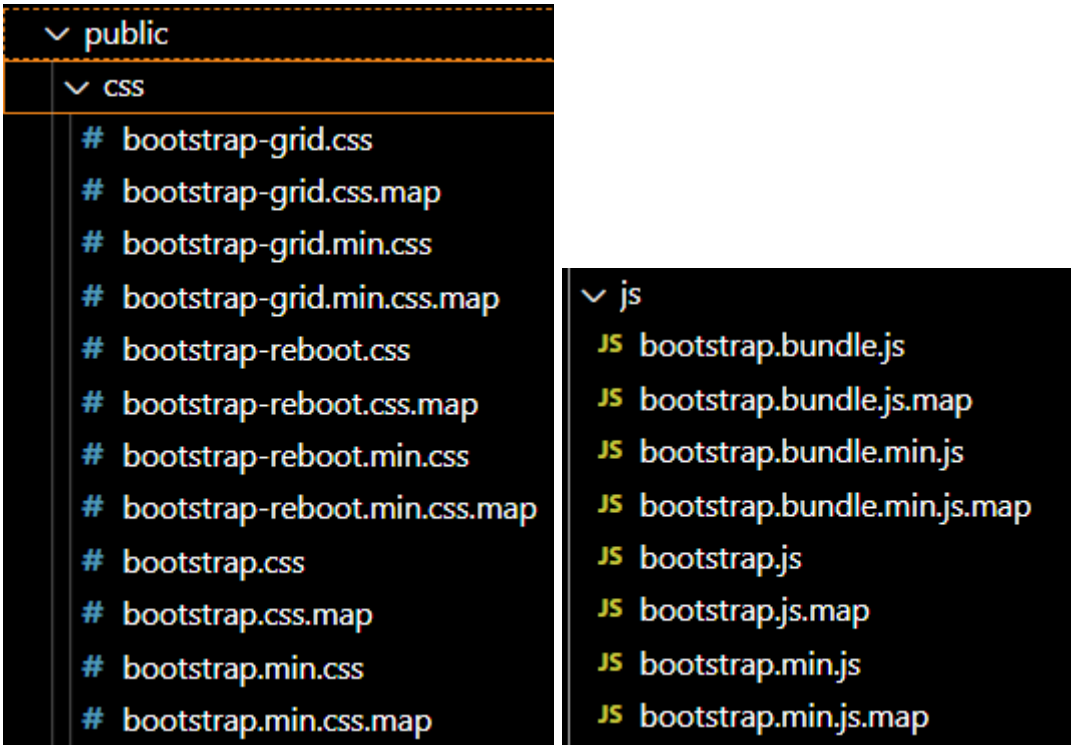
e. Asset

Setelah membuat *view*, maka selanjutnya akan membahas tentang asset pada *view*, berupa css, js dan lainnya. Pada modul ini css yang digunakan yaitu boostrap.

1) Download compiled CSS dan JS-nya.

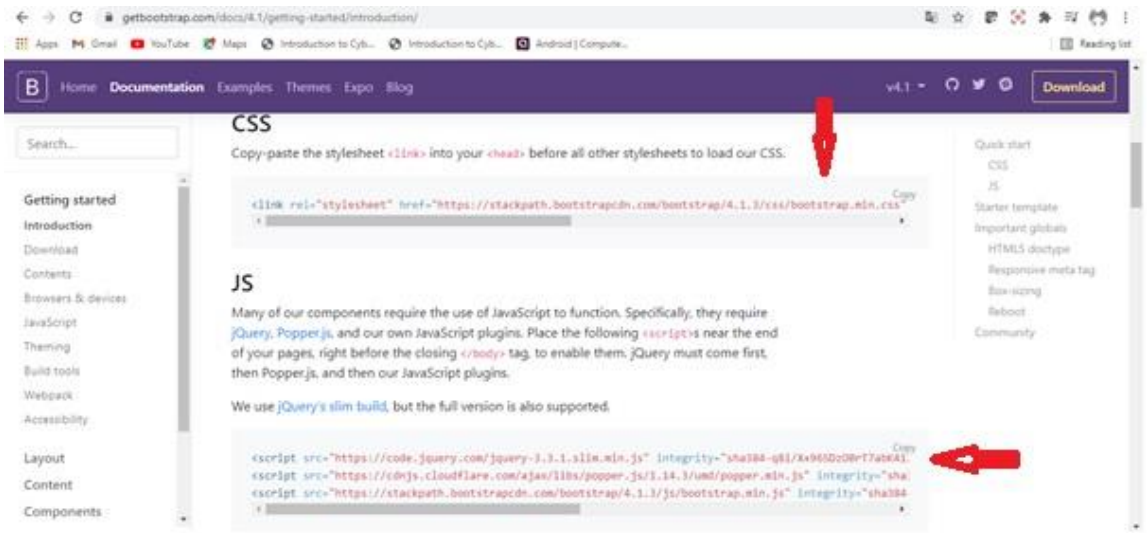


Buka *website* boostrap dan download assetnya. Link boostrap https://getbootstrap.com/docs/4.1/getting-started/download/. Setelah download extrak *folder* boostrapnya dan copy CSS serta JSnya pada *folder* public pada project. Setelah pada *folder* public tampilannya seperti dibawah.



2) Copy CDN lalu ganti link href menjadi server local

Buka dokumentasi dari bootstrap, kemudian Copy paste terlebih dahulu link untuk dimasukkan kehalaman web kita.



Untuk mengkoneksikan dengan css yang telah didownload dapat menggunakan absolute url seperti berikut :

http:localhost/phpmvc/public/css/bootstrap.css

Kemudian copy CDN kedalam **header.php** dan **footer.php**, kodenya akan menjadi seperti berikut:

header.php

app > view > templates > header.php

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport"
        content="width=device-width, initial-
        scale=1.0">
6       <meta http-equiv="X-UA-Compatible"
        content="ie=edge">
7       <title>Halaman <?= $data['judul']; ?></title>
8       <link rel="stylesheet"    href="http://
        localhost/phpmvc/public/css/bootstrap.css">
9     </head>
10    <body>
```

footer.php

app > view > templates > footer.php

```
1   <script src="https://code.jquery.com/jquery-3.3.1.
    slim.min.js" integrity="sha384-q8i/X
    +965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH
    +8abtTE1Pi6jizo" crossorigin="anonymous"></script>
2   <script src="https://cdnjs.cloudflare.com/ajax/
    libs/popper.js/1.14.7/umd/popper.min.js"
    integrity="ha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1
    clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
    crossorigin="anonymous"></script>
3   <script src="http:localhost:8080/phpmvc/public/js/
    bootstrap.js"></script>
4   </body>
5   </html>
```

Untuk bagian jquery dan popper dibiarkan online dan pastikan terkoneksi dengan internet agar bisa digunakan.

3) Menggunakan URL Constans

Jika ingin menggunakan asset yang sudah ada dapat menggunakan variable constant yang bisa digunakan berulang Ketika ingin mengakses *file* asset yang telah dibuat.

localhost/phpmvc/public

Link tersebut akan dijadikan sebagai constants url kemudian tinggal dipanggil setiap akan mengakses *file* asset. Hal itu bisa mempermudah jika terjadi perubahan *folder* pada url *web* maka tidak perlu mengubah satu persatu *file* yang mengakses url tersebut,namun tinggal mengganti di variable contanst nya saja.

Buat Contants.php di*folder* core

app > core > Constants.php > ...

```php
<?php
define('BASE_URL', "http://localhost/phpmvc/public");
?>
```

Require Constants.php di init.php

app > init.php

```php
<?php
require_once 'core/App.php';
require_once 'core/Controller.php';
require_once 'core/Constants.php';
```

Setelah itu mengkoneksikan header dan footer dengan BASE_URL yang sudah dibuat.

Header.php

app > view > templates > header.php

```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>Halaman <?= $data['judul']; ?></title>
    <link rel="stylesheet"    href="<?= BASE_URL; ?>/css/bootstrap.css">
  </head>
  <body>
   <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <div class="container">
        <a class="navbar-brand" href="<?= BASE_URL; ?>">MVC PHP</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
        aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
          <div class="navbar-nav">
            <a class="nav-item nav-link active" href="<?= BASE_URL; ?>">Home <span class="sr-only">(current)</span></a>
            <a class="nav-item nav-link" href="<?= BASE_URL; ?>/blog">Blog</a>
            <a class="nav-item nav-link" href="<?= BASE_URL; ?>/user/profile">User</a>
          </div>
        </div>
      </div>
    </nav>
```

Footer.php

app > view > templates > footer.php

```php
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X
  +965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
  integrity="ha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
  <script src="<?= BASEURL; ?>/js/bootstrap.js"></script>

  </body>
  </html>
```

4) Mempercantik halaman *Website*

Untuk sedikit mempercantik halaman *website*, maka harus memodifikasinya dengan beberapa component dari boostrap.

Edit *views*/home/index.php

app > view > home > index.php

```
<div class="container">
<div class="jumbotron mt-4">
<h1 class="display-4">Halo, Selamat Datang !</h1>
<p class="lead">This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.</p>
<hr class="my-4">
<p>It uses utility classes for typography and spacing to space content out within the larger container.</p>
<a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
</div>
</div>
```

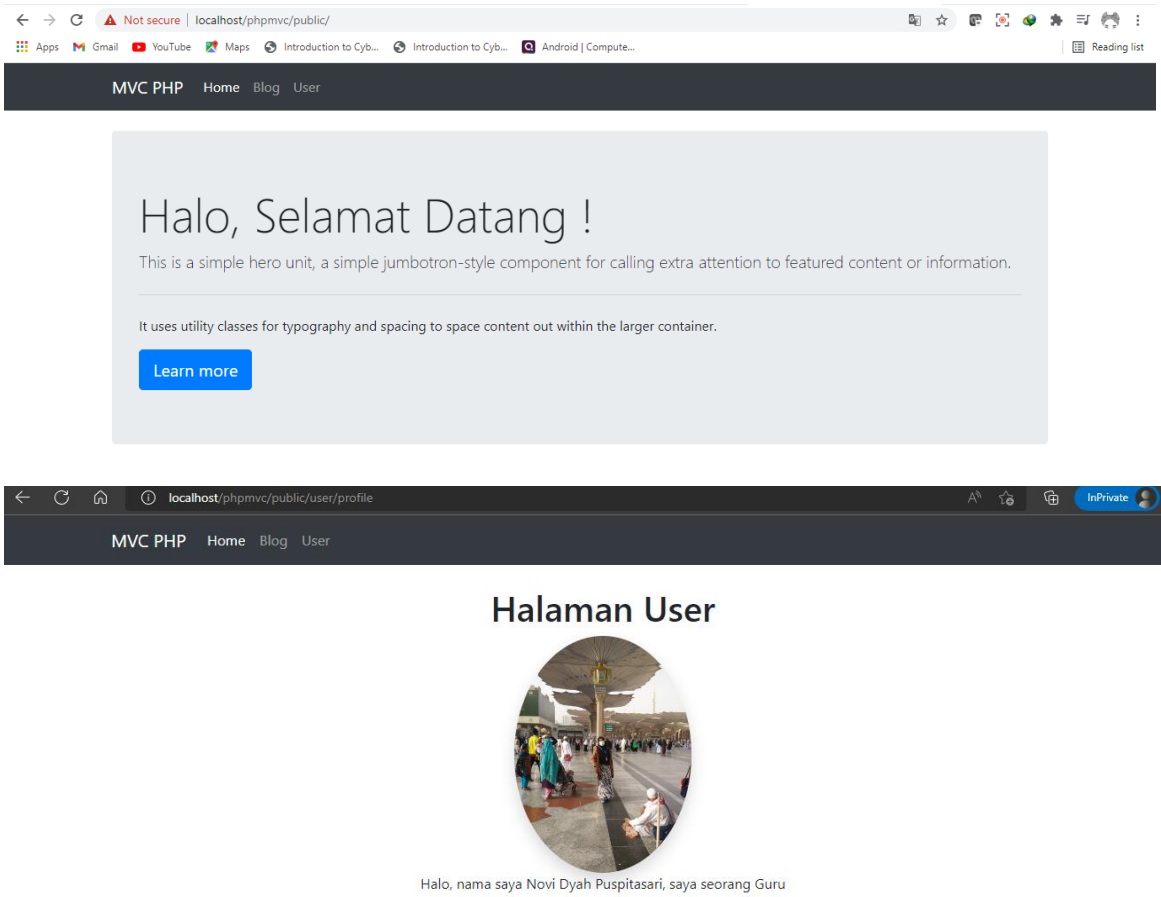
Edit *views*/user/pro*file*.php

Tambahkan gambar kalian terlebi dahulu di folder public/img, dan ubahlah data nama dan pekerjaan pada **app/controller/User.php** dengan data kalian.

app > view > user > profile.php

```
<div class="container text-center mt-4">
<h1>Halaman User</h1>
<img src="<?= BASE_URL; ?>/img/profile.jpg" alt="Novi Foto" width="200" class="rounded-circle shadow">
<p>Halo, nama saya <?= $data['nama']; ?>, saya seorang <?= $data['pekerjaan'];?></p>
</div>
```

Hasil:



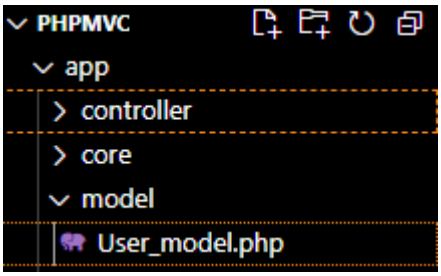
f. Model

Sesuai konsep dari MVC model merupakan representasi data dan yang mengelola bagian data, baik itu data lokal maupun data yang diambil dari database. Maka selanjutnya kita akan mencoba membuat model dengan data yang berasal dari lokal dan data yang berasal dari database.

Model dengan data lokal

Untuk percobaan model dengan data lokal ini kita akan mengoprek halaman home/index, disini kita akan mengubah bagian This is simple blala… itu menjadi nama kita yang kita deklarasikan dalam sebuah properti di model kita.

1. **Buat model User_model.php di*folder* model**



app > model > User_model.php > ...

```php
<?php
class User_model {
    private $nama = "Guru RPL";
    public function getUser()
    {
        return $this->nama;
    }
}
```

$nama merupakan data dummy (untuk keperluan simple model) yang akan kita kirim ketika controller memanggil method getUser().

2. **Membuat *method* model () di app/core/*Controller*.php**

```php
<?php
class Controller {
    public function view($view, $data = []) {
        require_once "../app/view/" . $view . '.php';
    }

    public function model($model) {
        require_once "../app/models/".$model.".php";
        return new $model;
        }

}
```

Mirip dengan *method view*() yang memilki fungsi untuk merequire sebuah *file*, *method* model() pun demikian, hanya saja pada *method* ini yang kita require bukan *file* **.php** yang berisi html melainkan sebuah class yang berisi data, oleh karena itu kita harus menginstance nya saat mereturn-nya

3. Ambil data Model di *Controller* Home

Pada *controller* Home tambahkan kode berikut :

$data['nama'] = $this->model('User_model')->getUser();

jadi kita menggunakan variabel data untuk menyimpan nilai dari data yang kita minta dari model, *method* model() akan meminta parameter nama dari model yang kita butuhkan lalu juga langsung memanggil *method* di dalamnya untuk mengembalikan data yang kita minta. Setelah itu kita kirimkan juga $data ke dalam *view* index-nya

Jadi keseluruhan kode pada *controller* **Home.php** menjadi seperti berikut :

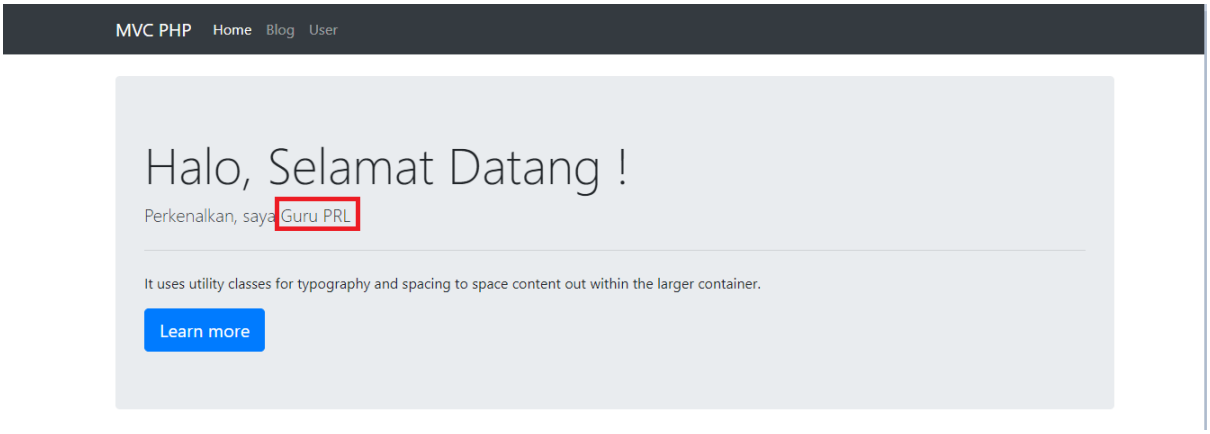
app > controller > Home.php > ...

```php
<?php
class Home extends Controller
{
    public function index()
    {
        $data['judul'] = "Home";
        $data['nama'] = $this->model('User_model')->getUser();
        $this->view('templates/header', $data);
        $this->view('home/index');
        $this->view('templates/footer');
    }
}
```

4. Edit *view* pada home/index.php

```php
<div class="container">
    <div class="jumbotron mt-4">
        <h1 class="display-4">Halo, Selamat Datang !</h1>
        <p class="lead">Perkenalkan, saya <?= $data['nama']; ?></p>
        <hr class="my-4">
        <p>It uses utility classes for typography and spacing to space content
            out within the larger container.</p>
        <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
    </div>
</div>
```

Setelah itu kita edit *view*-nya untuk mengoutputkan data yang sudah kirim dari *controller*. Sehingga bila benar maka tampilan yang muncul akan seperti ini :



Selanjutnya, masih dalam Model dengan data lokal, kita akan mecoba membuat sebuah *controller* baru yang kita beri nama **Blog**, nah kedepannya kita akan membuat *website* blog sederhana. Langkah-langkahnya sebagai berikut:

1. **Buat *Controller* Blog.php di*folder controllers***

app > controller > Blog.php > ...

```php
<?php
class Blog extends Controller
{
    public function index()
    {
        $data['judul'] = "Blog";
        $data['blog'] = $this->model("Blog_model")->getAllBlog();
        $this->view('templates/header', $data);
        $this->view('blog/index', $data);
        $this->view('templates/footer');
    }
}
```

Cara yang kita gunakan tidak jauh beda dengan pengambilan data dari model sebelumnya, bisa dilihat kita akan mengambil data dari class Blog_model dengan *method* **getAllBlog()z**

2. **Buat Model Blog_model.php di *folder* models**

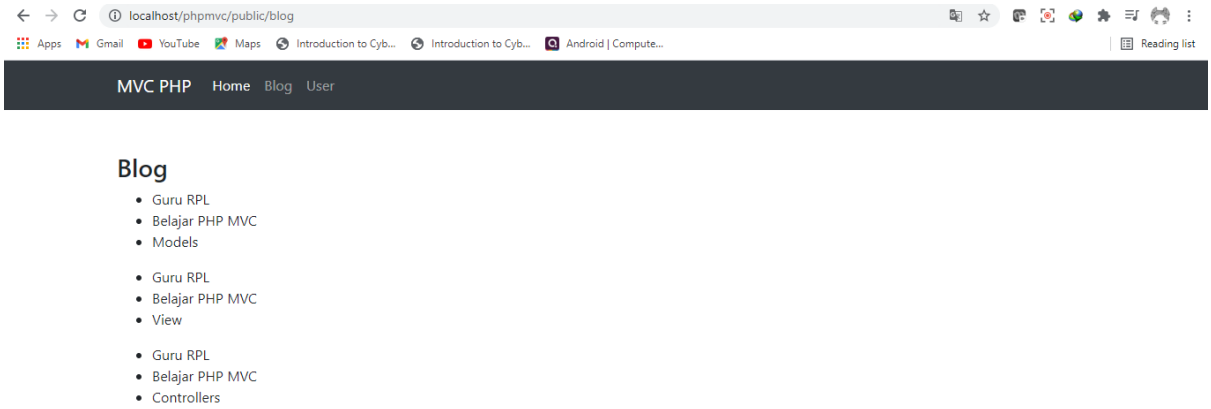
app > models > Blog_model.php > PHP Intelephense > Blog_model > $blog

```php
<?php
class Blog_model
{
    private $blog = [
        [
            "penulis" => "Guru RPL",
            "judul" => "Belajar PHP MVC",
            "tulisan" => "Models"
        ],
        [
            "penulis" => "Guru RPL",
            "judul" => "Belajar PHP MVC",
            "tulisan" => "View"
        ],
        [
            "penulis" => "Guru RPL",
            "judul" => "Belajar PHP MVC",
            "tulisan" => "Controllers"
        ]
    ];
    public function getAllBlog()
    {
        return $this->blog;
    }
}
```

3. **Buat *view* baru blog/index.php di *folder views***

```php
<body>
    <div class="container mt-5">
        <div class="row">
            <div class="col-6">
                <h3>Blog</h3>
                <?php foreach ($data['blog'] as $blog) : ?>
                    <ul>
                        <li><?= $blog['penulis']; ?></li>
                        <li><?= $blog['judul']; ?></li>
                        <li><?= $blog['tulisan']; ?></li>
                    </ul>
                <?php endforeach; ?>
            </div>
        </div>
    </div>
</body>
```

Tampilan Blog yang telah dibuat



E. TUGAS

1. Silakan praktikan untuk membuat halaman *web* bergaya MVC tanpa harus menggunakan framework seperti langkah-langkah di materi **2. Penerapan Konsep MVC Dalam Pembuatan *Web*.**
2. Ubahlah nama MVC PHP menjadi nama *website* kita (Wajib mengandung nama kita), Misal: Novi *Web*. Selanjutnya ubah isi menu menjadi Home, Data Guru, Data Siswa, dan About.

☺☺☺☺☺ SELAMAT BELAJAR ☺☺☺☺☺