**Alur Ngoding Codeigniter 4**

1. **Persiapan VS code**
   1. Install extension Intelephense
      1. Untuk mengaktifkannya, matikan dulu pengaturan bawaan :
         1. Ketik di pencarian extension “@builtin php”.
         2. Pilih PHP Language feature lalu Disable- kan.
   2. Install prettier > setelah install, restart vs code
   3. Masuk ke menu setting, ketik format. Lalu cheklist format on save
   4. Membuat snippet.
      1. Pilih menu file > preferences > snippets
      2. Pilih New Snippets > tulis nama snipet ‘html.json’
      3. Hapus isinya, lalu diganti dengan :

{

“PHP Tag”: {

“prefix”: “php”,

“body”: “<?php $1 ?>”

},

“Inline Echo”: {

“prefix”: “phpp”,

“body” : “<?= $$1; ?>”

}

}

1. **Install Codeigniter 4**
   1. Masuk ke folder htdocs
   2. Install dari composer menggunakan git bash
   3. Perintah untuk menginstall

composer create-project codeigniter4/appstarter *nama folder* --no-dev

* 1. Bisa jalankan built in server dengan ketikan perintah di folder ci4 nya

php spark serve

* 1. Tetapi untuk menggunakan XAMPP tidak perlu jalankan built in server
  2. Dalam folder ci 4 copy kan file ‘env’ buat namanya ‘.env’. Untuk mengaktifkan fungsi dalam .env ini hapus tanda #.
     1. Pada saat proses membangun aplikasi, aktifkan CI\_ENVIRONMENT dan ubah isinya menjadi = development
     2. Saat mode ini maka ketika ada error akan ditampilkan semua, kalau mode production, errornya disembunyikan.
     3. Ubah base url di .env dengan mengaktifkan bagian #app.baseURL ‘hilangkan tanda #’ lalu isi dengan base url sendiri.

1. **Konfigurasi CI 4 bagian Routes**
   1. Konfigurasi routes, letaknya di app>config>route.php. pada bagian routes bisa diubah untuk menentukan bagian awal aplikasi ketika dibuka.

Halaman pertama aplikasi yang dibuka yaitu :

$routes->get(‘/’, ‘Home::index’);

* 1. Jalur perpindahan halaman dilakukan dibagian routes ini dengan susunan perintah dasarnya :

$routes->Method(‘/link akses’, ‘Controller::method’);

Method bagian awal seperti => get, post, add, delete

Link akses sesuai dengan yang dibutuhkan,

Bagian Controller diisi dengan nama controllernya lalu diikuti dengan method yang terdapat pada controller tersebut.

* 1. Perilaku routes yang lain yaitu, bisa digunakan untuk menjalankan fuction tanpa memanggil controller dan method, contoh :

$routes->get(‘/link’, function() {

Isi function

});

* 1. Dalam routes juga bisa digunakan untuk mengambil data melalui link url.

Contoh :

$routes->get(‘/link/(:any)/(:num)’, ‘Controoler::method/$1/$2

Data yg diterima

Controller & Method

2 data yg diinputkan

Any untuk data apa saja yang diinputkan, num untuk data jenis numerik, alpha untuk data hanya alphabet, alphanum untuk angka dan huruf.

* 1. Penulisan link <a href> dalam codeigniter 4. Jika menggunakan php developmen server (php spark serve) penulisan link dilakukan menggunakan tanda /. Contoh :

<a href=”/pages/home”> Home </a>

**Tetapi** jika menggunakan server xampp penulisan link dilakukan setelah menulis base\_url. Contoh :

<a href=”<?= base\_url(‘/pages/home’) ?>” > Home </a>

1. **Penggunaan View**
   1. Sebaiknya disusun didalam folder sesuai dengan Controller yang memanggil view tersebut.
   2. Untuk menampilkan view dari controller menggunakan perintah

return view(‘namaView’); . Contoh : return view(‘beranda’);

* 1. Untuk menampilkan halaman yang terdiri dari beberapa view dilakukan dengan mengubah *return* menjadi *echo*.
  2. View bisa memisahkan beberapa bagian yang di sering dipanggil ulang seperti bagian header dan footer. Pemisahan dilakukan dengan membuat file baru tempat menyimpan masinng-masing header dan footer.
  3. Untuk memanggil keseluruhan view tersebut, pada controller menggunakan koodingan seperti ini :

Echo view(‘header’);

Echo view(‘namaView’);

Echo view(‘footer’);

* 1. Jika ingin mengirim data ke dalam view dapat dilakukan dengan menambah parmeter dalam pemanggilan view. Contoh :

$data = [‘judul’ => ‘Halaman Home’];

Echo view(‘header’, $data);

Echo view(‘namaView’);

Echo view(‘footer’);

Data yang telah dikirimkan dapat digunakan di kodingan halaman view tersebut.

* 1. Bagian Header dan Footer ini juga dapat di gabungkan kembali menjadi 1 file. Misalkan header dan footer digabungkan ke dalam sebuah file bernama template.php . Pada kodingan template tersebut, diantara kofingan header dan footer dimasukan kodingan untuk meletakan kontent dari view yang ingin ditampilkan. Kodingan yang ditulis :

<-- Ahir bagian header -->

<?= $this->renderSection(‘content’); ?>

<-- Awal bagian footer -->

* 1. Untuk menghubungkan Konten view dengan template diatas, ditambahkan kodingan seperti dibawah ini.

<?= $this->extend(‘template’); ?>

<?= $this->section(‘content’);?>

<--Ini isi Kontent view -->

<?= $this->endSection(); ?>

* 1. Jika menggunakan template, pemanggilan view pada bagian controller tidak perlu memanggil header dan footer lagi. View dapat langsung dipanggil, dan karena pemanggilan view hanya 1 line maka dapat menggunakan return.

return view(‘namaView’, $data);

* 1. Misalkan bagian Navbar di dalam template ingin dipisah jika dibutuhkan. Hal tersebut dapat dilakukan dengan pertama membuat file baru tempat meletakan navbar tersebut, contoh nabar.php. Pada bagian template ditambahkan kodingan untuk memanggil navbar tersebut :

<?= $this->include(‘layout/navbar’); ?>

1. **Penggunaan Controller**
   1. .
   2. .
   3. Menerima data dari form

**$this->request->getVar();**

Perintah ini berfungsi untuk menerima seluruh data yang dikirimkan memlalui form. Berikut contoh penggunaannya :

Misalkan pada view insert data terdapat form yang berfungsi untuk menambahkan data.

**<form action=”/buku/save” mehod=”post”>**

/buku => nama controller

/save => nama method yang terdapat pada controller tersebut

Lalu pada **Controller Save** dibuatlah method yang bernama **save()**

public function save()

{

$data = $this->request->getVar();

}

* 1. Menerima **File** dari form

**$data = $this->request->getFile(‘**gambar**’);**

**$data->move(‘img);**  // Kode ini memindahkan file gambar yang telah diupload ke dalam folder **img**

**$namaGambar = $data->getName();**

//Kode ini untuk mengambil nama file yang telah diupload

Nama ini dibutuhkan untuk menampilkan file ini nantinya.

1. **Penggunaan Model**
   1. Konfigurasi database
      1. Pertama edit pada bagian **.env** . Aktifkan semua yang ada di bagian databse dengan menghapus tanda pagar ( # ).
      2. Edit isi pada bagian database tersebut :
         1. Bagian hostname bisa diisi dengan **localhost** , tetapi jika ada masalah bisa diganti dengan ip **127.0.0.1 .**
         2. Bagian database sesuai dengan nama database di phpmyadmin.
         3. Uername = root, kalo pakai xampp password = , (kosong).
   2. Membuat Model
      1. Nama model dibuat biasanya disesuaikan dengan nama tabel database dan diikut kata model. Contoh : bukuModel
      2. Kodingan dasar dalam model :

<?php namespace App\Models;

use CodeIgniter\Model;

class BukuModel extends Model

{

}

* + 1. Untuk referensi isi dalam model dapat di lihat di dokumentasi Codeigniter 4 bagian **Modeling Data**.
    2. Setiap model yang dibuat mengacu pada file **parrent** Model yang berisi pengaturan dasar dari model. Letaknya ada di **\vendor\system\Model.php** .

Jika pengaturan sudah sesuai dengan default yang ada pada parrent, maka bisa dikosongkan pada model yang dibuat.

Jika ingin menngunakan fitur yang terdapat pada **parrent Model** dapat dilakukan dengan **mencopy** **fitur** dari parrent dan dipindahkan ke model yang dibuat. Contoh : fitur **protected $useTimestamps = false;** silahkan **copy** kodingan fitur ini lalu tempelkan pada model sendiri. Jika ingin menggunakannya maka ubah menjadi **protected $useTimestamps = true;** .

Dengan ini maka tabel dari database sudah terhubung dengan model kita.

* 1. Menghubungkan Model dengan Controller
     1. Pada Controller inisiasikan pemanggilan model dengan koding :

$bukuModel = new \App\Models\BukuModel();

* + 1. Mengambil isi database menggunakan model :

$buku = $bukuModel->findAll();

Perintah **findAll()** sama dengan **select \* from table.**

* + 1. Jika model ingin dipakai pada banyak **method**  dalam sebuah **Controller**, maka pemanggilan model dapat dibuat menjadi sebuah **Constructor**.

Protected $bukuModel;

public function \_\_construct ()

{

$this->bukuModel = new BukuModel();

}

Di dalam method digunakan dengan kodingan :

$buku = $this->bukuModel->findAll();

* 1. Model juga berguna untuk pengolahan database menggunakan function agar bisa diakses dari controller. Selengkapnya di tutorial CI4 video ke 8 menit ke 6.

Method untuk mengambil data dari database dari **Model**

Public function getBuku($slug = false)

{

If ($slug == false) {

Return $this->findAll();

}

Return $this->where([‘slug’ => $slug]->first();

}

Maksud parameter pada method tersebut adalah menerima parameter disimpan dalam variabel **$slug** dan bila tidak menerima data maka **$slug** secara default diisi **false.**

Untuk menngakses **method Model** tersebut, maka di **controller** dilakukan seperti ini :

**Controller Buku**  method **index()**

Public function index()

$data = [

‘title’ => ‘Daftar Buku’,

‘buku’ => $this->bukuModel->getBuku();

];

Return view(‘buku/index’, $data);

}

Controller **Buku** method **detail()**

Public function detail($slug) //parameter diterima dari route//

$buku = $this->bukuModel->getBuku($slug);

//parameter $slug dikirim ke Model??

dd($buku); // melihat hasil dari model //

}

* 1. Untuk fungsi perintah pada model seperti **finAll(), where(),**  dapat dilihat selenkapnya di website dokumentasi codeigniter.

1. **Perintah berguna pada CI 4**
2. Dump & Dump and Die

Perintah ini mirip dengan vardump pada php. Cara penggunaannya :

d($tes); => hanya dump (menampilkan) saja, tetap menjalankan program sesudahnya.

dd($tes); => melakukan dump and die, yaitu tidak menjalankan program sesudahnya.

1. Csrf (Cross Site Resource Forgery)

Fungsi ini sama seperti yang ada pada **Framework Laravel**. Fungsi nya yaitu membuat suatu form hanya bisa diakses oleh halaman yang menggunakan fungsi csrf ini saja. Hal ini dapat mencegah orang lain yang ingin menagkses form ini dari tempat lain. Arti csrf juga yaitu mencegah pemalsuan dari halaman lain.

Penggunaannya yaitu dengan meletakan kodingan csrf tepat dibawah tag **<form>.** Contoh :

<form action=”/buku/save” method=”post”>

**<?= csrf\_field(); ?>**

<-- Isi form -->

</form>

1. Exception

Pada CI 4 terdapat fungsi untuk menampilkan exception. Contoh cara penggunaannya yaitu.

**throw new \Codeigniter\Exceptions\PageNotFoundException(**‘Judul buku’ . $slug . ‘tidak ditemukan.’);

exception ini akan menampilkan halaman dengan tampilan Page Not Found kode 404, dan pesan yang diberikan yaiu **Judul buku (judul yang diterima) tidak ditemukan.**

1. Slug

CI 4 juga mempunyai fitur slug, yaitu fitur yang berfungsi untuk mengubah sebuah judul menjadi slug (judul dengan format url). Cara penggunaanya :

$slug = url\_title($this->request->getVar(‘judul’, ‘-‘, true);

Arti dari kode ini adalah membuat slug dari variabel judul, jika ada spasi akan diubah menjadi tanda strip ( - ), parameter berikutnya yaitu setiap huruf akan diubah menjadi huruf kecil = true.

1. Perlindungan database

Jika ingin mengakses database seperti menambah data, edit data, hapus data, kita harus membuat **ketentuan** pada **Model** kita field apa saja yang boleh diubah oleh kita.

Contoh penggunaanya :

Menuliskan kode pada Model

class bukuModel extends Model

{

protected $table = ‘buku’; //nama tabel yang digunakan //

protected $useTimestaps = true; //menggunakan fitur Time Stapms//

protected $allowedFields = [‘judul’, ‘slug’, ‘penulis’, ‘penerbit’, ‘sampul];

//menentukan bagian tabel yang boleh di ubah oleh kita//

<-- Isi method lain -->

}

1. Flash data

Menampilkan pemberitahuan / notifikasi setelah mengolah data. Sesuai dengan namanya Flash, data ini hanya ada 1 kali saja. Cara penggunaanya yaitu dengan menambahkan kodingan ini sebelum melakukan return redirect.

**session()->setFlashdata(**‘pesan’, ‘Data berghasil ditambahkan’**);**

Arti kode ini adalah menampilkan flash data dengan nama **pesan** dan pesan yang diberikan yaitu **Data berhasil ditambahkan.**

Menampilkan notifikasi ini dilakukan dibagian view, dengan memasukan kode :

<?php if(session()->getFlashdata(‘pesan’)) : ?>

//arti kode ini yaitu jika ada session flash data yang bernama pesan

<div class=”alert alert-success” role=”alert”>

<?= session()->getFlashdata(‘pesan’); ?> //menampilkan pesan dari flash data

</div>

<?php endif; ?>

1. Validasi data

Informasi mengeneai validasi data secara lengkap dapat dilihat di web dokumentasi codeigniter4 bagian **Validation**. Berikut contoh penggunaan validasi data di codeigniter :

If(!$this->validate([

‘judul’ => ‘required | is\_unique[buku.judul]’

//validasi is\_unique harus menyertakan field yang dituju

])) {

$validation = \Config\Services::validation(); //Mengambil pesan kesalahan yang didapat

return redirect()->to(‘/buku/create’)->**withInput()**->**with(‘validation’, $validation);**

}

Keterangan :

withInput() => mengirim seluruh input, namun input dikirim melalui **Session**. Jadi di bagian **Method di Controller** harus menangkap **Session** ini.

with(‘validation’, $validation) => validation yang menjadi parameter pertama dan ada diantara tanda kutip merupakan **key**, parameter kedua **$validation** merupakan data yang dikirimkan.

Tetapi jika kita ingin membuat **Pesan Error** kita sendiri, di CI 4 dapat dilakukan dengan kode seperti ini :

If(!$this->validate([

‘judul’ => [

‘rules’ => ‘required | is\_unique[buku.judul]’,

‘errors’ => [

‘required’ => ‘**{field}** buku harus diisi.’,

‘is\_unique’ => ‘**{field}** buku sudah terdaftar’

]

]

])) {

return redirect()->to(‘/buku/create’)->**withInput()**;

}

Berikut kode yang ditambahkan pada **Method di Controller** yang menerima input.

public function create()

{

**session();**

$data = [

‘title’ => ‘Form Tambag Data Buku’,

**‘validation’ => \Config\Service::validation()**

];

Selanjutnya pada bagian **View** untuk menampilkan pesan eror ini dapat menggungakan kode :

**$validation->hasError(‘judul’);** //ini untuk mengembalikan nilai **true** jika ada error

**$validation->getError(‘judul’);** //ini untuk mengembalikan **pesan error.**

1. Validasi File

Berbeda dengan validasi data, untuk validasi dasi File seperti gambar, video, dll memiki atribut validasi lain. Contoh

If(!$this->validate([

‘sampul’ => **‘uploaded[sampul]’**

])

Atribut **uploaded** sama fungsinya dengan **required.**

Untuk rule lain selengkapnya bisa dilihat di website **Dokumentasi CI 4** bagian validation. Contoh rule lain untuk memvalidasi apakah file yang diupload benar merupakan gambar, rules yang digunakan yaitu **mime\_in** dan **is\_image**. Di sini tidak hanya menggunakan is\_image tetapi harus ditambahkan dengan **mime\_in**. Hal ini dilakukan untuk memastikan bahwa file yang diupload benar benar gambar, bukan file lain seperti **script jahat** yang diubah formatnya jadi gambar (jpg,png, dll). Berikut contoh Kode penggunaannya :

‘sampul’ => [

‘rules’ => ‘uploaded[sampul] |max\_size[sampul**,**1024] | is\_image[sampul] | mime\_in[sampul, image/jpg**,**image/jpeg**,**image/png] ‘,

‘errors’ => ‘[

‘nama rules’ => ‘ Sesuaikan dengan keinginan, contoh seperti validation sebelumnya’

]

]

1. Old data

Old data ini berfungsi untuk mengisi atau membiarkan data yang sudah diisi tetap tampil di dalam form ketika halaman di refresh saat menerima Validasi error. Old data ini diambil dari **Data yang sudah kita simpan ke dalam Session** melalui perintah **withInput()** yang kita buat sebelumnya. Cara menggunakannya ialah dengan mengisikan pada bagian **value** di form..

**<?= old(‘judul’); ?>**

1. HTTP method spoofing

Membuat pengamanan data, misal dalam melakukan delete data. Masalah yang terjadi pada delete data yaitu operasi delete bisa dilakukan melalui link url. Untuk mengatasi masalah itu maka harus menggunakan **HTTP method spoofing** ini. Cara penggunaanya :

* 1. Pertama, link delete diubah menjadi sebuah **button** yang dimasukkan ke dalam form.

<form action=”**/buku/<?= $buku[‘id’]; ?>**” method=”POST” class=”d-inline”>

<?= csrf\_field(); ?>

**<input type=”hidden” name=”\_method” value=”DELETE”>**

<button type=”**submit**” class=”btn btn-danger”>Delete</button>

</form>

Keterangan:

* + - * + Link form action diarahkan ke Controller buku dengan parameter **id .** Hal ini agar tidak terlihat pada url bahwa ini merupakan method **delete**.
        + Bagian **input type=”hidden”** ini merupakan metode spoofing, yaitu metode untuk mengubah method form yang awalanya **POST** menjadi method **DELETE**.
  1. Selanjutnya untuk membedakan link form dengan link yang awalnya menunjukan detail buku, maka dilaukan penyesuaian di bagian **Routes**.

Buat routes baru :

$routes->**delete**(‘/buku/**(:num)**’, ‘Buku::delete/$1’);

\*perhatikan, jika pada routes untuk **detail buku** sebelumnya mengambil nilai dengan tipe **(:segment)** ubah menjadi **(:any)**. Karna segment menyebabkan method delete ikut dijalankan pada route detail buku.

1. save()

fungsi save merupakan fungsi yang sudah tersedia pada bagian **model.** Fungsi ini memiliki 2 (dua) kegunaan, yaitu *Memasukan data baru* dan *Mengubah data yang sudah ada*. Kalau pada CRUD yaitu **Create dan Update**. Contoh penggunaan :

**$this->bukuModel->save([ *data yang ingin ditambahkan / diubah* ]);**

Yang membedakan penggunaanya untuk create dan update adalah jika untuk **Update** ditambahkan variabel **$id**.

1. getError() = 4

fungsi ini untuk melihat error dengan kode 4, error **kode 4** adalah error ketika suatu file tidak di inputkan / diupload. Contoh pengecekan pada saat upload file **Gambar**.

$fileSampul = $this>request->getfile(‘sampul’); // ini utnuk mengambil file

If(**$fileSampul->getError()==4) {**  //ini untuk cek gambar terupload atau tidak

<--kode ketika true -->

**}**

1. Unlink file

Fungsi ini berguna untuk menghapus file yang tidak berguna lagi dari direktori kita. Misal, kita upload gambar dan gambar itu tersimpan di direktory program kita. Saat ingin menghapus gambar, agar gambar di direktori kita juga ikut terhapus maka digunakan fungsi ini. Cara penggunaannya :

**unlink(‘img/’. $sampul);**

Keterangan : paramaternya iyalah **letak folder** yang menyimpan file tersebut lalu titik (.) diikuti dengan **nama file** yang ingin dihapus.

1. Insert banyak data sekaligus ke database

**$builder->InsertBatch(***$data***);**

1. **Meletekan file dalam CI 4**
   1. Folder public

Meletakan folder css, js, dan img

1. **Tips CSS**
   1. Untuk menargetkan css pada bagian yang spesifik dapat dilakukan seperti contoh ini :

.table > tbody > tr > td { isi css; } **atau** .table > tbody > tr > \* { isi css; }

Keterangan : - tanda di depan suatu bagian

Tanda titik ( . ) ditujukan untuk nama Class

Tanda pagar ( # ) ditujukan untuk nama ID

* + - * + Tanda bintang( \* ) di akhir dimaksudkan untuk semua jenis item
  1. Untuk menambahkan tampilan validation dengan data validation yang diterima dari CI4, **Bootstrap** telah memiliki class yang mengatasi hal ini. Selengkapanya dapat melihat di dokumentasi Bootstrap dengan mengetik di mesin pencariannya **is-invalid**. Cari ke bawah yang berjudul **Server Side**.
  2. Untuk upload file, **Bootstrap** juga sudah menyediakan class nya. Untuk mencarinya ketikkan **File Browser** di mesin pencarian bootstrap.

**Ketika bekerja dengan file**, pada bagian **<form>** wajib ditambahkan atribut lagi yaitu **enctype=”multipart/form-data”.** Atribut ini berguna agar kita bisa bekerja dengan inputan biasa atau inputan file (dengan GET / POST / FILE).

* 1. Preview Gambar

Ketika kita ingin upload gambar dan ingin melihat apakah gambar yang kita input sudah benar. Untuk menampilkan **Preview** nya kita membutuhkan javascript.

Untuk kode nya bisa search **Preview Image / preview gambar**  di google. Atau contohnya ada di vide CI4 **episode 12.**

1. **Fitur CI4 = Migration, Seeder, dan Faker**

Fitur ini mirip dengan yang ada pada Framework Laravel juga.

1. **Migration**

Migration atau database Migration adalah fitur yang berfungsi untuk mengolah database hanya dengan menggunakan **kodingan**. Selengakap nya dapat dilihat di website dokumentasi codeigniter 4 bagian **Database Migrations.** Untuk inisiasinya dapat mengetikan di bagian terminal :

php spark migrate:create Orang

\*Lalu tekan enter, **Orang** merupakan nama tabel yang dibuat

File migration yang telah dibuat dapat dilihat di App/Database/Migration/

Setelah menemuka file migration, selanjutnya ubah isi nya menjadi kodingan di bawah ini, kodingan ini juga dapat ditemukan di dokumentasi CI4 :

**<?**php

**namespace** App\Database\Migrations;

**use** CodeIgniter\Database\Migration;

**class** **AddBlog** **extends** Migration

{

**public** **function** **up**()

{

$this**->**forge**->**addField([

'blog\_id' **=>** [

'type' **=>** 'INT',

'constraint' **=>** 5,

'unsigned' **=>** **true**,

'auto\_increment' **=>** **true**,

],

'blog\_title' **=>** [

'type' **=>** 'VARCHAR',

'constraint' **=>** '100',

],

'blog\_description' **=>** [

'type' **=>** 'TEXT',

'null' **=>** **true**,

],

]);

$this**->**forge**->**addKey('blog\_id', **true**);

$this**->**forge**->**createTable('blog');

}

**public** **function** **down**()

{

$this**->**forge**->**dropTable('blog');

}

}

Keterangan :

* + - * + Fungsi Up() untuk menambah/membuat tabel
        + Fungsi Down() untuk operasi penghapusan tabel
      1. **Seeder**

Seeder adalah metode pengisian data ke dalam tabel menggunakan kodingan. Penjelasan ada di vidieo CI4 eps 13 menit ke 17.

* + - 1. **Faker**

Faker adalah sebuah fitur untuk melakukan generate data **Dummy** atau data percobaan. Cara pakainya harus kunjungi dahulu librarynya, karena fitur ini tidak ada dalam ci atau bukan termasuk built-in dari CI nya. Untuk library faker dapat di search di google.

Cari di google **Faker Library**, paling pertama. Github namanya **fzaninotto**.