Konversi SharIF Judge dari CodeIgniter 3 ke CodeIgnier 4

Filipus–6181901074

1 Data Skripsi

Pembimbing utama/tunggal: **Pascal Alfadian Nugroho**
Pembimbing pendamping: -
Kode Topik : **PAN5492**
Topik ini sudah dikerjakan selama : **1** semester
Pengambilan pertama kali topik ini pada : Semester **54 - Genap 22/23**
Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**
Tipe Laporan : **B -** Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

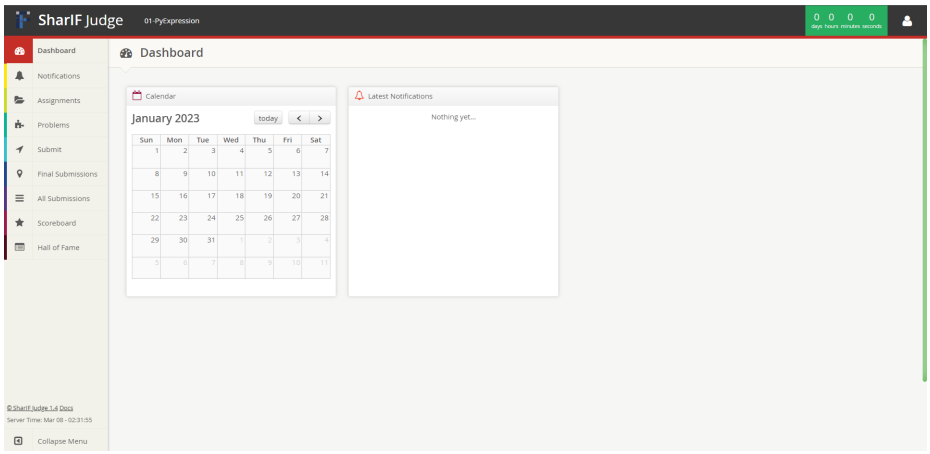
2 Latar Belakang

Tugas merupakan suatu bentuk pembelajaran dan penilaian yang diberikan oleh pengajar kepada pelajar untuk membantu pelajar mendalami materi yang sudah diberikan[1]. Pembagian tugas yang diberikan dapat dibagi menjadi 2 jenis yakni tugas individu dan tugas kelompok. Tugas individu merupakan tugas yang hanya ditanggung oleh satu individu sedangkan, tugas kelompok merupakan tugas yang ditanggung oleh beberapa individu. Tugas selanjutnya akan dikumpulkan kepada pengajar dan diberikan penilaian berdasarkan tingkat ketepatan jawaban dari tugas tersebut. Pengumpulan dan pengecekan tugas terutama *coding* secara manual memiliki kekurangan dimana diperlukan banyak langkah dalam melakukan pengecekan dan pengiriman nilai. Pengecekan secara manual juga terdapat kesulitan dalam pengecekan yakni, kekurangan dalam pengecekan plagiat antara tugas pelajar. Maka, dibutuhkan perangkat lunak untuk melakukan pengecekan secara otomatis salah satunya adalah *Online Judge*.

Online Judge merupakan sebuah perangkat lunak yang dapat melakukan pengecekan program sesuai dengan standar yang sudah diberikan. Perangkat lunak ini dapat menerima jawaban dari pelajar dan melakukan pengecekan secara otomatis dan memberikan keluaran berupa nilai dari pelajar tersebut[2]. Salah satu perangkat lunak *Online Judge* terdapat pada Universitas Katolik Parahyangan prodi Informatika bernama SharIF Judge (dapat dilihat pada Gambar 1).

[1] Prihantini, F. N. (2017, November 26). Kesadaran dan Perilaku Plagiarisme dikalangan Mahasiswa (Studi pada Mahasiswa Fakultas Ekonomi Jurusan Akuntansi Universitas Semarang). Prihantini | Jurnal Dinamika Sosial Budaya. https://journals.usm.ac.id/index.php/jdsb/article/view/559/370(19 Maret 2023)

[2] Kurnia, A., Lim, A., & Cheang, B. (2001). Online Judge. Computers & Education, 36(4), 299–315. https://doi.org/10.1016/s0360-1315(01)00018-5 (23 Maret 2023)



Gambar 1: Tampilan halaman *SharIF Judge*

SharIF Judge merupakan sebuah perangkat lunak *open source* untuk menilai kode dengan beberapa bahasa seperti C, C++, Java, dan Python secara online. SharIF Judge dibentuk menggunakan *framework* CodeIgniter 3 yang merupakan *framework* berbasis PHP (*Hypertext Preprocessor*) dan dimodifikasi sesuai dengan kebutuhan Informatika Unpar untuk mengumpulkan tugas dan ujian mahasiswa[3].

CodeIgniter 3 merupakan sebuah *framework opensource* yang bertujuan untuk mempermudah dalam membentuk sebuah aplikasi *website* menggunakan PHP. CodeIgniter 3 menggunakan struktur MVC yang membagi file menjadi 3 buah yaitu Model, View, Controller. Selain itu, CodeIgniter 3 merupakan *framework* ringan dan menyediakan banyak *library* untuk digunakan oleh penggunanya[4]. Namun, CodeIgniter 3 sudah memasuki fase *maintenance*[5] sehingga tidak akan mendapatkan *update* lebih lanjut dari pembentuknya. CodeIgniter 3 pada akhirnya akan tidak dapat dipakai dan akan hilangnya dokumentasi dari situs web resmi. Sehingga, perangkat lunak yang menggunakan CodeIgniter 3 perlu dikonversi ke *framework* CodeIgniter dengan versi terbaru yakni CodeIgniter 4.

CodeIgniter 4 merupakan versi terbaru dari *framework* CodeIgniter yang memiliki banyak perubahan fitur dari versi sebelumnya. CodeIgniter 4 dapat dijalankan menggunakan versi PHP 7.4 atau lebih baru sedangkan CodeIgniter 3 dapat dijalankan menggunakan versi PHP 5.6 atau lebih baru. CodeIgniter 4 juga membagi file menggunakan struktur MVC namun, memiliki struktrur folder berbeda dengan versi sebelumnya[6].

Pada skripsi ini, akan dilakukan konversi SharIF Judge dari CodeIgniter 3 sehingga dapat berjalan pada CodeIgniter 4.

3 Rumusan Masalah

- Bagaimana cara melakukan konversi CodeIgniter 3 menjadi CodeIgniter 4?
- Bagaimana mengevaluasi kode SharIF Judge dan mengubahnya agar dapat berjalan di CodeIgniter 4?

4 Tujuan

- Melakukan konversi dengan megubah kode sesuai dengan CodeIgniter 4.
- Melakukan evaluasi kode SharIF Judge dan mengubahnya agar dapat berjalan di CodeIgniter 4.

[3]SharIF Judge *Documentation* `https://github.com/ifunpar/SharIF-Judge/blob/docs/readme.md` (19 Maret 2023)
[4]Dokumentasi *CodeIgniter 3* `https://codeigniter.com/userguide3`(19 Maret 2023)
[5]Pemberitahuan fase *maintenance CodeIgniter 3. Welcome to CodeIgniter. (n.d.). https://codeigniter.com/download* `https://codeigniter.com/download`(19 Maret 2023)
[6]Dokumentasi *CodeIgniter 4* `https://codeigniter.com/user_guide`(19 Maret 2023)

5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencan kerja/laporan perkembangan terkahir :

1. **Melakkan studi literatur mengenai *Online Judge*.**[7]
   Status : baru ditambakan pada semester ini.
   Hasil : *Online Judge* sudah dipelajari.
   Online Judge merupakan sebuah perangkat lunak yang dibentuk untuk membantu pengguna dalam melakukan pengecekan dan penilaian terhadap sebuah program. *Online Judge* dibentuk untuk membantu pengajar dalam melakukan pengecekan secara otomatis dan membantu pengajar dalam menghindari program berbahaya seperti terdapat *malware*. Perangkat lunak ini dapat menerima masukan berupa program yang selanjutnya akan diperiksa dan memberikan keluaran berupa nilai dari program tersebut. *Online Judge* akan menilai program berdasarkan masukan dan keluaran sesuai dengan standar yang sudah ditentukan. Program akan dinilai menggunakan *secret test sets* sehingga, pengajar dapat mengetahui apakah program akan mengeluarkan keluaran sesuai dengan *test sets* yang diberikan.

   Penggunaan *online judge* dapat memiliki beberapa keuntungan dalam melakukan pengecekan program. Berikut merupakan beberapa keuntungan penggunaan *online judge*:

   - **Tidak ada solusi standart**
     Program tidak akan dinilai berdasarkan kodenya namun, akan dinilai berdasarkan keluarannya. Sehingga, setiap pengguna dapat mengumpulkan program berbeda dengan hasil yang sama.
   - **Aman dari program berbahaya**
     Pengajar dapat terhindar dari program berbahaya karena *online judge* akan memberhentikan program yang dianggap berbahaya
   - **Konsisten**
     Penilaian akan selalu konsisten tanpa terpengaruh oleh keadaan penilai karena dinilai menggunakan program dengan keadaan yang sama.
   - **Mudah untuk dinilai ulang**
     Pengajar dapat melakukan penilaian ulang dengan mudah tanpa harus menjalankan ulang program yang sudah terkumpul.
   - **Lebih banyak tugas**
     Pengajar dapat memberikan lebih banyak tugas tanpa harus menilai program lebih banyak.
   - **Penilaian cepat**
     Penilaian dilakukan lebih cepat sehingga, pengguna dapat menerima nilai dan pengajar dapat menyimpan nilai lebih cepat dibandingkan secara manual.

2. **Melakukan studi literatur mengenai *SharIF Judge*[8].**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *SharIF Judge* sudah dipelajari dan berhasil di jalankan pada *local*.
   SharIF Judge merupakan sebuah *Online Judge* percabangan dari *Sharif Judge* yang dibentuk oleh Mohammed Javad Naderi. *Sharif Judge* dibentuk menggunakan *CodeIgniter 3* dan dimodifikasi sesuai dengan kebutuhan di Informatika Universitas Katolik Parahyangan menjadi nama *SharIF Judge*. *SharIF Judge* dapat menilai kode berbahasa *C, C++, Java, dan Python* dengan mengunggah file ataupun mengetiknya secara langsung. *SharIF Judge* memiliki struktur aplikasi seperti berikut.

   ### Struktur Aplikasi

[7] Kurnia, A., Lim, A., & Cheang, B. (2001). Online Judge. Computers & Education, 36(4), 299–315. https://doi.org/10.1016/s0360-1315(01)00018-5 (23 Maret 2023)
[8] Dokumentasi *SharIF Judge* `https://github.com/ifunpar/SharIF-Judge/blob/docs/readme.md`(19 Maret 2023)

```
.
|-- application
|   |-- cache
|   |-- config
|   |-- controllers
|   |-- core
|   |-- helpers
|   |-- hooks
|   |-- language
|   |-- libraries
|   |-- logs
|   |-- models
|   |-- third_party
|   |-- vendor
|   |-- views
|-- assets
|   |-- images
|   |-- js
|   |-- styles
|-- restricted
|   |-- tester
|-- system
|-- assignments
```

Instalasi

Berikut merupakan persyaratan dan langkah-langkah melakukan *instalasi SharIF Judge*:

Persyaratan

SharIF Judge dapat dijalankan pada sistem operasi *Linux* dengan syarat sebagai berikut:

- Diperlukan *webserver* dengan versi PHP 5.3 atau lebih baru.
- Pengguna dapat menjalankan PHP pada *command line*. Pada *Ubuntu* diperlukan instalasi paket *php5-cli*.
- *MySql database* dengan ekstensi *Mysqli* untuk PHP atau *PostgreSql database*.
- PHP harus memiliki akses untuk menjalankan perintah melalui fungsi *shell_ exec*.

Kode 1: Kode untuk melakukah pengetesan fungsi

```
    echo shell_exec("php -v");
```

- *Tools* untuk melakukan kompilasi dan menjalankan kode yang dikumpulkan (*gcc, g++, javac, java, python2, python3*).
- *Perl* disarankan untuk diinstalasi untuk alasan ketepatan waktu, batas memori, dan memaksimalkan batas ukuran pada hasil kode yang dikirim.

Instalasi

(a) Mengunduh versi terakhir dari *SharIF Judge* dan melakukan *unpack* pada direktori *public html*.

(b) Memindahkan *folder system* dan *application* diluar direktori *public* dan mengubah *path* pada `index.php`(Opsional).

Kode 2: Contoh *path* pada halaman index.php

```
    $system_path = '/home/mohammad/secret/system';
    application_folder = '/home/mohammad/secret/application';
```

(c) Membentuk *database MySql* atau *PostgreSql* untuk *SharIF Judge*. Jangan melakukan instalasi paket koneksi *database* apapun untuk *C, C++, Java,* atau *Python*.

(d) Mengatur koneksi *database* pada *file* `application/config/database.example.php` dan menyimpannya dengan nama `database.php`. Pengguna dapat menggunakan awalan untuk nama tabel.

Kode 3: Contoh pengaturan koneksi untuk *database*

```
/*  Enter database connection settings here:  */
'dbdriver' => 'postgre',    // database driver (mysqli, postgre)
'hostname' => 'localhost',  // database host
```

```
'username' => '',          // database username
'password' => '',          // database password
'database' => '',          // database name
'dbprefix' => 'shj_',      // table prefix
```

(e) Mengatur *RADIUS server* dan *mail server* pada *file* `application/config/secrets.example.php` dan menyimpannya dengan nama `secrets.php`.

(f) Mengatur `application/cache/Twig` agar dapat ditulis oleh PHP.

(g) Membuka halaman utama SharIF Judge pada *web browser* dan mengikuti proses instalasi.

(h) Melakukan *Log in* dengan akun admin.

(i) Memindahkan direktori `tester` dan `assignments` diluar direktori publik dan mengatur kedua direktori agar dapat ditulis oleh PHP. Selanjutnya Menyimpan *path* kedua direktori pada halaman *Settings*. Direktori `assignments` digunakan untuk menyimpan *file-file* yang diunggah agar tidak dapat diakses publik.

Clean URLs

Secara asali, `index.php` merupakan bagian dari seluruh *urls* pada SharIF judge. Berikut merupakan contoh dari *urls* SharIF Judge.

`http://example.mjnaderi.ir/index.php/dashboard`

`http://example.mjnaderi.ir/index.php/users/add`

Pengguna dapat menghapus `index.php` pada *url* dan mendapatkan *url* yang baik apabila sistem pengguna mendukung *URL rewriting*.

`http://example.mjnaderi.ir/dashboard`

`http://example.mjnaderi.ir/users/add`

Cara Mengaktifkan *Clean URLs*

- Mengganti nama *file* `.htaccess2` pada direktori utama menjadi `.htaccess`.
- Mengganti `$config['index_page'] = 'index.php';` menjadi `$config['index_page'] = '';` pada *file* `application\config\config.php`.

Users

Pada perangkat lunak SharIF Judge, pengguna dibagi menjadi 4 buah. Keempat pengguna tersebut adalah *Admins, Head Instructors, Instructors*, dan *Students*. Tabel 1 merupakan pembagian tingkat setiap pengguna.

Tabel 1: *Tabel tingkat pengguna*

User Role	***User Level***
Admin	3
Head Instructor	2
Instructor	1
Student	0

Setiap pengguna memiliki akses untuk aksi yang berbeda berdasarkan tingkatnya. Tabel 2 merupakan aksi yang dapat dilakukan oleh setiap pengguna.

Tabel 2: *Tabel izin aksi setiap pengguna*

Aksi	*Admin*	*Head Instructor*	*Instructor*	*Student*
Mengubah *Settings*	✓	×	×	×
Menambah/Menghapus Pengguna	✓	×	×	×
Mengubah Peran Pengguna	✓	×	×	×
Menambah/Menghapus/Mengubah *Assignment*	✓	✓	×	×
Mengunduh *Test*	✓	✓	×	×
Menambah/Menghapus/Mengubah Notifikasi	✓	✓	×	×
Rejudge	✓	✓	×	×
Melihat/*Pause*/Melanjutkan/*Submission Queue*	✓	✓	×	×
Mendeteksi Kode yang Mirip	✓	✓	×	×
Melihat Semua Kode	✓	✓	✓	×
Mengunduh Kode Final	✓	✓	✓	×
Memilih *Assignment*	✓	✓	✓	✓
Submit	✓	✓	✓	✓

Menambahkan Pengguna

Admin dapat menambahkan pengguna melalui bagian *Add User* pada halaman *Users*. Admin harus mengisi setiap informasi dimana baris yang diawali # merupakan komen dan setiap baris lainnya mewakili pengguna dengan sintaks berikut:

Kode 4: Contoh sintaks untuk menambahkan pengguna

```
USERNAME,EMAIL,DISPLAY-NAME,PASSWORD,ROLE

* Usernames may contain lowercase letters or numbers and must be between 3 and 20 characters in length.
* Passwords must be between 6 and 30 characters in length.
* You can use RANDOM[n] for password to generate random n-digit password.
* ROLE must be one of these: 'admin', 'head_instructor', 'instructor', 'student'
```

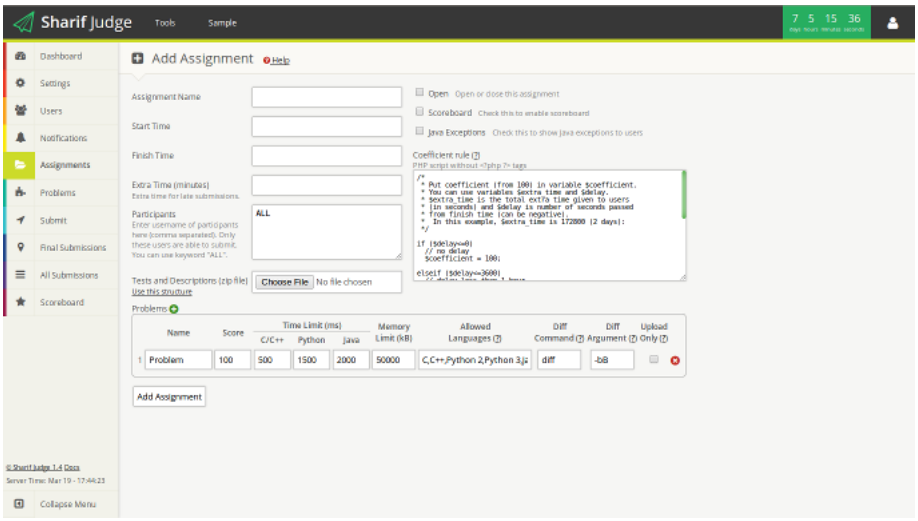
Dengan contoh sebagai berikut:

Kode 5: Contoh kode untuk menambahkan pengguna

```
# This is a comment!
# This is another comment!
instructor,instructor@sharifjudge.ir,Instructor One,123456,head_instructor
instructor2,instructor2@sharifjudge.ir,Instructor Two,random[7],instructor
student1,st1@sharifjudge.ir,Student One,random[6],student
student2,st2@sharifjudge.ir,Student Two,random[6],student
student3,st3@sharifjudge.ir,Student Three,random[6],student
student4,st4@sharifjudge.ir,Student Four,random[6],student
student5,st5@sharifjudge.ir,Student Five,random[6],student
student6,st6@sharifjudge.ir,Student Six,random[6],student
student7,st7@sharifjudge.ir,Student Seven,random[6],student
```

Menambah *Assignment*

Pengguna dapat menambahkan *assignment* baru melalui bagian *Add* pada halaman *Assignments*(dapat dilihat pada Gambar 2).



Gambar 2: Tampilan halaman *SharIF Judge* untuk menambahkan *assignment*

Berikut merupakan beberapa pengaturan pada halaman *Add Assignments*:

- ***Assignment Name***
  Assignment akan ditampilkan sesuai dengan masukan pada daftar *assignment*.
- ***Start Time***
  Pengguna tidak dapat mengumpulkan *assignment* sebelum waktu dimulai("*Start Time*"). Format pengaturan waktu untuk waktu mulai adalah `MM/DD/YYYY HH:MM:SS` dengan contoh `08/31/2013 12:00:00`.
- ***Finish Time, Extra Time***
  Pengguna tidak dapat mengumpukan *assignment* setelah *Finish Time* + *Extra Time*. Pengumpulan *Assignment* pada *Extra Time* akan dikalikan seusai dengan koefisien. Pengguna harus menulis skrip PHP untuk menghitung koefisien pada *field Coefficient Rule*. Format pengaturan waktu untuk waktu selesai sama seperti waktu mulai yakni `MM/DD/YYYY HH:MM:SS` dan format waktu tambahan menggunakan menit dengan contoh `120` (2 jam) atau `48*60` (2 hari).
- ***Participants***
  Pengguna dapat memasukan *username* setiap partisipan atau menggunakan kata kunci *ALL* untuk membiarkan seluruh pengguna melakukan pengumpulan. Contoh: *admin, instructor1 , instructor2 ,student1 , student2,student3 , student4*.
- ***Tests***
  Pengguna dapat mengunggah *test case* dalam bentuk *zip file* sesuai dengan struktur pada 2.
- ***Open***
  Pengguna dapat membuka dan menutup *assigment* untuk pengguna *student* melalui pilihan ini. Pengguna selain *student* tetap dapat mengumpukan *assignment* apabila sudah ditutup.
- ***Score Board***
  Pengguna dapat menghidupkan dan mematikan *score board* melalui pilihan ini.
- ***Java Exceptions***
  Pengguna dapat menghidupkan dan mematikan fungsi untuk menunjukan *java exceptions* kepada pengguna *students* dan tidak akan memengaruhi kode yang sudah di *judge* sebelumnya. Berikut merupakan tampilan apabila fitur *java exceptions* dinyalakan:

Kode 6: Contoh tampilan fitur *Java Exceptions*

```
1 Test 1
2 ACCEPT
```

```
Test 2
Runtime Error (java.lang.ArrayIndexOutOfBoundsException)
Test 3
Runtime Error (java.lang.ArrayIndexOutOfBoundsException)
Test 4
ACCEPT
Test 5
ACCEPT
Test 6
ACCEPT
Test 7
ACCEPT
Test 8
Runtime Error (java.lang.ArrayIndexOutOfBoundsException)
Test 9
Runtime Error (java.lang.StackOverflowError)
Test 10
Runtime Error (java.lang.ArrayIndexOutOfBoundsException)
```

- ***Archived Assignment***
  Pengguna dapat menghidupkan fitur ini dan *assignment* akan dibentuk dengan waktu selesai `2038-01-18 00:00:00` (UTC + 7) dengan kata lain pengguna memiliki waktut tidak terhingga untuk mengumpulkan *assignment*.
- ***Coefficient Rule***
  Pengguna dapat menuliskan skrip PHP pada bagian ini untuk menghitung koefisien dikalikan dengan skor. Pengguna harus memasukan koefisien (dari 100) dalam variabel `$coefficient`. Pengguna dapat menggunakan variabel `$extra_time` dan `$delay`. `$extra_time` merupakan total dari waktu tambahan yang diberikan kepada pengguna dalam detik dan `$delay` merupakan waktu dalam detik yang melewati waktu selesai(dapat berupa negatif). Skrip PHP pada bagian ini tidak boleh mengandung tag `<?php`, `<?`, dan `?>`. Berikut merupakan contoh skrip dimana `$extra_time` adalah 172800(2 hari):

Kode 7: Contoh skrip PHP

```php
if ($delay<=0)
  // no delay
  $coefficient = 100;

elseif ($delay<=3600)
  // delay less than 1 hour
  $coefficient = ceil(100-((30*$delay)/3600));

elseif ($delay<=86400)
  // delay more than 1 hour and less than 1 day
  $coefficient = 70;

elseif (($delay-86400)<=3600)
  // delay less than 1 hour in second day
  $coefficient = ceil(70-((20*($delay-86400))/3600));

elseif (($delay-86400)<=86400)
  // delay more than 1 hour in second day
  $coefficient = 50;

elseif ($delay > $extra_time)
  // too late
  $coefficient = 0;
```

- ***Name***
  Merupakan nama dari masalah pada *assignments*.
- ***Score***
  Merupakan skor dari masalah pada *assignments*.
- ***Time Limit***
  Pengguna dapat menentukan batas waktu untuk menjalankan kode dalam satuan milidetik. Bahasa *Python* dan *Java* biasanya memiliki waktu lebih lambat dari *C/C++* sehingga membutuhkan waktu lebih lama.
- ***Memory Limit***
  Pengguna dapat menentukan batas memori dalam *kilobytes* namun, pengguna pembatasan memori tidak terlalu akurat.

- ***Allowed Languages***
  Pengguna dapat menentukan bahasa setiap masalah pada *assignment*(dipisahkan oleh koma). Terdapat beberapa bahasa yang tersedia yaitu *C, C++, Java, Python 2, Python 3, Zip, PDF* ,dan *TXT*. Pengguna dapat memakai *Zip, PDF* ,dan *TXT* apabila opsi *Upload Only* dinyalakan. Contoh : *C, C++, Zip* atau *Python 2,Python 3*.
- ***Diff Command***
  Diff Command digunakan untuk membandingkan keluaran dengan keluaran yang benar. Secara asali, *SharIF Judge* menggunakan `diff` namun, pengguna dapat menggantinya pada bagian ini dan bagian ini tidak boleh mengandung spasi.
- ***Diff Arguments***
  Pengguna dapat mengatur argumen untuk *diff arguments* pada bagian ini. Pengguna dapat melihat `man diff` untuk daftar lengkap argumen *diff*. *SharIF Judge* terdapat dua buah opsi baru yakni `ignore` dan `identical`.
  - `ignore` : *SharIF Judge* mengabaikan semua baris baru dan spasi.
  - `identical` : *SharIF Judge* tidak mengabaikan apapun namun, keluaran dari *file* yang dikumpulkan harus identik dengan *test case* agar dapat diterima.
- ***Upload Only***
  Pengguna dapat menghidupkan *Upload only* namun, *SharIF Judge* tidak akan menilai masalah tersebut. Pengguna dapat memakai *ZIP*, *PDF*, dan *TXT* pada *allowed languanges* apabila pengguna menghidupkan bagian ini.

Sample Assignment

Berikut merupakan contoh dari *assignment* untuk melakukan pengujian *SharIF Judge*. Penambahan *Assignment* dapat dilakukan dengan memencet tombol *Add* pada halaman *Assignment*.

Problems

(a) *Problem 1 (Sum)*: Program pengguna dapat membaca *integer n*, membaca *n integers* dan mengeluarkan hasil dari *integer* tersebut.

Sample Input	*Sample Output*
5 53 78 0 4 9	145

(b) *Problem 2 (Max)*: Program pengguna dapat membaca *integer n*, membaca *n integer*, dan mengeluarkan total dari dua buah *integer* terbesar diantara *n integer*.

Sample Input	*Sample Output*
7 162 173 159 164 181 158 175	356

(c) *Problem 3 (Upload)*: Pengguna dapat mengunggah *file c* dan *zip* dan *problem* ini tidak akan dinilai karena hanya berupa *Upload Only*.

Tests

Pengguna dapat menemukan *file zip* pada direktori *Assignments*. Berikut merupakan susunan pohon dari ketiga *problems* diatas:

```
.
|-- p1
|   |-- in
|   |   |-- input1.txt
|   |   |-- input2.txt
```

```
|   |   |-- input3.txt
|   |   |-- input4.txt
|   |   |-- input5.txt
|   |   |-- input6.txt
|   |   |-- input7.txt
|   |   |-- input8.txt
|   |   |-- input9.txt
|   |   --- input10.txt
|   |-- out
|   |   --- output1.txt
|   |-- tester.cpp
|   --- desc.md
|-- p2
|   |-- in
|   |   |-- input1.txt
|   |   |-- input2.txt
|   |   |-- input3.txt
|   |   |-- input4.txt
|   |   |-- input5.txt
|   |   |-- input6.txt
|   |   |-- input7.txt
|   |   |-- input8.txt
|   |   |-- input9.txt
|   |   --- input10.txt
|   |-- out
|   |   |-- output1.txt
|   |   |-- output2.txt
|   |   |-- output3.txt
|   |   |-- output4.txt
|   |   |-- output5.txt
|   |   |-- output6.txt
|   |   |-- output7.txt
|   |   |-- output8.txt
|   |   |-- output9.txt
|   |   --- output10.txt
|   |-- desc.md
|   --- Problem2.pdf
|-- p3
|   --- desc.md
--- SampleAssignment.pdf
```

Problem 1 menggunakan metode *"Tester"* untuk mengecek hasil keluaran, sehingga memiliki *file* `tester.cpp`(*Tester Script*). *Problem 2* mengguanakan metode *"Output Comparison"* untuk mengecek hasil keluaran, sehingga memiliki dua buah direktori *in* dan *out* yang berisi *test case*. *Problem 3* merupakan *problem "Upload-Only"*.

Sample Solutions

Berikut merupakan *sample solutions* untuk *problem 1*:

Contoh solusi untuk bahasa *C*

Kode 8: Contoh skrip PHP

```c
#include<stdio.h>
int main(){
    int n;
    scanf("%d",&n);
    int i;
    int sum =0 ;
    int k;
    for(i=0 ; i<n ; i++){
        scanf("%d",&k);
        sum+=k;
    }
    printf("%d\n",sum);
    return 0;
}
```

Contoh solusi untuk bahasa *C++*

```c
#include<stdio.h>
int main(){
    int n;
    scanf("%d",&n);
    int i;
    int sum =0 ;
    int k;
    for(i=0 ; i<n ; i++){
        scanf("%d",&k);
        sum+=k;
    }
    printf("%d\n",sum);
    return 0;
}
```

Contoh solusi untuk bahasa *C*

```
import java.util.Scanner;
class sum
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int sum =0;
        for (int i=0 ; i<n ; i++)
        {
            int a = sc.nextInt();
            sum += a;
        }
        System.out.println(sum);
    }
}
```

Berikut merupakan contoh solusi untuk *problem 2*:

Contoh solusi untuk bahasa *C++*

```
#include<stdio.h>
int main(){
    int n , m1=0, m2=0;
    scanf("%d",&n);
    for(;n--;){
        int k;
        scanf("%d",&k);
        if(k>=m1){
            m2=m1;
            m1=k;
        }
        else if(k>m2)
            m2=k;
    }
    printf("%d",m1+m2);
    return 0;
}
```

contoh solusi untuk bahasa *C++*

```
#include<iostream>
using namespace std;
int main(){
    int n , m1=0, m2=0;
    cin >> n;
    for(;n--;){
        int k;
        cin >> k;
        if(k>=m1){
            m2=m1;
            m1=k;
        }
        else if(k>m2)
            m2=k;
    }
    cout << (m1+m2) << endl ;
    return 0;
}
```

Test Structure

Penambahan *assignment* harus disertakan dengan *file zip* berisikan *test cases*. *File zip* ini sebaiknya berisikan *folder* untuk setiap masalah dengan nama *p1,p1* dan *p3* selain masalah *Upload-Only*.

Metode Pemeriksaan

Terdapat dua buah metode untuk melakukan pemeriksaan yakni metode *Input Output* dan metode *Tester*.

(a) Metode perbandingan *Input Output*

Pada metode ini, pengguna harus memberi masukan dan keluaran pada *folder problem*. Perangkat lunak akan memberikan *file test input* ke kode pengguna dan melakukan perbandingan dengan hasil keluaran kode pengguna. *File input* harus berada didalam *folder in* dengan nama *input1.txt, input2.txt*, dst. *File output* harus berada di dalam *folder out* dengan nama *output1.txt,output2.txt*.

(b) Metode perbandingan *Tester*

Pada metode ini, pengguna harus menyediakan *file input test,* sebuah *file C++,* dan *file output test*(opsional). Perangkat lunak akan memberikan *file input test* ke kode pengguna dan mengambil keluaran pengguna. Selanjutnya, `tester.cpp` akan mengambil masukan pengguna, keluaran tes dan keluaran program pengguna. Jika keluaran pengguna benar maka perangkat lunak akan mengembalikan 1 sedangkan apabila salah maka perangkat lunak akan mengembalikan 0. Berikut adalah templat yang dapat digunakan pengguna untuk menuliskan *file tester.cpp*:

Kode 9: Templat kode *tester.cpp*

```
/*
 * tester.cpp
 */

#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(int argc, char const *argv[])
{

    ifstream test_in(argv[1]);    /* This stream reads from test's input file   */
    ifstream test_out(argv[2]);   /* This stream reads from test's output file  */
    ifstream user_out(argv[3]);   /* This stream reads from user's output file  */

    /* Your code here */
    /* If user's output is correct, return 0, otherwise return 1       */

    ...

}
```

Sample File

Pengguna dapat menemukan *file sample test* pada direktori *assignments.* Berikut merupakan contoh dari pohon *file* tersebut:

```
.
|-- p1
|   |-- in
|   |   |-- input1.txt
|   |   |-- input2.txt
|   |   |-- input3.txt
|   |   |-- input4.txt
|   |   |-- input5.txt
|   |   |-- input6.txt
|   |   |-- input7.txt
|   |   |-- input8.txt
|   |   |-- input9.txt
|   |   --- input10.txt
|   |-- out
|   |   --- output1.txt
|   |-- tester.cpp
|-- p2
|   |-- in
|   |   |-- input1.txt
|   |   |-- input2.txt
|   |   |-- input3.txt
|   |   |-- input4.txt
|   |   |-- input5.txt
|   |   |-- input6.txt
|   |   |-- input7.txt
|   |   |-- input8.txt
|   |   |-- input9.txt
|   |   --- input10.txt
|   |-- out
|   |   |-- output1.txt
|   |   |-- output2.txt
|   |   |-- output3.txt
|   |   |-- output4.txt
|   |   |-- output5.txt
|   |   |-- output6.txt
|   |   |-- output7.txt
|   |   |-- output8.txt
|   |   |-- output9.txt
|   |   --- output10.txt
```

Problem 1 menggunakan metode perbandingan *Tester*, sehingga memiliki *file tester.cpp.* Berikut merupakan *file* untuk *problem 1*:

Kode 10: Kode metode perbandingan *tester* dengan bahasa *tester.cpp*

```
/*
 * tester.cpp
```

```
 */

#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(int argc, char const *argv[])
{

    ifstream test_in(argv[1]);    /* This stream reads from test's input file   */
    ifstream test_out(argv[2]);   /* This stream reads from test's output file  */
    ifstream user_out(argv[3]);   /* This stream reads from user's output file  */

    /* Your code here */
    /* If user's output is correct, return 0, otherwise return 1       */
    /* e.g.: Here the problem is: read n numbers and print their sum:  */

    int sum, user_output;
    user_out >> user_output;

    if ( test_out.good() ) // if test's output file exists
    {
        test_out >> sum;
    }
    else
    {
        int n, a;
        sum=0;
        test_in >> n;
        for (int i=0 ; i<n ; i++){
            test_in >> a;
            sum += a;
        }
    }

    if (sum == user_output)
        return 0;
    else
        return 1;

}
```

Problem 2 menggunakan metode perbandingan *Input Output* sehingga memiliki *folder in* dan *folder out* berisikan *test case*. Sedangkan *problem 3* merupakan *Upload Only*, sehingga tidak memiliki *folder* apapun.

Deteksi Kecurangan

SharIF Judge menggunakan *Moss* untuk mendeteksi kode yang mirip. *Moss*(*Measure of Software Similarity*) merupakan sistem otomatis untuk menentukan kesamaan atau kemiripan dalam program. Penggunaan utama *Moss* adalah untuk melakukan pemeriksaan plagiarisme pada kelas *programming*. Pengguna dapat mengirim hasil kode terakhir(*Final Submission*) ke *server Moss* dengan satu klik.

Penggunaan *Moss* memiliki beberapa hal yang harus diatur oleh pengguna yakni:

(a) Pengguna harus mendapatkan *Moss User id* dan melakukan pengaturan pada *SharIF Judge*. Untuk mendapatkan *Moss User id*, pengguna dapat mendaftar pada halaman `http://theory.stanford.edu/~aiken/moss/`. Pengguna selanjutkan akan mendapatkan *email* berupa skrip *perl* berisikan *user id* pengguna. Berikut merupakan contoh dari potongan skrip *perl*:

Kode 11: Contoh potongan skrip *perl*

```

...

$server = 'moss.stanford.edu';
$port = '7690';
$noreq = "Request not sent.";
$usage = "usage: moss [-x] [-l language] [-d] [-b basefile1] ... [-b basefilen] [-m #] [-c \"string\"] file1 file2 file3 ...";

#
# The userid is used to authenticate your queries to the server; don't change it!
#
$userid=YOUR_MOSS_USER_ID;

#
# Process the command line options.  This is done in a non-standard
# way to allow multiple -b's.
#
$opt_l = "c";   # default language is c
$opt_m = 10;
$opt_d = 0;
```

```
21
22 ...
```

User id pada skrip *perl* diatas dapat digunakan pada *SharIF Judge* untuk mendeteksi kecurangan. Pengguna dapat menyimpan *user id* pada halaman *SharIF Judge Moss* dan *SharIF Judge* akan menggunakan *user id* tersebut.

(b) *Server* pengguna harus memiliki instalasi `perl` untuk menggunakan *Moss*.

(c) Pengguna dianjurkan untuk mendeteksi kode yang mirip setelah *assignment* selesai karena *SharIF Judge* akan mengirimkan hasil akhir kepada *Moss* dan pengguna(*students*) dapat mengganti hasil akhir sebelum *assignment* selesai.

Keamanan

Berikut merupakan langkah untuk melakukan pengaturan keamanan *SharIF Judge*:

(a) **Menggunakan *Sandbox***
Pengguna harus memastikan untuk menggunakan *sandbox* untuk bahasa *C/C++* dan menyalakan *Java Security Manager (Java Policy)* untuk bahasa *java*. Untuk menggunakan *sandbox* dapat dilihat pada sub bab 2.

(b) **Menggunakan *Shield***
Pengguna harus memastikan untuk menggunakan *shield* untuk bahasa *C, C++,* dan *Python*. Penggunaan *shield* dapat dilihat pada subbab 2.

(c) **Menjalankan sebagai *Non-Priviledge User***
Pengguna diwajibkan menjalankan kode yang telah dikumpulkan sebagai *non-priviledge user*. *Non-Priviledge User* adalah *user* yang tidak memiliki akses jaringan, tidak dapat menulis *file* apapun, dan tidak dapat membentuk banyak proses.

Diasumsikan bahwa PHP dijalankan sebagai pengguna `www-data` pada server. Membentuk *user* baru `restricted-user` dan melakukan pengaturan *password*. Selanjutnya, jalankan `sudo visudo` dan tambahkan kode `www-data ALL=(restricted_user) NOPASSWD: ALL` pada baris terakhir *file sudoers*.

- Pada *file* `tester\runcode.sh` ubah kode:

Kode 12: Kode *runcode.sh* awal

```
1 if $TIMEOUT_EXISTS; then
2     timeout -s9 $((TIMELIMITINT*2)) $CMD <$IN >out 2>err
3 else
4     $CMD <$IN >out 2>err
5 fi
```

menjadi:

Kode 13: Kode *runcode.sh* awal

```
1 if $TIMEOUT_EXISTS; then
2     sudo -u restricted_user timeout -s9 $((TIMELIMITINT*2)) $CMD <$IN >out 2>err
3 else
4     sudo -u restricted_user $CMD <$IN >out 2>err
5 fi
```

dan *uncomment* baris berikut:

Kode 14: Kode *runcode.sh* awal

```
1 sudo -u restricted_user pkill -9 -u restricted_user
```

- Mematikan akses jaringan untuk *restricted_ user*
Pengguna dapat mematikan akses jaringan untuk *restricted_ user* di *linux* menggunakan *iptables*. Setelah dimatikan lakukan pengujian menggunakan `ping` sebagai *restricted_ user*.

- Menolak izin menulis
  Pastikan tidak ada direktori atau *file* yang dapat ditulis oleh *restricted_user*.
- Membatasi jumlah proses
  Pengguna dapat membatasi jumlah proses dari *restricted_user* dengan menambahkan kode dibawah melalui *file /etc/security/limits.conf*.

Kode 15: Contoh kode untuk membatasi jumlah proses

```
restricted_user     soft    nproc   3
restricted_user     hard    nproc   5
```

(d) **Menggunakan dua *server***
Pengguna dapat memakai dua *server* untuk antar muka web dan menangani permintaan web dan mengguna *server* lainnya untuk menjalankan kode yang sudah dikumpulkan. Penggunaan dua *server* mengurangi risiko menjalankan kode yang sudah dikumpulkan. Pengguna harus mengganti *source code SharIF Judge* untuk memakai hal ini.

Sandboxing

SharIF Judge menjalankan banyak *arbitrary codes* yang pengguna kumpulkan. *SharIF Judge* harus menjalankan kode pada lingkungan terbatas sehingga memerlukan perkakas untuk *sandbox* kode yang sudah dikumpulkan. Pengguna dapat meningkatkan keamanan dengan menghidupkan *shield* dan *Sandbox*.

C/C++ Sandboxing

SharIF Judge menggunakan *EasySandbox* untuk melakukan *sandboxing* kode *C/C++*. *EasySandbox* berguna untuk membatasi kode yang berjalan menggunakan *seccomp*. *Seccomp* merupakan mekanisme *sandbox* pada *Linux kernel*. Secara asali *EasySandbox* dimatikan pada *SharIF Judge* namun, pengguna dapat menghidupkannya melalui halaman *Settings*. Selain itu, pengguna juga harus *"build EasySandbox"* sebelum menyalakannya. Berikut merupakan cara melakukan *build EasySandbox*:

File EasySandbox terdapat pada *tester/easysandbox*. Untuk membangun *EasySandbox* jalankan:

Kode 16: Kode *runcode.sh* awal

```
$ cd tester/easysandbox
$ chmod +x runalltests.sh
$ chmod +x runtest.sh
$ make runtests
```

Jika keluaran berupa *message All test passed!* maka, *EasySandbox* berhasil dibangun dan dapat dinyalakan pada perangkat lunak.

Java Sandboxing

Secara asali, *Java Sandbox* sudah dinyalakan menggunakan *Java Security Manager*. Pengguna dapat menghidupkan atau mematikannya pada halaman *Settings*.

Shield

Shield merupakan mekanisme sangat simpel untuk melarang jalannya kode yang berpotensi berbahaya. *Shield* bukan solusi *sanboxing* karena *shield* hanya menyediakan proteksi sebagian terhadap serangan remeh. Proteksi utama terhadap kode tidak terpercaya adalah dengan menghidupkan *Sandbox*(dapat dilihat pada subbab2).

Shield untuk *C/C++*

Dengan menghidupkan *shield* untuk *c/c++*, *SharIF Judge* hanya perlu menambahkan `#define` pada awal kode yang dikumpulkan sebelum menjalankannya. Sebagai contoh, pengguna dapat melarang penggunaan `goto` dengan menambahkan kode dibawah pada awal kode yang sudah dikumpulkan.

Kode 17: Kode *shield* untuk melarang penggunaan goto

```
#define goto YouCannotUseGoto
```

Dengan kode diatas, semua kode yang menggunakan `goto` akan mendapatkan *compilation error*. Apabila pengguna menghidupkan *shield*, semua kode yang mengandung `#undef` akan mendapatkan *compilation error*.

- Menghidupkan *shield* untuk *C/C++*
  Pengguna dapat menghidupkan atau mematikan *shield* pada halaman *settings*.
- Menambahkan aturan untuk *C/C++* Daftar aturan `#define` untuk bahasa *C* terdapat pada halaman *tester/shield/defc.h* dan *tester/shield/defcpp.h* untuk bahasa *C++*. Pengguna dapat menambahkan aturan baru pada *file* tersebut pada halaman *settings*. Berikut merupakan contoh sintaks pada untuk menambahkan aturan :

Kode 18: Sintaks aturan *#define*

```
/*

@file defc.h
There should be a newline at end of this file.
Put the message displayed to user after // in each line

e.g. If you want to disallow goto, add this line:
#define goto errorNo13    //Goto is not allowd

*/

#define system errorNo1      //"system" is not allowed
#define freopen errorNo2     //File operation is not allowed
#define fopen errorNo3       //File operation is not allowed
#define fprintf errorNo4     //File operation is not allowed
#define fscanf errorNo5      //File operation is not allowed
#define feof errorNo6        //File operation is not allowed
#define fclose errorNo7      //File operation is not allowed
#define ifstream errorNo8    //File operation is not allowed
#define ofstream errorNo9    //File operation is not allowed
#define fork errorNo10       //Fork is not allowed
#define clone errorNo11      //Clone is not allowed
#define sleep errorNo12      //Sleep is not allowed
```

Pada akhir baris *file* `defc.h` dan `defcpp.h` harus terdapat baris baru. Terdapat banyak aturan yang tidak dapat digunakan pada *g++*, seperti aturan `#define fopen errorNo3` untuk bahasa *C++*.

Shield untuk *Python*

Penggunaan *shield* untuk *python* dapat dihidupkan melalui halaman *settings*. Dengan menghidupkan *shield* untuk *python*, *SharIF Judge* hanya menambahkan beberapa kode pada baris awal kode yang sudah dikumpulkan sebelum dijalankan. Penambahan kode digunakan untuk mencegah pemakaian fungsi berbahaya. Kode-kode tersebut terletak pada *file tester/shield/shield_py2.py* dan *tester/shield/shield_py3.py*. Berikut merupakan cara untuk keluar dari *shield* untuk *python* menggunakan fungsi yang telah di daftar hitamkan:

Kode 19: Cara keluar dari *shield* untuk *python*

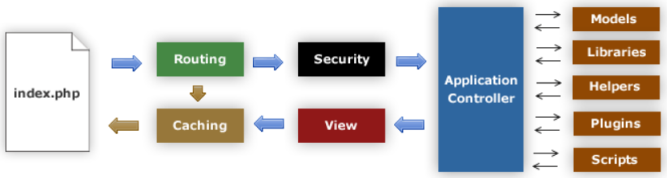
```

# @file shield_py3.py

import sys
sys.modules['os']=None

BLACKLIST = [
  #'__import__', # deny importing modules
  'eval', # eval is evil
  'open',
  'file',
  'exec',
  'execfile',
  'compile',
  'reload',
  #'input'
  ]
for func in BLACKLIST:
  if func in __builtins__.__dict__:
    del __builtins__.__dict__[func]
```

3. **Melakukan studi literatur mengenai *CodeIgniter 3*[9]**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *CodeIgniter 3* sudah dipelajari cara kerja dan dokumentasinya.
   CodeIgniter 3 merupakan sebuah *framework opensource* yang berfungsi untuk mempermudah pengguna dalam membentuk aplikasi *website* menggunakan bahasa PHP. *CodeIgniter 3* memiliki tujuan untuk membantu pengguna dalam membentuk aplikasi web lebih cepat dengan menyediakan beragam *library* dan tampilan dan *logic* yang simpel. *CodeIgniter 3* juga merupakan *framework* ringan yang menggunakan struktur *Model-View-Controller*, dan menghasilkan *URLs* yang bersih. *Code Igniter 3* memiliki *flow chart* aplikasi yang dapat dilihat pada gambar 3.



Gambar 3: *Flow Chart* Aplikasi *CodeIgniter 3*

Berikut merupakan pembagian *flow chart* aplikasi *CodeIgniter 3*:

(a) `index.php` berfungsi sebagai *front controller* yang berguna untuk melakukan inisiasi

(b) *Router* berfungsi dalam melakukan pemeriksaan dan menentukan penggunaan *HTTP Request*.

(c) *Cache* berfungsi untuk mengirimkan *file cache*(apabila ada) kepada *browser* secara langsung.

(d) *Security* berfungsi sebagai alat penyaringan setiap data dan *HTTP Request* yang masuk. Penyaringan data tersebut dilakukan sebelum *controller* aplikasi dimuat agar aplikasi menjadi lebih aman.

(e) *Controller* berguna sebagai alat untuk memuat *model, libraries*, dan sumber daya yang dibutuhkan untuk menjalankan permintaan spesifik.

(f) *View* akan dikirimkan menuju *browser* untuk dilihat oleh pengguna. Apabila *caching* dinyalakan, maka *view* akan dilakukan *cached* terlebih dahulu sehingga permintaan selanjutnya dapat diberikan.

Model-View-Controller

CodeIgniter 3 merupakan *framework* berbasis arsitektur *Model-View-Controller* atau yang selanjutnya akan disebut sebagai MVC. MVC merupakan sebuah pendekatan perangkat lunak yang memisahkan antara logika dengan presentasi atau tampilannya. Penggunaan struktur ini mengurangi penggunaan skrip pada halaman web karena tampilan terpisah dengan skrip PHP. Berikut merupakan penjelasan mengenai struktur MVC:

Model

Model berfungsi dalam mewakili struktur data perangkat lunak. *Model* berfungsi dalam mengambil, memasukan, dan memperbarui data pada *database*. Berikut merupakan contoh *file Model CodeIgniter 3* pada direktori `application/models/`:

Kode 20: Contoh *model* pada *CodeIgniter 3*

```
1 class Blog_model extends CI_Model {
2
3         public $title;
4         public $content;
5         public $date;
```

[9]Dokumentasi *CodeIgniter 3* `https://codeigniter.com/userguide3/`(19 Maret 2023)

```
6
7         public function get_last_ten_entries()
8         {
9                 $query = $this->db->get('entries', 10);
10                return $query->result();
11        }
12
13        public function insert_entry()
14        {
15                $this->title    = $_POST['title']; // please read the below note
16                $this->content  = $_POST['content'];
17                $this->date     = time();
18
19                $this->db->insert('entries', $this);
20        }
21
22        public function update_entry()
23        {
24                $this->title    = $_POST['title'];
25                $this->content  = $_POST['content'];
26                $this->date     = time();
27
28                $this->db->update('entries', $this, array('id' => $_POST['id']));
29        }
30
31}
```

Kode 20 membentuk beberapa fungsi yaitu:

- `get_last_ten_entries()` yang berfungsi untuk mengambil 10 data terakhir dari tabel `entries` menggunakan *query builder*.
- `insert_entry()` yang berfungsi untuk memasukan data *title,content*,dan *date* menuju tabel `entries`.
- `update_entry()` yang befungsi untuk memperbaharui data *title,content*,dan *date* pada tabel `entries` .

Model biasanya digunakan pada *file controller* dan dapat dipanggil menggunakan:

```
$this->load->model('model_name');.
```

View

View berfungsi dalam menyajikan informasi kepada pengguna. *View* biasanya merupakan halaman web namun, pada *CodeIgniter 3 view* dapat berupa pecahan halaman seperti *header* atau *footer*. Pecahan halaman dapat dimasukan pada halaman lain agar mempermudah dan membentuk kode yang lebih bersih.

Kode 21: Contoh *view* pada *CodeIgniter 3*

```
1 <?php
2 <html>
3 <head>
4         <title>My Blog</title>
5 </head>
6 <body>
7         <h1>Welcome to my Blog!</h1>
8 </body>
9 </html>
```

Kode 21 merupakan contoh *file view CodeIgniter 3* pada direktori `application/views/` yang berisikan judul My Blog dan *heading* Welcome to my Blog!. Pengguna dapat memanggil halaman yang sudah dibentuk pada *file controller* dengan cara sebagai berikut:

```
$this->load->view('name');
```

Controller

Controller berfungsi sebagai perantara antara *Model*, *View*, dan sumber daya yang dibutuhkan untuk melakukan proses *HTTP Request* dan menjalankan halaman web. Penamaan *controller* biasanya digunakan sebagai *url* pada perangkat lunak pengguna. Berikut merupakan contoh *controller CodeIgniter 3* pada direktori `application/controllers/`:

Kode 22: Contoh *controller* pada *CodeIgniter 3*

```
<?php
class Blog extends CI_Controller {

        public function index()
        {
                echo 'Hello World!';
        }

        public function comments()
        {
                echo 'Look at this!';
        }
}
```

Kode 22 berfungsi dalam mengembalikan *string* sesuai dengan *controller* yang dipanggil .Nama *controller* dan metode diatas akan dijadikan segmen pada *URL* seperti berikut:

`example.com/index.php/blog/index/`

Metode *index* akan secara otomatis dipanggil menjadi *URL* dan pengguna juga dapat memberi parameter untuk metode *controller* yang nantinya akan menjadi *URL*.

CodeIgniter URLs

CodeIgniter 3 menggunakan pendekatan *segment-based* dibandingkan menggunakan *query string* untuk membentuk *URL* yang mempermudah mesin pencari dan pengguna. Berikut merupakan contoh *URL* pada *CodeIgniter 3*:

`example.com/news/article/my_article`

Struktur segmen pada MVC menghasilkan *URL* sebagai berikut :

`example.com/class/function/ID`

Segmen tersebut dibagi menjadi tiga buah yakni:

(a) Segmen pertama mereprentasikan kelas *controller* yang dipanggil.

(b) Segmen kedua mereprentasikan kelas fungsi atau metode yang digunakan.

(c) Segmen ketiga dan segmen lainnya mereprentasikan *ID* dari variabel yang akan dipindahkan menuju *controller*.

Secara asali *URL* yang dihasilkan *CodeIgniter 3* terdapat nama *file* `index.php` seperti contoh dibawah ini:

`example.com/index.php/news/article/my_article`

Pengguna dapat menghapus *file* `index.php` file pada *url* menggunakan *file* `.htaccess` apabila *server Apache* pengguna menghidupkan `mod_rewrite`. Berikut merupakan contoh *file* `.htaccess` menggunakan metode *negative*:

Kode 23: Contoh *file* `.htaccess` pada halaman index.php

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

Aturan diatas menyebabkan *HTTP Request* selain yang berasal dari direktori atau *file* diperlakukan sebagai sebuah permintaan pada file `index.php`. Selain itu, pengguna juga dapat menambahkan akhirkan pada *URL* agar halaman pengguna dapat menampilkan halaman sesuai dengan tipe yang diingikan. Berikut merupakan contoh *URL* sebelum dan sesudah ditambahkan akhiran berupa:

```
example.com/index.php/products/view/shoes
```

```
example.com/index.php/products/view/shoes.html
```

Pengguna juga dapat menyalakan fitur *query strings* dengan cara sebagai mengubah *file application/config.php* seperti:

Kode 24: *File application/config.php*

```
$config['enable_query_strings'] = FALSE;
$config['controller_trigger'] = 'c';
$config['function_trigger'] = 'm';
```

Pengguna dapat mengubah `enable_query_strings` menjadi *TRUE*. *Controller* dan *function* dapat diakses menggunakan kata yang sudah diatur.

Helpers

Helpers merupakan fungsi pada *CodeIgniter 3* yang mempermudah pengguna dalam membentuk aplikasi web. Setipa *file helpers* terdiri dari banyak fungsi yang membantu sesuai kategori dan tidak ditulis dalanm format *Object Oriented*. *File helpers* terdapat pada direktori *system/helpers* atau *application/helpers*. Pengguna dapat memakai fitur *helpers* dengan cara memuatnya seperti berikut:

```
$this->load->helper('name');
```

Pemanggilan *helper* tidak menggunakan ekteksi .php melainkan hanya menggunakan nama dari *helper* tersebut. Pengguna dapat memanggil satu atau banyak *helper* pada metode *controller* ataupun *view* sesudah dimuat.

Libraries

CodeIgniter 3 menyediakan *library* yang dapat dipakai pengguna untuk mempermudah pembentukan aplikasi web. *Library* merupakan kelas yang tersedia pada direktori *application/libraries* dan dapat ditambahkan, diperluas, dan digantikan.

Kode 25: Contoh kelas *library* pada *CodeIgniter 3*

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Someclass {

        public function some_method()
        {
        }
}
```

Kode 25 merupakan contoh *file library* pada *CodeIgniter 3*. Setiap pembentukan *file library* diperlukan huruf kapital dan harus sama dengan nama kelasnya. Berikut merupakan contoh pemanggilan *file library* pada *file controller*:

```
$params = array('type' => 'large', 'color' => 'red');
```

```
$this->load->library('someclass', $params);
```

Pemanggilan kelas ini dapat dilakukan melalui *controller* manapun dan dapat diberikan parameter sesuai dengan metode yang dibentuk pada *library*. *CodeIgniter 3* menyediakan berbagai *library* yang dapat digunakan oleh pengguna seperti berikut:

JavaScript

Penggunaan kelas *javascript* dapat dipanggil pada konstruktor *controller* dengan cara berikut:

```
$this->load->library('javascript');
```

Pengguna selanjutnya harus melakukan inisiasi *library* pada halaman *view tag* `<head>` seperti berikut:

```
<?php echo $library_src;?>
<?php echo $script_head;?>
```

Sintaks `$library_src` merupakan lokasi *library* yang akan dimuat sedangkan `$script_head` merupakan lokasi untuk fungsi yang akan dijalankan. Selanjutnya *javascript* dapat diinisiasikan pada *controller* menggunakan sintaks berikut:

```
$this->javascript
```

Selain menggunakan *javascript*, pengguna dapat memakai *jQuery* dengan menambahkan *jQuery* pada akhir inisiasi kelas *javascript* seperti berikut:

```
$this->load->library('javascript/jquery');
```

Penggunga dapat memakai berbagai fungsi *library jquery* menggunakan sintaks berikut:

```
$this->jquery
```

Kelas *Email*

CodeIgniter 3 menyediakan kelas *email* dengan fitur sebagai berikut:

- Beberapa Protokol: *Mail, Sendmail*, dan *SMTP*
- Enkripsi *TLS* dan *SSL* untuk *SMTP*
- Beberapa Penerima
- *CC* dan *BCCs*
- *HTML* atau *email* teks biasa
- Lampiran
- Pembungkus kata
- Prioritas
- Mode*BCC Batch*, memisahkan daftar *email* besar menjadi beberapa *BCC* kecil.
- Alat *Debugging email*

Penggunaan *library email* dapat dikonfigurasikan pada *file config*. Pengguna dapat mengirim *email* dengan mudah menggunakan fungsi-fungsi yang telah disediakan *library email*. Kode 26 merupakan contoh pengiriman email melalui *controller*.

Kode 26: Contoh pengiriman email melalui *controller*

```
$this->load->library('email');

$this->email->from('your@example.com', 'Your Name');
$this->email->to('someone@example.com');
$this->email->cc('another@another-example.com');
$this->email->bcc('them@their-example.com');

$this->email->subject('Email Test');
$this->email->message('Testing the email class.');

$this->email->send();
```

Kode 26 merupakan contoh penggunaan *library email* untuk mengirim *email* dari `your@example.com` menuju `someone@example.com`. Konfigurasi ini juga mengirim dua buah salinan menuju `another@another-example.com` dan `them@their-example.com` dengan subjek berupa `Email Test` dengan pesan `Testing the email class.`. Selain itu, pengguna juga dapat melakukan konfigurasi preferensi *email* melalui dua puluh satu preferesi. Pengguna dapat melakukan konfigurasi secara otomatis melalui *file config* atau melakukan konfigurasi secara manual . Kode 27 merupakan contoh konfigurasi preferensi secara manual.

Kode 27: Contoh konfigurasi preferensi *library email* secara manual

```
$config['protocol'] = 'sendmail';
$config['mailpath'] = '/usr/sbin/sendmail';
$config['charset'] = 'iso-8859-1';
$config['wordwrap'] = TRUE;

$this->email->initialize($config);
```

Kode 27 merupakan contoh konfigurasi pengiriman *email* dengan protokol `sendmail` dan tujuan `usr/sbin/sendmail`.

Kelas *File Uploading*

Pengunggahan *file* terdapat empat buah proses sebagai berikut:

(a) Dibentuk sebuah form untuk pengguna memilih dan mengunggah *file*.

(b) Setelah *file* diunggah, *file* akan dipindahkan menuju direktori yang dipilih.

(c) Pada pengiriman dan pemindahan *file* dilakukan validasi sesuai dengan ketentuan yang ada.

(d) Setelah *file* diterima akan dikeluarkan pesan berhasil.

Perangkat lunak akan memindahkan *file* yang sudah diunggah pada *form* menuju *controller* untuk dilakukan validasi dan penyimpanan.

Kode 28: Contoh *controller* untuk melakukan validasi dan penyimpanan

```
<?php

class Upload extends CI_Controller {

        public function __construct()
        {
                parent::__construct();
                $this->load->helper(array('form', 'url'));
        }

        public function index()
        {
                $this->load->view('upload_form', array('error' => ' ' ));
        }

        public function do_upload()
        {
                $config['upload_path']          = './uploads/';
                $config['allowed_types']        = 'gif|jpg|png';
                $config['max_size']             = 100;
                $config['max_width']            = 1024;
                $config['max_height']           = 768;

                $this->load->library('upload', $config);

                if ( ! $this->upload->do_upload('userfile'))
                {
                        $error = array('error' => $this->upload->display_errors());

                        $this->load->view('upload_form', $error);
                }
                else
                {
                        $data = array('upload_data' => $this->upload->data());

                        $this->load->view('upload_success', $data);
                }
        }
}
?>
```

Kode 28 merupakan contoh kode dengan dua buah metode yakni:

(a) `index()` yang digunakan untuk mengembalikan *view* bernama `upload_form`

(b) `do_upload` yang digunakan untuk melakukan validasi berupa tipe, ukuran, lebar, dan panjang maksimal sebuah file. Metode ini juga mengembalikan *error* dan menyimpan *file* sesuai dengan direktori yang sudah diatur.

Direktori penyimpanan dapat diubah sesuai dengan kebutuhan namun perlu pengubahan izin direktori menjadi 777 agar dapat di baca, di tulis, dan di eksekusi oleh seluruh pengguna.

Kelas *Zip Encoding*

Zip merupakan format sebuah *file* yang berguna untuk melakukan kompress terhadap *file* untuk mengurangi ukuran dan menjadikannya sebuah *file*. *CodeIgniter 3* menyediakan *library Zip Encoding* yang dapat digunakan untuk membentuk arsip *Zip* yang dapat diunduh menuju *desktop* atau disimpan pada direktori. *Library* ini dapat diinsiasi dengan kode sebagai berikut:

```
$this->load->library('zip');
```

Setelah diinisiasi, pengguna dapat memanggil *library* tersebut menggunakan kode sebagai berikut:

```
$this->zip
```

Kode 29 merupakan contoh penggunaan *library Zip Encoding* untuk meyimpan dan menunduh data.

Kode 29: Contoh penggunaan *library Zip Encoding*

```
$name = 'mydata1.txt';
$data = 'A Data String!';

$this->zip->add_data($name, $data);

// Write the zip file to a folder on your server. Name it "my_backup.zip"
$this->zip->archive('/path/to/directory/my_backup.zip');

// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

Kode 29 merupakan contoh untuk melakukan penyimpanan *Zip file* pada direktori dan dapat mengunduh *file* menuju *desktop* pengguna. Selain menggunakan *library* yang sudah disediakan, pengguna dapat membentuk dan memperluas *libraries* sendiri sesuai dengan kebutuhan. Kode merupakan contoh *library* yang dibentuk.

Kode 30: Contoh *library* yang dibentuk

```
class Example_library {

        protected $CI;

        // We'll use a constructor, as you can't directly call a function
        // from a property definition.
        public function __construct()
        {
                // Assign the CodeIgniter super-object
                $this->CI =& get_instance();
        }

        public function foo()
        {
                $this->CI->load->helper('url');
                redirect();
        }
}
```

Library diatas merupakan contoh *library* yang dibentuk oleh pengguna digunakan untuk memanggil *helper* bernama *url*. Pengguna dapat memanggil kelas tersebut seperti memanggil kelas *library* lainnya. Selain itu, pengguna juga dapat mengganti *library* yang sudah ada dengan *library* yang dibentuk pengguna dengan mengubah nama kelas sama persis dengan nama *library* yang ingin digantikan.

Database

CodeIgniter 3 memiliki konfigurasi *database* yang menyimpan data-data terkait aturan *database*.

Kode 31: Contoh konfigurasi *database*

```
$db['default'] = array(
        'dsn'   => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'database_name',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => TRUE,
        'db_debug' => TRUE,
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'failover' => array()
);
```

Kode 31 merupakan contoh konfigurasi pada file *database* untuk *database* bernama `database_name` dengan *username* `root` tanpa sebuah *password*. *CodeIgniter 3* menyediakan fitur *query* untuk menyimpan, memasukan, memperbarui, dan menghapus data pada *database* sesuai dengan konfigurasi *database* yang sudah diatur. Kode 32 merupakan contoh *query* untuk melakukan *select* dan *join* pada *CodeIgniter 3*:

Kode 32: Contoh penggunaan *query*

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comments', 'comments.id = blogs.id');
$query = $this->db->get();
```

Kode 32 merupakan contoh kode untuk melakukan *query* pada tabel `blogs` yang melakukan *join* dengan tabel `comments`. Pengguna dapat mengambil hasil dari *query* menjadi *object* atau *array*. Selain itu, *database* pada *CodeIgniter 3* juga dapat digunakan untuk membentuk, menghapus, dan mengubah *database* ataupun menambahkan kolom pada *table*. Penggunaan *database* untuk mebentuk, menghapus, atau mengubah *database* harus dilakukan inisasi sebagai berikut:

```
$this->load->dbforge()
```

Setelah dilakukan inisiasi pengguna dapat membentuk *database* menggunakan kelas *Forge*. Kode 33 merupakan contoh untuk membentuk *database*.

Kode 33: Contoh membentuk *database* menggunakan *CodeIgniter3*

```
    $this->dbforge->create_database('db_name')
```

Selain itu, pengguna juga dapat menambahkan kolom dengan konfigurasinya. Kode 34 merupakan contoh penambahan kolom sesuai dengan kebutuhan pengguna.

Kode 34: Contoh menambahkan kolom dengan konfigurasinya menggunakan *CodeIgniter3*

```
    $fields = array(
        'blog_id' => array(
                'type' => 'INT',
                'constraint' => 5,
                'unsigned' => TRUE,
                'auto_increment' => TRUE
        ),
        'blog_title' => array(
                'type' => 'VARCHAR',
                'constraint' => '100',
                'unique' => TRUE,
        ),
        'blog_author' => array(
                'type' =>'VARCHAR',
                'constraint' => '100',
                'default' => 'King of Town',
        ),
        'blog_description' => array(
                'type' => 'TEXT',
```

```
                'null' => TRUE,
        ),
);
$this->dbforge->add_field($fields)
$this->dbforge->create_table('table_name');
```

Kode 34 merupakan contoh penggunaan *database* untuk menambahkan kolom sesuai dengan tipenya pada tabel `table_name`.

URI Routing

URL string biasanya menggunakan nama atau metode *controller* seperti pada berikut:

```
example.com/class/function/id/
```

Namun, pengguna dapat melakukan pemetaan ulang terhadap *url* yang dibentuk agar dapat memanggil beberapa metode.

Kode 35: Contoh *url* yang sudah dimetakan

```
example.com/product/1/
example.com/product/2/
example.com/product/3/
example.com/product/4/
```

Kode 35 merupakan contoh *url* yang sudah dimetakan ulang. Pengguna dapat menambahkan kode pemetaan pada *file application/config/routes.php* yang terdapat *array* bernama `$route`. Berikut merupakan beberapa cara melakukan pemetaan terhadap *url*:

WildCards

Route wildcard biasanya berisikan kode seperti berikut:

```
$route['product/:num'] = 'catalog/product_lookup';
```

Route diatas dibagi menjadi dua buah yakni:

(a) Bagian segmen *URL*

Bagian pertama merupakan segmen pertama *url* yang akan tampil pada *url*. Bagian kedua merupakan segmen kedua dapat berisikan angka atau karakter.

(b) Bagian kelas dan metode

Bagian kedua berisikan kelas dan metode dari *controller* yang akan digunakan pada *url*.

Ekspresi Reguler

Pengguna dapat memakai ekspresi reguler untuk melakukan pemetaan ulang *route*. Berikut merupakan contoh ekspresi reguler yang biasa digunakan:

```
$route['products/([a-z]+)/(\d+)'] = '$1/id_$2';
```

Ekpresi ini menghasilkan *URI products/shirts/123* yang memanggil kelas *controller* dan metode *id_ 123*. Pengguna juga dapat mengambil segmen banyak seperti berikut:

```
$route['login/(.+)'] = 'auth/login/$1';
```

Auto-loading

CodeIgniter 3 menyediakan sebuah fungsi untuk memuat berbagai kelas seperti *libraries, helpers,* dan *models*. Kelas dapat dimasukan pada `application/config/autoload.php` sesuai dengan petunjuk yang ada. *File autoload* akan di inisiasikan setiap aplikasi dijalankan sehingga pengguna tidak perlu memuat kelas tersebut berulang kali.

4. **Melakukan studi literatur mengenai *CodeIgniter 4*[10]**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : *CodeIgniter 4* sudah dipelajari cara kerja dan membentuknya.
   CodeIgniter 4 merupakan versi terbaru dari *framework CodeIgniter* yang berfungsi untuk membantu pembentukan web. *CodeIgniter 4* dapat dipasang menggunakan *composer* ataupun dipasang secara manual dengan mengunduhnya dari situs web resmi. Setelah dilakukan pemasangan *CodeIgniter 4* memiliki lima buah direktori dengan struktur aplikasi sebagai berikut:

   - `app/`

     Direktori *app* berisikan semua kode yang dibutuhkan untuk menjalankan aplikasi web yang dibentuk. Direktori ini memiliki beberapa direktori didalamnya yaitu:

     - `Config/` berfungsi dalam menyimpan *file* konfigurasi aplikasi web pengguna.
     - `Controllers/` berfungsi sebagai penentu alur program yang dibentuk.
     - `Database/` berfungsi sebagai penyimpan *file migrations* dan *seeds*.
     - `Filters/` befungsi dalam menyimpan *file* kelas *filter*.
     - `Helpers/` berfungsi dalam menyimpan koleksi fungsi mandiri.
     - `Language/` berfungsi sebagai tempat penyimpanan *string* dalam berbagai bahasa.
     - `Libraries/` berfungsi dalam penyimpan kelas yang tidak termasuk kategori lain.
     - `Models/` berfungsi untuk merepresentasikan data dari *database*.
     - `ThirdParty/` *library ThridParty* yang dapat digunakan pada aplikasi.
     - `Views/` berisikan *file HTML* yang akan ditampilkan kepada pengguna.

   - `public/`

     Direktori *public* merupakan *web root* dari situs dan berisikan data-data yang dapat diakses oleh pengguna melalui *browser*. Direktori ini berisikan *file* `.htaccess`, `index.php`, dan *assets* dari aplikasi web yang dibentuk seperti gambar, *CSS*, ataupun *JavaScript*.

   - `writable/`

     Direktori *writable* berisikan data-data yang mungkin perlu ditulis seperti *file cache*,*logs*, dan *uploads*. Pengguna dapat menambahkan direktori baru sesuai dengan kebutuhan sehingga menambahkan keamanan pada direktori utama.

   - `tests/`

     Direktori ini menyimpan *file* test dan tidak perlu dipindahkan ke *server* produksi. Direktori `_support` berisikan berbagai jenis kelas *mock* dan keperluan yang dapat dipakai pengguna dalam menulis *tests*.

   - `vendor/` atau `system/`

     Direktori ini berisikan *file* yang diperlukan dalam menjalani *framework* dan tidak boleh dimodifikasi oleh pengguna. Pengguna dapat melakukan *extend* atau membentuk kelas baru untuk menambahkan fungsi yang diperlukan.

Models-Views-Controllers

CodeIgniter 4 menggunakan pola *MVC* untuk mengatur *file* agar lebih simpel dalam menemukan *file* yang diperlukan. MVC menyimpan data, presentasi, dan alur program dalam *file* yang berbeda.

Models

Models berfungsi dalam menyimpan dan mengambil data dari tabel spesifik pada *database*. Data tersebut dapat berupa pengguna, pemberitahuan blog, transaksi, dll. *Models* pada umumnya disimpan

[10]Dokumentasi *CodeIgniter 4* `https://codeigniter.com/user_guide/`(19 Maret 2023)

pada direktori `app/Models` dan memiliki *namespace* sesuai dengan lokasi dari *file* tersebut. Kode 36 merupakan contoh dari *models*.

Kode 36: Contoh *Models*

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table      = 'users';
    protected $primaryKey = 'id';

    protected $useAutoIncrement = true;

    protected $returnType     = 'array';
    protected $useSoftDeletes = true;

    protected $allowedFields = ['name', 'email'];

    // Dates
    protected $useTimestamps = false;
    protected $dateFormat    = 'datetime';
    protected $createdField  = 'created_at';
    protected $updatedField  = 'updated_at';
    protected $deletedField  = 'deleted_at';

    // Validation
    protected $validationRules      = [];
    protected $validationMessages   = [];
    protected $skipValidation       = false;
    protected $cleanValidationRules = true;

}
```

Kode 36 merupakan contoh *Models* yang dapat digunakan pada *controllers*. *Models* tersebut terkoneksikan dengan tabel `users` dengan *primarykey* `id`. *Model* pada *CodeIgniter 4* juga dapat digunakan untuk mencari, menyimpan, dan menghapus data untuk setiap tabel spesifik. Kode 37 merupakan contoh penggunaan *model* untuk mencari data spesifik.

Kode 37: Contoh penggunaan *model* untuk mencari data spesifik

```
<?php

$users = $userModel->where('active', 1)->findAll();
```

Kode 37 menggabungkan *query* dengan metode pencarian *model* untuk mencari seluruh data `active`.

Views

Views merupakan halaman berisikan *HTML* dan sedikit PHP yang ditampilkan kepada pengguna ataupun dapat berupa pecahan halaman seperti *header, footer*, ataupun *sidebar*. *Views* biasanya terdapat pada `app/Views` dan mendapatkan data berupa variabel dari *controller* untuk ditampilkan.

Kode 38: Contoh *Views*

```
<html>
    <head>
        <title>My Blog</title>
    </head>
    <body>
        <h1>Welcome to my Blog!</h1>
    </body>
</html>
```

Kode 39 merupakan contoh *views* pada direktori `app/Views` yang akan menampilkan tulisan *Welcome to my Blog*. *View* ini dapat ditampilkan melalui controller seperti berikut:

Kode 39: Contoh *Views*

```
<?php

namespace App\Controllers;

```

```
use CodeIgniter\Controller;

class Blog extends Controller
{
    public function index()
    {
        return view('blog_view');
    }
}
```

Kode 39 merupakan contoh memanggil *views* pada *file controllers*. Kode ini mengembalikan halaman `blog_view` pada *controller* blog.

Controllers

Contollers merupakan kelas untuk mengambil atau memberikan data dari *models* menuju *views* untuk ditampilkan. Setiap pembentukan *controllers* dibentuk harus memperpanjang kelas *BaseController*. Kode 40 merupakan contoh *controllers* yang dibentuk.

Kode 40: Contoh *Controllers* pada *CodeIgniter 4*

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    public function index()
    {
        return 'Hello World!';
    }

    public function comment()
    {
        return 'I am not flat!';
    }
}
```

Kode 40 merupakan contoh *controllers* yang melakukan pengembalikan *Hello World* pada *url*:

`example.com/index.php/helloworld/`

Selain itu, *CodeIgniter 4* menyediakan fungsi bernama *Controller Filters* yang berfungsi untuk membiarkan pengguna membentuk sebuah kondisi sebelum ataupun sesudah *controller* dijalankan. Kode 41 merupakan contoh penggunaan *filters*.

Kode 41: Contoh *Controllers Filters* pada *CodeIgniter 4*

```
<?php

namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use Config\Database;

class MyFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $session = \Config\Services::session();
        $db = Database::connect();

        if ( !$db->tableExists('sessions'))
            return redirect()->to('/install');
        if ( !$session->get('logged_in')) // if not logged in
            return redirect()->to('/login');
    }

    public function after(RequestInterface $request, ResponseInterface $response, $arguments = null)
    {
        // Do something here
    }
}
```

Kode 41 merupakan contoh kode yang akan melakukan pengecekan tabel ataupun *session* sebelum *controller* dijalankan. Selanjutnya pengguna perlu menambahkan konfigurasi *filter* pada *routes* seperti sintaks berikut.

```
$routes->get('/', 'Dashboard::index',['filter' => 'check-install:dual,noreturn']);
```

Sintaks diatas akan melakukan pengecekan pada *controller* `Dashboard::index` sebelum dan setelah *controller* tersebut dijalankan.

CodeIgniter URLs

CodeIgniter 4 menggunakan pendekatan *segment-based* dibandingkan menggunakan *query-string* untuk menghasilkan *URL* sehingga ramah manusia dan mesin pencari. Berikut merupakan contoh *URL* yang dihasilkan *CodeIgniter 4*:

`https://www.example.com/ci-blog/blog/news/2022/10?page=2`

CodeIgniter 4 menghasilkan *URL* seperti diatas dengan membaginya menjadi:

- *Base URL* merupakan *URL* dasar dari aplikasi web yang dibentuk yaitu `https://www.example.com/ci-blog/`
- *URI Path* merupakan alamat yang dituju yaitu `/ci-blog/blog/news/2022/10`
- *Route* juga merupakan alamat yang dituju tanpa *URL* dasar yaitu `/blog/news/2022/10`
- *Query* merupakan hasil dari *query* yang ingin ditampilkan yaitu `page=2`

Secara asali *CodeIgniter 4* membentuk *URL* dengan `index.php` namun, pengguna dapat menghapus *file* `index.php` pada *URL* yang dibentuk. Pengguna dapat menghapus `index.php` sesuai dengan *server* yang digunakan. Berikut merupakan contoh dua buah *server* yang biasanya dipakai:

Apache Web Server

Pengguna dapat *URL* melalui *file* `.htaccess` dengan menyalakan ekstensi `mod_rewrite`. Kode 42 merupakan contoh *file* `.htaccess` untuk menghapus `index.php` pada *URL* yang dibentuk.

Kode 42: Contoh *file* `.htacess` pada *Apache Web Server*.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

File diatas memperlakukan semua *HTTP Request* selain dari direktori dan *file* yang ada sebagai permintaan.

NGINX

Pengguna dapat mengubah *URL* menggunakan `try_files` yang akan mencari *URI* dan mengirimkan permintaan pada *URL* yang ingin dihilangkan. Kode 43 merupakan contoh penggunaan `try_files` untuk menghapus `index.php` pada *URL*.

Kode 43: Contoh penggunaan `try-files`.

```
location / {
    try_files $uri $uri/ /index.php$is_args$args;
}
```

URI Routing

CodeIgniter 4 menyediakan dua buah *routing* yakni: ***Defined Route Routing*** Pengguna dapat mendefinisikan *route* secara manual untuk *URL* yang lebih fleksibel. Kode 44 merupakan contoh *route* yang didefinisikan secara manual.

Kode 44: Contoh *route* yang didefinisikan secara manual

```
<?php

$routes->get('product/(:num)', 'Catalog::productLookup');
```

Kode 44 merupakan contoh penggunaan *route* untuk menuju kelas `Catalog` dengan metode `productLookup`. Pengguna juga dapat memakai beberapa *HTTP verb* seperti *GET,POST,PUT, etc.* Selain menulis secara individu, pengguna dapat melakukan *grouping* pada route seperti Kode.

Kode 45: Contoh *route* yang menggunakan *grouping* manual

```php
<?php

$routes->group('admin', static function ($routes) {
    $routes->get('users', 'Admin\Users::index');
    $routes->get('blog', 'Admin\Blog::index');
});
```

Kode 45 merupakan contoh penggunaan *grouping* untuk *URI* `admin/users` dan `admin/blog`.

Auto Routing

Pengguna dapat mendefinisikan *route* secara otomatis melalui fitur *Auto Routing* apabila tidak terdapat *route*. Pengguna dapat menyalakan fitur ini pada `app/Config/Routes.php` dengan cara berikut:

```
$routes->setAutoRoute(true);
```

Pengguna juga perlu mengubah `$autoRoutesImproved` menjadi `true` pada direktori `app/Config/Feature.php`. Selain menggunakan *auto routing* baru, pengguna dapat menggunakan *Auto Routing (Legacy)* yang terdapat pada *CodeIgniter 3* dengan cara seperti berikut:

Databases

CodeIgniter 4 menyediakan kelas *database* yang dapat menyimpan, memasukan, memperbarui, dan menghapus data pada *database* sesuai dengan konfigurasi. Pengguna dapat melakukan konfigurasi untuk *database* yang ingin dikoneksikan melalui direktori `app/Config/Database.php` atau *file* `.env`. Kode 46 merupakan contoh pada direktori `Database.php`.

Kode 46: Contoh konfigurasi *database* pada *CodeIgniter 4*.

```php
<?php

namespace Config;

use CodeIgniter\Database\Config;

class Database extends Config
{
    public $default = [
        'DSN'      => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'database_name',
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => true,
        'DBDebug'  => true,
        'charset'  => 'utf8',
        'DBCollat' => 'utf8_general_ci',
        'swapPre'  => '',
        'encrypt'  => false,
        'compress' => false,
        'strictOn' => false,
        'failover' => [],
        'port'     => 3306,
    ];

    // ...
}
```

Kode 46 merupakan contoh konfigurasi untuk database bernama `database_name` dengan *username* root. Selain untuk melakukan koneksi *database*, kelas ini dapat digunakan untuk menambahkan, menghapus, dan memperbaharui data pada *database*. Berikut merupakan contoh penggunaan *query* pada *database*:

Kode 47: Contoh konfigurasi *database* pada *CodeIgniter 4*.

```php
<?php
```

```
2
3 $builder = $db->table('users');
4 $builder->select('title, content, date');
5 $builder->from('mytable');
6 $query = $builder->get();
```

Kode 47 merupakan contoh penggunaan *query* untuk mengambil data *title,content* ,dan *date* pada tabel *mytable*. *CodeIgniter 4* juga menyediakan fitur untuk membentuk *database* melalui fitur bernama *Database Forge*. Pengguna dapat membentuk,mengubah, menghapus tabel dan juga menambahkan *field* pada tabel tersebut. Kode 48 merupakan contoh pembentukan *database*.

Kode 48: Contoh pembentukan tabel melalui *database forge*.

```
1 <?php
2
3 $fields = [
4     'id' => [
5         'type'           => 'INT',
6         'constraint'     => 5,
7         'unsigned'       => true,
8         'auto_increment' => true,
9     ],
10    'title' => [
11        'type'       => 'VARCHAR',
12        'constraint' => '100',
13        'unique'     => true,
14    ],
15    'author' => [
16        'type'       => 'VARCHAR',
17        'constraint' => 100,
18        'default'    => 'King of Town',
19    ],
20    'description' => [
21        'type' => 'TEXT',
22        'null' => true,
23    ],
24    'status' => [
25        'type'       => 'ENUM',
26        'constraint' => ['publish', 'pending', 'draft'],
27        'default'    => 'pending',
28    ],
29 ];
30 $forge->addField($fields);
31 $forge->createTable('table_name');
```

Kode merupakan contoh pembentukan *database* dengan tabel bernama *table_name* yang berisikan beberapa *field*.

Library

CodeIgniter 4 menyediakan berbagai *library* untuk membantu pengguna dalam pembentukan aplikasi web. Berikut merupakan contoh *library* yang disediakan oleh *CodeIgniter 4*: **Kelas *Email*** *CodeIgniter* menyediakan kelas *email* dengan fitur sebagai berikut:

- Beberapa Protokol: *Mail*,*Sendmail*,dan *SMTP*
- Enkripsi *TLS* dan *SSL* untuk *SMTP*
- Beberapa Penerima
- *CC* dan *BCCs*
- *HTML* atau *email* teks biasa
- Lampiran
- Pembungkus kata
- Prioritas
- Mode*BCC Batch*, memisahkan daftar *email* besar menjadi beberapa *BCC* kecil.
- Alat *Debugging email*

Pengguna dapat melakukan konfigurasi pada *file* `app/Config/Email.php` untuk melakukan pengiriman *email*. Kode 49 merupakan contoh konfigurasi preferensi *email* secara manual.

Kode 49: Contoh kode untuk melakukan konfigurasi *email*.

```
<?php

$config['protocol'] = 'sendmail';
$config['mailPath'] = '/usr/sbin/sendmail';
$config['charset']  = 'iso-8859-1';
$config['wordWrap'] = true;

$email->initialize($config);
```

Selain itu, pengguna dapat melakukan pengiriman *email* sesuai dengan kebutuhan. Kode 50 merupakan contoh penggunaan kelas *email* untuk mengirim *email*.

Kode 50: Contoh kode untuk melakukan pengiriman *email*.

```
<?php

$email = \Config\Services::email();

$email->setFrom('your@example.com', 'Your Name');
$email->setTo('someone@example.com');
$email->setCC('another@another-example.com');
$email->setBCC('them@their-example.com');

$email->setSubject('Email Test');
$email->setMessage('Testing the email class.');

$email->send();
```

Kode 50 merupakan contoh penggunaan kelas *email* untuk mengirimim *email* dari `your@example.com` kepada `someone@example.com` dengan subjek `Email Test` dan pesan `Testing the email class.`

Working with Uploaded Files

Pengunggahan *file* terdapat empat buah proses sebagai berikut:

(a) Dibentuk sebuah form untuk pengguna memilih dan mengunggah *file*.

(b) Setelah *file* diunggah, *file* akan dipindahkan menuju direktori yang dipilih.

(c) Pada pengiriman dan pemindahan *file* dilakukan validasi sesuai dengan ketentuan yang ada.

(d) Setelah *file* diterima akan dikeluarkan pesan berhasil.

Perangkat lunak akan menerima *file* dari *views* yang nantinya akan dilakukan validasi pada *controller*. Kode 51 merupakan contoh *view* untuk melakukan pengunggahan *file*.

Kode 51: Contoh kode untuk melakukan pengunggahan *file*.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Upload Form</title>
</head>
<body>

<?php foreach ($errors as $error): ?>
    <li><?= esc($error) ?></li>
<?php endforeach ?>

<?= form_open_multipart('upload/upload') ?>
    <input type="file" name="userfile" size="20">
    <br><br>
    <input type="submit" value="upload">
</form>

</body>
</html>
```

Kode 51 merupakan contoh *file view* menggunakan *form helper* dan dapat memberitahu apabila terdapat *error*. Setelah dilakukan penerimaan *file*, perangkat lunak akan mengirimkan *file* kepada *controller* untuk dilakukan validasi dan penyimpanan. Kode merupakan contoh *controller* untuk melakukan validasi dan penyimpanan.

Kode 52: Contoh kode *controller* untuk melakukan validasi dan penyimpanan.

```
<?php
```

```
2
3  namespace App\Controllers;
4
5  use CodeIgniter\Files\File;
6
7  class Upload extends BaseController
8  {
9      protected $helpers = ['form'];
10
11     public function index()
12     {
13         return view('upload_form', ['errors' => []]);
14     }
15
16     public function upload()
17     {
18         $validationRule = [
19             'userfile' => [
20                 'label' => 'Image File',
21                 'rules' => [
22                     'uploaded[userfile]',
23                     'is_image[userfile]',
24                     'mime_in[userfile,image/jpg,image/jpeg,image/gif,image/png,image/webp]',
25                     'max_size[userfile,100]',
26                     'max_dims[userfile,1024,768]',
27                 ],
28             ],
29         ];
30         if (! $this->validate($validationRule)) {
31             $data = ['errors' => $this->validator->getErrors()];
32
33             return view('upload_form', $data);
34         }
35
36         $img = $this->request->getFile('userfile');
37
38         if (! $img->hasMoved()) {
39             $filepath = WRITEPATH . 'uploads/' . $img->store();
40
41             $data = ['uploaded_fileinfo' => new File($filepath)];
42
43             return view('upload_success', $data);
44         }
45
46         $data = ['errors' => 'The file has already been moved.'];
47
48         return view('upload_form', $data);
49     }
50 }
```

Kode 52 terdapat dua buah fungsi yaitu:

- `index()` yang mengembalikan *view* bernama `upload_form`
- `upload()` yang memberikan aturan untuk melakukan validasi dan melakukan penyimpanan pada direktori `uploads`.

Helpers

Helpers merupakan fungsi pada *CodeIgniter 4* yang menyediakan beberapa fungsi untuk pengguna dalam membentuk aplikasi web. *Helpers* dapat dimuat oleh pengguna seperti berikut:

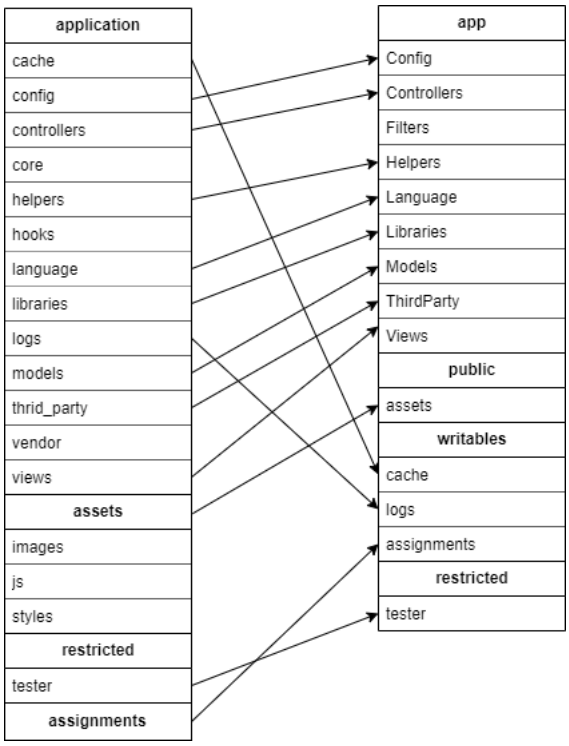
```
<?php

helper('helpers_name');
```

Setelah dilakukan pemanggilan, pengguna dapat memakai fungsi-fungsi yang disediakan sesuai dengan *helpers* yang digunakan.

5. **Melakukan studi literatur mengenai konversi *CodeIgniter 3* ke *CodeIgniter 4*** [11]
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Sudah dipelajari cara melakukan konversi dan melakukan konversi secara sederhana. Aplikasi akan dikonversi menuju *CodeIgniter 4* dengan struktur sebagai berikut.

[11] Dokumentasi *CodeIgniter 4* `https://codeigniter.com/user_guide/installation/upgrade_4xx.html`(22 Maret 2023)



Gambar 4: Pemindahan struktur aplikasi

Konversi CodeIgniter 3 ke CodeIgniter 4 diperlukan penulisan ulang karena terdapat banyak implementasi yang berbeda. Konversi ke CodeIgniter 4 diawali dengan melakukan instalasi projek baru CodeIgniter 4.

Struktur Aplikasi

Struktur direktori pada CodeIgniter 4 memiliki perubahan yang terdiri *app, public, dan writable.* Direktori app merupakan perubahan dari direktori application dengan isi yang hampir sama dengan beberapa perubahan nama dan perpindahan direktori. Pada CodeIgniter 4 terdapat direktori *public* yang bertujuan sebagai direktori utama pada aplikasi website. Selanjutnya terdapat direktori *writable* yang berisikan *cache data, logs,* dan *session data.*

Routing

CodeIgniter 4 meletakan *route* pada *file* `app\Config\Routes.php`. CodeIgniter 4 memiliki fitur *auto routing* seperti pada CodeIgniter 3 namun, pada *default* di matikan. Fitur *auto routing* memungkinkan untuk dinyalakan serupa dengan pada CodeIgniter 3 namun tidak direkomendasikan karena alasan *security.*

Model, View, and Controller

Struktur MVC pada CodeIgniter 4 berbeda dibandingkan CodeIgniter 3 dimana terdapat perbedaan penyimpanan direktori untuk ketiga *file* tersebut. Berikut merupakan penjelasan mengenai struktur MVC:

Model

Model pada *CodeIgniter 4* terdapat pada direktori `app\Models`. Pembentukan *file* untuk *Model* perlu ditambahkan `namespace App\Models;` dan `use CodeIgniter\Model;` pada awal *file* setelah membuka tag PHP. Selanjutnya nama fungsi perlu diubah dari `extends CI_Model` menjadi `extends Model`. *Model* dapat dilakukan pembaharuan melalui cara berikut:

(a) Pertama pengguna harus memindahkan seluruh *file model* menuju direktori `app/Models`

(b) Selanjutnya pengguna harus menambahkan `namespace App\Models;` setelah pembukaan *tag PHP*.

(c) Pengguna juga harus menambahkan `use CodeIgniter\Model;` setelah kode diatas.

(d) Pengguna harus mengganti `extends CI_Model` menjadi `extends Model`.

(e) Terakhir pemanggilan *model* berubah dari sintaks:

```
$this->load->model('x');
```

menjadi sintaks berikut:

```
$this->x = new X();.
```

View

View pada CodeIgniter 4 terdapat di `app\Views` dengan sintaks yang harus diubah. Sintaks yang harus diubah merupakan sintaks untuk memanggil *view* pada CodeIgniter 3 `$this->load->view('x');` sedangkan pada CodeIgniter 4 dapat menggunakan `return view('x);`. Selanjutnya, sintaks `<?php echo $title?>` pada halaman *view* dapat diubah menjadi `<?= $title ?>`. Berikut merupakan cara melakukan pembaharuan *view*:

(a) Pertama pengguna perlu memindahkan seluruh *file views* menuju `app/Views`

(b) Selanjutnya pengguna perlu mengubah sintaks:

```
$this->load->view('directory_name/file_name')
```

menjadi sintaks berikut:

```
return view('directory_name/file_name');
```

(c) Pengguna juga perlu mengubah sintaks:

```
$content = $this->load->view('file', $data, TRUE);
```

menjadi sintaks berikut:

```
$content = view('file', $data);.
```

(d) Pada *file views* pengguna dapat mengubah sintaks:

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

menjadi sintaks berikut:

```
<?= $title ?>.
```

(e) Pengguna juga perlu menghapus apabila terdapat sintaks `defined('BASEPATH') OR exit('No direct script access allowed');`.

Controller

Controller pada CodeIgniter 4 terdapat di `app\Controllers` dan diperlukan beberapa perubahan. Pertama, perlu ditambahkan `namespace App\Controllers;` pada awal *file* setelah membuka tag PHP. Selanjutnya, perlu mengubah `extends CI_Controller` menjadi `extends BaseController`. Selanjutnya, diperlukan pengubahan nama pada pemanggilan *file* menjadi `App\Controllers\User.php`. Pengguna dapat melakukan pembaharuan *controller* menggunakan cara berikut:

(a) Pertama pengguna harus memindahkan seluruh *file controller* menuju `app/Controllers`.

(b) Pengguna juga harus menambahkan sintaks `namespace App\Controllers;` setelah pembukaan *tag PHP*.

(c) Selanjutnya pengguna harus mengubah `extends CI_Controller` menjadi `extends BaseController`.

(d) Pengguna juga harus menghapus baris `defined('BASEPATH') OR exit('No direct script access allowed');` apabila ada.

Libraries

CodeIgniter 4 menyediakan *library* untuk digunakan dan dapat diinstall apabila diperlukan. Pemanggilan *library* berubah dari `$this->load->library('x');` menjadi `$this->x = new X();`. Terdapat beberapa *library* yang harus di perbaharui dengan sedikit perubahan. Berikut merupakan beberapa *libraries* yang terdapat pembaharuan: ***Configuration***

File configuration CodeIgniter 4 terdapat pada `app\Config` dengan penulisan sedikit berbeda dengan versi sebelumnya. Pengguna hanya perlu melakukan pemindahan menuju CodeIgniter 4 dan apabila menggunakan *file config custom* maka, diperlukan penulisan ulang pada direktori Config dengan melakukan *extend* pada `CodeIgniter\Config\BaseConfig`.

Database

Penggunaan *database* pada CodeIgniter 4 hanya berubah sedikit dibandingkan dengan versi sebelumnya. Data-data penting kredensial diletakan pada `app\Config\Database.php` dan perlu dilakukan beberapa perubahan sintaks dan *query*. Sintaks untuk memuat database diubah menjadi `$db = db_connect();` dan `$db = db_connect('group_name');` apabila menggunakan beberapa *database*. Semua *query* harus diubah dari `$this->db` menjadi `$db` dan beberapa sintaks perlu diubah seperti `$query->result();` menjadi `$query->getResult();`. Selain itu, terdapat *class* baru yakni *Query Builder Class* yang harus di inisiasi `$builder = $db->table('mytable');` dan dapat dipakai untuk menjalankan *query* dengan mengganti `$this->db` seperti `$this->db->get();` menjadi `$builder->get();`.

Emails

Perubahan *email* hanya terdapat pada nama dari *method* dan pemanggilan *library email*. Pemanggilan *library* berubah dari `$this->load->library('email);` menjadi `$email = service('email');` dan selanjutnya perlu dilakukan perubahan pada semua `$this->email` menjadi `$email`. Selanjutnya beberapa pemanggilan *method* berubah dengan tambahan *set* didepannya seperti *from* menjadi *setFrom*.

Working with Uploaded Files

Terdapat banyak perubahan dimana pada CodeIgniter 4 pengguna dapat mengecek apakah *file* telah terunggah tanpa *error* dan lebih mudah untuk melakukan penyimpanan *file*. Pada CodeIgniter 4 melakukan akses pada *uploaded file* dilakukan dengan sintaks berikut:

```
$file = $this->request->getFile('userfile')
```

selanjutnya dapat dilakukan validasi dengan cara sebagai berikut:

```
$file->isValid()
```

Penyimpanan *file* yang sudah diunggah dapat dilakukan dengan sintaks berikut.

```
$path = $this->request->getFile('userfile')->store('head_img/', 'user_name.jpg');.
```

File yang sudah diunggah dan di validasi akan tersimpan pada `writable/uploads/head_img/user_name.jpg`.

HTML Tables

Tidak terdapat banyak perubahan pada *framework* versi terbaru hanya perubahan pada nama *method* dan pemanggilan *library*. Perubahan pemanggilan *library* dari `$this->load->library('table');`

menjadi `$table = new \CodeIgniter\View\Table();` dan perlu dilakukan perubahan setiap `$this->table` menjadi `$table`. Selain itu, terdapat bebera perubahan pada penamaan *method* dari *underscored* menjadi *camelCase*.

Localization

CodeIgniter 4 mengembalikan *file* bahasa menjadi *array* sehingga perlu dilakukan beberapa perubahan. Pertama, perlu dilakukan konfigurasi *default language* pada perangkat lunak. Selanjutnya melakukan pemindahan *file* bahasa pada *CodeIgniter 3* menuju `app\Language\<locale>`. *File-file* bahasa *CodeIgniter 3* perlu dilakukan penghapusan semua kode `$this->lang->load($file, $lang);` dan mengubah *method* pemanggilan bahasa dari `$this->lang->line('error_email_missing')` menjadi `echo lang('Errors.errorEmailMissing');`

Migrations

Perubahan perlu dilakukan pada nama *file* menjadi nama dengan cap waktu. Selanjutnya dilakukan penghapusan kode `defined('BASEPATH') OR exit('No direct script access allowed');` dan menambahkan dua buah kode setelah membuka tag PHP yaitu:

- `namespace App\Database\Migrations;`
- `use CodeIgniter/Database/Migration;`

Setelah itu, `extends CI_Migration` diubah menjadi `extends Migration`. Terakhir, terdapat perubahan pada nama *method Forge* dari `$this->dbforge->add_field` menjadi `$this->forge->addField` dalam bentuk *camelcase*.

Routing

Pengguna dapat melakukan pembaharuan *routing* dengan cara berikut:

(a) Pengguna dapat memakai *Auto Routing 4* seperti pada *CodeIgniter 3* dengan menyalakan *Auto Routing(Legacy)*.

(b) Terdapat perubahan dari `(:any)` menjadi `(:segment)`.

(c) Pengguna juga harus mengubah sintaks pada `app/Config/Routes.php` seperti berikut:

- `$route['journals'] = 'blogs';` menjadi `$routes->add('journals', 'Blogs::index');` untuk memanggil fungsi `index` pada *controller* `Blogs`.
- `$route['product/(:any)'] = 'catalog/product_lookup'` menjadi

```
$routes->add('product/(:segment)', 'Catalog::productLookup');.
```

Validations

Pengguna dapat melakukan pembaharuan pada *validations* melalui cara berikut:

(a) Pengguna harus mengubah kode pada *view* dari `<?php echo validation_errors(); ?>` menjadi `<?= validation_list_errors() ?>`

(b) Pengguna perlu mengubah beberapa kode pada *controller* seperti berikut:

- `$this->load->helper(array('form', 'url'));` menjadi `helper(['form', 'url']);`
- Pengguna perlu menghapus kode `$this->load->library('form_validation');`
- `if ($this->form_validation->run() == FALSE)` menjadi `if (! $this->validate([]))`
- `$this->load->view('myform');` menjadi seperti berikut:

```
return view('myform', ['validation' => $this->validator,]);
```

(c) Pengguna juga perlu mengubah kode (dapat dilihat pada kode53) untuk melakukan validasi.

Kode 53: Perubahan kode untuk melakukan validasi.

```php
<?php

$isValid = $this->validate([
    'name'  => 'required|min_length[3]',
    'email' => 'required|valid_email',
    'phone' => 'required|numeric|max_length[10]',
]);
```

Helpers

Helpers tidak terdapat banyak perubahan namun, beberapa *helpers* pada *CodeIgniter 3* tidak terdapat pada *CodeIgniter 4* sehingga perlu perubahan pada implementasi fungsinya. *Helpers* dapat di dimuat secara otomatis menggunakan `app\Config\Autoload.php`

Events

Events merupakan pembaharuan dari *Hooks*. Pengguna harus mengubah

```
$hook['post_controller_constructor']
```

menjadi

```
Events::on('post_controller_constructor', ['MyClass', 'MyFunction']);}
```

Dan menambahkan *namespace* `CodeIgniter\Events\Events;` pada awal pembukaan tag PHP.

Framework

Pengguna tidak membutuhkan direktori *core* dan tidak membutuhkan kelas `MY_X` pada direktori *libraries* untuk memperpanjang atau mengganti potongan *CI4*. Pengguna dapat membentuk kelas dimanapun dan menambahkan metode pada `app/Config/Services.php`.

6. **Menulis dokumen skripsi**
   Status : Ada sejak rencana kerja skripsi.
   Hasil : Menulis sebagian dokumen skripsi yaitu bab 1, 2, dan 3.

7. **Melakukan studi literatur *Twig*[12]**
   Status : baru ditambahkan pada semester ini
   Hasil : Sudah dipelajari cara kerja *twig* untuk implementasi pada *view*.
   Twig merupakan sebuah *template engine* untuk bahasa pemrograman *PHP*. *Twig* digunakan pada *view* aplikasi *ShaIF Judge* untuk menampilkan data berupa variabel dan melanjutkan atau memasukan sebuah *template view* pada *view* lainnya. Kode 54 merupakan contoh penggunaan *Twig* pada salah satu halaman *SharIF Judge*.

Kode 54: Contoh penggunaan *twig* pada *view*.

```
<h1>Members</h1>
<ul>
    {% for user in users %}
        <li>{{ user.username|e }}</li>
    {% endfor %}
</ul>
```

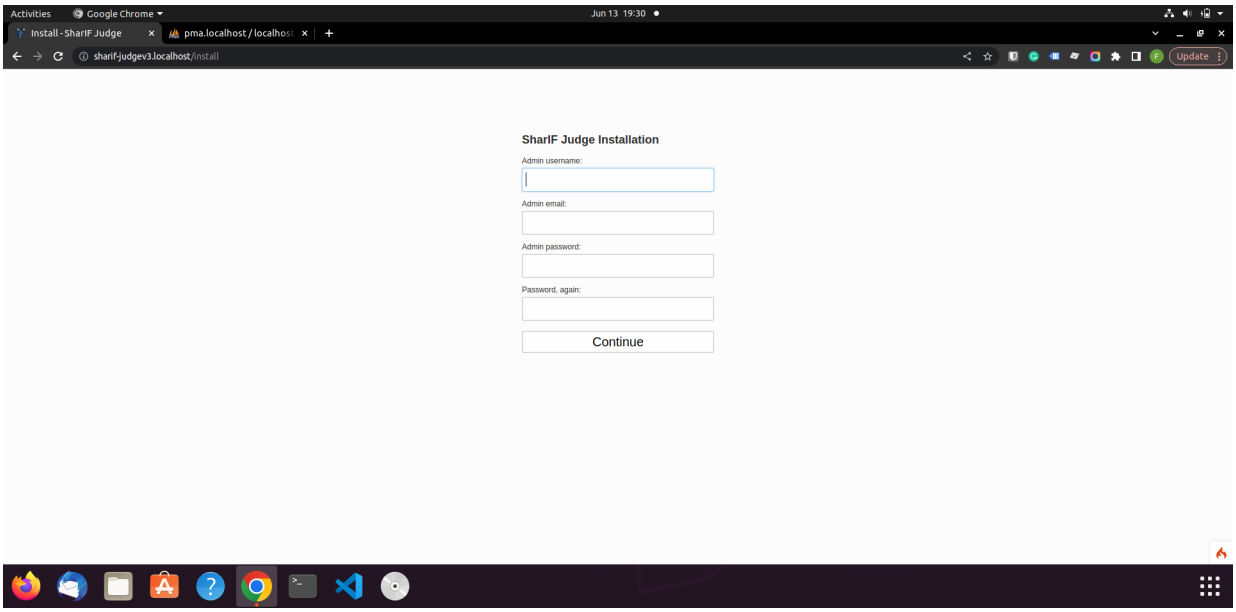
Kode 54 menggunakan dua buah *delimiters* yaitu `{%..%}` dan `{{..}}`. *Delimites* `{%..%}` berfungsi untuk menjalankan perintah PHP seperti *for-loops*. Sedangkan *delimiters* `{{..}}` berfungsi untuk menampilkan nilai dari variabel.

8. **Melakukan implementasi sederhana**
   Status : baru ditambahkan pada semester ini

[12] Dokumentasi *Twig* `https://twig.symfony.com/doc/3.x/functions/block.html`(10 Maret 2023)

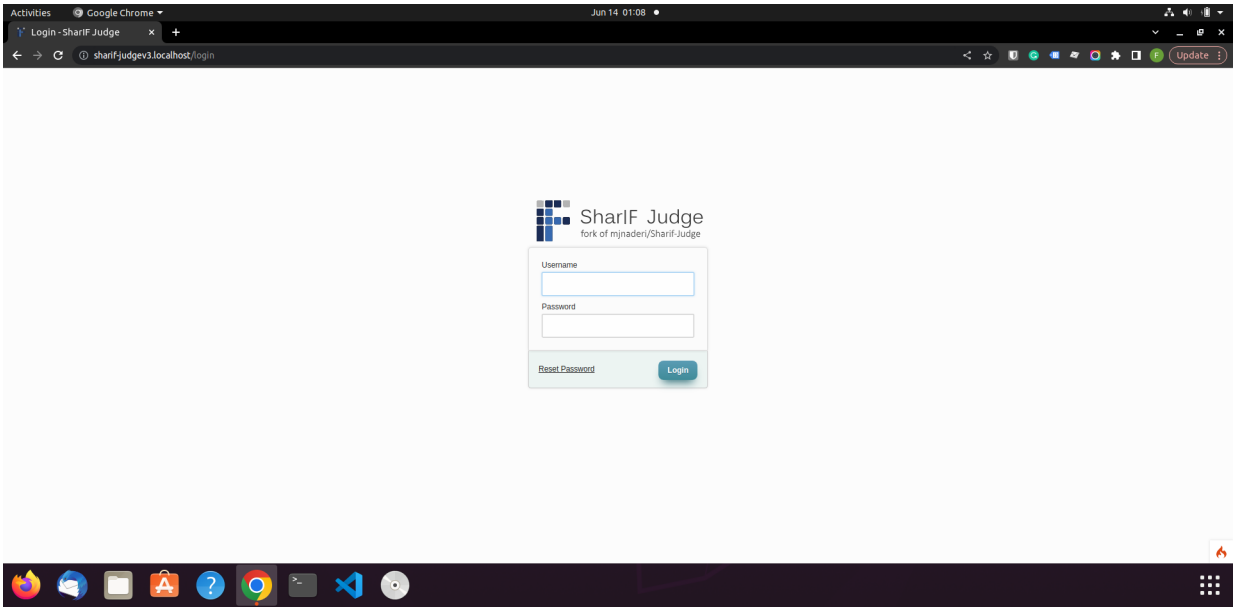
Hasil : Sudah berhasil melakukan implementasi sederhana pada beberapa fitur.
Mengimplementasikan kode pada projek *CodeIgniter 4* antara lain fitur *install, login, logout, dashboard, profile* ,dan *setting*. Berikut merupakan fitur yang sudah di implementasikan:

- **Install**



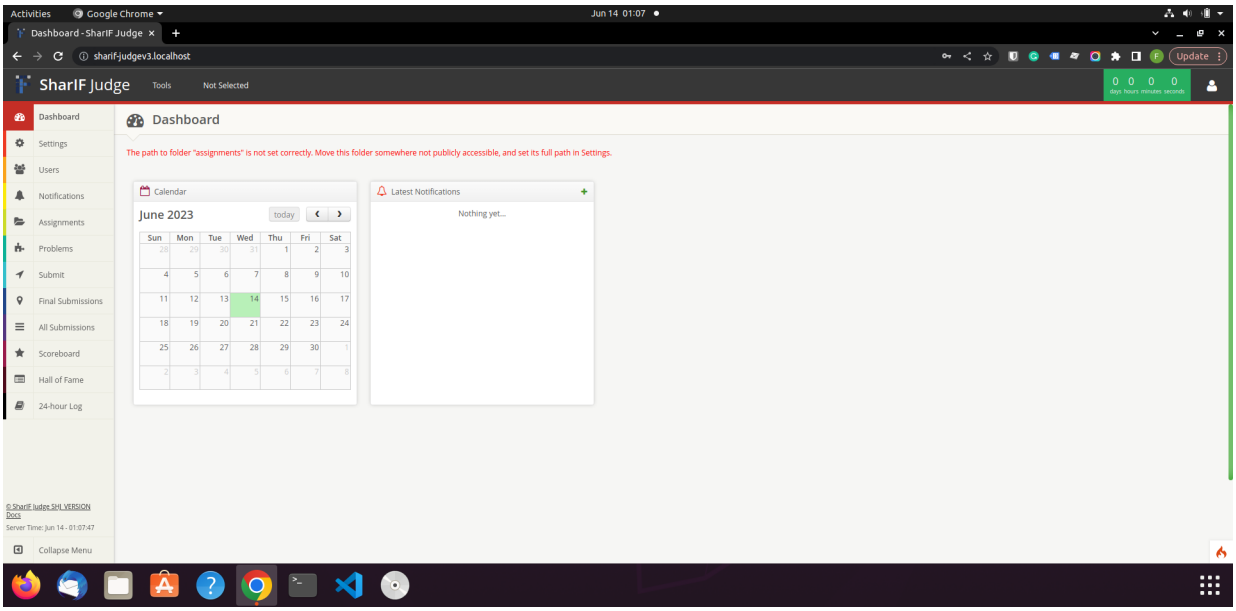
Gambar 5: Tampilan Halaman Install

- **Login**



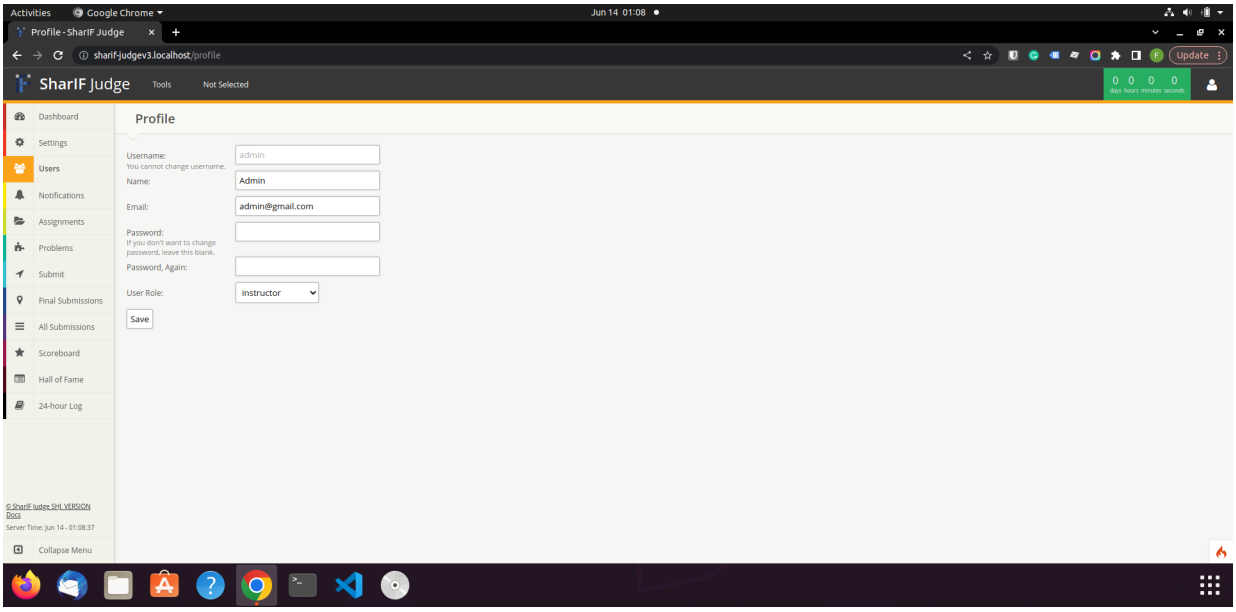
Gambar 6: Tampilan Halaman Login

- **Dashboard**



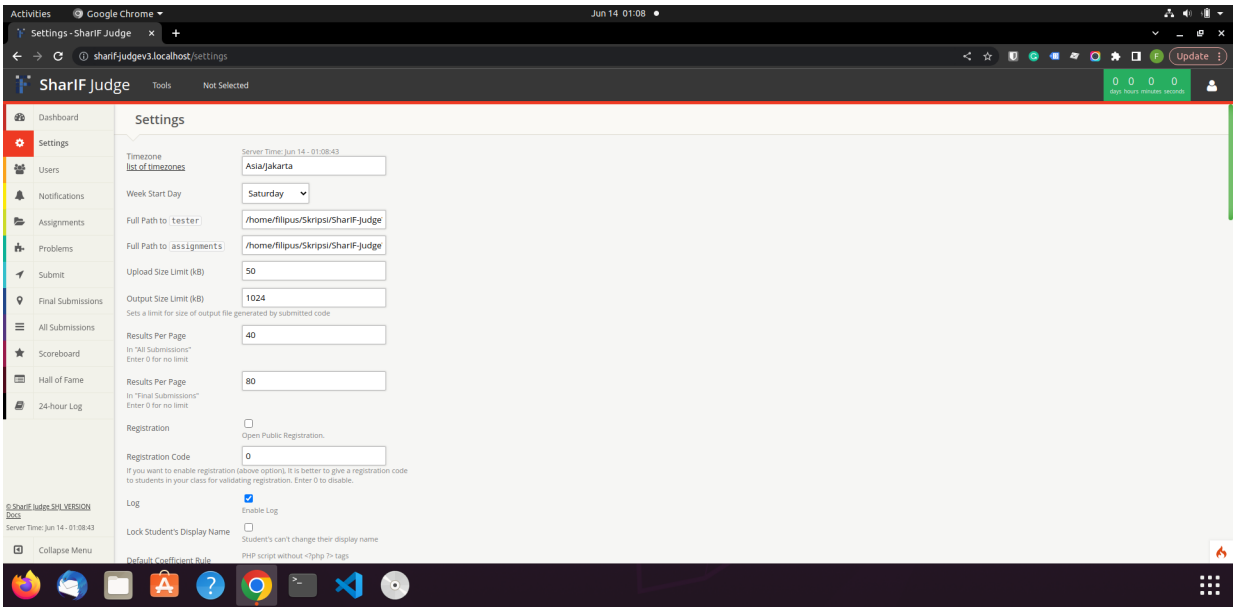
Gambar 7: Tampilan Halaman Dashboard

- **Profile**



Gambar 8: Tampilan Halaman Profile

- **Settings**



Gambar 9: Tampilan Halaman Settings

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Mempelajari cara kerja SharIF Judge.
2. Melakukan studi literatur mengenai *CodeIgnier 3* dan *CodeIgniter 4*.
3. Melakukan studi literatur mengenai cara konversi *CodeIgniter 3* menjadi *CodeIgnier 4*.
4. Menulis dokumen skripsi yaitu bab 1, 2, dan 3.

Bandung, 14/06/2023

Filipus

Menyetujui,

Nama: Pascal Alfadian Nugroho
Pembimbing Tunggal