SKRIPSI

KONVERSI SHARIF JUDGE DARI CODEIGNITER 3 KE CODEIGNITER 4

Filipus

NPM: 6181901074

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2023

DAFTAR ISI

Daftar Isi iii

Daftar Gambar v

1 Pendahuluan 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 3
- 1.3 Tujuan . . . 3
- 1.4 Batasan Masalah . . . 3
- 1.5 Metodologi . . . 3
- 1.6 Sistematika Pembahasan . . . 3

2 Landasan Teori 5
- 2.1 CodeIgniter 3[1] . . . 5
  - 2.1.1 *Model-View-Controller* . . . 5
  - 2.1.2 *CodeIgniter URLs* . . . 7
  - 2.1.3 *Helpers* . . . 8
  - 2.1.4 *Libraries* . . . 9
  - 2.1.5 *Database* . . . 13
  - 2.1.6 *URI Routing* . . . 14
  - 2.1.7 *Auto-loading* . . . 15
- 2.2 SharIF Judge[2] . . . 15
  - 2.2.1 Struktur Aplikasi . . . 15
  - 2.2.2 Instalasi . . . 15
  - 2.2.3 *Clean URLs* . . . 17
  - 2.2.4 Users . . . 17
  - 2.2.5 Menambah *Assignment* . . . 18
  - 2.2.6 *Sample Assignment* . . . 22
  - 2.2.7 *Test Structure* . . . 24
  - 2.2.8 Deteksi Kecurangan . . . 26
  - 2.2.9 Keamanan . . . 27
  - 2.2.10 *Sandboxing* . . . 29
  - 2.2.11 *Shield* . . . 29
- 2.3 CodeIgniter 4[3] . . . 31
  - 2.3.1 *Models-Views-Controllers* . . . 31
  - 2.3.2 *CodeIgniter URLs* . . . 34
  - 2.3.3 *URI Routing* . . . 35
  - 2.3.4 *Database* . . . 36
  - 2.3.5 *Library* . . . 37
  - 2.3.6 *Helpers* . . . 41
- 2.4 Konversi CodeIgniter 3 ke CodeIgniter 4[3] . . . 41
  - 2.4.1 Struktur Aplikasi . . . 42
  - 2.4.2 *Routing* . . . 42

2.4.3 *Model, View, and Controller* . . . 42
2.4.4 *Configuration* . . . 43
2.4.5 *Database* . . . 44
2.4.6 *Migrations* . . . 44
2.4.7 *Routing* . . . 44
2.4.8 *Libraries* . . . 44
2.4.9 *Helpers* . . . 46
2.4.10 *Events* . . . 46
2.4.11 *Framework* . . . 46

3 Analisis **47**
3.1 Analisis Sistem Kini . . . 47
3.1.1 *Model* . . . 47
3.1.2 *View* . . . 52
3.1.3 *Controller* . . . 60
3.2 *Library* . . . 66
3.3 Analisis Sistem Usulan . . . 69
3.3.1 Persiapan *CodeIgniter 4* . . . 69
3.3.2 Struktur Aplikasi . . . 69
3.3.3 *Routing* . . . 71
3.3.4 *Model, View, and Controller* . . . 71
3.3.5 *Libraries* . . . 73
3.3.6 *Configuration* . . . 76
3.3.7 *Database* . . . 76
3.3.8 *Helpers* . . . 77

4 Perancangan **79**
4.1 Instalasi *CodeIgniter 4* . . . 79
4.2 Perubahan Struktur Aplikasi . . . 79
4.2.1 app/Config . . . 79
4.2.2 *Controllers* . . . 80
4.2.3 *Filters* . . . 81
4.2.4 *Libraries* . . . 82
4.2.5 *Model* . . . 90
4.2.6 *View* . . . 90
4.2.7 public . . . 91

Daftar Referensi **93**

A Kode Program **95**

B Hasil Eksperimen **97**

DAFTAR GAMBAR

1.1 Tampilan halaman *SharIF Judge* 1
1.2 *Pemindahan struktur aplikasi menuju CodeIgniter 4* 2

2.1 *Flow Chart* Aplikasi *CodeIgniter 3* 5
2.2 Tampilan halaman *SharIF Judge* untuk menambahkan *assignment* 19

3.1 Tampilan Halaman *Dashboard* 52
3.2 Tampilan Halaman *Profile* 53
3.3 Tampilan Halaman *Settings* 53
3.4 Tampilan Halaman *Users* 54
3.5 Tampilan Halaman *Notifications* 54
3.6 Tampilan Halaman *Assignments* 55
3.7 Tampilan Halaman *Problems* 55
3.8 Tampilan Halaman *Submit* 56
3.9 Tampilan Halaman *Final Submission* 56
3.10 Tampilan Halaman *All Submission* 57
3.11 Tampilan Halaman *Scoreboard* 57
3.12 Tampilan Halaman *Hall of Fame* 58
3.13 Tampilan Halaman *24-hour Log* 58
3.14 Tampilan Halaman *ReJudge* 59
3.15 Tampilan Halaman *Submission Queue* 59
3.16 Tampilan Halaman *Cheat Detection* 60
3.17 *Pemindahan struktur aplikasi menuju CodeIgniter 4* 70

B.1 Hasil 1 97
B.2 Hasil 2 97
B.3 Hasil 3 97
B.4 Hasil 4 97

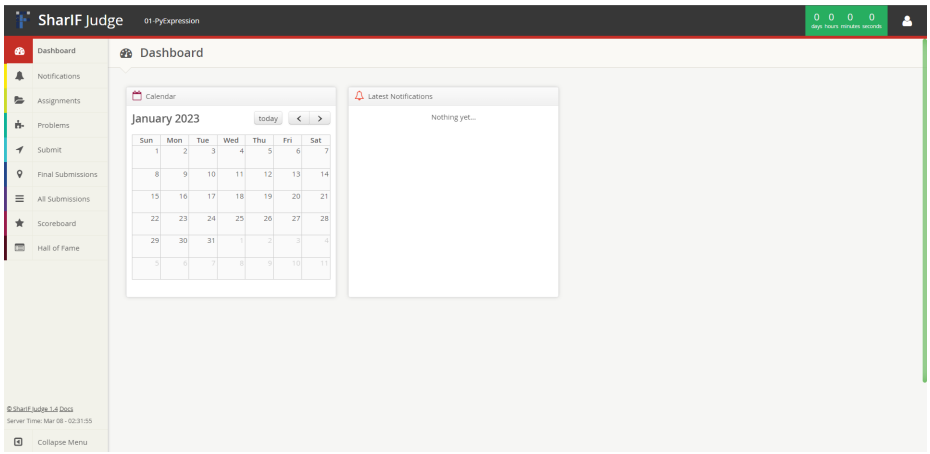
BAB 1

PENDAHULUAN

1.1 Latar Belakang

Tugas merupakan suatu bentuk pembelajaran dan penilaian yang diberikan oleh pengajar kepada pelajar untuk membantu pelajar mendalami materi yang sudah diberikan[4]. 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[5]. Salah satu perangkat lunak *Online Judge* terdapat pada Universitas Katolik Parahyangan prodi Informatika bernama SharIF Judge (dapat dilihat pada Gambar 1.1).



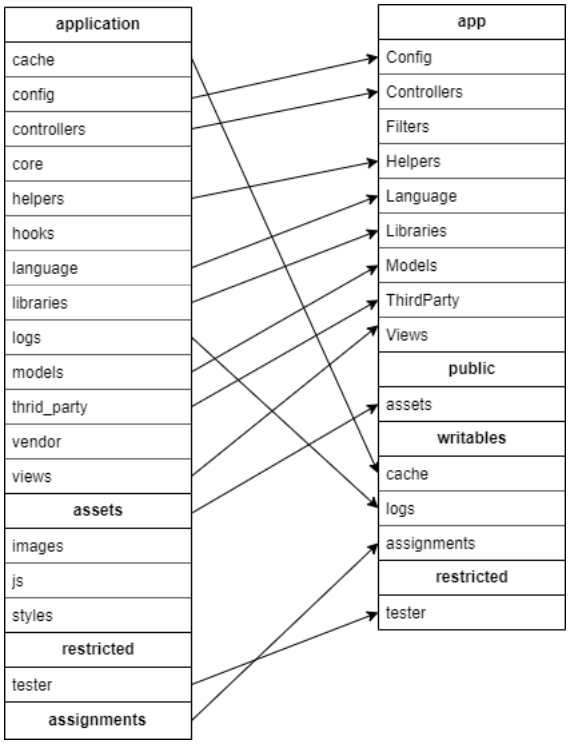
Gambar 1.1: Tampilan halaman *SharIF Judge*

SharIF Judge pada awalnya bernama Sharif Judge yang merupakan sebuah perangkat lunak *open source* untuk menilai kode dengan beberapa bahasa seperti C, C++, Java, dan Python secara online.

Sharif Judge pada awalnya dibentuk oleh Mohammad Javad menggunakan *framework* CodeIgniter 3 yang merupakan *framework* berbasis PHP (*Hypertext Preprocessor*). Sharif Judge kemudian di *fork* dan dimodifikasi menjadi SharIF Judge dengan penambahan fungsi sesuai kebutuhan Informatika Unpar untuk mengumpulkan tugas dan ujian mahasiswa[2].

CodeIgniter 3 merupakan sebuah *framework opensource* yang bertujuan untuk mempermudah dalam membangun 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[1]. Namun, CodeIgniter 3 sudah memasuki fase *maintenance*[1] 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 struktur direktori berbeda dengan versi sebelumnya[3]. Perubahan direktori dapat dilihat pada gambar 1.2.



Gambar 1.2: *Pemindahan struktur aplikasi menuju CodeIgniter 4*

Gambar 1.2 merupakan perubahan struktur yang terdapat pada CodeIgniter 4. Rincian perubahan dapat dilihat pada bab 3

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

[1]Pemberitahuan fase *maintenance CodeIgniter 3* https://codeigniter.com/download(19 Maret 2023)

1.2 Rumusan Masalah

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

1.3 Tujuan

Tujuan dari skripsi ini adalah sebagai berikut:

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

1.4 Batasan Masalah

Batasan masalah pada pembentukan skripsi ini adalah sebagai berikut:

- EasySandbox tidak dijalankan karena tidak mendukung perangkat berbasis sistem operasi *MACOS*

1.5 Metodologi

Metodologi yang dilakukan dalam melakukan penelitian ini adalah sebagian berikut:

1. Melakukan analisis dan eksplorasi fungsi-fungsi perangkat lunak SharIF Judge.
2. Melakukan studi literatur kebutuhan konversi dari CodeIgniter 3 menjadi CodeIgniter 4.
3. Melakukan konversi perangkat lunak dari CodeIgniter 3 menjadi CodeIgniter 4.
4. Melakukan pengujian dan eksperimen terhadap perangkat lunak yang sudah di konversi.
5. Menyelesaikan pembentukan dokumen

1.6 Sistematika Pembahasan

Penelitian ini akan dibahas dalam enam bab yang masing-masing berisi:

1. **Bab 1:** Pendahuluan
   Bab ini berisi latar belakang, rumusan masalah,tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
2. **Bab 2:** Landasan Teori
   Bab ini berisi pembahasan dasar-dasar teori yang akan digunakan dalam melakukan konversi SharIF Judge dari CodeIgniter 3 ke CodeIgniter 4. Landasan Teori yang digunakan diantaranya adalah SharIF Judge, CodeIgniter 3, CodeIgniter 4, dan Konversi CodeIgniter 3 ke CodeIgniter 4.
3. **Bab 3:** Analisis
   Bab ini berisi analisis *SharIF Judge* dan analisis kebutuhan konversi menuju *CodeIgniter 3*.

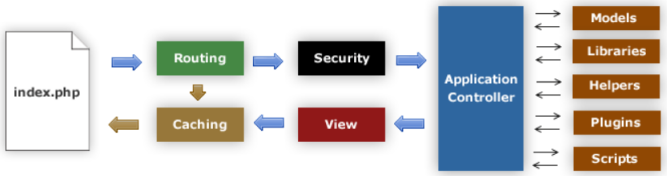
4. **Bab 4:** Perancangan
   Bab ini berisikan mengenai rancangan yang perangkat lunak yang akan dikonversi.
5. **Bab 5:** Implementasi dan Pengujian
   Bab ini berisikan hasil implementasi dan pengujian yang telah dilakukan untuk melakukan konversi SharIF Judge dari *CodeIgniter 3* ke *CodeIgniter 4*.
6. **Bab 6:** Kesimpulan dan Saran
   Bab ini berisikan kesimpulan dari hasil konversi yang telah dilakukan dan saran-saran terhadap perangkat lunak.

BAB 2

LANDASAN TEORI

2.1 CodeIgniter 3[1]

CodeIgniter 3 merupakan sebuah *framework opensource* yang berfungsi untuk mempermudah pengguna dalam membangun aplikasi *website* menggunakan bahasa PHP. CodeIgniter 3 memiliki tujuan untuk membantu pengguna dalam membangun 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 2.1.



Gambar 2.1: *Flow Chart* Aplikasi *CodeIgniter 3*

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

1. `index.php` berfungsi sebagai *front controller* yang berguna untuk melakukan inisiasi
2. *Router* berfungsi dalam melakukan pemeriksaan dan menentukan penggunaan *HTTP Request.*
3. *Cache* berfungsi untuk mengirimkan *file cache*(apabila ada) kepada *browser* secara langsung.
4. *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.
5. *Controller* berguna sebagai alat untuk memuat *model, libraries*, dan sumber daya yang dibutuhkan untuk menjalankan permintaan spesifik.
6. *View* akan dikirimkan menuju *browser* untuk dilihat oleh pengguna. Apabila *caching* dinyalakan, maka *view* akan dilakukan *cached* terlebih dahulu sehingga permintaan selanjutnya dapat diberikan.

2.1.1 *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 2.1: Contoh *model* pada *CodeIgniter 3*

```
class Blog_model extends CI_Model {

        public $title;
        public $content;
        public $date;

        public function get_last_ten_entries()
        {
                $query = $this->db->get('entries', 10);
                return $query->result();
        }

        public function insert_entry()
        {
                $this->title    = $_POST['title']; // please read the below note
                $this->content  = $_POST['content'];
                $this->date     = time();

                $this->db->insert('entries', $this);
        }

        public function update_entry()
        {
                $this->title    = $_POST['title'];
                $this->content  = $_POST['content'];
                $this->date     = time();

                $this->db->update('entries', $this, array('id' => $_POST['id']));
        }

}
```

Kode 2.1 terdapat 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 kode sebagai berikut:

`$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 membangun kode yang lebih bersih.

Kode 2.2: Contoh *view* pada *CodeIgniter 3*

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

Kode 2.2 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 2.3: 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 2.3 berfungsi dalam mengembalikan *string* sesuai dengan fungsi *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 dapat diambil dari *URL*.

2.1.2 *CodeIgniter URLs*

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

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

URL diatas dibentuk berdasarkan *Controller* sebagai berikut :

```
example.com/class/function/ID
```

Segmen tersebut dibagi menjadi tiga buah yakni:

1. Segmen pertama mereprentasikan kelas *controller* yang dipanggil.
2. Segmen kedua mereprentasikan kelas fungsi atau metode yang digunakan.
3. 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 2.4: 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 berikut:

Kode 2.5: *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*. Pengubahan fitur tersebut dapat memperbolehkan pengguna untuk menambahkan *query* pada *URL* yang dibentuk. Berikut merupakan contoh pengaksesan *URL* melalui *query strings*:

`index.php?c=controller&m=method`

URL dapat mengakses fungsi ataupun metode dari *controller* menggunakan *query* tanpa harus menggunakan *URL* biasa.

2.1.3 *Helpers*

Helpers merupakan fungsi pada *CodeIgniter 3* yang mempermudah pengguna dalam membangun 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* yang ingin digunakan. Pengguna dapat memanggil satu atau banyak *helper* pada metode *controller* ataupun *view* sesudah dimuat.

2.1.4 *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 2.6: Contoh kelas *library* pada *CodeIgniter 3*

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

class Someclass {

        public function some_method()
        {
        }
}
```

Kode 2.6 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);
```

Kode diatas memanggil *library* `someclass` dan 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 2.7 merupakan contoh pengiriman email melalui *controller*.

Kode 2.7: 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 2.7 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 2.8 merupakan contoh konfigurasi preferensi secara manual.

Kode 2.8: 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 2.8 merupakan contoh konfigurasi pengiriman *email* dengan protokol `sendmail`, tujuan `usr/sbin/sendmail`, *character set* berjenis `iso-8859-1`, dan menyalakan fitur `wordwrap`. Selanjutnya konfigurasi dapat diinisialisasikan menggunakan `initialize`.

Kelas *File Uploading*

Pengunggahan *file* terdapat empat buah proses sebagai berikut:

1. Dibentuk sebuah form untuk pengguna memilih dan mengunggah *file*.
2. Setelah *file* diunggah, *file* akan dipindahkan menuju direktori yang dipilih.
3. Pada pengiriman dan pemindahan *file* dilakukan validasi sesuai dengan ketentuan yang ada.
4. 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 2.9: 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 2.9 merupakan contoh kode dengan dua buah metode yakni:

1. `index()` yang digunakan untuk mengembalikan *view* bernama `upload_form`
2. `do_upload` yang digunakan untuk melakukan validasi berupa tipe, ukuran, lebar, dan panjang maksimal sebuah file. Metode ini juga mengembalikan *error* dan menyimpan *file* pada direktori `./uploads/`.

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 membangun 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 2.10 merupakan contoh penggunaan *library Zip Encoding* untuk meyimpan dan menunduh data.

Kode 2.10: 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 2.10 merupakan contoh untuk melakukan penyimpanan *Zip file* pada direktori dan dapat mengunduh *file* menuju *desktop* pengguna. Selain menggunakan *library* yang sudah disediakan, pengguna dapat membangun dan memperluas *libraries* sendiri sesuai dengan kebutuhan. Kode merupakan contoh *library* yang dibentuk.

Kode 2.11: 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.

2.1.5 *Database*

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

Kode 2.12: 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 2.12 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 2.13 merupakan contoh *query* untuk melakukan *select* dan *join* pada *CodeIgniter 3*:

Kode 2.13: Contoh penggunaan *query*

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

Kode 2.13 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 membangun, 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 membangun *database* menggunakan kelas *Forge*. Kode 2.14 merupakan contoh untuk membangun *database* dengan nama `db_name`.

Kode 2.14: Contoh membangun *database* menggunakan *CodeIgniter3*

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

Selain itu, pengguna juga dapat menambahkan kolom dengan konfigurasinya. Kode 2.15 merupakan contoh penambahan kolom sesuai dengan kebutuhan pengguna.

Kode 2.15: 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 2.15 merupakan contoh penggunaan *database* untuk menambahkan kolom sesuai dengan tipe, batas dari data yang disimpan, penambahan otomatis, dan lainnya pada tabel `table_name`.

2.1.6 *URI Routing*

URL string yang dihasilkan *CodeIgniter 3* 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 metode dengan penambahaan *segment* yang diinginkan.

Kode 2.16: Contoh *url* yang sudah dimetakan

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

Kode 2.16 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:

1. Bagian segmen *URL*
   Bagian pertama merupakan segmen pertama *url* yang akan tampil pada *url*. Bagian kedua merupakan segmen kedua dapat berisikan angka atau karakter.
2. 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';
```

2.1.7 *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.

2.2 SharIF Judge[2]

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.

2.2.1 Struktur Aplikasi

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

2.2.2 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 2.17: 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

1. Mengunduh versi terakhir dari *SharIF Judge* dan melakukan *unpack* pada direktori *public html*.
2. Memindahkan *folder system* dan *application* diluar direktori *public* dan mengubah *path* pada `index.php`(Opsional).

Kode 2.18: Contoh *path* pada halaman index.php

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

3. Membangun *database MySql* atau *PostgreSql* untuk *SharIF Judge*. Jangan melakukan instalasi paket koneksi *database* apapun untuk *C, C++, Java,* atau *Python*.
4. Mengatur koneksi *database* pada *file* `application/config/database.example.php` dan menyimpannya dengan nama `database.php`. Pengguna dapat menggunakan awalan untuk nama tabel.

Kode 2.19: 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
```

5. Mengatur *RADIUS server* dan *mail server* pada *file* `application/config/secrets.example.php` dan menyimpannya dengan nama `secrets.php`.
6. Mengatur `application/cache/Twig` agar dapat ditulis oleh PHP.
7. Membuka halaman utama SharIF Judge pada *web browser* dan mengikuti proses instalasi.
8. Melakukan *Log in* dengan akun admin.
9. 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.

2.2.3 *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`.

2.2.4 Users

Pada perangkat lunak SharIF Judge, pengguna dibagi menjadi 4 buah. Keempat pengguna tersebut adalah *Admins, Head Instructors, Instructors*, dan *Students*. Tabel 2.1 merupakan pembagian tingkat setiap pengguna.

Tabel 2.1: *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 2.2 merupakan aksi yang dapat dilakukan oleh setiap pengguna.

Tabel 2.2: *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 2.20: 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'
```

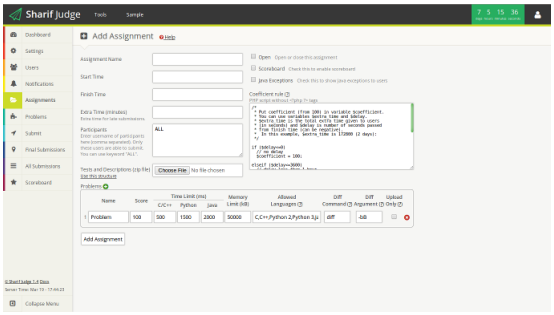
Dengan contoh sebagai berikut:

Kode 2.21: 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
```

2.2.5 Menambah *Assignment*

Pengguna dapat menambahkan *assignment* baru melalui bagian *Add* pada halaman *Assignments*(dapat dilihat pada Gambar 2.2).



Gambar 2.2: 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 2.2.7.

- ***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 2.22: Contoh tampilan fitur *Java Exceptions*

```
Test 1
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 2.23: 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.

2.2.6 *Sample Assignment*

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

Problems

1. *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

2. *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

3. *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 2.24: 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*

```java
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++*

```cpp
#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++*

```cpp
#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;
}
```

2.2.7 *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*.

1. 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*.

2. 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 2.25: Templat kode *tester.cpp*

```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 2.26: Kode metode perbandingan *tester* dengan bahasa *tester.cpp*

```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.

2.2.8 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:

1. 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 2.27: 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;

...
```

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.

2. *Server* pengguna harus memiliki instalasi `perl` untuk menggunakan *Moss*.
3. 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.

2.2.9 Keamanan

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

1. **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 2.2.10.

2. **Menggunakan *Shield***

   Pengguna harus memastikan untuk menggunakan *shield* untuk bahasa *C, C++*, dan *Python*. Penggunaan *shield* dapat dilihat pada subbab 2.2.11.

3. **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 membangun banyak proses.
Diasumsikan bahwa PHP dijalankan sebagai pengguna `www-data` pada server. Membangun *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 2.28: 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 2.29: 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 2.30: 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 2.31: Contoh kode untuk membatasi jumlah proses

```
1 restricted_user     soft    nproc   3
2 restricted_user     hard    nproc   5
```

4. **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.

2.2.10 *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 2.32: 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*.

2.2.11 *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 subbab2.2.10).

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 2.33: 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 2.34: 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 2.35: 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]
```

2.3 CodeIgniter 4[3]

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 membangun kelas baru untuk menambahkan fungsi yang diperlukan.

2.3.1 *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 pada direktori `app/Models` dan memiliki *namespace* sesuai dengan lokasi dari *file* tersebut. Kode 2.36 merupakan contoh dari *models*.

Kode 2.36: Contoh *Models*

```php
<?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 2.36 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 2.37 merupakan contoh penggunaan *model* untuk mencari data spesifik.

Kode 2.37: Contoh penggunaan *model* untuk mencari data spesifik

```php
<?php

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

Kode 2.37 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 2.38: Contoh *Views*

```php
<html>
    <head>
```

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

Kode 2.39 merupakan contoh *views* pada direktori `app/Views` yang akan menampilkan tulisan *Welcome to my Blog*. *View* ini dapat ditampilkan melalui controller seperti berikut:

Kode 2.39: Contoh *Views*

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

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

Kode 2.39 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 2.40 merupakan contoh *controllers* yang dibentuk.

Kode 2.40: 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 2.40 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 membangun sebuah kondisi sebelum ataupun sesudah *controller* dijalankan. Kode 2.41 merupakan contoh penggunaan *filters*.

Kode 2.41: Contoh *Controllers Filters* pada *CodeIgniter 4*

```
<?php

namespace App\Filters;

```

```php
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 2.41 merupakan contoh kode yang akan melakukan pengecekan tabel ataupun *session* sebelum *controller* dijalankan. Kode ini akan disimpan pada direktori `app/Filters`. 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.

2.3.2 *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* membangun *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 2.42 merupakan contoh *file* `.htaccess` untuk menghapus `index.php` pada *URL* yang dibentuk.

Kode 2.42: 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 2.43 merupakan contoh penggunaan `try_files` untuk menghapus `index.php` pada *URL*.

Kode 2.43: Contoh penggunaan `try-files`.

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

2.3.3 *URI Routing*

CodeIgniter 4 menyediakan dua buah *routing* yakni:

Defined Route Routing

Pengguna dapat mendefinisikan *route* secara manual untuk *URL* yang lebih fleksibel. Kode 2.44 merupakan contoh *route* yang didefinisikan secara manual.

Kode 2.44: Contoh *route* yang didefinisikan secara manual

```
<?php

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

Kode 2.44 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 2.45: Contoh *route* yang menggunakan *grouping* manual

```
<?php

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

Kode 2.45 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 perlu mengubah `$autoRoutesImproved` menjadi `true` pada file `app/Config/Feature.php`. Selain menggunakan *auto routing* baru, pengguna dapat menggunakan *Auto Routing (Legacy)* yang terdapat pada *CodeIgniter 3* dengan cara seperti berikut:

2.3.4 *Database*

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 2.46 merupakan contoh pada direktori `Database.php`.

Kode 2.46: Contoh konfigurasi *database* pada *CodeIgniter 4*.

```
<?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 2.46 merupakan contoh konfigurasi untuk database bernama `database_name` dengan *username* root. Selain itu, konfigurasi juga dapat dilakukan pada file `.env` untuk mempermudah dalam pengubahan pada saat melakukan *deploy*. Kode 2.47 merupakan contoh konfigurasi pada file `.env`:

Kode 2.47: Contoh konfigurasi *database* pada file `.env`.

```
database.default.username = 'root';
database.default.password = '';
database.default.database = 'ci4';
```

Kode 2.47 akan menyimpan konfigurasi pada grup *default* dengan *username* berupa root, tanpa menggunakan *password*, dan juga dengan nama *database* ci4.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 2.48: Contoh penggunaan *query* menggunakan konfigurasi pada *CodeIgniter 4*.

```
<?php

$builder = $db->table('users');
$builder->select('title, content, date');
$query = $builder->get();
```

Kode 2.48 merupakan contoh penggunaan *query* untuk mengambil data *title,content* ,dan *date* pada tabel *users*. *CodeIgniter 4* juga menyediakan fitur untuk membangun *database* melalui fitur bernama *Database Forge*. Pengguna dapat membangun,mengubah, menghapus tabel dan juga menambahkan *field* pada tabel tersebut. Kode 2.49 merupakan contoh pembentukan *database*.

Kode 2.49: Contoh pembentukan tabel melalui *database forge*.

```
<?php

$fields = [
    'id' => [
        'type'           => 'INT',
        'constraint'     => 5,
        'unsigned'       => true,
        'auto_increment' => true,
    ],
    'title' => [
        'type'       => 'VARCHAR',
        'constraint' => '100',
        'unique'     => true,
    ],
    'author' => [
        'type'       => 'VARCHAR',
        'constraint' => 100,
        'default'    => 'King of Town',
    ],
    'description' => [
        'type' => 'TEXT',
        'null' => true,
    ],
    'status' => [
        'type'       => 'ENUM',
        'constraint' => ['publish', 'pending', 'draft'],
        'default'    => 'pending',
    ],
];
$forge->addField($fields);
$forge->createTable('table_name');
```

Kode 2.49 merupakan contoh pembentukan *database* dengan tabel bernama *table_name* yang berisikan beberapa *field*.

2.3.5 *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 2.50 merupakan contoh konfigurasi preferensi *email* secara manual.

Kode 2.50: 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 2.51 merupakan contoh penggunaan kelas *email* untuk mengirim *email*.

Kode 2.51: 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 2.51 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:

1. Dibentuk sebuah form untuk pengguna memilih dan mengunggah *file*.
2. Setelah *file* diunggah, *file* akan dipindahkan menuju direktori yang dipilih.
3. Pada pengiriman dan pemindahan *file* dilakukan validasi sesuai dengan ketentuan yang ada.
4. Setelah *file* diterima akan dikeluarkan pesan berhasil.

Perangkat lunak akan menerima *file* dari *views* yang nantinya akan dilakukan validasi pada *controller*. Kode 2.52 merupakan contoh *view* untuk melakukan pengunggahan *file*.

Kode 2.52: 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 2.52 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 2.53: Contoh kode *controller* untuk melakukan validasi dan penyimpanan.

```
<?php

namespace App\Controllers;

use CodeIgniter\Files\File;

class Upload extends BaseController
{
    protected $helpers = ['form'];

    public function index()
    {
        return view('upload_form', ['errors' => []]);
    }

    public function upload()
    {
        $validationRule = [
            'userfile' => [
                'label' => 'Image File',
                'rules' => [
                    'uploaded[userfile]',
                    'is_image[userfile]',
                    'mime_in[userfile,image/jpg,image/jpeg,image/gif,image/png,image/webp]',
                    'max_size[userfile,100]',
                    'max_dims[userfile,1024,768]',
                ],
            ],
        ];
        if (! $this->validate($validationRule)) {
            $data = ['errors' => $this->validator->getErrors()];

            return view('upload_form', $data);
        }

        $img = $this->request->getFile('userfile');

        if (! $img->hasMoved()) {
            $filepath = WRITEPATH . 'uploads/' . $img->store();

            $data = ['uploaded_fileinfo' => new File($filepath)];

            return view('upload_success', $data);
        }

        $data = ['errors' => 'The file has already been moved.'];

        return view('upload_form', $data);
    }
}
```

Kode 2.53 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`.

Validation

CodeIgniter 4 menyediakan *library* untuk melakukan validasi terhadap data yang dikirimkan oleh pengguna. Data yang divalidasi dapat diberikan aturan-aturan sesuai dengan konfigurasi pengguna. Kode 2.54 merupakan contoh penggunaan *validation*.

Kode 2.54: Contoh kode untuk melakukan pengumpulan data.

```
<html>
<head>
    <title>My Form</title>
</head>
<body>

    <?= validation_list_errors() ?>

    <?= form_open('form') ?>

        <h5>Username</h5>
        <input type="text" name="username" value="<?= set_value('username') ?>" size="50">

        <h5>Password</h5>
        <input type="text" name="password" value="<?= set_value('password') ?>" size="50">

        <h5>Password Confirm</h5>
        <input type="text" name="passconf" value="<?= set_value('passconf') ?>" size="50">

        <h5>Email Address</h5>
        <input type="text" name="email" value="<?= set_value('email') ?>" size="50">

        <div><input type="submit" value="Submit"></div>

    <?= form_close() ?>

</body>
</html>
```

Kode 2.54 merupakan contoh penggunaan *validation* pada halaman *view*. Pengambilan *error* dapat menggunakan sintaks `validation_list_errors()`. Selanjutnya akan digunakan fungsi `form_open` untuk membuka *tag form* sesuai dengan *url* yang sudah dibentuk. Setiap input akan diberikan *name* sesuai dengan aturan yang ingin diberikan. Setelah *tag form* selesai maka akan ditutup dengan sintaks `form_close`. Setelah itu data-data yang sudah dimasukan akan dikirimkan menuju *controller* seperti pada kode 2.55.

Kode 2.55: Contoh kode untuk melakukan validasi data yang sudah dikumpulkan.

```
<?php

namespace App\Controllers;

class Form extends BaseController
{
    protected $helpers = ['form'];

    public function index()
    {
        if (! $this->request->is('post')) {
            return view('signup');
        }

        $rules = [];

        if (! $this->validate($rules)) {
            return view('signup');
        }

        // If you want to get the validated data.
```

```
        $validData = $this->validator->getValidated();

        return view('success');
    }
}
```

Data-data yang sudah diberikan oleh pengguna akan dijalankan menggunakan *controller* diatas. Sintaks `if (! $this->request->is('post'))` akan melakukan pengecekan apakah *request* yang diberikan berupa *post* atau tidak. Selanjutnya dapat ditentukan aturan pada variabel `rules` yang nantinya akan dilakukan validasi menggunakan fungsi `validate`. Fungsi `validate` akan mengecek data yang diberikan dan menentukan apakah sudah sesuai dengan aturan yang ada atau belum. Kode 2.56 merupakan contoh pembentukan aturan sesuai dengan nama *form* yang ada.

Kode 2.56: Contoh kode untuk menetapkan aturan untuk validasi data yang sudah dikumpulkan.

```
$rules = [
    'username' => 'required|max_length[30]',
    'password' => 'required|max_length[255]|min_length[10]',
    'passconf' => 'required|max_length[255]|matches[password]',
    'email'    => 'required|max_length[254]|valid_email',
];
```

Kode 2.56 merupakan contoh aturan yang ditetapkan untuk setiap *form* yang ada. Aturan-aturan tersebut dapat diganti sesuai dengan kebutuhan dari pengguna. Selain menggunakan aturan yang disediakan *CodeIgniter 4*, pengguna dapat membentuk aturannya sendiri pada file *Validation.php*. Kode 2.57 merupakan contoh aturan yang dibentuk secara manual.

Kode 2.57: Contoh kode pembentukan aturan secara manual pada file `Validation.php`.

```
<?php

class MyRules
{
    public function even($value): bool
    {
        return (int) $value % 2 === 0;
    }
}
```

Kode 2.57 merupakan contoh pembentukan aturan secara manual. Penggunaan aturan yang dibentuk secara manual sama seperti penggunaan aturan lainnya.

2.3.6 *Helpers*

Helpers merupakan fungsi pada *CodeIgniter 4* yang menyediakan beberapa fungsi untuk pengguna dalam membangun 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. Fungsi-fungsi itu antara lain adalah *form*, *array*, dan lainnya.

2.4 Konversi CodeIgniter 3 ke CodeIgniter 4[3]

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.

2.4.1 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.*

2.4.2 *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.*

2.4.3 *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 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:

1. Pertama pengguna harus memindahkan seluruh *file model* menuju direktori `app/Models`
2. Selanjutnya pengguna harus menambahkan `namespace App\Models;` setelah pembukaan *tag PHP.*
3. Pengguna juga harus menambahkan `use CodeIgniter\Model;` setelah kode diatas.
4. Pengguna harus mengganti `extends CI_Model` menjadi `extends Model`.
5. 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*:

1. Pertama pengguna perlu memindahkan seluruh *file views* menuju `app/Views`
2. Selanjutnya pengguna perlu mengubah sintaks:

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

menjadi sintaks berikut:

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

3. Pengguna juga perlu mengubah sintaks:

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

menjadi sintaks berikut:

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

4. Pada *file views* pengguna dapat mengubah sintaks:

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

menjadi sintaks berikut:

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

5. 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:

1. Pertama pengguna harus memindahkan seluruh *file controller* menuju `app/Controllers`.
2. Pengguna juga harus menambahkan sintaks `namespace App\Controllers;` setelah pembukaan *tag PHP*.
3. Selanjutnya pengguna harus mengubah `extends CI_Controller` menjadi `extends BaseController`.
4. Pengguna juga harus menghapus baris `defined('BASEPATH') OR exit('No direct script access allowed');` apabila ada.

2.4.4 *Configuration*

File configuration CodeIgniter 4 terdapat pada `app\Config` dengan penulisan sedikit berbeda dengan versi sebelumnya. Penulisan berudah dari yang sebelumnya menggunakan *array* akan berubah menjadi menggunakan variabel. 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`. Beberapa konfigurasi juga akan dipindahkan menuju file `.env`.

2.4.5 *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 `$db = db_connect();` dan apabila menggunakan beberapa *database* maka sintaks menjadi `$db = db_connect('group_name');`. 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();`.

2.4.6 *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 metode *Forge* dari yang sebelumnya bernama `$this->dbforge->add_field` menjadi menggunakan *camelCase* `$this->forge->addField`.

2.4.7 *Routing*

Pengguna dapat melakukan pembaharuan *routing* dengan cara berikut:

1. Pengguna dapat memakai *Auto Routing* 2.3.3 seperti pada *CodeIgniter 3* dengan menyalakan *Auto Routing(Legacy)*.
2. Terdapat perubahan dari `(:any)` menjadi `(:segment)`.
3. Pengguna juga harus mengubah sintaks pada `app/Config/Routes.php` seperti berikut:
   - `$route['journals'] = 'blogs';`
     menjadi
     ```
     $routes->add('journals', 'Blogs::index');
     ```
     Sintak diatas berguna untuk memanggil fungsi `index` pada *controller* `Blogs`.
   - `$route['product/(:any)'] = 'catalog/product_lookup'`
     menjadi
     ```
     $routes->add('product/(:segment)', 'Catalog::productLookup');.
     ```

2.4.8 *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:

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()
```

File yang sudah diunggah dapat disimpan dengan sintaks berikut:

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

Sintaks diatas akan mengambil file dengan atribut nama `userfile` dan menyimpannya pada direktori `head_img` dengan nama file `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');`

Validations

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

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

2. 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,]);
```

3. Pengguna juga perlu mengubah kode (dapat dilihat pada kode2.58) untuk melakukan validasi.

Kode 2.58: 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]',
]);
```

2.4.9 *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`

2.4.10 *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;`.

2.4.11 *Framework*

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

BAB 3

ANALISIS

3.1 Analisis Sistem Kini

Seperti dibahas pada bab 2.2, *SharIF Judge* merupakan sebuah *online judge* dan di kustomisasi sesuai dengan kebutuhan Informatika UNPAR. *SharIF Judge* dibentuk menggunakan *framework CodeIgniter 3* yang menerapkan arsitektur *Model-View-Controller* atau MVC. Arsitektur ini memisahkan pemrosesan data pada *Model*, memisahkan logika pada *Controller*, dan memisahkan tampilan pada *View*.

3.1.1 *Model*

Model terdapat pada direktori `application/models`. Direktori ini berisikan kelas *model* dengan fungsi-fungsi untuk mengolah data pada aplikasi. Berikut merupakan *model* pada *SharIF Judge* beserta fungsi-fungsinya.

`Assignment_model.php`

Model Assignment terdapat beberapa fungsi untuk memproses data pada tabel *assignment*. Berikut merupakan fungsi-fungsi dari *model* tersebut:

- `add_assignment`
  Fungsi ini berguna untuk menambahkan atau memperbaharui *assignment* pada *database*.
- `delete_assignment`
  Fungsi ini berguna untuk menghapus *assignment* pada *database*.
- `all_assignments`
  Fungsi ini berguna untuk mengembalikan seluruh *assignment* beserta informasi *assignment* tersebut.
- `new_assignment_id`
  Fungsi ini berguna untuk mencari `id` terkecil yang dapat digunakan untuk menambahkan *assignment* baru.
- `all_problems`
  Fungsi ini berguna untuk mengembalikan seluruh *problems* dari *assignment* yang ada.
- `problem_info`
  Fungsi ini berguna untuk mengembalikan baris tabel untuk *problem* tertentu.
- *assignment_info*
  Fungsi ini berguna untuk mengembalikan baris tabel untuk `assignment` tertentu.

- `is_participant`
  Fungsi ini berguna untuk mengecek apakah *username* merupakan peserta atau tidak.
- `increase_total_submits`
  Fungsi ini berguna untuk menambahkan satu buah total *submit*.
- `set_moss_time`
  Fungsi ini berguna untuk memperbaharui *"Moss Update Time"* untuk *assignment* tertentu.
- `get_moss_time`
  Fungsi ini berguna untuk mengembalikan *"Moss Update Time"* untuk *assignment* tertentu.
- `save_problem_description`
  Fungsi ini berguna untuk menyimpan atau memperbaharui deskripsi *problem*.
- `_update_coefficients`
  Fungsi ini dipanggil pada fungsi `add_assignment` yang berguna untuk memperbaharui koefisien dari *assignment* tertentu.

`Hof_model.php`

Berikut merupakan fungsi-fungsi dari `Hof_model.php` yang berguna untuk mengambil data untuk ditampilkan pada halaman *Hall of Fame*.

- `get_all_final_submission`
  Fungsi ini berguna untuk mengambil data untuk *Hall of Fame* berdasarkan *username*.
- `get_all_user_assignments`
  Fungsi ini berguna untuk mengambil detail *assignment* dan *problem* berdasarkan pengguna.

`Logs_model.php`

Berikut merupakan fungsi-fungsi dari `Logs_model.php` yang berguna untuk mencatat *log* pada beberapa tabel.

- `insert_to_logs`
  Fungsi ini berguna untuk mencatat *log* pada tabel *login*.
- `get_all_logs`
  Fungsi ini berguna untuk mengembalikan seluruh *log* dalam bentuk *array*.

`Notifications_model.php`

Notifications Assignment terdapat beberapa fungsi untuk memproses data pada tabel *notifications*. Berikut merupakan fungsi-fungsi dari *model* tersebut:

- `get_all_notifications`
  Fungsi ini berguna untuk mengembalikan seluruh *notifications* dalam bentuk *array*.
- `get_latest_notifications`
  Fungsi ini berguna untuk mengembalikan sepuluh *notification* terakhir.
- `add_notification`
  Fungsi ini berguna untuk menambahkan *notification* baru.
- `update_notification`
  Fungsi ini berguna untuk memperbaharui *notification* tertentu.

- `delete_notification`
  Fungsi ini berguna untuk menghapus *notification* tertentu.
- `get_notification`
  Fungsi ini berguna untuk mengembalikan *notification* dalam bentuk *array*.
- `have_new_notification`
  Fungsi ini berguna untuk mengecek apakah terdapat *notification* setelah waktu tertentu.

`Queue_model.php`

Berikut merupakan fungsi-fungsi dari `Queue_model.php` yang berguna untuk memproses data pada halaman *queue*.

- `in_queue`
  Fungsi ini berguna untuk mengecek apakah *submission* pengguna tertentu sudah dalam antrean.
- `get_queue`
  Fungsi ini berguna untuk mengembalikan data seluruh antrian.
- `empty_queue`
  Fungsi ini berguna untuk menghapus seluruh tabel *queue*.
- `add_to_queue`
  Fungsi ini berguna untuk memasukan *submission* kedalam tabel *queue*.
- `rejudge`
  Fungsi ini berguna untuk menambahkan *submission* kedalam antrean untuk dilakukan *rejudge*.
- `rejudge_single`
  Fungsi ini berguna untuk menambahkan satu buah *submission* kedalam antrean untuk dilakukan *rejudge*.
- `get_first_item`
  Fungsi ini berguna untuk mengambil data pertama dalam antrean.
- `remove_item`
  Fungsi ini berguna untuk menghapus data tertentu dalam antrean.
- `save_judge_result_in_db`
  Fungsi ini berguna untuk menyimpan hasil dari *judge* ke dalam *database*.
- `add_to_queue_exec`
  Fungsi ini berguna untuk menambahkan data *dummy* pada antrean.

`Scoreboard_model.php`

Berikut merupakan fungsi-fungsi dari `Scoreboard_model.php` yang berguna untuk memproses data untuk ditampilkan pada halaman *Score Board*.

- `_generate_scoreboard`
  Fungsi ini dipanggil pada fungsi `update_scoreboard` dan berfungsi untuk menghasilkan *scoreboard* dari *final submission*.
- `update_scoreboards`
  Fungsi ini berguna untuk memperbaharui *cache scoreboard* dari seluruh *assignment*. Fungsi ini dipanggil setiap terdapat penghapusan pengguna atau seluruh *assignment* pengguna.

- `update_scoreboard`
  Fungsi ini berguna untuk memperbaharui *cache scoreboard* dari seluruh *assignment*. Fungsi ini dipanggil setelah *judge* atau *rejudge*.
- `get_scoreboard`
  Fungsi ini berguna untuk mengambil seluruh *cache scoreboard* dari *assignment* tertentu.

`Settings_model.php`

Berikut merupakan fungsi-fungsi dari `Settings_model.php` yang berguna untuk memproses data untuk ditampilkan pada tabel *settings*.

- `get_setting`
  Fungsi ini berguna untuk mengembalikan data *setting* tertentu.
- `set_setting`
  Fungsi ini berguna untuk memperbaharui *setting*.
- `get_all_settings`
  Fungsi ini berguna untuk mengembalikan seluruh data *setting*.
- `set_settings`
  Fungsi memperbaharui beberapa *setting*.

`Submit_model.php`

Berikut merupakan fungsi-fungsi dari `Submit_model.php` yang berguna untuk memproses data yang berkaitan dengan *submission*.

- `get_submission`
  Fungsi ini berguna untuk mengembalikan baris data *submission* tertentu.
- `get_final_submissions`
  Fungsi ini berguna untuk mengembalikan seluruh *final submission*.
- `get_all_submissions`
  Fungsi ini berguna untuk mengembalikan seluruh *submission*.
- `count_final_submissions`
  Fungsi ini berguna untuk menghitung seluruh *final submission*.
- `count_all_submissions`
  Fungsi ini berguna untuk menghitung seluruh *submission*.
- `set_final_submission`
  Fungsi ini berguna untuk memperbaharui *submission* terentu menjadi *final submission*.
- `add_upload_only`
  Fungsi ini berguna untuk menambahkan hasil dari *upload only* ke dalam *database*.

User_model.php

Berikut merupakan fungsi-fungsi dari `User_model.php` yang berguna untuk memproses data pada tabel *users*.

- `have_user`
  Fungsi ini berguna untuk mengecek apakah terdapat pengguna dengan *username* tertentu.

- `user_id_to_username`
  Fungsi ini berguna untuk mengembalikan *user id* menjadi *username*.
- `username_to_user_id`
  Fungsi ini berguna untuk mengembalikan *username* menjadi *user id*.
- `have_email`
  Fungsi ini berguna untuk mengecek apakah terdapat *username* dengan *email* tertentu.
- `add_user`
  Fungsi ini berguna untuk menambahkan sebuah pengguna.
- `add_users`
  Fungsi ini berguna untuk menambahkan banyak pengguna.
- `delete_user`
  Fungsi ini berguna untuk menghapus pengguna tertentu.
- `delete_submissions`
  Fungsi ini berguna untuk menghapus seluruh *submission* pada pengguna tertentu.
- `validate_user`
  Fungsi ini berguna untuk mengecek *username* dan *password* apakah sesuai dengan data.
- `selected_assignment`
  Fungsi ini berguna untuk mengembalikan *assignment* untuk pengguna tertentu.
- `get_names`
  Fungsi ini berguna untuk mengembalikan nama dari pengguna tertentu.
- `update_profile`
  Fungsi ini berguna untuk memperbaharui profil dari pengguna seperti nama, *email, password*, dan *role*.
- `send_password_reset_mail`
  Fungsi ini berguna untuk menghasilkan *password reset key* dan mengirim *email* untuk melakukan *reset password*.
- `passchange_is_valid`
  Fungsi ini berguna untuk mengecek apakah *password key* yang diberikan sesuai atau tidak.
- `reset_password`
  Fungsi ini berguna untuk mengatur ulang *password* sesuai dengan *password key* tertentu.
- `get_all_users`
  Fungsi ini berguna untuk mengembalikan seluruh pengguna untuk halaman *users*.
- `get_user`
  Fungsi ini berguna untuk mengembalikan baris data untuk pengguna tertentu.
- `update_login_time`
  Fungsi ini berguna untuk memperbaharui *login time* dan *last login time* untuk pengguna tertentu.

User.php

Berikut merupakan fungsi-fungsi dari `User.php` yang berguna untuk memproses data pada tabel *users*.

- `select_assignment`

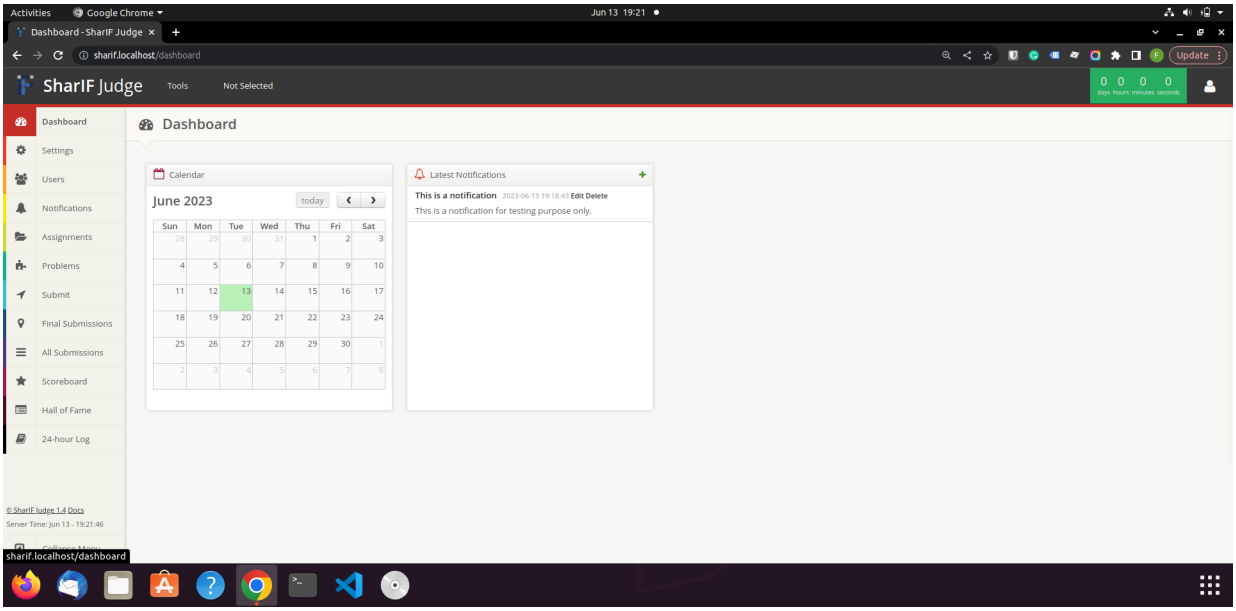
Fungsi ini berguna untuk mengatur *assignment* yang dipilih oleh pengguna.

- `save_widget_positions`
  Fungsi ini berguna untuk memperbaharui posisi dari *dashboard widget* pada *database*.
- `get_widget_positions`
  Fungsi ini berguna untuk mengembalikan data *dashboard widget*.

3.1.2 *View*

View terdapat pada direktori `application/views`. Direktori ini berisikan seluruh *file* untuk tampilan halaman *SharIF Judge*. *File* tersebut dipisahkan oleh direktori sesuai dengan fungsinya. Direktori tersebut dibagi menjadi tiga buah direktori utama yakni *error, pages*, dan *templates*. Direktori *error* berisikan tampilan halaman *error* yang akan dilihat oleh pengguna. Direktori *pages* merupakan tampilan utama *SharIF Judge* yang terbagi lagi menjadi dua buah direktori yakni *admin* dan *authentication*. Direktori *admin* berisikan tampilan halaman untuk *role admin*. Direktori *authentication* berisikan tampilan halaman untuk akses pengguna seperti *LOgin, Register,* dan *Reset Password*. Direktori *templates* terdiri dari tampilan yang digunakan oleh seluruh tampilan halaman seperti *header* dan *side bar*. Berikut merupakan tampilan halaman pada *SharIF Judge*:

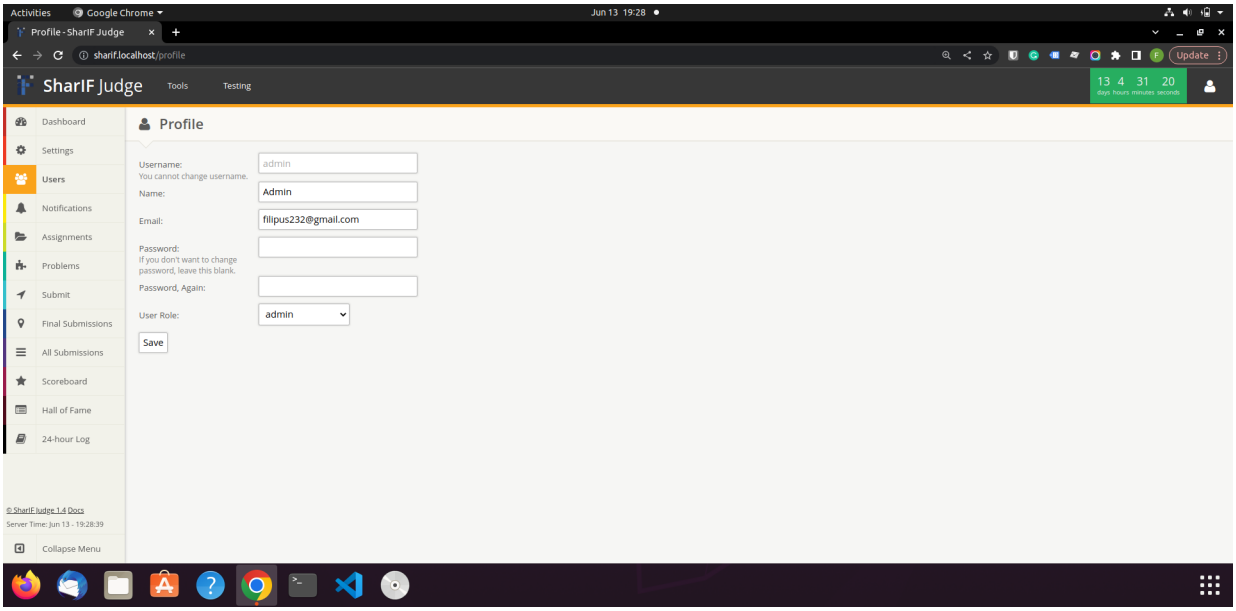
Dashboard



Gambar 3.1: Tampilan Halaman Dashboard

Gambar 3.1 merupakan tampilan halaman *dashboard* yang terdapat pada semua *role* pengguna.

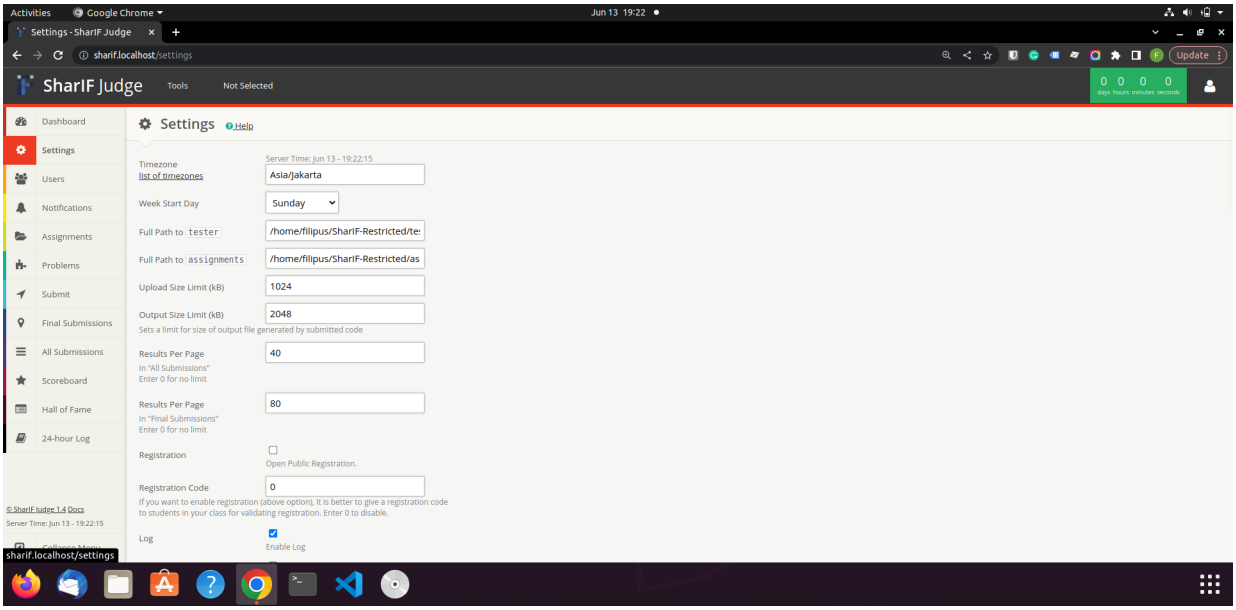
Profile



Gambar 3.2: Tampilan Halaman Profile

Gambar 3.2 merupakan tampilan halaman *profile* yang terdapat pada semua *role* pengguna. Namun, terdapat fitur yang tidak dapat digunakan oleh *siswa* dan *instructor* yakni mengganti role.

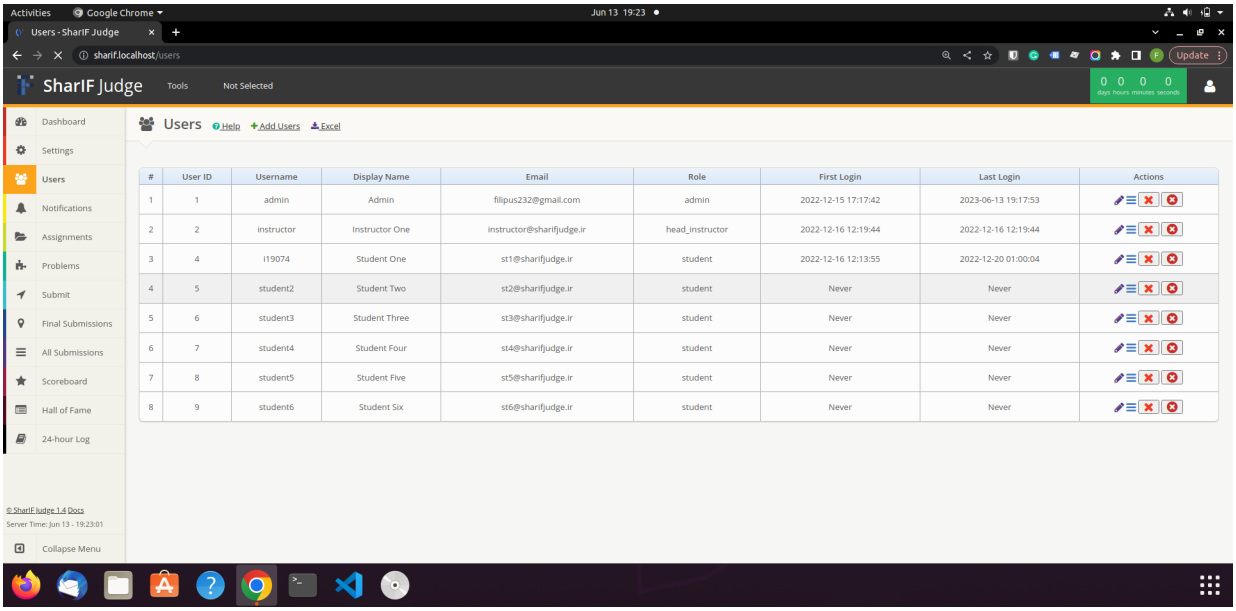
Settings



Gambar 3.3: Tampilan Halaman Settings

Gambar 3.3 merupakan tampilan halaman *settings* yang terdapat hanya pada *role* admin dan *head instructor*.

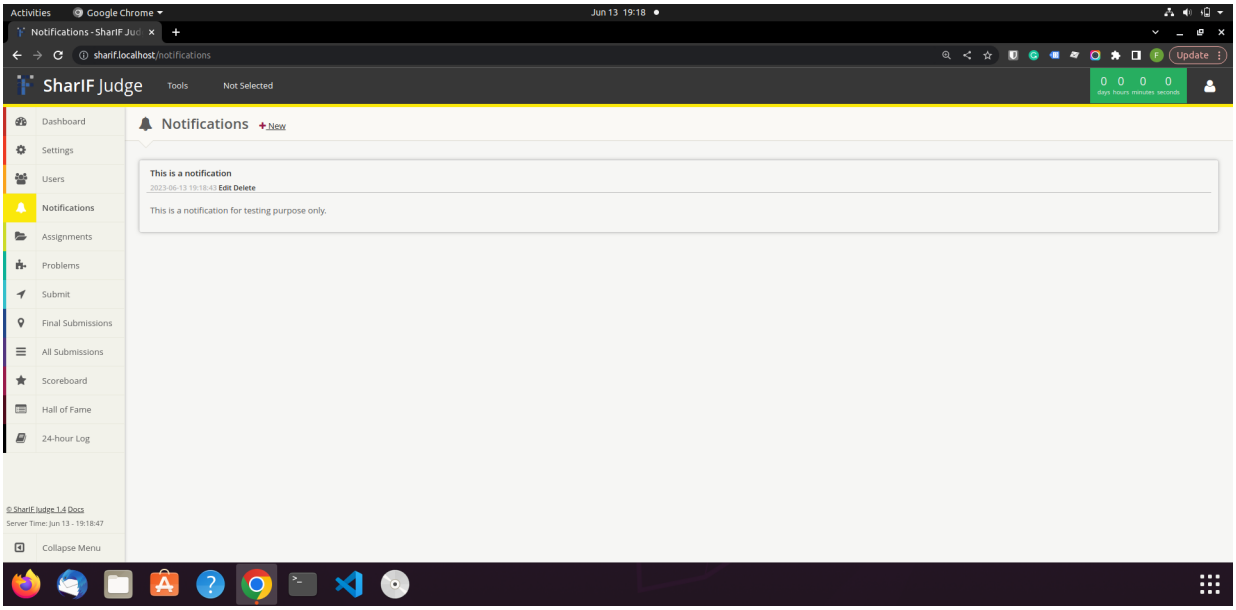
Users



Gambar 3.4: Tampilan Halaman Users

Gambar 3.4 merupakan tampilan halaman *users* yang terdapat hanya pada *role* admin dan *head instructor*.

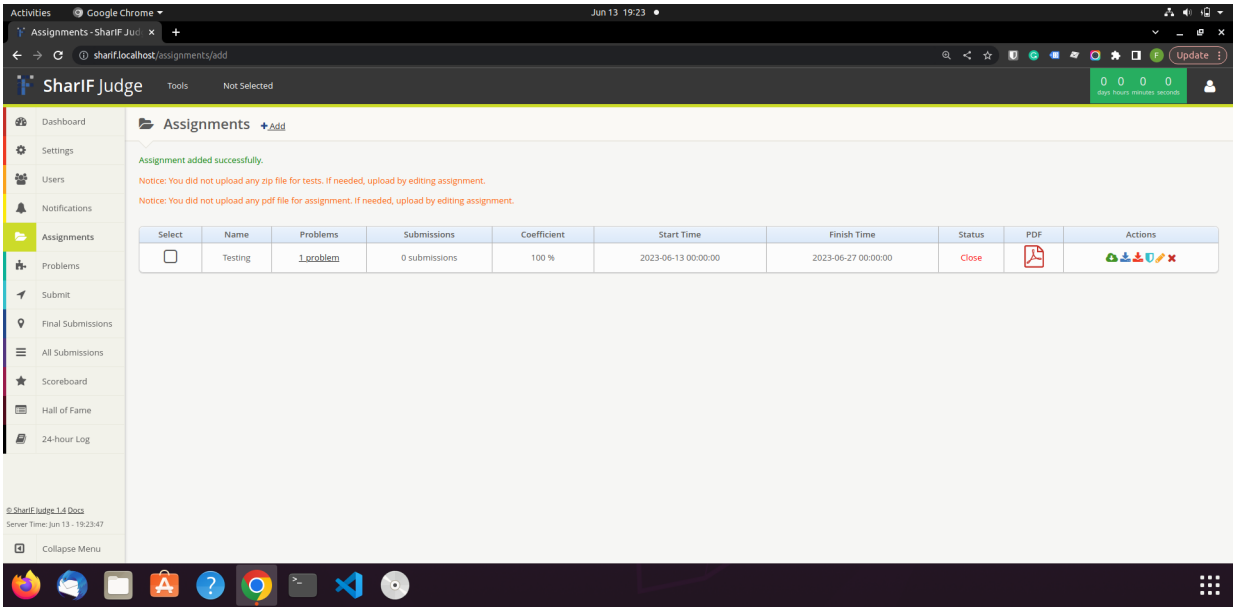
Notifications



Gambar 3.5: Tampilan Halaman Notifications

Gambar 3.5 merupakan tampilan halaman *notifications* yang terdapat pada semua *role* pengguna.

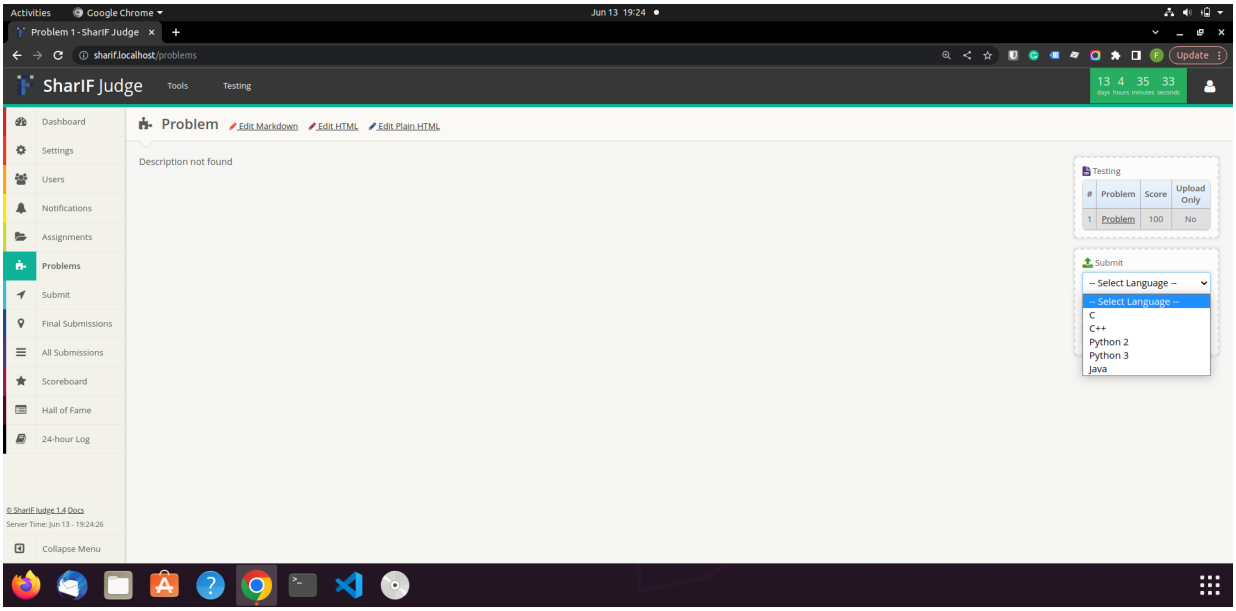
Assignments



Gambar 3.6: Tampilan Halaman Assignments

Gambar 3.6 merupakan tampilan halaman *assignments* yang terdapat pada semua *role* pengguna. Namun, terdapat bagian yang tidak dapat diakses oleh *role* siswa dan *instructor* yakni bagian *actions*.

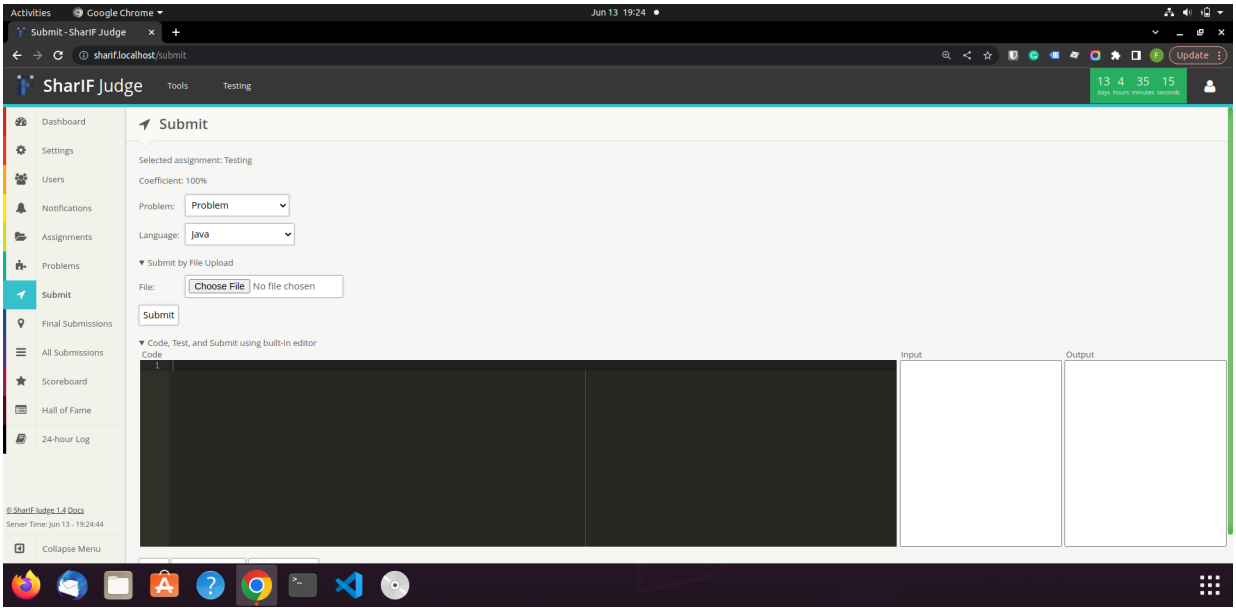
Problems



Gambar 3.7: Tampilan Halaman Problems

Gambar 3.7 merupakan tampilan halaman *problems* yang terdapat pada semua *role* pengguna.

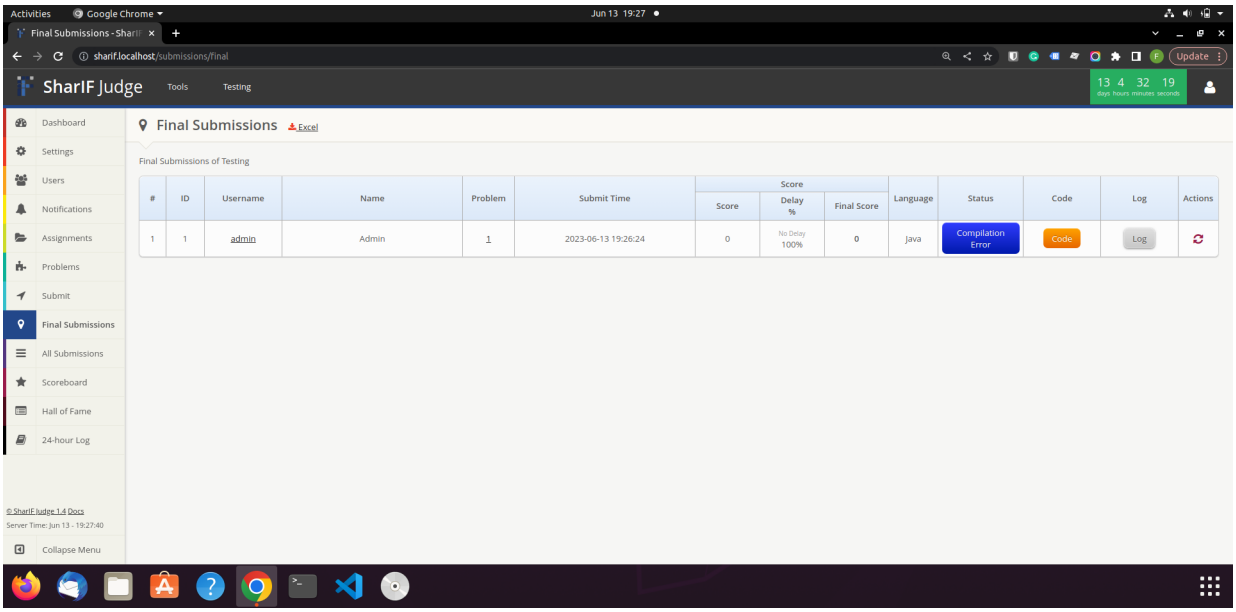
Submit



Gambar 3.8: Tampilan Halaman Submit

Gambar 3.8 merupakan tampilan halaman *submit* yang terdapat pada semua *role* pengguna.

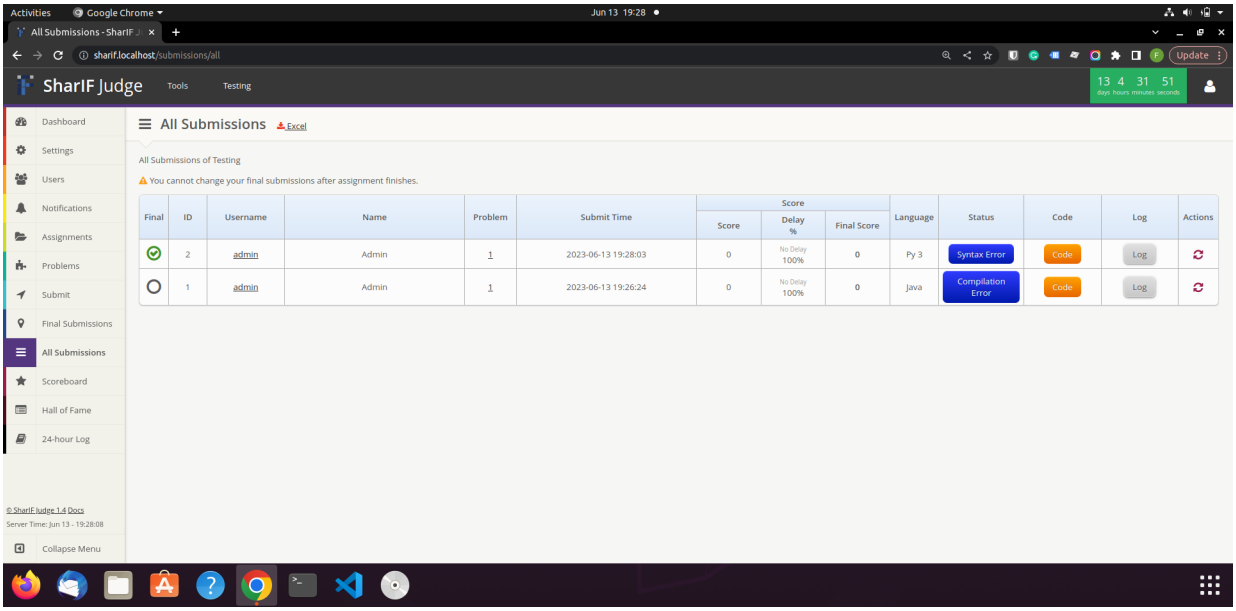
Final Submissions



Gambar 3.9: Tampilan Halaman Final Submission

Gambar 3.9 merupakan tampilan halaman *submit* yang terdapat pada semua *role* pengguna.

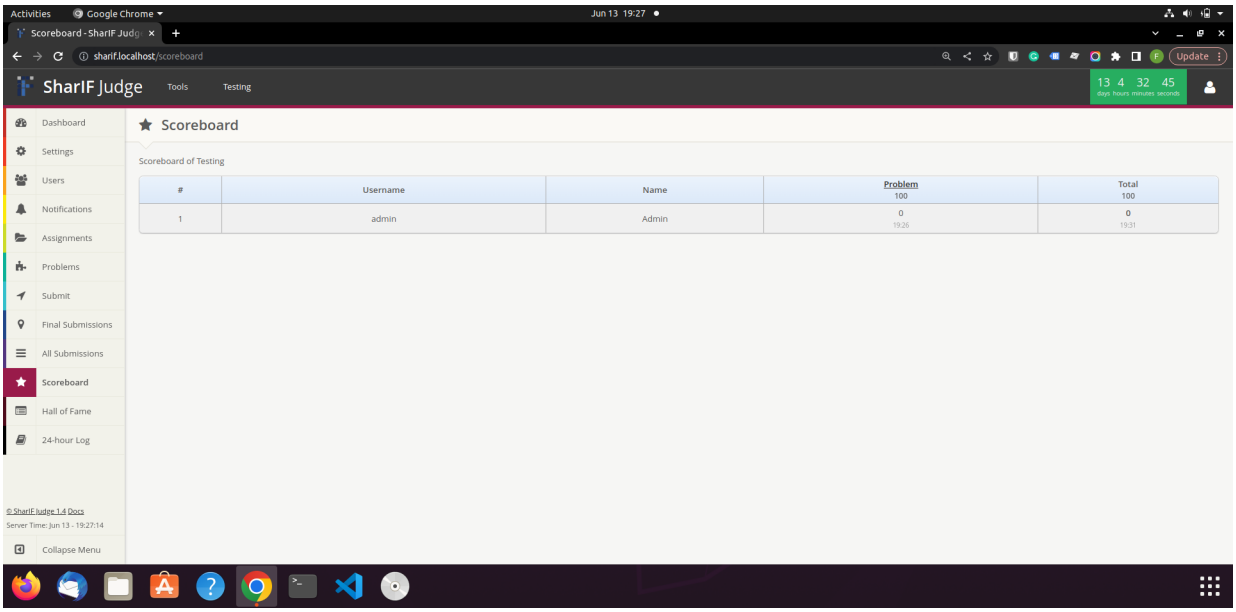
1 ***All Submissions***



Gambar 3.10: Tampilan Halaman All Submission

2 Gambar 3.10 merupakan tampilan halaman *All Submission* yang terdapat pada semua *role* pengguna.

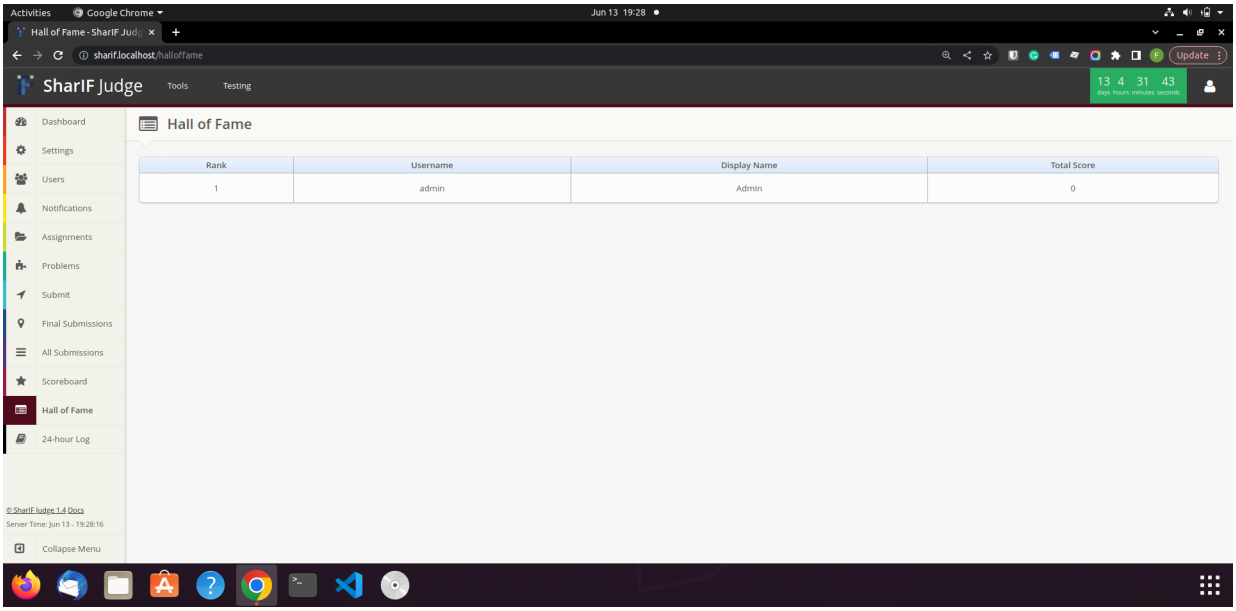
3 ***Scoreboard***



Gambar 3.11: Tampilan Halaman Scoreboard

4 Gambar 3.10 merupakan tampilan halaman *All Submission* yang terdapat pada semua *role* pengguna.

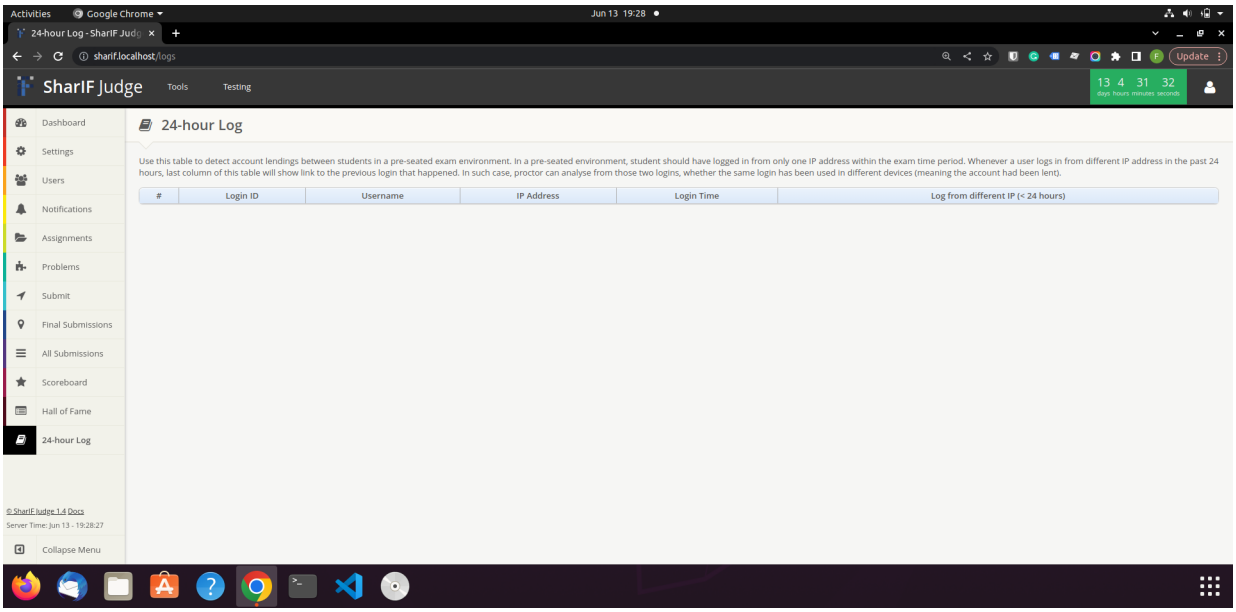
Hall of Fame



Gambar 3.12: Tampilan Halaman Hall of Fame

Gambar 3.12 merupakan tampilan halaman *Hall of Fame* yang terdapat pada semua *role* pengguna.

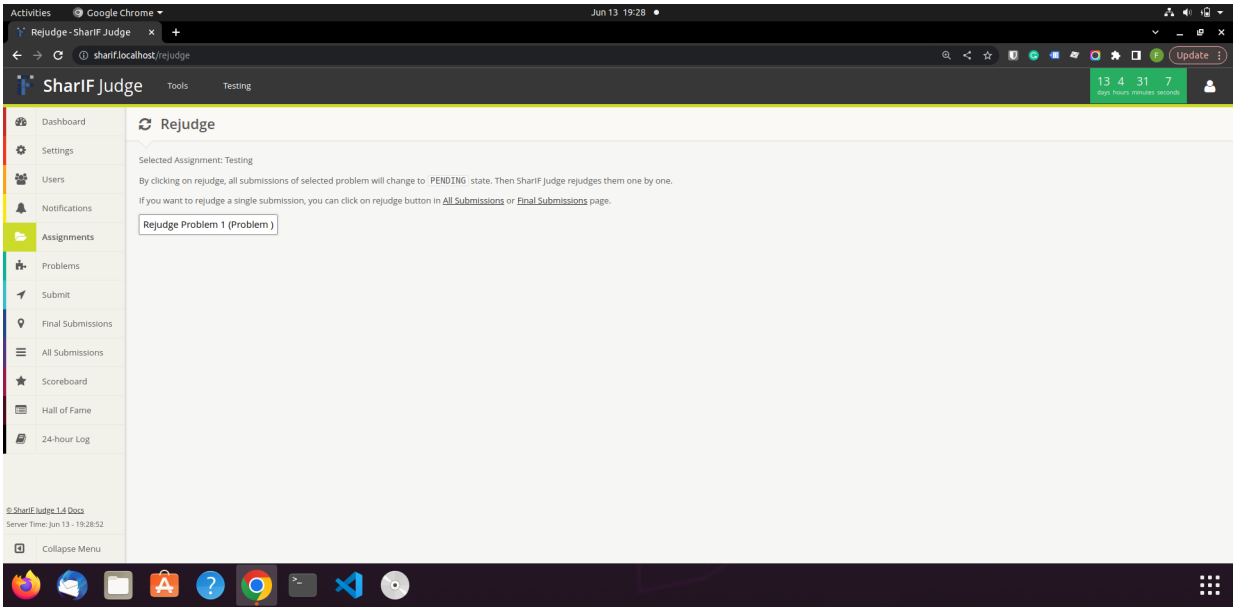
24-hour Log



Gambar 3.13: Tampilan Halaman 24-hour Log

Gambar 3.13 merupakan tampilan halaman *24-hour Log* yang terdapat hanya pada *role admin* dan *head instructor*.

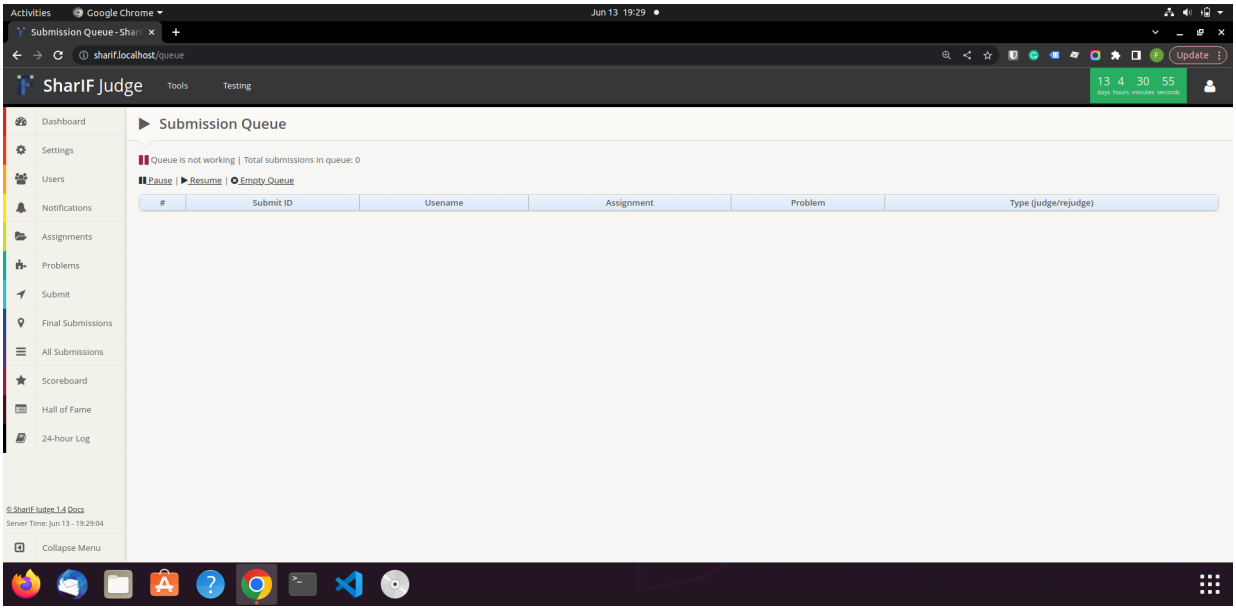
Rejudge



Gambar 3.14: Tampilan Halaman ReJudge

Gambar 3.14 merupakan tampilan halaman *ReJudge* yang terdapat hanya pada *role admin* dan *head instructor*.

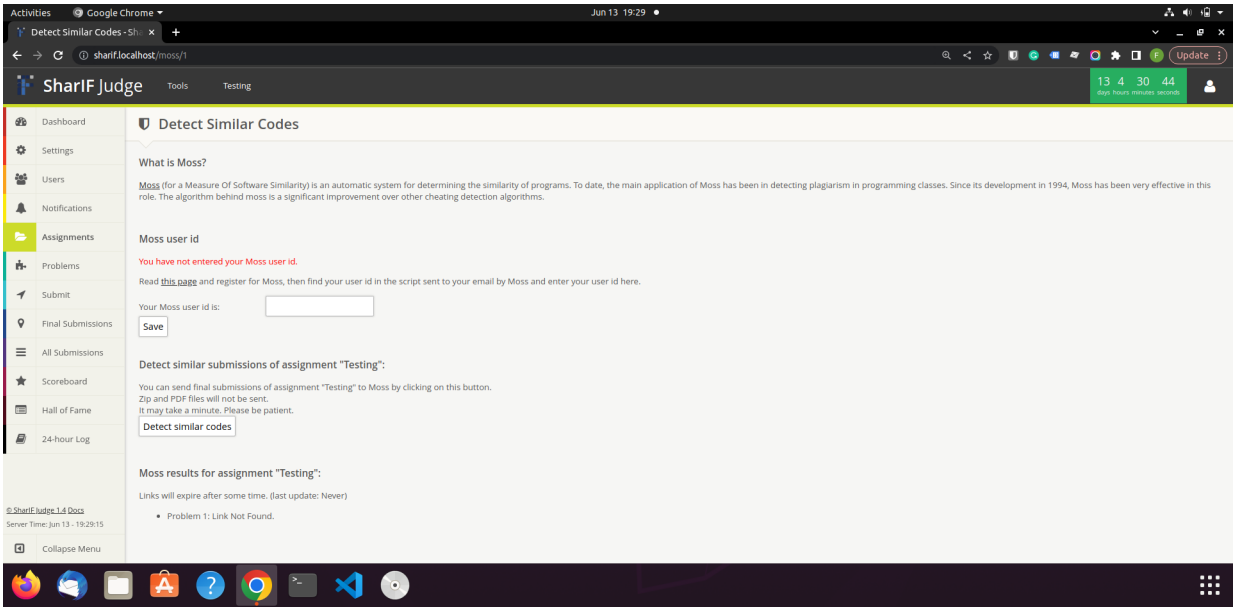
Submission Queue



Gambar 3.15: Tampilan Halaman Submission Queue

Gambar 3.15 merupakan tampilan halaman *Submission Queue* yang terdapat hanya pada *role admin* dan *head instructor*.

Cheat Detection



Gambar 3.16: Tampilan Halaman Cheat Detection

Gambar 3.16 merupakan tampilan halaman *Cheat Detection* yang terdapat hanya pada *role admin* dan *head instructor*.

3.1.3 *Controller*

Controller pada direktori `application/controller`. Direktori ini berisikan kelas *controller* dengan fungsi-fungsi dalam mengambil atau memberikan data *models* untuk dialihkan menuju *views* untuk ditampilkan. Berikut merupakan *controller* pada *SharIF Jugde* beserta fungsi-fungsinya.

`Assignments.php`

Berikut merupakan fungsi-fungsi pada *controller* `Assignments.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data menuju halaman `assignments.twig` menggunakan `Assignment_model`.
- `select`
  Fungsi ini berguna untuk memilih *assignment* menggunakan *ajax*.
- `pdf`
  Fungsi ini berguna untuk mengunduh *assignment* atau *problem* dalam bentuk pdf.
- `downloadtestsdesc`
  Fungsi ini berguna untuk mengunduh dan mengompres data *test* dan deskripsi sebuah *assignment*.
- `download_submissions`
  Fungsi ini berguna untuk mengunduh dan mengompres kode terakhir sebuah *assignment* pengguna.

- `delete`
  Fungsi ini berguna untuk menghapus *assignment*.
- `add`
  Fungsi ini berguna untuk menambah atau mengubah *assignment* berdasarkan masukan pengguna.
- `_add`
  Fungsi ini berguna untuk menambah atau mengubah *assignment*.
- `edit`
  Fungsi ini berguna untuk mengecek *role* pengguna dapat mengubah *assignment*. Selanjutnya akan dikembalikan pada fungsi `add`.
- `pdfCheck` Fungsi ini berguna untuk mengecek *file* pdf dari sebuah *assignment*.

`Dashboard.php`

Berikut merupakan fungsi-fungsi pada *controller* `Dashboard.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data menuju halaman *dashboard* menggunakan tiga buah *model*. *Model* tersebut terdiri dari `Assignment_model`, `Settings_model`, dan `Notifications_model`.
- `widget_positions`
  Fungsi ini berguna untuk menyimpan data *widget* pengguna.

`Install.php`

Controller `Install.php` hanya memiliki satu buah fungsi bernama `index`. Fungsi ini berguna untuk membentuk tabel yang dibutuhkan oleh *SharIF Judge* pada *database*. Selain itu, fungsi ini juga berguna untuk memasukan data pengguna *admin* yang pertama kali memasang *SharIF Judge* pada perangkat.

`Login.php`

Berikut merupakan fungsi-fungsi pada *controller* `Login.php`.

- `_registration_code`
  Fungsi ini beguna untuk memeriksa kode registrasi.
- `index`
  Fungsi ini berguna untuk melakukan validasi *username* dan *password* pengguna. Selain itu, fungsi ini juga memperbaharui *log* pada tabel *login*.
- `register`
  Fungsi ini berguna untuk melakukkan validasi dalam pembetukan akun.
- `logout`
  Fungsi ini berguna untuk menghancurkan *session* dari pengguna dan memindahkan pengguna ke halaman *login*.
- `lost`
  Fungsi ini berguna untuk mengirim *email* lupa password.

- `rest`
  Fungsi ini berguna untuk melakukan *reset password* pengguna.

`Logs.php`

Controller `Logs.php` hanya memiliki satu buah fungsi bernama `index`. Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *logs* menggunakan `Logs_model`.

`Moss.php`

Berikut merupakan fungsi-fungsi pada *controller* `Moss.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *moss*.
- `update`
  Fungsi ini berguna untuk memperbaharui *moss_userid* yang dimasukan oleh pengguna.
- `_detec`
  Fungsi ini berguna untuk melakukan pengecekan terhadap *submission*. !!TODO

`Notification.php`

Berikut merupakan fungsi-fungsi pada *controller* `Notification.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikam data pada halaman *notifications* menggunakan `assignment_model` dan `notifications_model`.
- `add`
  Fungsi ini berguna untuk menambahkan data *notifications*.
- `edit`
  Fungsi ini berguna untuk memperbaharui data *notifications*.
- `delete`
  Fungsi ini berguna untuk menghapus data *notifications*.
- `check`
  Fungsi ini berguna memeriksa *notifications* baru.

`Problems.php`

Berikut merupakan fungsi-fungsi pada *controller* `Problems.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data *problems* sesuai dengan *assignment* tertentu pada halaman *problems*.
- `edit`
  Fungsi ini berguna untuk memperbaharui deskripsi *problems* pada *assignment* tertentu.

`Profile.php`

Berikut merupakan fungsi-fungsi pada *controller* `Profile.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *profile*. Selain itu, fungsi ini berguna untuk melakukan pembaharuan data *profile*.
- `_password_check`
  Fungsi ini berguna untuk melakukan validasi terhadap *password* yang akan dimasukkan pengguna sesuai dengan aturan.
- `_password_again_check`
  Fungsi ini berguna untuk melakukan validasi terhadap pengulangan *password* yang dimasukkan pengguna.
- `_email_check`
  Fungsi ini berguna untuk melakukan validasi terhadap *email* yang dimasukkan pengguna
- `_role_check`
  Fungsi ini berguna untuk melakukan validasi *role* pengguna.

`Queue.php`

- `index`
  Fungsi ini berguna untuk mengambil dan meberikan data pada halaman *queue* menggunakan tiga buah *model*. *Model* tersebut adalah `Assignment_model`, `queue_model`, dan `settings_model`.
- `pause`
  Fungsi ini berguna untuk memperbaharui data pada tabel *settings*.
- `resume`
  Fungsi ini berguna untuk melanjutkan proses *queue*.
- `empty_queue`
  Fungsi ini berguna untuk menghapus data tabel *queue*.

`Queueprocess.php`

Controller `Queueprocess.php` hanya memiliki satu buah fungsi bernama `index`.Fungsi ini berguna untuk menjalankan proses *judge* berdasarkan *queue* satu demi satu sesuai antrean. Fungsi ini menggunakan beberapa *model* yaitu `Queue_model`, `Submit_model`, `Assignments_model`, dan `Settings_model`.

`Rejudge.php`

Berikut merupakan fungsi-fungsi pada *controller* `Rejudge.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *rejudge* menggunakan `Assignment_model`.
- `rejudge_single`
  Fungsi ini berguna untuk melakukan *rejudge* pada satu buah masalah tertentu.

Scoreboard.php

Berikut merupakan fungsi-fungsi pada *controller* `Scoreboard.php`.

- `index`
  Fungsi ini berguna untuk memberikan data pada halaman *scoreboard* menggunakan dua buah *model*. *Model* tersebut adalah `Assignment_model` dan `Scoreboard_model`.

Server_time.php

Controller `Server_time.php` hanya memiliki satu buah fungsi bernama `index`. Fungsi ini berguna untuk mengeluarkan *server_time*.

Settings.php

Berikut merupakan fungsi-fungsi pada *controller* `Settings.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *settings* menggunakan `Settings_model` dan `Assignment_model`.
- `update`
  Fungsi ini berguna untuk mengambil masukan dan memperbaharui data pada halaman *settings* berdasarkan masukan tersebut. Data tersebut nantinya akan disimpan pada *database* menggunakan fungsi `Settings_model`.

Submission.php

Berikut merupakan fungsi-fungsi pada *controller* `Submission.php`.

- `_download_excel`
  Fungsi ini berguna untuk mengubah data-data dari *submission* yang dipilih menjadi format *excel*.
- `final_excel`
  Fungsi ini berguna untuk mengunduh data *final submissions*.
- `all_excel`
  Fungsi ini berguna untuk mengunduh data seluruh *submissions*.
- `the_final`
  Fungsi ini berguna untuk memberikan data pada halaman *Final Submissions* menggunakan beberapa *model*. *Model* tersebut terdiri dari `Submit_model`, `Settings_model`, dan `User_model`.
- `all`
  Fungsi ini berguna untuk memberikan data pada halaman *All Submissions* menggunakan beberapa *model*. *Model* tersebut terdiri dari `Submit_model`, `Settings_model`, dan `User_model`.
- `select`
  Fungsi ini berguna untuk memilih *submission* yang akan dijadikan *submission final* oleh pengguna.
- `_check_type`
  Fungsi ini berguna untuk melakukan pengecekan tipe *submission* yang telah dikumpulkan oleh pengguna.

- `view_code`
  Fungsi ini berguna untuk memperlihatkan *submission* yang telah dikumpulkan oleh pengguna sesuai dengan tipenya.
- `download_file`
  Fungsi ini berguna untuk mengunduh hasil dari *submission* yang telah dikumpulkan oleh pengguna.

`Submit.php`

Berikut merupakan fungsi-fungsi pada *controller* `Submit.php`.

- `_language_to_type`
  Fungsi ini berguna untuk mengubah bahasa pemrograman menjadi tipe sesuai dengan pilihan pengguna.
- `_language_to_ext`
  Fungsi ini berguna untuk mengubah bahasa pemrograman menjadi ekstensi sesuai dengan pilihan pengguna.
- `_match`
  Fungsi ini berguna untuk mencocokan tipe dengan ekstensi dari bahasa pemrogramannya.
- `_check_language`
  Fungsi ini berguna untuk melakukan pengecekan terhadap bahasa pemrograman yang digunakan.
- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *submit* menggunakan `Assignment_model`.
- `_upload`
  Fungsi ini berguna untuk menyimpan jawaban dan memasukannya ke *queue* untuk dinilai.
- `load`
  Fungsi ini berguna untuk memuat kode dari *editor file*.
- `save`
  Fungsi ini berguna untuk meyimpan kode menuju *editor file* dan mengirim ataupun menjalankannya.
- `_submit`
  Fungsi ini berguna untuk menambahkan kode pada *queue* untuk dilakukan *judge*.
- `_execute`
  Fungsi ini berguna untuk menambahkan kode untuk dijalankan atau di *queue*.
- `get_output` Fungsi ini berguna untuk memuat *file* menjadi hasil eksekusi.

`User.php`

Berikut merupakan fungsi-fungsi pada *controller* `User.php`.

- `index`
  Fungsi ini berguna untuk mengambil dan memberikan data pada halaman *users*.
- `add`
  Fungsi ini berguna untuk menambahkan pengguna baru sesuai dengan masukan.

- `delete`
  Fungsi ini berguna untuk menghapus pengguna yang dipilih.
- `delete_submissions`
  Fungsi ini berguna untuk menghapus *submission* dari sebuah pengguna.
- `list_excel`
  Fungsi ini berguna untuk menghasilkan dan mengunduh data pengguna pada format *excel*.

3.2 *Library*

SharIF Judge menggunakan beberapa *library* yang dibentuk secara manual maupun yang sudah tersedia pada *Codeigniter 3*. Berikut merupakan *library* yang dipakai oleh *SharIF Judge*:

Unzip

Unzip merupakan sebuah *library* yang dibentuk oleh Phil Sturgeon. *Library* ini mewajibkan pengguna untuk menyalakan *extension Zlib* sebelum dapat digunakan. *Library Unzip* berfungsi untuk mengextraksi file dengan *extension* `.zip` menuju direktori yang ditentukan dan dapat mengeluarkan *error* yang sesuai. *Library* ini juga dapat memberi batasan *extension* apa yang diinginkan dari file tersebut. Kode 3.1 merupakan contoh penggunaan *library Unzip* pada *SharIF Judge*.

Kode 3.1: Contoh kode penggunaan *Library Unzip*

```
1
2
3
```

```
$this->load->library('unzip');
$this->unzip->allow(array('txt', 'cpp', 'html', 'md', 'pdf'));
$extract_result = $this->unzip->extract($u_data['full_path'], $tmp_dir);
```

Kode 3.1 merupakan contoh penggunakan *library Unzip*. Sintaks `$this->load->library('unzip');` berfungsi untuk melakukan *load library Unzip* agar dapat digunakan pada fungsi tersebut. Selanjutnya sintaks `$this->unzip->allow(array('txt', 'cpp', 'html', 'md', 'pdf'));` berfungsi untuk memberikan batasan *extension* apa saja yang diinginkan dari file tersebut. Terakhir sintaks `$extract_result = $this->unzip->extract($u_data['full_path'], $tmp_dir);` berfungsi untuk melakukan ekstraksi terhadap file zip pada `$u_data['full_path']` tersebut menuju direktori pada variabel tmp_dir.

Twig

Twig merupakan sebuah *template engine library* yang digunakan untuk mempermudah dalam membentuk *view* pada aplikasi. *Twig* terintegrasi dengan fungsi-fungsi pada *CodeIgniter 3* sehingga dapat menggunakan seluruh fungsi yang terdapat pada *CodeIgniter 3*. Kode 3.2 merupakan contoh penggunaan *Twig* pada *SharIF Judge*.

Kode 3.2: Contoh *view* menggunakan *library Twig*

```
1
2
3
4
5
6
7
8
```

```
{% set title = 'Login' %}
{% include 'templates/simple_header.twig' %}

{{ form_open() }}
    <div class="box login">

        <div class="judge_logo">
            <a href="{{ site_url() }}"><img src="{{ base_url('assets/images/banner.png') }}"/></a>
```

```
        </div>

        <div class="login_form">
            <div class="login1">
                <p>
                    <label for="form_username">Username</label><br/>
                    <input id="form_username" type="text" name="username" required="required" pattern="[0-9a-z]{3,20}" title="The
                        Username field must be between 3 and 20 characters in length, and contain only digits and lowercase
                        letters" class="sharif_input" value="{{ set_value('username') }}" autofocus="autofocus"/>
                    {{ form_error('username', '<div class="shj_error">', '</div>') }}
                </p>
                <p>
                    <label for="form_password">Password</label><br/>
                    <input id="form_password" type="password" name="password" required="required" pattern=".{6,200}" title="The
                        Password field must be at least 6 characters in length" class="sharif_input"/>
                    {{ form_error('password', '<div class="shj_error">', '</div>') }}
                </p>
                {% if error %}
                    <div class="shj_error">Incorrect username or password.</div>
                {% endif %}
            </div>
            <div class="login2">
                <p style="margin:0;">
                    {% if registration_enabled %}
                    <a href="{{ site_url('register') }}">Register</a> |
                    {% endif %}
                    <a href="{{ site_url('login/lost') }}">Reset Password</a>
                    <input type="submit" value="Login" id="sharif_submit"/>
                </p>
            </div>
        </div>

    </div>
</form>
</body>
</html>
```

Twig pada *view SharIF Judge* menggunakan dua buah *delimiters* yakni `{{ }}` dan `{% %}`. *Delimiters* `{{ }}` memiliki fungsi untuk mengembalikan *expression* seperti variabel ataupun fungsi *CodeIgniter 3*. Contoh fungsi yang dikembalikan oleh *delimiters* pada kode diatas adalah `form_open` yang merupakan sebuah fungsi pada *CodeIgniter 3* untuk membuka *tag form*. Sedangkan *delimiters* `{% %}` memiliki fungsi untuk mengeksekusi fungsi PHP seperti *for-loops* atau *if else*. Contoh fungsi yang dieksekusi pada kode diatas adalah `if` yang berfungsi untuk mengecek kondisi tertentu.

Password_hash

Password_hash merupakan sebuah *library* yang dibentuk oleh *phpass*. *Library* ini berfungsi untuk melakukan enkripsi *password* dan melakukan verifikasi *password*. *Library* ini mendukung beberapa metode enkripsi antara lain *CRYPT_BLOWFISH* dan *CRYPT_EXT_DES*. Kode 3.3 merupakan contoh penggunaan *library* ini pada *SharIF Judge*.

Kode 3.3: Contoh kode penggunaan *Library Password_hash*

```
    $this->load->library('password_hash', array(8, FALSE));
    $user['password'] = $this->password_hash->HashPassword($this->input->post('password'));
```

Kode 3.3 merupakan contoh penggunaan *library Passwors_hash*. Sintaks pada baris pertama akan melakukan *load* pada *library Password_hash*. Sintaks pada baris selanjutnya akan melakukan *hashing* pada *input* menggunakan algoritma yang ditentukan dan menyimpannya pada sebuah variabel.

MY_Form_validation

MY_Form_validation merupakan *library* yang dibentuk secara manual untuk menambahkan fungsi validasi yang sudah tersedia pada *CodeIgniter 3*. *Library* ini memiliki dua buah fungsi yakni:

- `required`
  Fungsi ini berguna untuk melakukan pengecekan terhadap sebuah *input* apakah berisikan sebuah *array* kosong ataupun *string* kosong.
- `lowercase`
  Fungsi ini berguna untuk melakukan pengecekan apakah *input* berisikan kata-kata dengan huruf kecil atau tidak.

MY_Profiler

Parsedown

Parsedown merupakan sebuah *library* yang dibentuk oleh Emanuil Rusev. *Library* ini berfungsi untuk mengubah teks dengan sintaks *markdown* menjadi teks dalam bentuk file lain seperti HTML. Kode 3.4 merupakan contoh penggunaan *library parsedown*.

Kode 3.4: Contoh kode penggunaan *Library Parsedown*

```
1 $this->load->library('parsedown');
2 $html = $this->parsedown->parse(file_get_contents("$assignment_dir/p$i/desc.md"));
```

Kode 3.4 akan menginisiasi *library parsedown* pada sintaks baris pertama. Selanjutnya isi dari file `desc.md` akan diambil dan dilakukan *parsedown* menjadi file HTML. Berikut merupakan contoh dari teks sebelum dan sesudah di *parsedown*:

Ini adalah list :

1. Satu
2. Dua
3. Tiga

Kode 3.5: Contoh kode sesudah dilakukan *parsedown*

```
1 <p>Ini adalah list</p>
2 <ol>
3 <li>Satu</li>
4 <li>Dua</li>
5 <li>Tiga</li>
```

Kode 3.5 merupakan contoh kode HTML sesudah dilakukan *parsedown*. Angka 1, 2, dan 3 akan diubah menjadi *list* yang dapat ditampilkan pada HTML.

Phpexcel

Phpexcel merupakan sebuah *library* yang dibentuk oleh *PHPExcel*. *Library* ini memiliki fungsi untuk mengubah data yang ada pada PHP menjadi file excel.

Shj_pagination

Shj_pagination merupakan *library* yang dibentuk secara manual dengan fungsi untuk membatasi maksimal data pada setiap halaman sesuai dengan konfigurasinya.

upload

Library ini merupakan fungsi yang tersedia pada *CodeIgniter 3*. *Library* ini berguna untuk menerima masukan dan mengunggah file yang dimasukan oleh pengguna.

email

Library ini merupakan fungsi yang tersedia pada *CodeIgniter 3*. *Library* ini berguna untuk mengirimkan *email* kepada orang yang dituju sesuai dengan konfigurasinya.

form_validation

Library ini merupakan fungsi yang tersedia pada *CodeIgniter 3*. *Library* ini berguna untuk melakukan validasi terhadap data yang dimasukan oleh pengguna sesuai dengan aturan yang telah ditetapkan.

3.3 Analisis Sistem Usulan

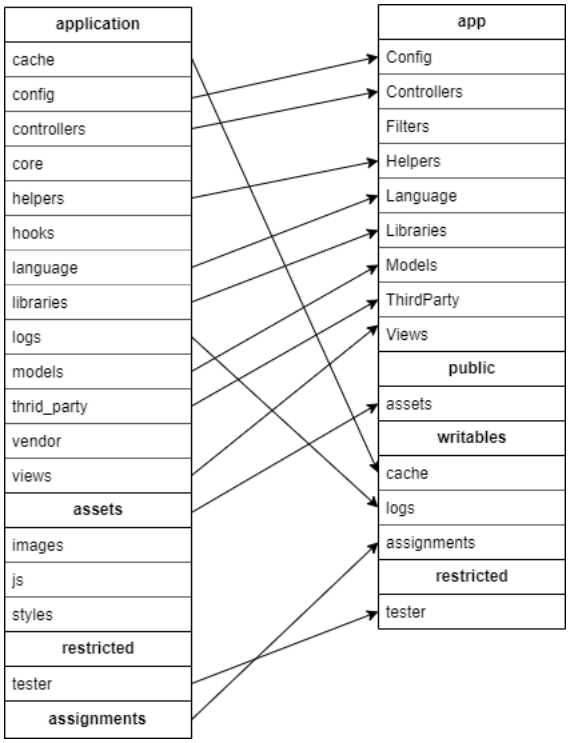
Konversi *CodeIgniter 3* menuju *CodeIgniter 4* diperlukan penulisan ulang karena terdapat perubahan struktur aplikasi dan beberapa fungsi yang memiliki pemanggilan berbeda dan harus dilakukan pembaharuan.

3.3.1 Persiapan *CodeIgniter 4*

Konversi dimulai dengan mempersiapkan aplikasi *CodeIgniter 4* dengan mengunduh ataupun memasangnya melalui *Composer*. Pengguna juga perlu memasang komponen pendukung seperti *Twig*, *phpoffice*, *radius*, dan *adldap2*.

3.3.2 Struktur Aplikasi

Struktur aplikasi pada *CodeIgniter 3* dan *CodeIgniter 4* memiliki perubahan sehingga perlu dilakukan pemindahan *file-file* menuju *CodeIgnter 4*. Gambar 3.17 merupakan pemindahan struktur aplikasi *SharIF Judge* pada *CodeIgniter 3* menuju *CodeIgniter 4*.



Gambar 3.17: *Pemindahan struktur aplikasi menuju CodeIgniter 4*

Berikut merupakan rincian direktori yang akan dipindahkan menuju *CodeIgniter 4*.

Application

Direktori-direktori `application` pada *CodeIgniter 3* akan dipindahkan dengan penyesuaian menuju direktori `app` terkecuali direktori `vendor`, `cache` dan `core`. Berikut merupakan direktori yang dipindahkan dari direktori *application* menuju direktori *app*.

- `application/config` akan dipindahkan menuju `app/Config`.
- `application/controllers` akan dipindahkan menuju `app/Controllers`.
- `application/helpers` akan dipindahkan menuju `app/Helpers`.
- `application/languange` akan dipindahkan menuju `app/Languange`.
- `application/libraries` akan dipindahkan menuju `app/Libraries`.
- `application/models` akan dipindahkan menuju `app/Models`.
- `application/views` akan dipindahkan menuju `app/views`.

Public

CodeIgniter 3 tidak menyediakan direktori akar berupa *public* sehingga terdapat perubahan struktur dimana direktori yang sebelumnya ada pada `application` akan dipindah menuju direktori `public`. Berikut merupakan direktori yang dipindahkan menuju direktori `public`.

- `assets` akan dipindahkan menuju `public/assets`.

Selain stuktur aplikasi diatas, *SharIF Judge* memiliki dua buah direktori terpisah diluar direktori utama bernama *assignments* dan *tester*. Direktori *assignments* ini berfungsi untuk menyimpan seluruh *file* yang telah dikumpulkan sedangkan direktori *tester* digunakan untuk melakukan perco-

baan untuk keamanan *sandbox*. Kedua direktori ini harus dapat ditulis oleh *PHP* sehingga akan dipindahkan menuju direktori *writables*.

Writables

Direktori ini merupakan direktori berisikan seluruh direktori yang dapat ditulis oleh *PHP*. Berikut merupakan direktori yang dipindahkan menuju *writables*.

- `application/cache` akan dipindahkan menuju `writables/cache`.
- `application/log` akan dipindahkan menuju `writables/logs`.
- `assignments` akan dipindahkan menuju `writables/assignments`.

3.3.3 *Routing*

Routing pada aplikasi *SharIF Judge* menggunakan *auto routing* yang telah disediakan oleh *CodeIgniter 3*. *Auto routing* akan membentuk *url* sesuai dengan *controller* dan *method* yang telah dibentuk tanpa harus didefinisikan secara manual. Penggunaan *auto routing* seperti pada *CodeIgniter 3* memiliki kekurangan pada bagian keamanan dimana *filter* pada *controller* dan proteksi *CSRF* akan dilewati. Sehingga, konversi pada aplikasi *SharIF Judge* akan menggunakan *URI Routing* yang didefinisikan secara manual untuk alasan keamanan dan *url* yang fleksibel. Berikut merupakan contoh *route* yang didefinisikan secar a manual.

```
$routes->post("login/register",'Login::register');
```

Route akan didefinisikan secara eksplisit sesuai dengan fungsi dan metodenya.

3.3.4 *Model, View, and Controller*

CodeIgniter 4 memiliki perubahan baik dari kegunaan dan cara pemanggilan *Model, View, and Controller*. Berikut merupakan perubahan yang terjadi:

Model

Model pada *CodeIgniter 4* memiliki perubahan dimana *model* dapat digunakan untuk mengambil data pada satu buah tabel spesifik. Konversi *model* dari *CodeIgniter 3* menuju *CodeIgniter 4* dapat dilakukan menggunakan dua buah cara yakni:

1. Menggunakan *Model* dari *CodeIgniter 3*
   Model pada *CodeIgniter 3* memiliki kekurangan dimana pengguna harus membentuk secara manual seluruh fungsi untuk mengambil, memasukan, dan memperbaharui data dari sebuah tabel spesifik pada *model* tersebut. Kode 3.6 merupakan contoh fungsi yang digunakan untuk mengambil data dari sebuah tabel.

Kode 3.6: Contoh fungsi untuk mengambil data seluruh user

```
1 public function get_all_users()
2     {
3         return $this->db->order_by('role', 'asc')->order_by('id')->get('users')->result_array();
4     }
```

Kode 3.6 mengambil data dari tabel `users` dengan hasil berupa *array* dan diurutkan sesuai dengan `role` dan `id`nya.

2. Menggunakan *Model* pada *CodeIgniter 4*

   CodeIgnier 4 menyediakan fungsi *model* yang dapat dibentuk melalui *command line* untuk sebuah tabel spesifik. Pengguna dapat membentuk *model* melalui *command line* menggunakan sintaks sebagai berikut.

```
make:model <name>
```

   Model pada *CodeIgniter 4* menyediakan fungsi untuk mengambil, memasukan, dan memperbaharui data dari sebuah tabel spesifik tanpa harus membentuk secara manual fungsi-fungsi tersebut. Pengguna dapat melakukan inisiasi dan memakai fungsi untuk mengambil data menggunakan kode berikut.

```
$userModel = new \App\Models\UserModel();

$user = $userModel->findAll();
```

Kode diatas akan mengambil seluruh data dari `$userModel` sesuai dengan konfigurasi yang telah dilakukan pengguna. Pengguna juga dapat melakukan *create, update,* dan *delete* melalui kode berikut.

Kode 3.7: Contoh kode untuk menghapus data pada model

```
$this->Notifications_model->delete('notifications', array('id' => $id));
```

Konversi aplikasi *SharIF Judge* akan menggunakan kedua buah cara dengan penghapusan beberapa fungsi yang terdapat pada *model* dari *CodeIgniter 4* seperti mengambil, menghapus, dan menambahkan data. Sedangkan untuk fungsi-fungsi lain pada *SharIF Judge* akan dilakukan pembaharuan sesuai dengan dokumentasi yang telah ada.

View

View pada aplikasi *SharIF Judge* menggunakan *template engine* bernama *Twig*. *Twig* merupakan sebuah *template engine* untuk bahasa pemrograman *PHP* yang berguna untuk mempermudah dalam mebentuk tampilan sebuah aplikasi. *Twig* tidak terintegrasi pada *CodeIgniter 4* sehingga akan terdapat beberapa pemasangan dan perubahan pada sintaks yang telah dipasang. Selain itu, terdapat beberapa perubahan fungsi pada *CodeIgniter 4* sehingga perlu dilakukan penyesuaian seperti pengubahan *file extension* dari `.twig` menjadi `.php`. Konversi *SharIF Judge* menuju *CodeIgniter 4* akan mengubah *view* yang sebelumnya menggunakan *twig* menjadi menggunakan *PHP* sesuai dengan dokumentasi *CodeIgniter 4*. Seluruh *delimiters* akan diubah menggunakan fitur yang terdapat pada *CodeIgniter 4*. Kode 3.8 merupakan contoh konversi yang dilakukan.

Kode 3.8: Contoh *view* menggunakan *twig*

```
{% if registration_enabled %}
    <a href="{{ site_url('register') }}">Register</a> |
{% endif %}
```

menjadi kode berikut:

Kode 3.9: Contoh *view* menggunakan *php*

```
<?php if($registration_enabled): ?>
    <a href="<?= site_url('register') ?>">Register</a> |
<?php endif ?>
```

Seluruh sintaks *twig* akan diubah menjadi sintaks *PHP* dari *CodeIgniter 4* seperti yang terdapat pada kode 3.9.

Controller

Controller pada *CodeIgniter 4* memiliki fungsi sama dengan pada *CodeIgniter 3* sehingga hanya perlu dilakukan penghapusan dan perubahan pada sintaks yang ada. Namun, terdapat perubahan pada `constructor` dimana pada *CodeIgniter 4* terdapat `initController`. *Constructor* pada *PHP* tidak diperbolehkan untuk mengembalikan apapun sehingga terdapat beberapa pemindahan fungsi seperti `redirect()` menuju *filters*. Selain itu, konversi akan tetap menggunakan `__construct` dengan pemindahan beberapa sintaks menuju `initController` seperti pemanggilan *helpers* dan variabel yang dapat diakses pada seluruh *controller*. Berikut merupakan contoh penggunaan `initController` untuk variabel.

Kode 3.10: Contoh penggunaan initController untuk variabel

```
protected $config;

public function initController(RequestInterface $request, ResponseInterface $response, LoggerInterface $logger)
    {
        // Do Not Edit This Line
        parent::initController($request, $response, $logger);

        // Preload any models, libraries, etc, here.

        // E.g.: $this->session = \Config\Services::session();
        $this->config = Config('Secrets');
    }
```

Kode 3.10 akan memanggil variabel `$config` berisikan data dari *file config* `Secrets.php` yang dapat digunakan seluruh *controller*.

Fungsi lainnya akan dipindahkan sesuai dengan yang ada pada *CodeIgniter 3* dengan pembaharuan sesuai dengan dokumentasi *CodeIgniter 4*. Selain pembaharuan, akan terdapat pemindahan variabel *global* yang sebelumnya telah diinisiasikan menuju *controller*.

3.3.5 *Libraries*

Libraries pada *CodeIgniter 3* memiliki perubahan dan penghapusan pada *CodeIgnier 4* sehingga perlu dilakukan pembaharuan. Berikut merupakan *libraries* yang dipakai pada *SharIF Judge*:

Emails

Emails pada *CodeIgnier 4* terdapat perubahan sintaks dan cara pemanggilan sehingga akan dipindahkan sesuai dengan sintaks yang baru. Sintaks berubah dari yang sebelumnya menggunakan *snakecase* menjadi menggunakan *camelcase*. Kode 4.6 merupakan contoh penggunaan *library email*.

Kode 3.11: Contoh perubahan *library emails*

```
$this->email->setFrom($this->settings_model->get_setting('mail_from'), $this->settings_model->get_setting('mail_from_name'));
                $this->email->setTo($user[1]);
                $this->email->setSubject('SharIF Judge Username and Password');
                $text = $this->settings_model->get_setting('add_user_mail');
                $text = str_replace('{SITE_URL}', base_url(), $text);
                $text = str_replace('{ROLE}', $user[4], $text);
                $text = str_replace('{USERNAME}', $user[0], $text);
                $text = str_replace('{PASSWORD}', htmlspecialchars($user[3]), $text);
                $text = str_replace('{LOGIN_URL}', base_url(), $text);
                $this->email->setMessage($text);
                $this->email->send()
```

Kode 4.6 memiliki sintaks dengan nama sama namun terdapat perubahan menjadi *camelcase*.

Working with Uploaded Files

Upload akan digantikan dengan fungsi *Working with oploaded files* dengan beberapa penggantian dan penghapusan fungsi. *Working with uploaded files* terdapat perubahan pada beberapa sintaks dan validasi terhadap *file* yang telah diunggah. Konversi aplikasi *SharIF Judge* akan menggunakan fungsi ini dengan beberapa perubahan sintaks sesuai dengan dokumentasinya.

Validation

Form_validation akan digantikan dengan fungsi *validation* dengan perubahan dan pengapusan beberapa fungsi. Berikut merupakan contoh pembentukan aturan untuk mengumpulkan sebuah data pada *form*.

```
$validate->setRule('username', 'username', 'requiredmin_length[3]|max_length[20]);|
```

Sintaks diatas akan melakukan validasi terhadap *input* yang akan masukan oleh pengguna. Namun, *CodeIgniter 4* tidak menyediakan fungsi `form_error` sehingga akan diubah dengan menggunakan fungsi baru bernama `validation_errors()`. Fungsi tersebut dapat digunakan untuk mengembalikan *error* apabila terdapat data yang tidak sesuai dengan aturan. *Error* tersebut dapat ditampilkan pada halaman *view* menggunakan sintaks berikut.

```
<?= $validation->getError('username'); ?>
```

Sintaks diatas akan mengembalikan *error* terhadap *form* dengan nama *username* apabila tidak sesuai dengan aturan yang sudah ditentukan. Variabel `validation` akan dikirimkan dari *controller* berisikan *library* dari *validation* tersebut.

Zip Archive

Zip Encoding akan digantikan dengan fungsi PHP *zip archive* karena sudah tidak tersedia pada *CodeIgniter 4*. Fungsi *zip archive* terdapat beberapa perbedaan sehingga akan disesuaikan dengan fungsi-fungsi yang ada.
Library yang terdapat pada *CodeIgniter 4* juga dapat di*extend* dan dibentuk sesuai dengan kebutuhan. Berikut merupakan *library* yang dibentuk oleh pengguna. Berikut merupakan *library* yang dibentuk oleh pengguna.

Twig

Library ini tidak akan digunakan untuk membentuk *view* pada *CodeIgniter 4* namun, akan ada penggunaan sebuah fungsi *Twig* yang akan dibentuk pada direktori `app/Libraries`. Fungsi tersebut bernama `extra_time_formatter` yang memiliki fungsi untuk mengubah input yang diberikan menjadi format jam dikali enam puluh menit.

Unzip

Library ini akan digunakan kembali dan dipindahkan menuju direktori `app/Libraries`. *Library* terdapat pengapusan sintaks *defined* dan juga penambahan *namespace*.

Password_hash

Library ini tidak akan digunakan dan akan digantikan oleh *password hash* yang disediakan oleh PHP .*Library Password_hash* merekomendasikan pegguna untuk menggunakan fungsi *native* yang disediakan oleh PHP apabila aplikasi mendukung PHP versi 5.5 ke atas. Sehingga, akan dilakukan konversi menggunakan fungsi yang disediakan oleh PHP bernama `password_hash()`. Seluruh penggunaan *library* ini akan diubah menggunakan fungsi yang disediakan oleh PHP dengan metode *hashing* sama yaitu *CRYPT_BLOWFISH*. Perubahan fungsi *hashing* ini bersifat *backward compatible* sehingga dapat menggunakan *database* aplikasi terdahulu tanpa perlu membentuk data baru. Berikut merupakan contoh pengubahan kode dari *phpass* menjadi *password_hash*.

```
'password' => $this->password_hash->HashPassword($password)
```

menjadi

```
'password' => password_hash($password,PASSWORD_BCRYPT)
```

Sintaks `password_hash()` diatas menerima dua buah parameter yakni data yang ingin di enkripsi dan tipe enkripsi. Enkripsi akan menggunakan sintaks `PASSWORD_BCRYPT` yang menggunakan tipe *hash* berupa `CRYPT_BLOWFISH`.

MY_Form_validation

Library MY_Form_validation akan dipindahkan menuju direktori `app/Libraries`. *Library* ini akan digunakan kembali dengan perubahan *extends* menjadi menuju `Validation`, penghapusan sintaks `defined` , dan akan ada penambahan *namespace* pada baris awal file. Kode 4.16 merupakan contoh penambahan *namespace* dan penggantian *extends* pada *library* ini.

Kode 3.12: Contoh perubahan *library MY_Form_validation* pada *CodeIgniter 4*

```
1 namespace App\Libraries;
2
3 use CodeIgniter\Validation\Validation;
4
5 class MY_Form_validation extends Validation
```

Kode 4.16 mengapus sintaks `defined` dan menggantikannya dengan penambahan *namespace*. Selain itu, kelas *library* akan *extends Validation*.

MY_Profiler

Parsedown

Library Parsedown akan dipindahkan menuju direktori `app/Libraries`. *Library* ini akan digunakan kembali dengan penambahan *namespace* pada baris awal file dan penghapusan sintaks `defined`. Kode 4.17 merupakan contoh penambahan *namespace* dan juga penambahan sintaks *defined*.

Kode 3.13: Contoh perubahan *library Parsedown* pada *CodeIgniter 4*

```
1 namespace App\Libraries;
2
3 class Parsedown
```

Kode 4.17 menghapus sintaks `defined` dan menggantikannya dengan penambahan *namespace*.

Phpexcel

Library ini akan digunakan kembali namun tidak akan dipindahkan menuju `app/Libraries`. *Library* akan dilakukan instalasi melalui *composer* dengan sintaks berikut:

```
composer require phpoffice/phpexcel
```

Sintaks diatas akan dijalankan pada akar dari aplikasi dan tidak terdapat perubahan terhadap penggunaan sintaks ini.

Shj_pagination

Library ini akan digunakan kembali dan dipindahkan menuju direktori `app/Libraries`. Selain itu, terdapat penambahan *namespace* pada baris awal file dan penghapusan sintaks *defined*.

3.3.6 *Configuration*

Configuration terdapat perubahan nama dari yang sebelumnya `application/config/config.php` menjadi `app/Config/App.php` dan penambahan *file* dengan nama `app/Config/Secrets.php`. Berhubung dengan perubahan nama tersebut, terdapat beberapa perpindahan sintaks menuju direktori baru tersebut. Berikut merupakan sintaks yang dipindahkan menuju `app/Config/Security.php`:

- `$config['csrf_protection'] = TRUE;`
- `$config['csrf_token_name'] = 'shj_csrf_token';`
- `$config['csrf_cookie_name'] = 'shjcsrftoken';`
- `$config['csrf_expire'] = 7200;`
- `$config['csrf_regenerate'] = FALSE;`

Configurations yang telah dipindahkan akan diubah dari yang sebelumnya menggunakan *array* menjadi menggunakan *variable*. Seluruh *configurations* pada *CodeIgniter 3* akan dipindahkan menuju *CodeIgniter 4* sesuai dengan direktorinya dan fungsinya. Sedangkan `app/Config/App.php` dan dipindahkan menuju file `.env` karena alasan kemudahan untuk penggantian *environment* dalam melakukan *deploy*.

3.3.7 *Database*

Database pada *CodeIgniter 4* fungsi yang sama pada *CodeIgniter 3* sehingga akan dilakukan pemindahan konfigurasi sesuai dengan yang ada pada *CodeIgniter 3*. Namun, terjadi beberapa perubahan pada sintaks untuk melakukan koneksi ke *database* dan beberapa sintaks untuk melakukan *query*. Sintaks koneksi *database* akan berubah dari `$this->load->database();` menjadi `db = db_connect()`. Selain itu, pemanggilan fungsi *query builder* berubah menggunakan *camelcase* dari yang sebelumnya menggunakan *snakecase*. Seperti dari yang sebelumnya menggunakan sintaks `get_where` menjadi `getWhere`.

Database pada aplikasi *SharIF Judge* menggunakan *autoload* yang dapat memuat beberapa fungsi secara otomatis. Berikut merupakan contoh penggunakan *autoload* pada *CodeIgniter 3*.

```
$autoload['libraries'] = array('database');
```

Sintaks diatas memuat *library database* dan akan ditambahkan pada *file* `autoload.php`. Namun, pada konversi ini tidak akan menggunakan *autoload* karena *CodeIgniter 3* tidak mengikuti standar *PSR 4* sehingga pada *CodeIgniter 4* akan dimuat menggunakan `db = db_connect()` pada `initController`.

3.3.8 *Helpers*

Helpers tidak terdapat banyak perubahan namun, terdapat perubahan pemanggilan *helpers* dan beberapa penghapusan *helpers*. Berikut merupakan *helpers* yang dihapus dan diubah cara pemanggilannya.

- *Download Helper Helper* ini sudah tidak tersedia pada *CodeIgniter 4* sehingga perlu dilakukan pengubahan dengan menggunakan fungsi *HTTP Response*. *HTTP Response* menyediakan fungsi bernama *Force File Download* yang berguna untuk mengunduh sebuah *file* menuju perangkat pengguna. Fungsi ini dapat dipanggil menggunakan sintaks berikut.

```
return $this->response->download($name, $data);
```

  Sintaks diatas mengembalikan *file* yang ingin diunduh pengguna dengan dua buah parameter. Parameter pertama berupa nama *file* yang ingin diunduh sedangkan parameter kedua merupakan data dalam *file* tersebut.
- `redirect()`
  Fungsi ini memiliki perubahan pada *CodeIgniter 4* dimana `redirect()` tidak langsung mengarahkan kepada *url* yang dibentuk. Pengguna harus mengembalikan fungsi *redirect()* menggunakan `return` dengan sintaks sebagai berikut.

```
return redirect()->to('login/form')
```

  Sintaks diatas akan mengembalikan pengguna menuju *url* `login/form` yang sudah dibentuk pada *routes*.

Konversi *helpers* akan dipindahkan dari yang sebelumnya menggunakan *autoload* menuju `initController` dengan menambahkan pada variabel *helpers*.

BAB 4

PERANCANGAN

Bab ini membahas perancangan untuk seluruh implementasi *SharIF Judge* pada *CodeIgniter 4*.

4.1 Instalasi *CodeIgniter 4*

CodeIgniter 4 akan dilakukan instalasi menggunakan *composer*. *Composer* merupakan sebuah *dependency manager* untuk PHP yang memungkinkan pengguna untuk melakukan instalasi seluruh kebutuhan untuk menjalankan program berbasis PHP. Instalasi akan dilakukan menggunakan kode sebagai berikut:

4.2 Perubahan Struktur Aplikasi

Struktur aplikasi *SharIF Judge* akan dipindahkan seperti pemetaan pada bab 3 gambar 3.17. Struktur aplikasi pada *CodeIgniter 4* akan berisikan sebagai berikut :

4.2.1 app/Config

File *config* pada *CodeIgniter 3* akan dipindahkan sesuai dengan pemetaan pada gambar. Direktori berisikan data-data pada `application/Config`. Beberapa data pada direktori ini akan dipindahkan menuju file yang terdapat pada *CodeIgniter 4*. Terdapat juga penambahan *file* `Secrets.php` yang dibentuk secara manual. Berikut merupakan rincian isi direktori ini:

`App.php`

File ini tidak akan digunakan sehingga akan dibiarkan kosong dan seluruh data akan dipindahkan menuju file `.env`. Kode 4.1 merupakan isi dari file `.env` yang dipindahkan dari direktori `application/config`:

Kode 4.1: Kode `application/config/App.php` yang dipindahkan menuju `.env`

```
app.baseURL = 'http://sharif.localhost/'
```

Kode 4.1 akan menentukan *url* dasar dari aplikasi menjadi `http://sharif.localhost/`.

`Autoload.php`

File ini tidak akan digunakan karena pada *CodeIgniter 3* tidak mengikuti standar pada PSR-4. Sedangkan pada *CodeIgniter 4* mengikuti standar PSR-4 sehingga dapat melakukan inisiasi terhadap

sebuah kelas menggunakan sintaks `new Kelas`. Beberapa kelas yang diinisiasi pada file ini akan dipindahkan menuju `BaseController` dan juga akan dipanggil menggunakan PSR-4 pada file-file yang menggunakan kelas tersebut.

`Cache.php`

File ini tidak terdapat perubahan karena akan tetap menggunakan konfigurasi *default*.

`Constant.php`

File ini akan berisikan seluruh data yang dipindahkan dari `application/config/constants.php`. Kode .

`Filters.php`

4.2.2 *Controllers*

Controller terdapat perubahan pada bagian fungsi `__construct()` dimana sekarang tidak dapat mengembalikan sesuatu. Oleh karena itu, akan dibentuk beberapa *filters* 4.2.1 untuk melakukan pengecekan terhadap fungsi yang sebelumnya terdapat pada `__construct()`.

app/Controllers

Direktori ini berisikan seluruh *controller* yang dipindahkan dari *CodeIgniter 3*. Berikut merupakan rincian isi pada direktori ini:

- `Assignments.php`
- `BaseController.php`
- `Dashboard.php`
- `Halloffame.php`
- `Install.php`
- `Login.php`
- `Logs.php`
- `Moss.php`
- `Notifications.php`
- `Problems.php`
- `Profile.php`
- `Queue.php`
- `Queueprocess.php`
- `Rejudge.php`
- `Scoreboard.php`
- `Server_time.php`
- `Settings.php`
- `Submissions.php`
- `Submit.php`
- `Users.php`

Controllers terdapat perubahan dan penambahan baik dalam *extends* maupun dalam pemanggilan kelas lain seperti *model*. Kode 4.2 merupakan perubahan yang terdapat pada *controller* `Logs.php`.

Kode 4.2: Perubahan kode *controllers* pada *CodeIgniter 4*

```
namespace App\Controllers;

use App\Controllers\BaseController;
use App\Models\AssignmentModel;
use App\Models\LogsModel;
use App\Models\User;

class Logs extends BaseController
{
    protected $session;
    protected $user;
    protected $logs_model;
    protected $assignment_model;

    public function __construct()
    {
        $this->session = session();
        $this->logs_model = new LogsModel();
        $this->assignment_model = new AssignmentModel();
        $this->user = new User();
        if ( $this->user->level <= 2) // permission denied
            throw \CodeIgniter\Exceptions\PageNotFoundException::forPageNotFound();
    }

    public function index()
    {

        $data = array(
            'logs' => $this->logs_model->get_all_logs(),
            'selected' => 'logs',
            'user' => $this->user,
            'all_assignments' => $this->assignment_model->all_assignments(),
            'finish_time' => $this->user->selected_assignment['finish_time'],
            'extra_time' => $this->user->selected_assignment['extra_time'],
        );

        return view('pages/admin/logs', $data);
    }
}
```

Kode 4.2 terdapat perubahan dimana sekarang akan *extends BaseController*. Terdapat juga penghapusan sintaks `defined('BASEPATH') OR exit('No direct script access allowed');`. Terdapat penambahan sintaks *namespace* dan juga beberapa sintaks untuk memanggil *models*. *Controller* juga memiliki perubahan dalam mengembalikan *view* dimana sekarang menggunakan sintaks `return view`. Selain itu terdapat penambahan pada `BaseController.php` untuk melakukan inisiasi terhadap *helpers* dan juga beberapa *library* yang akan digunakan.

4.2.3 *Filters*

Pada *CodeIgniter 4* `__construct()` tidak dapat mengembalikan sesuatu oleh karena itu akan dibentuk beberapa *filters* untuk melakukan pengecekan. Beberapa *filters* yang dibentuk antara lain berfungsi untuk mengecek apakah dijalankan dari *command line interface*, apakah sudah *install* dan *login*, apakah sudah *login*, apakah sudah *login* dan dijalankan dari *command line interface*, apakah sudah *login* dan apakah *request* berupa *AJAX*, dan apakah sudah *login* dan pengecekan terhadap *role* pengguna. Kode 4.3 merupakan sintaks untuk melakukan pengecekan apakah dijalankan dari *command line interface*.

Kode 4.3: Pemindahan kode pada *Filters*

```
    public function before(RequestInterface $request, $arguments = null)
    {
        $request = \Config\Services::request();

        if ($request->isCLI())
            throw \CodeIgniter\Exceptions\PageNotFoundException::forPageNotFound();
    }
```

Kode 4.3 merupakan pemindahan kode dari `__construct` menuju *filters* `CheckCLI`. Kode ini akan mengecek apakah *request* yang diberikan oleh pengguna merupakan *command line interface*. Apabila *request* yang diberikan bukan berupa itu maka akan diberikan error berupa halaman tidak ditemukan. Setelah dibentuk *filters*, selanjutnya akan ditambahkan menuju file `Filters.php` untuk mendefiniskan nama untuk dimasukkan menuju *routes*. Kode 4.4 merupakan penambahan nama pada file `Filters.php`.

Kode 4.4: Penambahan nama *filters* untuk didefinisikan menuju *routes*

```
    public array $aliases = [
        'csrf'          => CSRF::class,
        'toolbar'       => DebugToolbar::class,
        'honeypot'      => Honeypot::class,
        'invalidchars'  => InvalidChars::class,
        'secureheaders' => SecureHeaders::class,
        'check-installandlogin' => CheckInstallAndLogin::class,
     ]
```

Kode 4.4 merupakan penambahan nama *filters* untuk dipanggil menuju *routes*. Penamaan ini akan ditambahkan pada *array* `$aliases` dengan *index* dan nama kelasnya. Setelah ditambahkan, *filters* akan dipanggil pada *routes* yang membutuhkan pengecekan. Kode 4.5 merupakan penambahan *filters* pada *routes* sesuai dengan kebutuhannya.

Kode 4.5: Penambahan *filter* pada *routes*

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

Kode 4.5 menambahkan *filter* yang sudah dibentuk dan dinamakan setelah penulisan nama *controller* dan fungsinya.

4.2.4 *Libraries*

Libraries terdapat beberapa perubahan dan penghapusan fungsi sehingga akan digantikan. Berikut merupakan perancangan perubahan fungsi pada *CodeIgniter 4*.

Emails

Emails pada *CodeIgnier 4* terdapat perubahan sintaks dan cara pemanggilan sehingga akan dipindahkan sesuai dengan sintaks yang baru. Sintaks berubah dari yang sebelumnya menggunakan *snakecase* menjadi menggunakan *camelcase*. Kode 4.6 merupakan contoh penggunaan *library email*.

Kode 4.6: Contoh perubahan *library emails*

```
$this->email->setFrom($this->settings_model->get_setting('mail_from'), $this->settings_model->get_setting('mail_from_name'));
                $this->email->setTo($user[1]);
                $this->email->setSubject('SharIF Judge Username and Password');
                $text = $this->settings_model->get_setting('add_user_mail');
                $text = str_replace('{SITE_URL}', base_url(), $text);
                $text = str_replace('{ROLE}', $user[4], $text);
                $text = str_replace('{USERNAME}', $user[0], $text);
                $text = str_replace('{PASSWORD}', htmlspecialchars($user[3]), $text);
                $text = str_replace('{LOGIN_URL}', base_url(), $text);
                $this->email->setMessage($text);
                $this->email->send()
```

Kode 4.6 memiliki sintaks dengan nama sama namun terdapat perubahan menjadi *camelcase*.

Working with Uploaded Files

Working with uploaded files terdapat perubahan pada beberapa sintaks dan validasi terhadap *file* yang telah diunggah. Konversi aplikasi *SharIF Judge* akan menggunakan fungsi ini dengan beberapa perubahan sintaks sesuai dengan dokumentasinya. Kode 4.7 merupakan perubahan yang terdapat pada *library* ini.

Kode 4.7: Perancangan perubahan *library upload* pada *CodeIgniter 4*

```
$zip_uploaded = $this->request->getFile('tests_desc');
        if ( $_FILES['tests_desc']['error'] === UPLOAD_ERR_NO_FILE ){
            $this->messages[] = array(
                'type' => 'notice',
                'text' => "Notice: You did not upload any zip file for tests. If needed, upload by editing assignment."
            );
        }
        elseif ( !$zip_uploaded){
            $this->messages[] = array(
                'type' => 'error',
                'text' => "Error: Error uploading tests zip file: ".$zip_uploaded->getErrorString()
            );
        }
        else{
            $zip_uploaded->move($assignments_root);
            $this->messages[] = array(
                'type' => 'success',
                'text' => "Tests (zip file) uploaded successfully."
            );
        }
```

Kode 4.7 merupakan perubahan yang terdapat pada *library upload*. . Pengambilan file akan digantikan dengan sintaks `$this->request->getFile('')` dengan parameter berupa nama dari *tag form* yang sudah dibentuk. Selanjutnya akan dilakukan pengecekan terhadap file yang sudah di unggah dan memberikan beberapa *error message* sesuai dengan kondisinya. File yang sudah di unggah akan dipindahkan menuju direktori `assignments_root`.

Validation

Form_validation akan digantikan dengan fungsi *validation* dengan perubahan dan pengapusan beberapa fungsi. *Validation* akan diinisiasikan pada fungsi `__construct` pada setiap *controller* yang membutuhkan. Kode 4.8 merupakan inisiasi *validation* pada *controller*.

Kode 4.8: Perancangan inisiasi *validation* pada `__construct`

```

protected $validation;

    public function __construct()
    {
        $this->validation = \Config\Services::validation();
    }
```

Kode 4.8 merupakan sintaks untuk melakukan inisiasi terhadap *validation*. Setiap *controller* yang menggunakan *validation* akan dideklarasikan sebuah variabel bernama *validation* agar dapat dipanggil pada seluruh fungsi yang membutuhkan pada kelas tersebut. *Validation* dilanjutkan dengan penetapan aturan untuk *tag form* yang diinginkan. Berikut merupakan contoh pembentukan aturan untuk mengumpulkan sebuah data pada *form*.

Kode 4.9: Perancangan perubahan konfigurasi aturan pada *library validation*

```
$this->validation->setRule('username', 'username', 'required|min\_length[3]|max\_length[20]');
```

Kode 4.9 akan menetapkan aturan terhadap *input* yang akan masukan oleh pengguna sesuai dengan nama *form*nya. Sedangkan penetapan aturan berubah menggunakan *camelCase*. Aturan yang dibentuk secara manual akan dipindahkan menuju file `Validation.php`. Kode 4.10 merupakan aturan yang dibentuk secara manual.

Kode 4.10: Perancangan aturan yang dibentuk secara manual pada file `Validation.php`

```
class MyRules
{
    public function password_check($str): bool
    {
        if (strlen($str) == 0 OR (strlen($str) >= 6 && strlen($str) <= 200))
            return TRUE;
        return FALSE;
    }

    public function password_again_check($str) :bool
    {
        $request = \Config\Services::request();
        if ($request->getPost('password') !== $request->getPost('password_again'))
            return FALSE;
        return TRUE;
    }

    public function role_check($user,$role)
    {
        $user = new User();
        if ($user->level <= 2)
            return ($role == '');

        // Admins can change everybody's user role:
        $roles = array('admin', 'head_instructor', 'instructor', 'student');
        return in_array($role, $roles);
    }

    /**
     * Checks whether a user with this email exists
     */
    public function email_check($email,$edit_username):bool
    {
        $user_model = new UserModel();
        if ($user_model->have_email($email, $edit_username))
            return FALSE;
        return TRUE;
    }

    /**
     * checks whether the entered registration code is correct or not
     *
     */
    public function _registration_code($code){
        $settings_model = new SettingsModel();
        $rc = $settings_model->get_setting('registration_code');
        if ($rc == '0')
            return TRUE;
        if ($rc == $code)
            return TRUE;
        return FALSE;
    }

    /**
     * Required
     *
     * @param   string
     * @return  bool
     */
    public function required($str)
    {
        return is_array($str) ? (bool) count($str) : ($str !== '');
    }


    // --------------------------------------------------------------------

```

```
    /**
     * Is Lowercase
     *
     * @param $str
     * @return bool
     */
    public function lowercase($str)
    {
        return (strtolower($str) === $str);
    }

        // --------------------------------------------------------------------


    public function _check_language($str)
    {
        if ($str=='0')
            return FALSE;
        if (in_array( strtolower($str),array('c', 'c++', 'python 2', 'python 3', 'java', 'zip', 'pdf', 'txt')))
            return TRUE;
        return FALSE;
    }


    // --------------------------------------------------------------------
    // Used in Submissions.php

    public function _check_type($type)
    {
        return ($type === 'code' || $type === 'result' || $type === 'log');
    }
}
```

Aturan-aturan diatas dipindahkan dari *controller* menuju `Validation.php` agar dapat digunakan untuk melakukan validasi. Aturan yang sudah dibentuk dapat digunakan seperti aturan lainnya dengan cara menulis nama kelasnya. Setelah aturan ditetapkan akan dieksekusi berdasarkan *request* dari pengguna dan dilakukan validasi. Kode 4.11 merupakan perubahan penggunaan *validation* terhadap data yang sudah diberikan oleh pengguna.

Kode 4.11: Perancangan perubahan penggunaan *validation* pada *CodeIgniter 4*

```
if ($this->validation->withRequest($this->request)->run())
        {
            if ( !$this->request->isAJAX() ){
                exit;
            }else{
                list($ok, $error) = $this->user_model->add_users(
                    $this->request->getPost('new_users'),
                    $this->request->getPost('send_mail'),
                    $this->request->getPost('delay')
                );
            return view('pages/admin/add_user_result', array('ok' => $ok, 'error' => $error));
            }
        }
```

Kode 4.11 akan menjalankan *validation* berdasarkan *request* dari pengguna. *Validation* akan tetap menggunakan sintaks `run()` namun akan ada penambahan sintaks `withRequest` dimana validasi akan dijalankan setiap ada *HTTP Request* dari pengguna. Namun, *CodeIgniter 4* tidak menyediakan fungsi `form_error` sehingga akan diubah dengan menggunakan fungsi baru bernama `validation_errors()`. Fungsi tersebut dapat digunakan untuk mengembalikan *error* apabila terdapat data yang tidak sesuai dengan aturan. *Error* tersebut dapat ditampilkan pada halaman *view* menggunakan sintaks berikut.

```
<?= $validation->hasError('username') ? $validation->getError('username') : '' ?>
```

Sintaks diatas akan melakukan pengecekan apakah terdapat *error* dari *form* bernama `username`

dan apabila terdapat maka akan dikembalikan dara *error* yang berasal dari *validation*. Variabel `validation` akan dikirimkan dari *controller* berisikan *library validation*.

Zip Archive

Zip Encoding akan digantikan dengan fungsi PHP *zip archive* karena sudah tidak tersedia pada *CodeIgniter 4*. Fungsi *zip archive* terdapat beberapa perbedaan sehingga akan disesuaikan dengan fungsi-fungsi yang ada. Kode 4.12 merupakan perubahan yang terdapat pada *zip encoding*.

Kode 4.12: Perancangan perubahan *zip encoding* menjadi *zip archive*

```
$this->zip = new \ZipArchive();
        $this->zip->open($zipname, ZipArchive::CREATE);
        for ($i=1 ; $i<=$number_of_problems ; $i++)
        {

            $path = "$root_path/p{$i}/in";
            $options = ['add_path' => "p{$i}/in/", 'remove_all_path' => TRUE];
            $this->zip->addGlob($path.'/*.{txt}', GLOB_BRACE, $options);

            $path = "$root_path/p{$i}/out";
            $options = ['add_path' => "p{$i}/out/", 'remove_all_path' => TRUE];
            $this->zip->addGlob($path.'/*.{txt}', GLOB_BRACE, $options);

            $path = "$root_path/p{$i}/tester.cpp";
            if (file_exists($path))
                $this->zip->addFile($path,"p{$i}/tester.cpp");

            $pdf_files = glob("$root_path/p{$i}/*.pdf");
            if ($pdf_files)
            {
                $path = $pdf_files[0];
                $this->zip->addFile($path,"p{$i}/".shj_basename($path));
            }

            $path = "$root_path/p{$i}/desc.html";
            if (file_exists($path))
                $this->zip->addFile($path,"p{$i}/desc.html");

            $path = "$root_path/p{$i}/desc.md";
            if (file_exists($path))
                $this->zip->addFile($path,"p{$i}/desc.md");
        }

        $pdf_files = glob("$root_path/*.pdf");
        if ($pdf_files)
        {
            $path = $pdf_files[0];
            $this->zip->addFile($path,shj_basename($path));
        }
        $this->zip->close();

        header('Content-Type: application/zip');
        header('Content-disposition: attachment; filename=' . $zipname);
        header('Content-Length: ' . filesize($zipname));
        readfile($zipname);
```

Kode 4.12 merupakan perubahan dari *zip encoding* menjadi *zip archive*. *Zip archive* akan dilakukan inisiasi pada variabel `zip`. Selanjutnya akan dibuka file *zip* menggunakan sintaks `open` yang menerima dua buah parameter. Parameter pertama berisikan nama *zip* file yang ingin dibentuk sedangkan parameter kedua berisikan mode *zip* yang diinginkan. Fungsi *Library* yang terdapat pada *CodeIgniter 4* juga dapat di*extend* dan dibentuk sesuai dengan kebutuhan. Berikut merupakan *library* yang dibentuk oleh pengguna.

Twig

Library ini tidak akan digunakan untuk membentuk *view* pada *CodeIgniter 4* namun, akan ada penggunaan sebuah fungsi *Twig* yang akan dibentuk pada direktori `app/Libraries`. Fungsi tersebut

bernama `extra_time_formatter` yang memiliki fungsi untuk mengubah input yang diberikan menjadi format jam dikali enam puluh menit. Kode 4.13 merupakan fungsi yang akan dibentuk pada direktori `app/Libraries` dengan nama `Twig.php`.

Kode 4.13: Perancangan perubahan *library MY_Form_validation* pada *CodeIgniter 4*

```php
<?php
namespace App\Libraries;

class Twig
{

    /**
     * Required
     *
     * @param   string
     * @return  bool
     */
    public function extra_time_formatter($extra_time)
    {
        // convert to minutes
        $extra_time = floor($extra_time/60);
        // convert to H*60
        if ($extra_time % 60 == 0 )
            $extra_time = ($extra_time/60) . '*60';
        return $extra_time;
    }
}
```

Kode 4.13 merupakan fungsi yang akan dibentuk pada file `Twig.php`. Fungsi ini akan dipanggil pada halaman `add_assignment.php`.

Unzip

Library ini akan digunakan kembali dan dipindahkan menuju direktori `app/Libraries`. *Library* terdapat penghapusan sintaks *defined* dan juga penambahan *namespace*. Kode 4.14 merupakan penghapusan sintaks *defined* dan juga penambahan *namespace*.

Kode 4.14: Perancangan perubahan *library Unzip* pada *CodeIgniter 4*

```php
namespace App\Libraries;

class Unzip
{
```

Kode 4.14 merupakan perubahan yang terdapat pada *library* ini. Terdapat penghapusan sintaks *defined* dan penggantian menjadi *namespace*. Selanjutnya, *library* ini akan diinisiasikan pada fungsi *controller* yang menggunakan. Kode 4.15 merupakan inisiasi *library unzip* pada *controller*.

Kode 4.15: Perancangan inisiasi *library unzip* pada *controller*

```php

        $this->unzip = new Unzip();
        // Create a temp directory
        $tmp_dir_name = "shj_tmp_directory";
        $tmp_dir = "$assignments_root/$tmp_dir_name";
        shell_exec("rm -rf $tmp_dir; mkdir $tmp_dir;");

        // Extract new test cases and descriptions in temp directory
        $this->unzip->allow(array('txt', 'cpp', 'html', 'md', 'pdf'));
        $extract_result = $this->unzip->extract("$assignments_root/".$zip_uploaded->getName(), $tmp_dir);
```

Kode 4.15 merupakan inisiasi *library unzip* pada *controller*. Inisiasi akan dilakukan pada variabel `unzip` yang sudah dibentuk diluar fungsi tersebut. Penggunaan akan tetap sama seperti sebelumnya sehingga tidak terdapat perubahan sintaks.

Password_hash

Library ini tidak akan digunakan dan akan digantikan oleh *password hash* yang disediakan oleh PHP .*Library Password_hash* merekomendasikan pegguna untuk menggunakan fungsi *native* yang disediakan oleh PHP apabila aplikasi mendukung PHP versi 5.5 ke atas. Sehingga, akan dilakukan konversi menggunakan fungsi yang disediakan oleh PHP bernama `password_hash()`. Seluruh penggunaan *library* ini akan diubah menggunakan fungsi yang disediakan oleh PHP dengan metode *hashing* sama yaitu *CRYPT_BLOWFISH*. Perubahan fungsi *hashing* ini bersifat *backward compatible* sehingga dapat menggunakan *database* aplikasi terdahulu tanpa perlu membentuk data baru. Berikut merupakan contoh pengubahan kode dari *phpass* menjadi *password_hash*.

```
'password' => $this->password_hash->HashPassword($password)
```

menjadi

```
'password' => password_hash($password,PASSWORD_BCRYPT)
```

Sintaks `password_hash()` diatas menerima dua buah parameter yakni data yang ingin di enkripsi dan tipe enkripsi. Enkripsi akan menggunakan sintaks `PASSWORD_BCRYPT` yang menggunakan tipe *hash* berupa `CRYPT_BLOWFISH`. Selain itu, terdapat fungsi untuk melakukan pengecekan *password* yang sudah di enkripsi. Berikut merupakan contoh pengubahan kode untuk melakukan pengecekan *password* yang sudah di enkripsi.

```
password_verify($password, $query->getRow()->password)
```

Sintaks diatas menerima dua buah parameter dengan parameter pertama berupa masukan dari pengguna dan parameter berikutnya merupakan *hash* dari *password* yang sudah disimpan. Fungsi ini akan mengembalikan data berupa *true* apabila *password* sama dan *false* apabila *password* berbeda.

MY_Form_validation

Library MY_Form_validation akan dipindahkan menuju direktori `app/Libraries`. *Library* ini akan digunakan kembali dengan perubahan *extends* menjadi menuju `Validation`, penghapusan sintaks `defined` , dan akan ada penambahan *namespace* pada baris awal file. Kode 4.16 merupakan contoh penambahan *namespace* dan penggantian *extends* pada *library* ini.

Kode 4.16: Contoh perubahan *library MY_Form_validation* pada *CodeIgniter 4*

```
1 namespace App\Libraries;
2
3 use CodeIgniter\Validation\Validation;
4
5 class MY_Form_validation extends Validation
```

Kode 4.16 mengapus sintaks `defined` dan menggantikannya dengan penambahan *namespace*. Selain itu, kelas *library* akan *extends Validation*.

MY_Profiler

Parsedown

Library Parsedown akan dipindahkan menuju direktori `app/Libraries`. *Library* ini akan digunakan kembali dengan penambahan *namespace* pada baris awal file dan penghapusan sintaks `defined`. Kode 4.17 merupakan contoh penambahan *namespace* dan juga penambahan sintaks *defined*.

Kode 4.17: Perancangan perubahan *library Parsedown* pada *CodeIgniter 4*

```
namespace App\Libraries;

class Parsedown
```

Kode 4.17 menghapus sintaks `defined` dan menggantikannya dengan penambahan *namespace*. Penggunaan *library* ini tidak akan berubah sehingga tidak terdapat perbedaan sintaks. Namun, terdapat perubahan cara inisiasi *library* ini dimana sekarang akan menggunakan sintaks *new* dan dilakukan inisiasi pada *BaseController*. Kode 4.18 merupakan perubahan cara melakukan inisiasi *library parsedown*.

Kode 4.18: Perancangan perubahan inisiasi *library Parsedown* pada *controller CodeIgniter 4*

```
    protected $parsedown;

    /**
     * Constructor.
     */
    public function initController(RequestInterface $request, ResponseInterface $response, LoggerInterface $logger)
    {
        // Do Not Edit This Line
        parent::initController($request, $response, $logger);

        // Preload any models, libraries, etc, here.

        // E.g.: $this->session = \Config\Services::session();
        $this->parsedown = new Parsedown();
    }
```

Kode 4.18 merupakan perubahan inisiasi pada *CodeIgniter 4*. *Library parsedown* akan dilakukan inisiasi menuju variabel `parsedown` yang sudah dibentuk pada luar fungsi. Inisiasi dilakukan pada *BaseController* karena terdapat pemakaian pada beberapa *model* dan *controller*.

Phpexcel

Library ini akan digunakan kembali namun tidak akan dipindahkan menuju `app/Libraries`. *Library* akan dilakukan instalasi melalui *composer* dengan sintaks berikut:

```
composer require phpoffice/phpexcel
```

Sintaks diatas akan dijalankan pada akar dari aplikasi dan tidak terdapat perubahan terhadap penggunaan sintaks ini.

Shj_pagination

Library ini akan digunakan kembali dan dipindahkan menuju direktori `app/Libraries`. Selain itu, terdapat penambahan *namespace* pada baris awal file dan penghapusan sintaks *defined*. Kode 4.19 merupakan penambahan perubahan dan penambahan sintaks pada *library* ini.

Kode 4.19: Perancangan perubahan *library Shj_pagination* pada *CodeIgniter 4*

```
namespace App\Libraries;

class Shj_pagination
{
```

4.2.5 *Model*

Model akan dipindahkan sesuai dengan direktori 4.2 dan diubah sesuai dengan dokumentasi *CodeIgniter 4*. Seluruh *Model* akan diganti penamaannya dari yang sebelumnya menggunakan *snakecase* menjadi *camelcase*. Berikut merupakan rincian

4.2.6 *View*

View akan diubah menggunakan *extension* `.php` sesuai pada dokumentasi *CodeIgniter 4*. Seluruh *file view* akan diubah menjadi *extension* `.php` dari yang sebelumnya menggunakan `.twig`. Seluruh *delimiters* juga akan diubah menggunakan fungsi pada *CodeIgniter 4*. Perubahan *view* dapat dilihat pada kode 4.20.

Kode 4.20: Perubahan *view* pada *Login.php*

```
<!-- {#
 # SharIF Judge
 # file: login.twig
 # author: Mohammad Javad Naderi <mjnaderi@gmail.com>
 #} -->
<!DOCTYPE html>
<html lang="en">
<?= $this->include('templates/simple_header')?>

<?= form_open() ?>
    <div class="box login">

        <div class="judge_logo">
            <a href="<?= site_url() ?>"><img src="<?= base_url('assets/images/banner.png') ?>"/></a>
        </div>

        <div class="login_form">
            <div class="login1">
                <p>
                    <label for="form_username">Username</label><br/>
                    <input id="form_username" type="text" name="username" required="required" pattern="[0-9a-z]{3,20}" title="The
                        Username field must be between 3 and 20 characters in length, and contain only digits and lowercase
                        letters" class="sharif_input" value="<?= set_value('username') ?>" autofocus="autofocus"/>
                    <?= isset($this->errors['username'])?>
                </p>
                <p>
                    <label for="form_password">Password</label><br/>
                    <input id="form_password" type="password" name="password" required="required" pattern=".{6,200}" title="The
                        Password field must be at least 6 characters in length" class="sharif_input"/>
                    <?= isset($this->errors['password'])?>
                </p>
                <?php if ($error): ?>
                    <div class="shj_error">Incorrect username or password.</div>
                <?php endif ?>
            </div>
            <div class="login2">
                <p style="margin:0;">
                    <?php if ($registration_enabled): ?>
                    <a href="<?= site_url('register') ?>">Register</a> |
                    <?php endif ?>
                    <a href="<?= site_url('login/lost') ?>">Reset Password</a>
                    <input type="submit" value="Login" id="sharif_submit"/>
                </p>
            </div>
        </div>

    </div>q
<?= form_close() ?>
</body>
</html>
```

Kode 4.20 merupakan perubahan yang terdapat pada halaman *view*. *Delimiters* `{{ }}` akan digantikan menjadi `<?= ?>` sedangkan `{% %}` akan digantikan menjadi `<?php ?>`. *Delimiters* yang memanggil fungsi pada *CodeIgniter 4* akan digantikan menjadi `<?= ?>`. Perubahan juga terdapat pada sintaks dari yang sebelumnya menggunakan `include` akan digantikan menggunakan

fungsi *CodeIgniter 4* berupa `$this->include`. Selain terdapat perubahan *extension* dan *delimiters*, terdapat juga penambahan kode pada *Controller* karena tidak mendukung pembentukan variabel *global* pada *View*. Kode 4.21 merupakan contoh penambahan kode pada *Controller*.

Kode 4.21: Penambahan kode pada *Login.php*

```
    $data = [
            'error' => FALSE,
            'registration_enabled' => $this->settings_model->get_setting('enable_registration'),
            'title' => 'Login',
            'validationError' => $this->validation
        ];
```

Kode 4.21 merupakan contoh penambahan data pada *controller*. Penambahan data terjadi karena halaman *view* PHP tidak dapat mendeklarasikan variabel secara *global* sehingga data-data seperti *title* tidak dapat diakses oleh *view* lainnya.

4.2.7 public

assets

Direktori ini berisikan seluruh data yang dapat dilihat oleh pengguna seperti *javascript*, gambar, dan lainnya. Berikut merupakan rincian isi pada direktori ini:

- `ace`
- `font`
- `fullcalendar`
- `gridster`
- `images`
- `js`
- `nano_scroller`
- `noty`
- `pdfjs`
- `reveal`
- `snippet`
- `styles`
- `tinymce`

DAFTAR REFERENSI

[1] Version 3.1.13 (2022) *CodeIgniter User Guide*. CodeIgniter Foundation. Richmond,Canada.

[2] Version 1.4 (2023) *SharIF Judge Documentation*. Informatika UNPAR. Jl. Ciumbuleuit No. 94, Bandung.

[3] Version 4.3 (2023) *CodeIgniter User Guide*. CodeIgniter Foundation. Richmond,Canada.

[4] Prihatini, F. N. dan Indudewi, D. (2016) Kesadaran dan Perilaku Plagiarisme dikalangan Mahasiswa(Studi pada Mahasiswa Fakultas Ekonomi Jurusan Akuntansi Universitas Semarang). *Dinamika Sosial Budaya*, **18**, 68–75.

[5] Kurnia, A., Lim, A., dan Cheang, B. (2001) Online judge. *Computers & Education*, **18**, 299–315.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

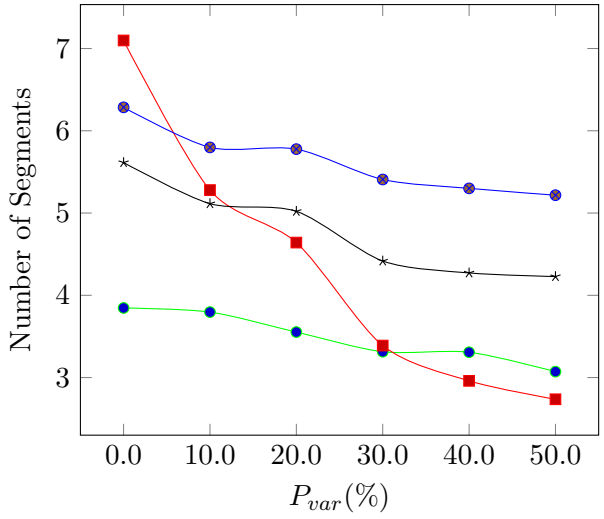
Kode A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

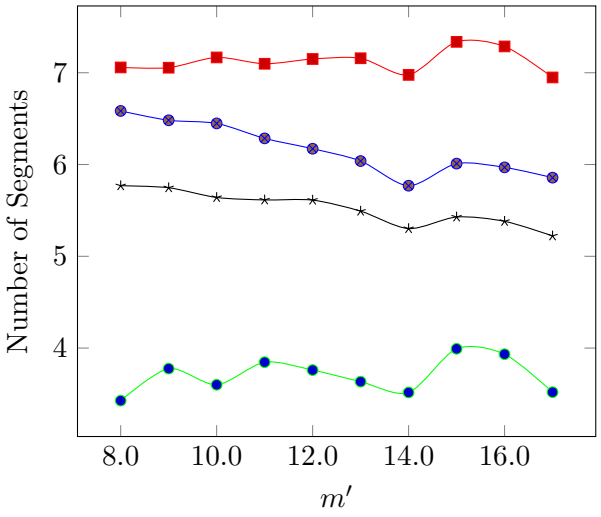
LAMPIRAN B

HASIL EKSPERIMEN

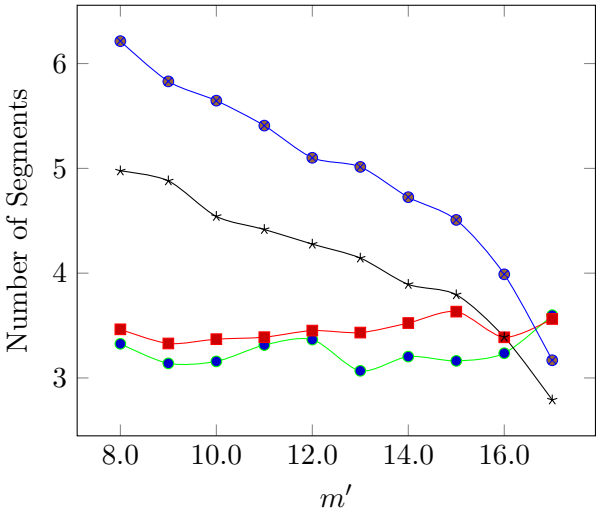
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



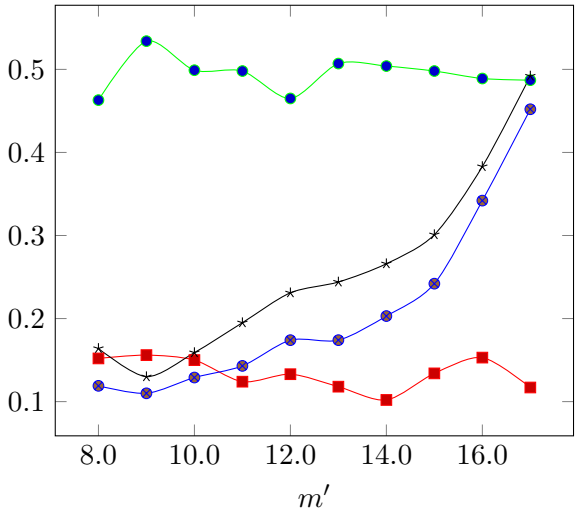
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4