

## SKRIPSI

### PEMUTARAN ULANG KETIKAN MAHASISWA PADA SHARIF JUDGE



Andreas Ronaldi

NPM: 6182101026

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2025



# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>iii</b>
<b>DAFTAR GAMBAR</b>	<b>v</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	4
1.3 Tujuan . . . . .	4
1.4 Batasan Masalah . . . . .	4
1.5 Metodologi . . . . .	4
1.6 Sistematika Pembahasan . . . . .	4
<b>2 LANDASAN TEORI</b>	<b>7</b>
2.1 SharIF Judge . . . . .	7
2.1.1 Instalasi . . . . .	7
2.1.2 Users . . . . .	8
2.2 CodeIgniter 3 . . . . .	8
2.2.1 Model-View-Controller . . . . .	9
2.2.2 CodeIgniter URLs . . . . .	11
2.2.3 <i>Helpers</i> . . . . .	11
2.3 Twig . . . . .	12
2.4 Integrated Development Environment . . . . .	12
2.5 Ace . . . . .	13
2.5.1 Perekaman Event . . . . .	15
2.6 Chart.js . . . . .	16
<b>3 ANALISIS</b>	<b>19</b>
3.1 Analisis Sistem Kini . . . . .	19
3.1.1 Model, View, Controller . . . . .	19
3.1.2 Assets . . . . .	44
3.1.3 Penyimpanan Kode Submission . . . . .	45
3.1.4 Antrian Penilaian Kode . . . . .	46
3.2 Analisis Sistem Usulan . . . . .	46
3.2.1 Fitur perekaman perubahan atau event . . . . .	47
3.2.2 Fitur penyimpanan rekaman perubahan . . . . .	48
3.2.3 Fitur melihat daftar rekaman . . . . .	49
3.2.4 Fitur pemutaran ulang rekaman . . . . .	50
<b>4 PERANCANGAN</b>	<b>53</b>
4.1 Rancangan Antarmuka . . . . .	53
4.1.1 Sistem Rekaman . . . . .	53
4.1.2 Sistem Pemutaran ulang . . . . .	53
4.2 Rancangan Penyimpanan Rekaman . . . . .	53

4.3	Rancangan Perubahan Kode . . . . .	54
4.3.1	Merekam perubahan atau event . . . . .	54
4.3.2	Menyimpan rekaman . . . . .	54
4.3.3	Melihat daftar rekaman . . . . .	55
4.3.4	Pemutaran ulang rekaman . . . . .	56
<b>5</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>57</b>
5.1	Implementasi . . . . .	57
5.1.1	Merekam Peristiwa pada IDE . . . . .	57
5.1.2	Menyimpan Rekaman pada Sistem . . . . .	60
5.1.3	Melihat Daftar Rekaman . . . . .	61
5.1.4	Pemutaran Ulang Rekaman . . . . .	62
5.2	Pengujian Fungsional . . . . .	64
5.3	Pengujian Eksperimental . . . . .	64
5.3.1	Lingkungan pengujian . . . . .	64
5.3.2	Pengujian . . . . .	66
<b>DAFTAR REFERENSI</b>		<b>69</b>
<b>A KODE PROGRAM</b>		<b>71</b>
<b>B HASIL EKSPERIMEN</b>		<b>73</b>

## DAFTAR GAMBAR

1.1	Sistem Tradisional Pemberian Tugas . . . . .	1
1.2	Sistem Integrasi oleh <i>Online Judge</i> . . . . .	2
1.3	Tampilan Awal SharIF Judge . . . . .	3
2.1	<i>Flow Chart</i> CodeIgniter . . . . .	8
2.2	Hasil Web Page <i>Library</i> Ace . . . . .	15
2.3	Hasil Web Page <i>Library</i> Chart.js . . . . .	18
3.1	Struktur MVC pada SharIF Judge . . . . .	20
3.2	Struktur Kelas Model pada SharIF Judge . . . . .	21
3.3	Struktur Direktori View pada SharIF Judge . . . . .	26
3.4	Struktur Kelas Controller pada SharIF Judge . . . . .	28
3.5	Halaman Assignments . . . . .	29
3.6	Halaman Dashboard . . . . .	30
3.7	Halaman Hall of Fame . . . . .	31
3.8	Halaman Install . . . . .	31
3.9	Halaman Login . . . . .	32
3.10	Halaman 24-Hour Log . . . . .	33
3.11	Halaman Moss . . . . .	34
3.12	Halaman Notifications . . . . .	35
3.13	Halaman Problems . . . . .	36
3.14	Halaman Profile . . . . .	37
3.15	Halaman Queue . . . . .	38
3.16	Halaman Rejudge . . . . .	39
3.17	Halaman Scoreboard . . . . .	39
3.18	Halaman Settings . . . . .	40
3.19	Halaman Final Submissions . . . . .	41
3.20	Halaman All Submissions . . . . .	42
3.21	Halaman Submit . . . . .	43
3.22	Halaman Users . . . . .	44
3.23	Usecase analisis sistem usulan . . . . .	47
3.24	Sequence Diagram Fitur Perekaman Perubahan . . . . .	48
3.25	Sequence Diagram Fitur Penyimpanan Rekaman . . . . .	49
3.26	Sequence Diagram Membuka Halaman Rekaman . . . . .	50
3.27	Sequence Diagram Membuka Halaman Rekaman . . . . .	51
5.1	Bagan Heatmap Perubahan Saat <i>Debugging</i> . . . . .	67
5.2	Bagan Heatmap Perubahan Tanpa <i>Debugging</i> . . . . .	67
5.3	Bagan Histogram Perubahan Saat <i>Debugging</i> . . . . .	68
5.4	Bagan Histogram Perubahan Tanpa <i>Debugging</i> . . . . .	68
B.1	Hasil 1 . . . . .	73
B.2	Hasil 2 . . . . .	73

B.3 Hasil 3 . . . . .	73
B.4 Hasil 4 . . . . .	73

1

## BAB 1

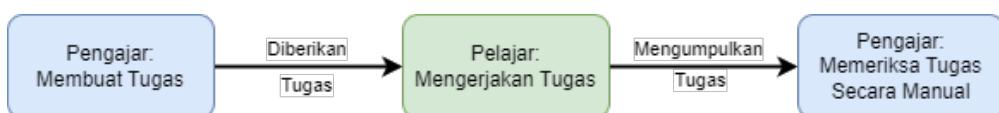
2

### PENDAHULUAN

#### 3 1.1 Latar Belakang

4 Institusi yang memberikan pendidikan, perlu memiliki cara untuk mengetahui pemahaman pelajarannya. Salah satu caranya adalah dengan memberikan tugas. Tugas merupakan sebuah bentuk penilaian dari pengajar kepada pelajarnya [1]. Tugas diberikan kepada pelajar untuk membantu pelajar mendalami materi yang sudah diberikan sebelumnya oleh pengajar dan juga untuk melihat seberapa jauh pemahaman pelajar terhadap materi yang sudah diberikan.

9 Pada bidang informatika, banyak materi pembelajaran yang dapat diberikan. Salah satu pembelajaran utama dalam bidang informatika adalah keterampilan pemrograman. Dikarenakan itu, perlu sebuah sistem untuk melatih keterampilan pemrograman yaitu dengan memberikan tugas menulis kode program sesuai dengan petunjuk yang diberikan dan program tersebut dapat berjalan sesuai dengan petunjuk [2]. Secara tradisional, tugas ini diberikan dengan cara pengajar menyiapkan dan mendistribusikan tugas tersebut kepada pelajar, kemudian dikumpulkan kembali hasil program pekerjaan pelajar, dan pengajar akan menilai kode program sesuai ketepatan dengan program yang diinginkan secara manual seperti Gambar 1.1. Karena menilaian kode program mencakup keluaran program dan juga analisis kode, maka proses tersebut memakan waktu yang cukup lama untuk dilakukan. Walaupun begitu, cara tradisional ini masih bekerja jika jumlah pelajarnya sedikit. Tetapi semakin banyak kode program yang harus di periksa maka semakin banyak waktu yang dibutuhkan dan semakin banyak pula kesalahan yang berhubungan dengan manusia. Salah satu masalah lain yang muncul juga adalah pelajar tidak dapat mengetahui apakah kode program berada pada jalur yang benar dalam menemukan solusi tugas tersebut.

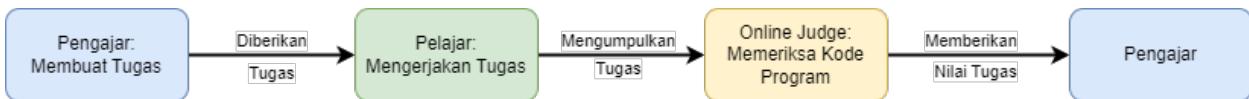


Gambar 1.1: Sistem Tradisional Pemberian Tugas

23 Pemberian tugas menulis kode program memiliki banyak masalah. Oleh karena itu, dibutuhkan-  
24 nya sistem baru untuk memberikan tugas kepada pelajar bidang informatika. Sistem baru yang  
25 dimaksud tentunya untuk melakukan penilaian secara otomatis. Sebuah sistem yang mengambil  
26 kode program pelajar dan memberikan sebuah nilai numerik yang menandakan hasil dari kode  
27 program tersebut [3]. Suatu hal yang menarik, Tugas kode program dapat dibagi menjadi 2 jenis  
28 yaitu tugas individu dan tugas kelompok. Pada tugas kelompok merupakan tugas yang ditanggung

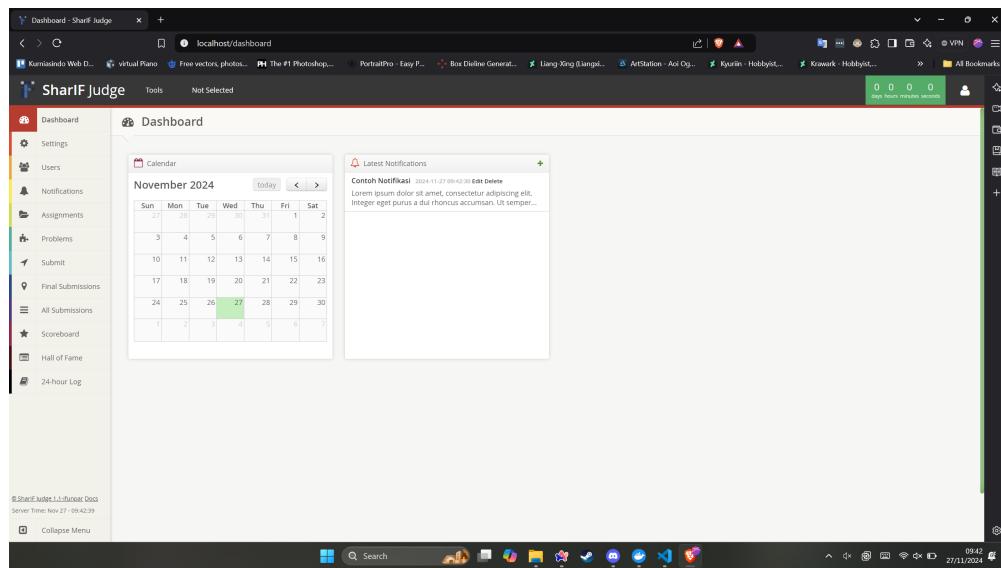
oleh banyak pelajar, biasanya program yang dibuat memiliki antarmuka dan harus diperiksa oleh pengguna khusus yang mengetahui fitur-fitur yang dibutuhkan. Sedangkan tugas individu merupakan sebuah tugas yang diberikan untuk satu individu, biasanya program yang dibuat bersifat algoritmik dan tidak memerlukan antarmuka untuk dijalankan. Program algoritmik sebuah jenis program yang dibuat berdasarkan algoritma untuk menyelesaikan masalah tertentu. Algoritma sendiri adalah langkah-langkah dalam pemecahan masalah secara sistematis [4]. Algoritma itu seperti resep makanan, dimana akan ada bahan-bahan yang dibutuhkan dan serangkaian langkah untuk membuat suatu makanan yang dijelaskan.

Sebagian besar program yang bersifat algoritmik hanya perlu mengambil *input* dari *input* standar seperti angka, huruf, dan sebuah kata atau kalimat dengan format yang sudah ditentukan, seolah-olah *input* ini merupakan *output* dari program lain. Kemudian program algoritmik akan memproses *input* tersebut dalam komputer dan mengeluarkan hasil komputasinya dalam format yang sudah ditentukan untuk dibaca oleh program lain dan memanfaatkan hasil komputasi tersebut. Singkatnya, program algoritmik itu seperti filter antar program. Dengan ini, sistem penilaian secara otomatis dapat dibuat dengan membuat sebuah program yang mengambil kode program, memasukkan *input* sesuai format ke dalam program tersebut, membaca hasil keluaran program, dan menilai hasil keluaran program tersebut [3]. Sistem penilaian otomatis ini diberikan nama *Online Judge*. Terlebih lagi sistem ini dapat dilakukan secara *offline* maupun *online*. Gambar 1.2 menunjukkan bagaimana *online judge* berintegrasi dengan sistem pemberian tugas yang sudah ada.



Gambar 1.2: Sistem Integrasi oleh *Online Judge*

Tugas pemrograman sudah menjadi keseharian dalam pembelajaran pada bidang informatika. Termasuk pada perguruan tinggi pada bidang informatika, maka *online judge* menjadi sebuah kebutuhan termasuk pada Universitas Katolik Parahyangan atau yang biasa disebut UNPAR. *Online Judge* yang digunakan oleh UNPAR dinamakan SharIF-Judge [5] yang merupakan hasil dimodifikasi oleh Stillmen Vallian terhadap Sharif-Judge [6] buatan Mohammad Javad Naderi yang dibuat menggunakan *framework* CodeIgniter dan Bash. Gambar 1.3 merupakan halaman utama setelah masuk ke dalam SharIF-Judge.



Gambar 1.3: Tampilan Awal SharIF Judge

Ujian juga merupakan sebuah bentuk penilaian dari pengajar kepada pelajarnya. Tentunya pelajar maupun mahasiswa ingin memperoleh nilai yang memuaskan dalam ujiannya. Banyak cara yang dilakukan oleh pelajar maupun mahasiswa untuk memperoleh nilai tersebut, salah satunya adalah dengan melakukan kecurangan yaitu *copy paste* atau menyalin jawaban teman atau rekan mereka [1]. Praktek ini diperparah jika ujian dilakukan secara *online*, dikarenakan pelajar dapat mengakses berbagai fasilitas di internet. Oleh karena itu, diperlukan sebuah sistem pada sistem *online judge* untuk mengawasi saat terjadinya ujian online.

Pada saat siswa mengerjakan tugas maupun ujian pembuatan kode program, umumnya pekerjaan kode tersebut dilakukan pada aplikasi eksternal seperti *visual studio code* atau *notepad*. Hal ini juga terjadi pada sistem dalam UNPAR dimana mahasiswa akan membuat kode program pada aplikasi eksternal. Ini membuat pengawasan saat pembuatan kode program lebih sulit untuk dilakukan, terlebih jika ujian dilakukan secara *online*. Maka dari itu, Nicholas Aditya Halim memodifikasi SharIF Judge agar semua sistem pemberian tugas seperti pada Gambar 1.2 dapat dilakukan dalam sistem yang sama yaitu pada SharIF Judge. Sistem yang bangun oleh Nicholas Aditya Halim adalah “Implementasi editor kode pada Sharif Judge” [7], dimana SharIF Judge ditambahkan sebuah *Integrated Development Environment* atau yang disebut dengan IDE. IDE merupakan sebuah sistem yang memiliki kemampuan untuk membuat kode dalam editor kode dan menjalankan kode program tersebut. Dengan adanya IDE, seluruh proses pembuatan kode program dapat dilakukan dalam SharIF Judge. Maka dari itu, seluruh proses sistem pemberian tugas dapat dilakukan dalam satu sistem saja, yaitu SharIF Judge.

Walaupun begitu, pada dasarnya IDE tidak dapat mengawasi jika terjadinya praktek *copy paste*. Maka dari itu pada Tugas akhir ini, IDE pada SharIF Judge akan dimodifikasi untuk menangani hal tersebut dengan ditambahkannya fitur untuk merekam semua ketikan atau kejadian dalam editor kode dalam IDE. Lalu ketikan atau kejadian dalam editor dapat di putar kembali seperti rekaman. Fitur ini akan membuat pengawasan terhadap kegiatan kuliah lebih mudah untuk pengawas dan dapat menjadi bukti kecurangan jika dibutuhkan.

## 1 1.2 Rumusan Masalah

2 Rumusan Masalah yang akan dibahas pada tugas akhir ini adalah:

- 3 1. Bagaimana merencanakan dan mengimplementasikan perekaman dan pemutaran ulang ketikan  
4 mahasiswa pada IDE SharIF-Judge?
- 5 2. Bagaimana cara menyimpan data pemutaran ulang mahasiswa dan tidak mengambil penyim-  
6 panan *file* sangat besar?
- 7 3. Bagaimana tanggapan pengguna terhadap implementasi perekaman dan pemutaran ulang  
8 kode ketikan pada SharIF Judge?

## 9 1.3 Tujuan

10 Tujuan yang ingin dicapai skripsi ini adalah sebagai berikut:

- 11 1. Merencanakan dan mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa  
12 pada IDE SharIF-Judge.
- 13 2. Mencari cara penyimpanan data efektif dalam sebuah *file* dan mengimplementasikannya pada  
14 perekaman dan pemutaran ulang ketikan.
- 15 3. Mendapatkan umpan balik dari tanggapan pengguna terhadap perekaman dan pemutaran  
16 ulang ketikan mahasiswa pada SharIF-Judge.

## 17 1.4 Batasan Masalah

18 Pada penggerjaan tugas akhir ini terhadap batasan sebagai berikut:

- 19 • Perangkat lunak SharIF Judge hanya digunakan pada lingkungan Teknik Informatika Unpar.
- 20 • Perangkat lunak hanya dapat diuji pada mata kuliah pemrograman di mana dosen pembimbing  
21 terlibat.

## 22 1.5 Metodologi

23 Metodologi penggerjaan tugas akhir ini adalah sebagai berikut:

- 24 1. Melakukan studi mengenai komponen yang diperlukan untuk membuat sistem perekaman  
25 dan pemutaran ulang ketikan pada IDE berbasis web.
- 26 2. Merancang sistem perekaman dan pemutaran ulang ketikan berbasis web untuk SharIF Judge
- 27 3. Mengimplementasikan IDE berbasis web pada SharIF Judge.
- 28 4. Melakukan pengujian dan eksperimen.
- 29 5. Menulis dokumen tugas akhir.

## 30 1.6 Sistematika Pembahasan

31 Sistematika pembahasan skripsi ini adalah sebagai berikut:

- 32 • **Bab 1:** Pendahuluan
  - 33 Membahas latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan  
34 sistematika pembahasan.

1     • **Bab 2:** Landasan Teori

2       Membahas teori-teori yang berhubungan dengan penelitian ini, yaitu SharIF Judge, CodeIgniter  
3       3, Twig, IDE, dan Ace.

4     • **Bab 3:** Analisis

5       Membahas analisis terhadap perangkat lunak SharIF Judge dan IDE pada SharIF Judge.

6     • **Bab 4:** Perancangan

7       Membahas perancangan fitur yang diimplementasikan pada SharIF Judge.

8     • **Bab 5:** Implementasi dan Pengujian

9       Membahas implementasi fitur pada SharIF Judge dan pengujian yang dilakukan.

10    • **Bab 6:** Kesimpulan dan Saran

11      Membahas kesimpulan dari penelitian ini dan saran untuk penelitian berikutnya.



1

## BAB 2

2

### LANDASAN TEORI

#### 3 2.1 SharIF Judge

4 SharIF Judge merupakan modifikasi dari *open source* bernama Sharif Judge, sebuah website judge  
5 gratis dengan kemampuan mengkompilasi bahasa C, C++, Java, dan Python. Sharif Judge dibuat  
6 oleh Mohammad Javad Naderi dengan interface web berbahasa PHP menggunakan *framework*  
7 CodeIgniter 3 dan BASH [?]. Modifikasi dilakukan untuk menambahkan fitur pada Sharif Judge  
8 dan juga untuk menyesuaikan sesuai dengan kebutuhan Teknik Informatika UNPAR.

##### 9 2.1.1 Instalasi

10 Ada beberapa prasyarat yang diperlukan dalam menjalankan SharIF Judge pada sebuah *server*  
11 Linux adalah sebagai berikut:

- 12 • *Webserver* dengan PHP versi 5.3 atau lebih dengan `mysqli` extension
- 13 • PHP Command Line Interface (CLI)
- 14 • *Database MySQL* atau PostgreSQL
- 15 • PHP harus memiliki akses untuk menjalankan *shell commands* dengan fungsi `shell_exec`
- 16 • Kemampuan untuk mengompilasi dan menjalankan kode yang dikumpulkan (`gcc, g++, javac,`  
17 `java, python2, dan python3`)
- 18 • Perl

19 Setelah perangkat yang sudah memenuhi prasyarat, berikut merupakan cara instalasi SharIF  
20 Judge:

- 21 1. Unduh versi terakhir dari Sharif Judge dan menempatkannya pada direktori publik.
- 22 2. Pindahkan folder `system` dan `application` ke luar direktori publik. Kemudian simpan  
23 alamatnya pada `index.php`.
- 24 3. Buat sebuah *Database MySQL* atau PostgreSQL.
- 25 4. Atur pengaturan koneksi *database* pada `application/config/database.php`.
- 26 5. Atur pengaturan koneksi RADIUS dan SMTP pada `application/config/secrets.php` jika  
27 dibutuhkan.
- 28 6. Atur agar direktori `application/cache/Twig` dapat ditulis oleh php.
- 29 7. Buka halaman utama SharIF Judge pada *browser* dan ikuti proses instalasi.
- 30 8. Log in dengan akun admin
- 31 9. Pindahkan folder `tester` dan `assignments` ke luar direktori publik. Kemudian simpan  
32 alamatnya pada halaman pengaturan.

### 2.1.2 Users

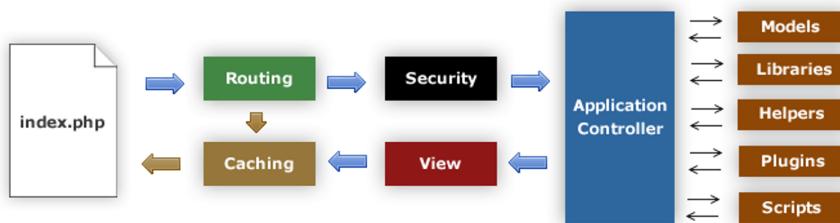
- Pada SharIF Judge, pengguna dibagi menjadi 4 buah *role*. Role yang tersedia adalah sebagai berikut:
1. *admin*
  2. *head instructor*
  3. *instructor*
  4. *student*
- Setiap *role* memiliki akses pada aksi yang berbeda berdasarkan *role*-nya. Tabel 2.1 merupakan aksi-aksi yang dapat dilakukan untuk setiap pengguna pada SharIF Judge.

Tabel 2.1: *Tabel fitur untuk setiap role*

Aksi	Admin	Head Instructor	Instructor	Student
Mengubah <i>Settings</i>	✓	✗	✗	✗
Mengelola Pengguna	✓	✗	✗	✗
Mengelola <i>Assignment</i>	✓	✓	✗	✗
Mengelola Notifikasi	✓	✓	✗	✗
<i>Rejudge</i>	✓	✓	✗	✗
Mengelola <i>Queue</i>	✓	✓	✗	✗
Mendeteksi Kode yang Mirip	✓	✓	✗	✗
Melihat Semua <i>Submission</i>	✓	✓	✓	✗
Mengunduh Kode Final	✓	✓	✓	✗
Memilih <i>Assignment</i>	✓	✓	✓	✓
<i>Submit</i> Kode	✓	✓	✓	✓

### 2.2 CodeIgniter 3

- CodeIgniter 3 adalah sebuah *framework opensource* untuk mempermudah pengguna dalam menggunakan sebuah aplikasi *website* menggunakan bahasa PHP. CodeIgniter 3 bertujuan untuk membantu pengguna dalam membangun sebuah aplikasi *website* lebih cepat dengan menyediakan *library* yang beragam dengan fungsi yang umum digunakan dan tampilan dan *logic* yang simpel. Gambar 2.1 merupakan bagaimana data mengalir pada sistem CodeIgniter.



Gambar 2.1: *Flow Chart* CodeIgniter

- Berikut merupakan penjelasan sederhana dari *flow chart* sistem CodeIgniter 3:
1. `index.php` berfungsi sebagai *front controller* yang akan melakukan inisiasi *resource* utama untuk menjalankan CodeIgniter.

- 1 2. Router meneliti *request* HTTP dan menentukan apa yang harus dilakukan dengan *request* tersebut.
- 3 3. Jika terdapat *file cache*, maka langsung dikirimkan ke *browser* melewati eksekusi sistem yang biasanya.
- 4 4. Sebelum *controller* dimuat, seluruh *request* HTTP dan data dari user disaring terlebih dahulu untuk keamanan.
- 5 5. *Controller* memuat *model*, *library* utama, dan *resource* lainnya yang diperlukan.
- 6 6. *View* akhir lalu dikirim ke *browser* untuk dilihat. *Cache* akan dibuat terlebih dahulu bila diaktifkan.

### 10 2.2.1 Model-View-Controller

11 CodeIgniter merupakan framework berbasis arsitektur Model-View-Controller atau yang selanjutnya  
 12 akan disebut dengan MVC. MVC adalah pendekatan *software* yang memisahkan *logic* aplikasi  
 13 dan tampilannya. Pendekatan ini membuat *website* hanya memiliki sedikit *script* karena tampilan  
 14 *website* terpisah dari *scripting* PHP. Berikut merupakan penjelasan mengenai struktur MVC:

#### 15 Model

16 *Model* mewakili struktur data pada sistem untuk mengambil, memasukkan, dan memperbarui data  
 17 pada *database*. *Model* dapat dibuat dengan membuat sebuah kelas yang mengekstensi `CI_Model`  
 18 dan diletakkan pada `application/models/`.

Kode 2.1: Contoh *model*

```
19 class Blog_model extends CI_Model {
20
21     public $title;
22     public $content;
23     public $date;
24
25
26     public function get_last_ten_entries()
27     {
28         $query = $this->db->get('entries', 10);
29         return $query->result();
30     }
31
32     public function insert_entry()
33     {
34         $this->title    = $_POST['title'];
35         $this->content  = $_POST['content'];
36         $this->date     = time();
37
38         $this->db->insert('entries', $this);
39     }
40
41     public function update_entry()
42     {
43         $this->title    = $_POST['title'];
44         $this->content  = $_POST['content'];
45         $this->date     = time();
46
47         $this->db->update('entries', $this, array('id' => $_POST['id']));
48     }
49
50 }
```

52 Kode 2.1 merupakan contoh model kelas bernama `Blog_model` pada CodeIgniter. *Model*  
 53 `Blog_model` dapat mengambil, menambahkan, dan memperbarui *database* bernama ‘entries’. File  
 54 *model* tersebut akan disimpan pada `application/models/Blog_model`. Selanjutnya, pengguna

- 1 dapat memanggil *Model* tersebut pada *file controller* (akan dijelaskan pada bagian [Controller](#)) untuk  
2 memanggil model pada Kode [2.1](#) dengan menggunakan notasi sebagai berikut:

```
3     $this->load->model('Blog_model');
```

- 4 Untuk memanggil *method* yang terdapat pada model tersebut, notasi yang digunakan adalah  
5 sebagai berikut:

```
6         $this->Blog_model->get_last_ten_entries();
```

- 7 Notasi diatas akan memuat *model* dengan nama `Blog_model` dan akan memanggil *method*  
8 `get_last_ten_entries`.

## 9 View

- 10 *View* adalah informasi yang akan di tunjukkan kepada user. Biasanya *view* merupakan sebuah  
11 halaman web, tetapi pada CodeIgniter, view dapat berupa pecahan halaman seperti *header*, *footer*,  
12 *sidebar*, dan lainnya. Pecahan halaman tersebut dapat dimasukkan secara fleksibel ke dalam *view*  
13 lainnya apabila dibutuhkan.

Kode 2.2: Contoh *view*

```
14
15 1 <html>
16 2 <head>
17 3     <title>My Blog</title>
18 4 </head>
19 5 <body>
20 6     <h1>Welcome to my Blog!</h1>
21 7 </body>
22 8 </html>
```

- 24 Kode [2.2](#) merupakan contoh dari *file view* pada CodeIgniter. File akan disimpan pada direktori  
25 `application/views/`. Untuk dapat diperlihatkan dibutuhkannya penggalian halaman pada *file*  
26 *controller* dengan cara sebagai berikut:

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

- 28 Notasi diatas akan mengembalikan halaman *view* dengan nama `name` yang terletak pada direktori  
29 `application/views/name.php` dan menampilkannya kepada pengguna.

## 30 Controller

- 31 *Controller* adalah bagian utama dari aplikasi CodeIgniter, berfungsi sebagai perantara antara  
32 *model*, *view*, dan *resources* lainnya yang dibutuhkan untuk memproses HTTP *request* dan mem-  
33 buat sebuah web page. Kelas *Controller* akan mengekstensi `CI_Controller` dan disimpan pada  
34 `application/controllers/`. Contoh *controller* ditunjukkan pada Kode [2.3](#).

Kode 2.3: Contoh *controller*

```
35
36 1 <?php
37 2 class Blog extends CI_Controller {
38 3
39 4     public function index()
40 5     {
41 6         echo 'Hello_World!';
42 7     }
}
```

```
1|     public function comments()
2|     {
3|         echo 'Look_at_this!';
4|     }
5| }
```

8 Kode 2.3 berfungsi dalam mengembalikan string sesuai dengan fungsi *controller* yang dipanggil.  
9 Nama file *controller* pada direktori `application/controllers/blog.php` dan metode diatas akan  
10 dijadikan segmen pada URL seperti berikut:

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

<sup>12</sup> URL diatas akan mengembalikan sebuah teks ‘Hello World!’.

Kode 2.4: Contoh memuat *model* dan menampilkan *view*

```
13
14 1 class Blog_controller extends CI_Controller {
15 2     public function blog()
16 3     {
17 4         $this->load->model('blog');
18 5
19 6         $data['query'] = $this->blog->get_last_ten_entries();
20 7
21 8         $this->load->view('blog', $data);
22 9     }
23 0 }
```

Pada CodeIgniter, *model* dan *view* hanya dapat dimuat melalui controller. Seperti contoh, Kode 2.4 akan memuat *model blog* dan mengambil data dari *database*, lalu menampilkan *view* yang memuat data tersebut.

## 28 2.2.2 CodeIgniter URLs

29 URL pada CodeIgniter menggunakan *segment-based approach* dibandingkan dengan *query string*  
30 *approach* yang biasanya dipakai. *Segment-based approach* dirancang untuk *search-engine* dan dapat  
31 mempermudah pengguna juga. Berikut merupakan contoh dari URL CodeIgniter:

[example.com/news/article/my\\_article](http://example.com/news/article/my_article)

Struktur URL pada CodeIgniter juga mengikuti pendekatan MVC (Referensi 2.2.1) dan biasanya memiliki struktur sebagai berikut:

example.com/class/function/ID

- 36 1. Segmen pertama mewakili kelas *controller* yang ingin dipanggil.  
37 2. Segmen berikutnya mewakili fungsi kelas atau *method* yang ingin di panggil.  
38 3. Segmen ketiga dan selanjutnya mewakili *identifier* atau pengenal dan variable-variable lain  
39 yang akan di kirimkan ke *controller*.

40 2.2.3 *Helpers*

41 *Helpers* merupakan sebuah kumpulan fungsi untuk membantu dalam sebuah kategori tertentu. *File*  
42 *helpers* terdapat pada direktori `system\helpers` atau `application\helpers`. Penggunaan *helpers*  
43 dalam *CodeIgniter* adalah dengan memuat file helpers dalam fungsi atau kelas *Controller* dengan  
44 cara seperti berikut ini:

```
1 $this->load->helper('name')
```

2 Setelah *helper* dimuat dalam fungsi, maka kumpulan fungsi dalam *file helper* dapat langsung  
 3 dipanggil.

### 4 2.3 Twig

5 Twig merupakan sebuah *template engine* untuk PHP. Ada beberapa *expression*, *expression*, atau  
 6 *statement* yang ditemukan pada template Twig adalah sebagai berikut:

- 7 • Pewarisan *Template*
- 8 • Struktur Kontrol (menggunakan kondisional, *looping*)
- 9 • Filter
- 10 • Variable pada PHP

11 Pada saat template dievaluasi, semua *variable* atau *expression* akan dibuang menjadi value dan  
 12 *tag* yang mengontrol logika template.

Kode 2.5: Contoh template Twig

```
13
14 1  {% extends "base.html" %} 
15 2  {% block navigation %} 
16 3  <ul id="navigation">
17 4  {% for item in navigation %} 
18 5  <li>
19 6  <a href="{{item.href}}>
20 7  {% if item.level == 2 %}&nbsp;&nbsp;{% endif %}
21 8  {{ item.caption|upper }} 
22 9  </a>
23 0  </li>
24 1  {% endfor %}
25 2  </ul>
26 3  {% endblock navigation %}
```

28 Kode 2.5 merupakan contoh sebuah template Twig. Terdapat dua jenis *delimiter*, yaitu `{% ... %}`  
 29 dan `{{ ... }}`. *Delimiter* `{% ... %}` digunakan untuk menjalankan sebuah *statement* seperti *for-loops*, sedangkan *delimiter* `{{ ... }}` digunakan untuk mengubah sebuah *variable* atau *expression* menjadi nilai sesungguhnya.

### 32 2.4 Integrated Development Environment

33 Intergrated Development Environment (IDE) merupakan sebuah aplikasi yang menyediakan berbagai  
 34 peralatan yang diperlukan untuk membantu pengembangan perangkat lunak. Beberapa peralatan  
 35 umum yang dimiliki oleh sebuah IDE adalah sebagai berikut:

- 36 • *Editor*  
 37 Editor teks sebagai tempat untuk mengetik kode, dapat dilengkapi dengan berbagai fitur  
 38 seperti *syntax highlighting* (menampilkan teks dengan warna yang berbeda untuk mengintensifkan  
 39 keterbacaan kode) dan *word completion* (menampilkan prediksi kata yang sedang atau yang  
 40 akan diketik pengguna).
- 41 • *Complier*  
 42 Digunakan untuk menterjemahkan kode program yang dibuat pada editor teks ke dalam  
 43 sebuah program yang dapat dijalankan oleh komputer.

1     • *Execution*

2       Menjalankan kode program yang sudah dikompilasi, dengan input jika dibutuhkan, dan  
3       mengembalikan hasilnya.

4 **2.5 Ace**

5 Ace merupakan *library* yang menyediakan sebuah editor kode yang dapat dimasukkan ke dalam  
6 sebuah web page dan dikembangkan menggunakan bahasa *Javascript*. Ace memiliki kemampuan  
7 yang sama seperti editor kode pada umumnya. Berikut merupakan beberapa fitur utama yang  
8 dimiliki oleh Ace:

- 9     • *Syntax highlighting* untuk bahasa pemrograman.  
10    • Automatic indent dan outdent.  
11    • Kemampuan *cut*, *copy*, dan *paste*.  
12    • Kemampuan *drag and drop* teks menggunakan mouse.  
13    • Banyak *Cursors* dan *selections*  
14    • *Line wrapping*  
15    • *Code folding*

16      Untuk mengintegrasikan *library* Ace dalam sebuah web page, Ace perlu ditanam dalam sebuah  
17     web page. Salah satu cara untuk menanam Ace ke dalam sebuah web page adalah dengan mem-*build*  
18     *library* atau menngunduh hasil dari *build* folder bernama *src* versi *pre-packaged* yang disediakan  
19     oleh Ace. Hasil dari *building library* Ace adalah sebuah folder yang dapat ditaruh dalam direktori  
20     lokal. Dalam folder tersebut, terdapat file *javascript* bernama *ace.js* yang dapat dipanggil dalam  
21     web page untuk menanam *library* Ace dalam web page. Cara untuk menanamkan file tersebut sama  
22     dengan cara untuk memasukkan file *javascript* pada umumnya yaitu dengan cara seperti berikut:

23            <script src="/ace-builds/ace.js" type="text/javascript"></script>

24      Setelah *library* Ace ditanam untuk mengakses berbagai macam fitur yang disediakan, maka kelas  
25     yang disediakan Ace dapat dipanggil. Berikut merupakan beberapa kelas penting yang terdapat  
26     pada *library* Ace adalah sebagai berikut:

27     • **Ace**

28       Kelas **Ace** merupakan kelas utama untuk menyiapkan editor kode Ace pada *browser*. Ace  
29       memiliki fungsi utama yang penting yaitu fungsi **edit** yang akan membuat sebuah editor  
30       dalam web page pada element beridentitas argumen yang diberikan saat dipanggil. Fungsi  
31       **edit** akan mengembalikan kelas **Editor**.

32     • **Editor**

33       Entri utama untuk fungsionalitas *library* Ace. Editor sendiri merepresentasikan editor kode  
34       yang dibuat pada web page. Editor juga menjadi kelas utama untuk mengakses kelas-kelas yang  
35       berhubungan dengan editor kode dengan mengakses kelas variable **Editor** seperti **session**  
36       dalam editor kode. Kelas **Editor** sendiri dapat dikonfigurasikan sesuai dengan fungsi yang  
37       disediakan seperti **setTheme** yang mengubah warna editor kode sesuai dengan *theme* yang di  
38       pasang.

39     • **EditSession**

40       Sebuah kelas yang menyimpan semua status dalam editor seperti isi editor, *selection*, dan

1 lain-lain. Kelas ini dinamakan `EditSession`, tetapi untuk mengakses dari kelas `Editor`,  
 2 variable `EditSession` dinamakan `session`.

3 • **Anchor**

4 Menangani posisi *pointer* pada dokumen. Saat teks dimasukkan atau dihapus, posisi *anchor*  
 5 akan diperbarui.

6 • **Document**

7 Menyimpan teks dokumen.

8 • **Range**

9 Kelas ini digunakan di berbagai tempat untuk mengindikasikan suatu wilayah di dalam editor.

10 Kelas ini menyimpan posisi baris awal dan kolom awal, serta baris akhir dan kolom akhir.

11 • **Selection**

12 Kelas ini menyimpan posisi yang dipilih oleh pengguna dalam editor.

13 • **Commands**

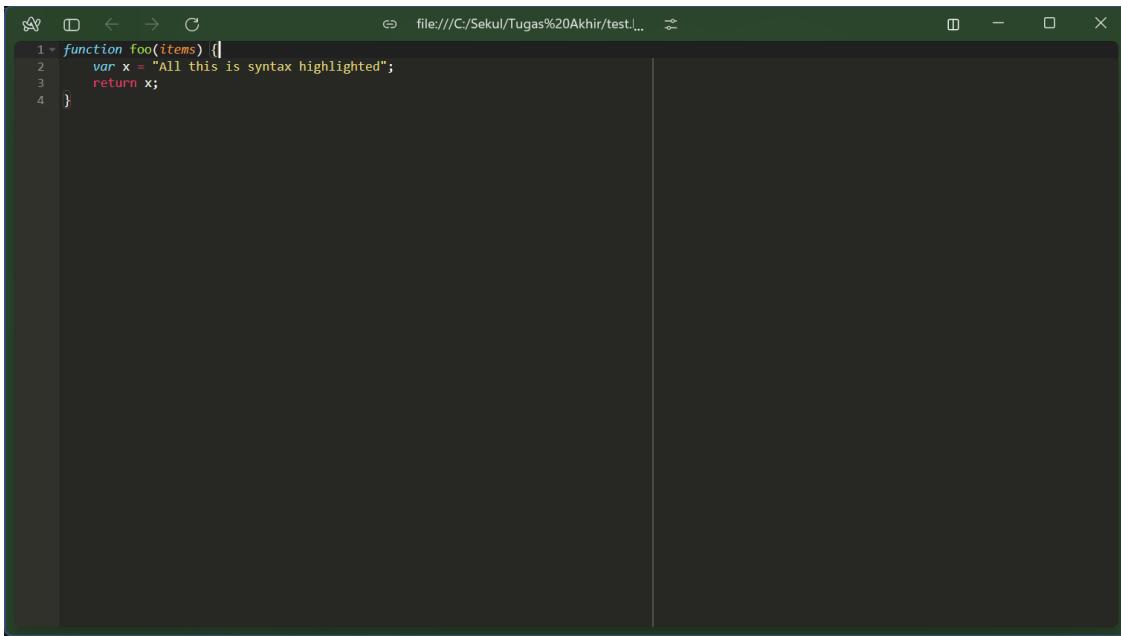
14 Kelas ini digunakan untuk menjalankan perintah pada sebuah editor. Contoh perintah yang  
 15 sudah ada dalam editor yaitu *insert*, *copy*, *paste*.

Kode 2.6: Contoh kode penggunaan Ace

```

16 1<!DOCTYPE html>
17 2<html lang="en">
18 3<head>
19 4<title>ACE in Action</title>
20 5<style type="text/css" media="screen">
21 6    #editor {
22 7        position: absolute;
23 8        top: 0;
24 9        right: 0;
25 0        bottom: 0;
26 1        left: 0;
27 2    }
28 3</style>
29 4</head>
30 5<body>
31 6
32 7<div id="editor">function foo(items) {
33 8    var x = "All_this_is_syntax_highlighted";
34 9    return x;
35 0}</div>
36 1
37 2<script src="/ace-builds/ace.js" type="text/javascript"></script>
38 3<script>
39 4    var editor = ace.edit("editor");
40 5    editor.setTheme("ace/theme/monokai");
41 6    editor.session.setMode("ace/mode/javascript");
42 7</script>
43 8</body>
44 9</html>
```

47 Kode 2.6 merupakan cara penggunaan Ace pada sebuah `div` dengan id `editor`. Ace juga memiliki  
 48 beberapa konfigurasi, seperti contoh ini yaitu menggunakan tema *monokai* dan menggunakan *syntax*  
 49 *highlighting* untuk bahasa pemrograman JavaScript. Gambar 2.2 menunjukkan hasil web page yang  
 50 dibuka dalam *browser* menggunakan Kode 2.6.



Gambar 2.2: Hasil Web Page *Library Ace*

### **2.5.1 Perekaman Event**

Pada editor kode Ace, disediakannya fungsi *event listener* atau pendengar *event* atau kejadian yang berhubungan dengan sebuah kelas. Pada *event listener* ini akan disediakannya sebuah fungsi *callback* yang akan dipanggil saat *event* tersebut terjadi. Berikut merupakan beberapa *event listener* dalam sebuah kelas:

- **Editor**

Pada editor sendiri disediakannya satu *event listener* yaitu `mouseup` yang akan mendengarkan saat melepaskan tombol pada tetikus atau *mouse*.

- **EditSession**

Pada kelas *session* ada satu *event listener* yaitu `change` yang akan mendengarkan perubahan pada isi atau kode pada editor kode. Pada fungsi *callback* yang akan dijalankan oleh *event listener* ini akan diberikan parameter `delta` yang menunjukkan perubahan apa yang terjadi pada editor kode.

- **Selection**

Pada kelas *selection* ada beberapa *event listener* yaitu sebagai berikut:

- `changeCursor` : Mendengarkan perubahan pada kurSOR atau *anchor* dalam editor kode.
- `changeSelection` : Mendengarkan perubahan pemilihan isi kode dalam editor kode.

- **Commands**

Pada kelas ini tersedia dua *event listener* yaitu `exec` dan `afterExec`. `exec` akan mendengarkan saat perintah akan dijalankan pada editor kode, sedangkan `afterExec` akan mendengarkan perintah yang sudah selesai dijalankan pada editor kode. Pada fungsi *callback* yang akan dijalankan oleh *event listener* ini akan diberikan perintah yang dijalankan oleh kelas *Commands*.

Untuk menggunakan fungsi *event listener* pada kelas yang diinginkan, dibutuhkan fungsi `on` pada kelas tersebut. Fungsi `on` memiliki dua parameter yaitu nama *event* ingin didengar (`exec` atau `change`) dan sebuah fungsi *callback* yang akan dijalankan saat *event* terjadi. Kode 2.7 merupakan

- 1 perhubahan kode yang dilakukan dalam tag `<script>` pada Kode 2.6 agar perubahan isi editor
- 2 dapat didengar.

Kode 2.7: Contoh kode event listener

```

3 1 <script src="/ace-builds/ace.js" type="text/javascript"></script>
5 2 <script>
6 3   var editor = ace.edit("editor");
7 4   editor.setTheme("ace/theme/monokai");
8 5   editor.session.setMode("ace/mode/javascript");
9 6
10 7   editor.session.on("change", (delta) => {
11 8     console.log(delta);
12 9     // Contoh Keluaran :
13 10    // {
14 11      //   action: "insert"
15 12      //   end: {row: 3, column: 5}
16 13      //   id: 1
17 14      //   lines: ['a']
18 15      //   start: {row: 3, column: 4}
19 16    // }
20 17  });
21 18 </script>

```

Kode 2.7 akan menggunakan *event listener* `change` dalam kelas `EditSession`, dengan mengakses kelas `EditSession` melalui `editor` yang dinamakan `session`. Pada kelas tersebut akan dijalankan fungsi `on` dengan parameter “`change`” dan sebuah fungsi anonimous sebagai fungsi *callback* yang akan memprint ke `console` isi perubahan pada editor kode.

## 2.6 Chart.js

Chart.js merupakan sebuah *library javascript open-source* untuk membuat visualisasi data bagan interaktif berbasis `canvas` dalam web page [8]. Chart.js memiliki fitur-fitur yang dapat digunakan untuk mendukung dan mempermudah visualisasi data dalam web page. Berikut merupakan beberapa fitur yang dimiliki oleh Chart.js:

- Chart.js menyediakan berbagai tipe bagan yang dapat digunakan dan juga memiliki opsi penyesuaian yang sering digunakan.
- Chart.js memiliki konfigurasi bawaan yang bagus dan mudah untuk diintegrasikan dalam sebuah web page.
- Chart.js menggunakan *canvas HTML5 rendering* yang membuat sangat cepat terutama untuk data yang besar.

Identik dengan *library Ace* untuk menintegrasikan *library Chart.js* dalam sebuah web page, *library Chart.js* dapat dibuild dan dimasukkan ke dalam folder projek dan menambahkan file `javascript` bernama `chart.js` dalam folder `dist` ke dalam web page menggunakan cara yang identik dengan cara memasukkan file `javascript` pada umumnya yaitu dengan cara sebagai berikut:

```
42 <script src="/chartjs/dist/chart.js" type="text/javascript"></script>
```

Setelah itu *library Chart.js* dapat digunakan dengan membuat sebuah kelas `javascript` baru bernama `Chart`. Untuk membuat kelas `Chart` dibutuhkan 2 argumen yaitu elemen `canvas` dalam HTML web page dan sebuah objek `javascript` yang dapat diisi dengan opsi-opsi yang diinginkan dengan menspesifikasi `key` dan `value` yang sesuai dengan opsi yang diinginkan. Berikut merupakan beberapa `key` dan `value` yang dapat digunakan ada dalam opsi *library Chart.js*:

1     • **type**

2       **type** hanya menerima sebuah kata yang menjadi tipe utama bagan yang dibuat oleh *library*  
 3       Chart.js, tetapi tipe ini dapat berubah mengikuti data yang diberikan. Berikut merupakan  
 4       beberapa tipe-tipe yang ada dalam *library* Chart.js:

5       – **bar**

6           Bagan **bar** menyediakan cara untuk menvisualisasikan data sebagai batang vertikal.  
 7           Bagan ini biasanya digunakan untuk menunjukkan data tren dan perbandingan beberapa  
 8           set data secara berdampingan.

9       – **line**

10          Bagan **line** adalah cara menvisualisasikan data sebagai titik data yang disambungkan  
 11          dengan garis. Identik dengan Bagan **bar**, Bagan **line** juga digunakan untuk menunjukkan  
 12          data tren dan perbandingan beberapa set data secara berdampingan.

13     • **data**

14       **data** sendiri menerima *value* objek *javascript* dengan isi **labels** dan **dataset**. Kedua *key*  
 15       menerima sebuah *array* dengan isi yang berbeda. **labels** hanya menerima sebuah *array* berisi  
 16       teks untuk label data horizontal atau vertikal. **dataset** menerima *array* primitive type, *array*  
 17       dengan isi *array*, dan *array objek*. *objek* dalam **dataset** menerima *key data* dan juga memiliki  
 18       beberapa *key* opsi yaitu **label** untuk melabelkan data dalam horizontal maupun vertikal. *Key*  
 19       **data** menerima *array* dengan isi *array objek* dengan data yang ingin divisualisasikan.

20     • **options**

21       *Key options* merupakan fitur utama dari kelas **Chart** dan hanya menerima sebuah objek  
 22       *javascript* yang memiliki banyak *key* untuk menyesuaikan bagan yang dibuat oleh *library*  
 23       Chart.js. Salah satu *key* dalam **options** adalah **scales** yang digunakan untuk mengatur data  
 24       yang ditampilkan untuk aksis X dan Y. Salah satu pengaturannya adalah untuk menumpuk  
 25       data dengan data yang sama di aksis yang sama yaitu dengan menggunakan *key stacked*  
 26       dalam aksis X atau Y.

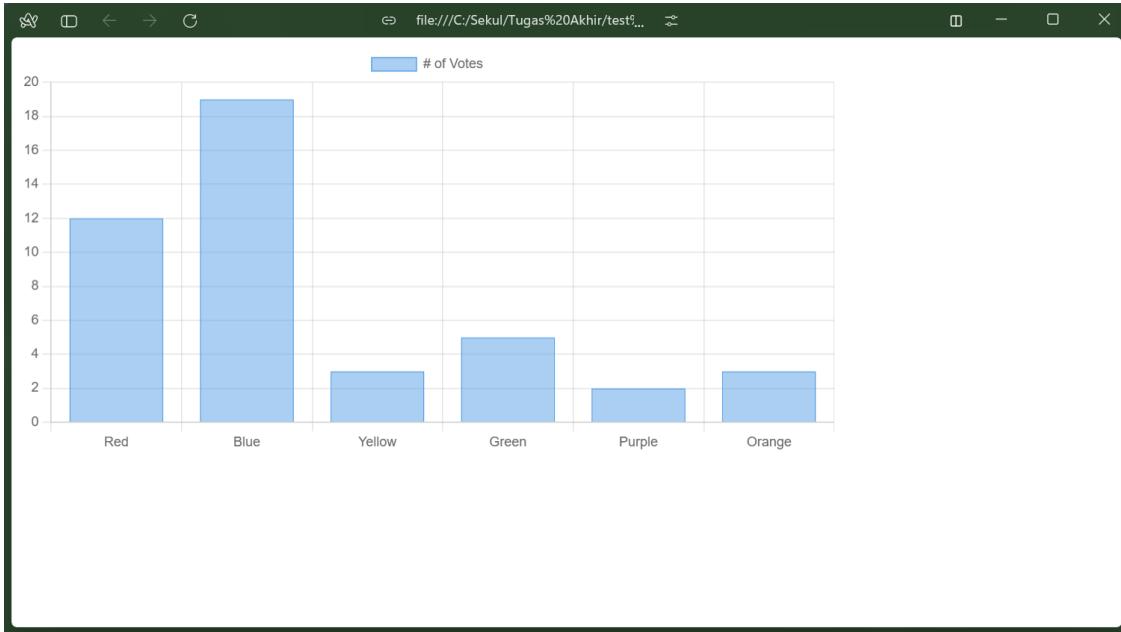
Kode 2.8: Contoh kode penggunaan Chart.js

```

27
28 1 <!DOCTYPE html>
29 2 <html lang="en">
30 3 <body>
31 4 <div>
32 5   <canvas id="myChart"></canvas>
33 6 </div>
34 7
35 8 <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
36 9
37 0 <script>
38 1   const ctx = document.getElementById('myChart');
39 2
40 3   new Chart(ctx, {
41 4     type: 'bar',
42 5     data: {
43 6       labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
44 7       datasets: [{
45 8         label: '# of Votes',
46 9         data: [12, 19, 3, 5, 2, 3],
47 10        borderWidth: 1
48 11      }]
49 12    },
50 13    options: {
51 14      scales: {
52 15        y: {
53 16          beginAtZero: true
54 17        }
55 18      }
56 19    }
57 20  }>
```

```
B0    });
B1 </script>
B2 </body>
B3 </html>
```

- 6 Kode 2.8 merupakan contoh penggunaan *library* Chart.js pada sebuah *canvas* dengan id **myChart**.  
7 *Key options* pada contoh ini menggunakan **beginAtZero** yang membuat data dimulai dari nol.  
8 Gambar 2.3 merupakan hasil web page yang dibuka dalam *browser* dengan Kode 2.8.



Gambar 2.3: Hasil Web Page *Library* Chart.js

1

## BAB 3

2

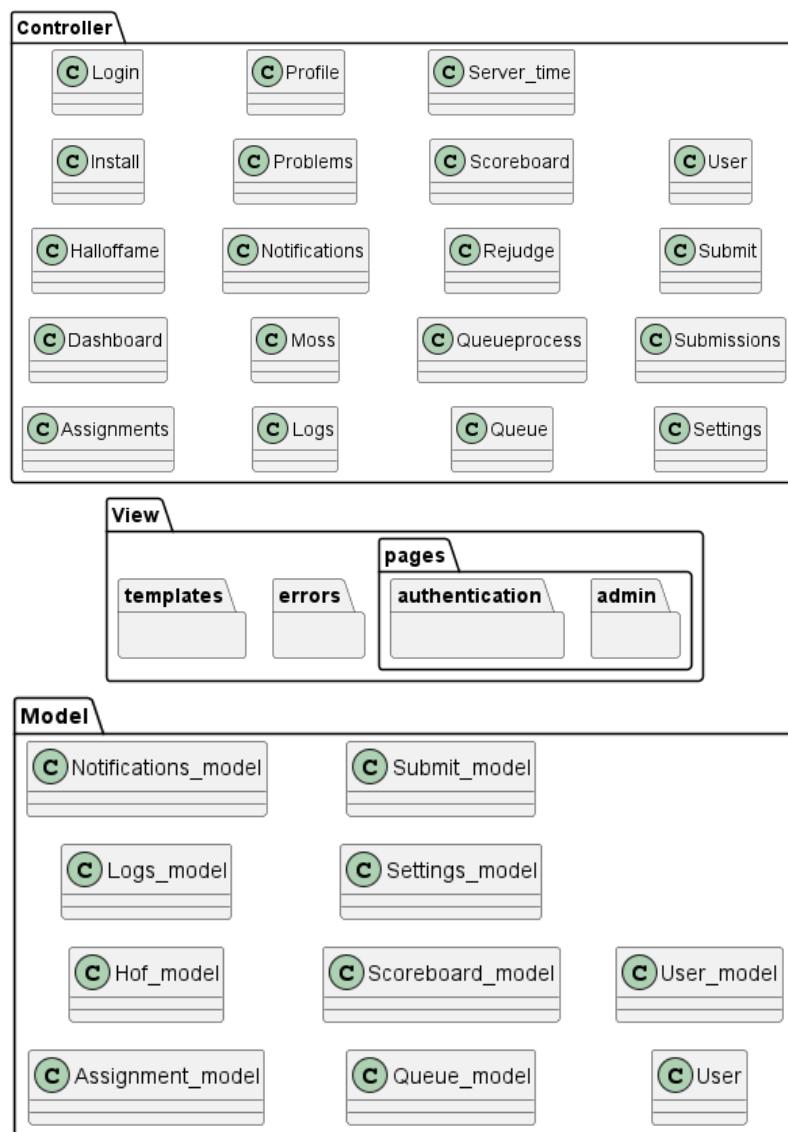
## ANALISIS

### 3 3.1 Analisis Sistem Kini

4 Seperti yang sudah dibahas pada subbab 2.1, SharIF Judge merupakan sebuah website judge yang  
5 dimodifikasi sesuai dengan kebutuhan Teknik Informatika UNPAR. Analisis diawali dengan MVC  
6 aplikasi SharIF Judge. Berikut merupakan hasil eksplorasi SharIF Judge yang telah dilakukan:

#### 7 3.1.1 Model, View, Controller

8 SharIF Judge menggunakan *framework* CodeIgniter 3 yang berbasis arsitektur Model-View-Controller  
9 seperti yang dijelaskan pada subbab 2.2.1. Gambar 3.1 merupakan kelas diagram struktur MVC  
10 pada SharIF Judge.



Gambar 3.1: Struktur MVC pada SharIF Judge

1 Berikut merupakan hasil eksplorasi dari struktur MVC pada SharIF Judge:

## 2 Model

3 Analisis MVC akan dimulai dengan *model* yang berada pada direktori `application/models`. Direktori *Model* berisi kelas-kelas yang digunakan untuk mengelola dan mengembalikan data dari *database*. Gambar 3.2 merupakan struktur kelas *model* dalam SharIF Judge.



Gambar 3.2: Struktur Kelas Model pada SharIF Judge

1 Berikut merupakan penjelasan dari kelas *model* dan fungsi-fungsinya yang terdapat pada SharIF  
 2 Judge:

3 • **Assignment\_model.php**

4 Model ini digunakan untuk mengelola tabel *assignments* dan mengembalikan informasi yang  
 5 digunakan dalam halaman *assignment* dan *problem*. Fungsi yang dimiliki adalah sebagai  
 6 berikut:

- 7 – `add_assignment($id, $edit)`  
     Menambahkan atau memperbarui sebuah *assignment*.
- 8 – `delete_assignment($assignment_id)`  
     Menghapus sebuah *assignment*.
- 9 – `all_assignments()`  
     Mengembalikan daftar semua *assignment* dan informasinya.
- 10 – `new_assignment_id()`  
     Mendapatkan nomor terkecil dan dapat digunakan sebagai *id assignment* terbaru.
- 11 – `all_problems($assignment_id)`  
     Mengembalikan daftar semua *problems* dari sebuah *assignment*.
- 12 – `problem_info($assignment_id, $problem_id)`  
     Mengembalikan semua informasi sebuah *problem*
- 13 – `assignment_info($assignment_id)`  
     Mengembalikan semua informasi sebuah *assignment*

```

1   – is_participant($participants, $username)
2     Mengembalikan sebuah boolean yang menyatakan bahwa $username terdapat dalam
3     $participants.
4   – increase_total_submits($assignment_id)
5     Menambahkan jumlah total submits sebanyak satu pada sebuah assignment.
6   – set_moss_time($assignment_id)
7     Memperbarui “Moss Update Time” pada sebuah assignment.
8   – get_moss_time($assignment_id)
9     Mengembalikan “Moss Update Time” pada sebuah assignment.
10  – save_problem_description($assignment_id, $problem_id, $text, $type)
11    Menambahkan atau memperbarui deskripsi pada sebuah problem.
12  – _update_coefficients($a_id, $extra_time, $finish_time, $new_late_rule)
13    Memperbarui koefisien dari sebuah assignment.

```

- **Hof\_model.php**

Model ini digunakan untuk mengembalikan informasi yang digunakan dalam *hall of fame* dari tabel **submissions**. Fungsi yang dimiliki adalah sebagai berikut:

```

17  – get_all_final_submission()
18    Mengembalikan seluruh total nilai final submission untuk semua user.
19  – get_all_user_assignments($username)
20    Mengembalikan nilai final submission pada semua problem untuk user tertentu.

```

- **Logs\_model.php**

Model ini berfungsi untuk mengelola tabel **logins** dan mengembalikan catatan *login*. Fungsi yang dimiliki adalah sebagai berikut:

```

24  – insert_to_logs($username, $ip_address)
25    Mencatat login sebuah user dan menghapus catatan jika melebihi 24 jam.
26  – get_all_logs()
27    Mengembalikan semua catatan login.

```

- **Notifications\_model.php**

Model ini digunakan untuk mengelola tabel **notifications**. Fungsi yang dimiliki adalah sebagai berikut:

```

31  – get_all_notifications()
32    Mengembalikan semua notifications.
33  – get_latest_notifications()
34    Mengembalikan 10 notifications terbaru.
35  – add_notification($title, $text)
36    Menambahkan notification baru.
37  – update_notification($id, $title, $text)
38    Memperbarui sebuah notification.
39  – delete_notification($id)
40    Menghapus sebuah notification.
41  – get_notification($notif_id)
42    Mengembalikan sebuah notification.

```

- ```
1   - have_new_notification($time)
2     Mengembalikan sebuah boolean yang menyatakan bahwa terdapatnya notification baru.
3 • Queue_model.php
4   Model ini digunakan untuk mengelola tabel queue dan menampilkan data queue. Fungsi yang
5   dimiliki adalah sebagai berikut:
6   - in_queue($username, $assignment, $problem)
7     Mengembalikan sebuah boolean yang menyatakan bahwa username masih memiliki queue
8     dalam sebuah problem.
9   - get_queue()
10    Mengambil semua submission queue.
11   - empty_queue()
12    Menghapus semua queue.
13   - add_to_queue($submit_info)
14    Menambahkan sebuah submission ke dalam queue.
15   - rejudge($assignment_id, $problem_id)
16    Menambahkan seluruh submissions dalam sebuah problem ke dalam queue untuk dinilai
17    ulang.
18   - rejudge_single($submission)
19    Menambahkan sebuah submission ke dalam queue untuk dinilai ulang.
20   - get_first_item()
21    Mengembalikan item pertama dalam tabel queue.
22   - remove_item($username, $assignment, $problem, $submit_id)
23    Menghapus sebuah item tertentu dalam tabel queue.
24   - save_judge_result_in_db ($submission, $type)
25    Menyimpan hasil penilaian judge ke dalam database.
26   - add_to_queue_exec($submit_info)
27    Menambahkan sebuah dummy submission yang digunakan hanya untuk dijalankan ke
28    dalam queue.
```
- Scoreboard\_model.php
- Model ini digunakan untuk mengelola tabel **scoreboard**. Fungsi yang dimiliki adalah sebagai berikut:
- ```
32  - _generate_scoreboard($assignment_id)
33   Menghasilkan scoreboard untuk sebuah assignment dari nilai akhir semua submission.
34  - update_scoreboards()
35   Memperbaharui scoreboard untuk semua assignment.
36  - update_scoreboard($assignment_id)
37   Memperbaharui scoreboard untuk sebuah assignment.
38  - get_scoreboard($assignment_id)
39   Mengembalikan scoreboard pada sebuah assignment.
```
- Settings\_model.php
- Model ini digunakan untuk mengelola tabel **settings**. Fungsi yang dimiliki adalah sebagai berikut:

- 1     – `get_setting($key)`  
2       Mengembalikan nilai dari sebuah `$key` pada tabel `settings`.
- 3     – `set_setting($key, $value)`  
4       Memperbarui nilai dari pada `setting` `$key`.
- 5     – `get_all_settings()`  
6       Mengembalikan seluruh `settings`.
- 7     – `set_settings($settings)`  
8       Memperbarui seluruh nilai perubahan `settings`.

- 9     • **Submit\_model.php**

Model ini digunakan untuk mengelola tabel `submission`. Fungsi yang dimiliki adalah sebagai berikut:

- 12    – `get_submission($username, $assignment, $problem, $submit_id)`  
13      Mengembalikan sebuah baris data `submission` tertentu.
- 14    – `get_final_submissions($a_id, $u_vl, $uname, $p_num, $fil_u, $fil_prob)`  
15      Mengembalikan seluruh `final submission` pada sebuah `assignment`. *User* dengan role  
16      *student* hanya dapat melihat `final submission` dirinya sendiri.
- 17    – `get_all_submissions($a_id, $u_vl, $uname, $p_num, $fil_u, $fil_prob)`  
18      Mengembalikan seluruh `submission` pada sebuah `assignment`. *User* dengan role *student*  
19      hanya dapat melihat `submission` dirinya sendiri.
- 20    – `count_final_submissions($a_id, $u_vl, $uname, $fil_u, $fil_prob)`  
21      Mengembalikan jumlah `final submission` pada sebuah `assignment`.
- 22    – `count_all_submissions($a_id, $u_vl, $uname, $fil_u, $fil_prob)`  
23      Mengembalikan jumlah `submission` pada sebuah `assignment`.
- 24    – `set_final_submission($username, $assignment, $problem, $submit_id)`  
25      Memperbarui sebuah `submission` menjadi `final submission`.
- 26    – `add_upload_only($submit_info)`  
27      Menyimpan hasil `upload only problem` ke dalam tabel `database`.

- 28     • **User.php**

Model ini digunakan untuk menyimpan `settings` sebuah `user`. Fungsi yang dimiliki adalah sebagai berikut:

- 31    – `select_assignment($assignment_id)`  
32      Menyimpan `assignment` yang dipilih oleh `user`.
- 33    – `save_widget_positions($positions)`  
34      Menyimpan posisi `widget` sebuah `user`.
- 35    – `get_widget_positions()`  
36      Mendapatkan posisi `widget` sebuah `user`.

- 37     • **User\_model.php**

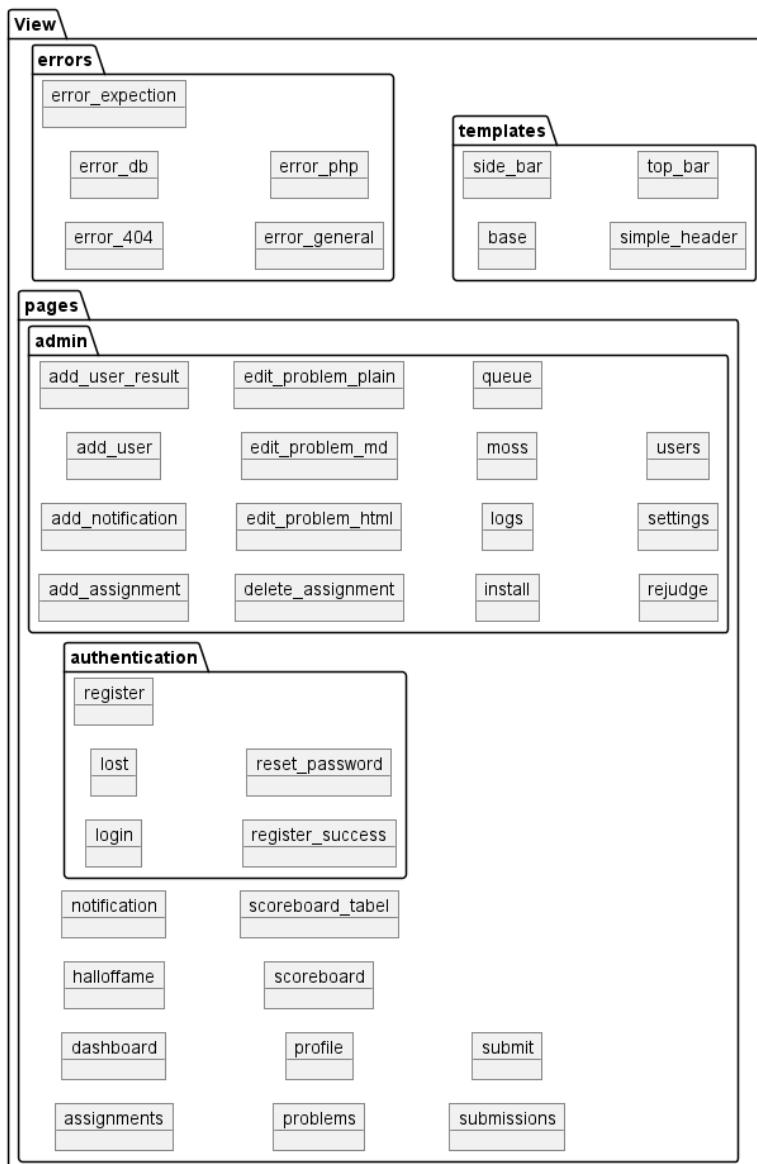
Model ini digunakan untuk mengelola tabel `users`. Fungsi yang dimiliki adalah sebagai berikut:

- 40    – `have_user($username)`  
41      Mengembalikan sebuah `boolean` yang menyatakan `$username` sudah ada pada `database`.
- 42    – `user_id_to_username($user_id)`

1 Mengembalikan *username* dari `$user_id`.  
2 – `username_to_user_id($username)`  
3 Mengembalikan *user id* dari *username*.  
4 – `have_email($email, $username)`  
5 Mengembalikan sebuah *boolean* yang menyatakan jika *user* memiliki *email* pada *database*.  
6 – `add_user($username, $email, $display_name, $password, $role)`  
7 Menambahkan satu *user* baru ke dalam *database*.  
8 – `add_users($text, $send_mail, $delay)`  
9 Menambahkan banyak *user* baru ke dalam *database*.  
10 – `delete_user($user_id)`  
11 Menghapus sebuah *user* dalam *database*.  
12 – `delete_submissions($user_id)`  
13 Mendelete semua *submissions* yang di *submit* oleh sebuah *user*.  
14 – `validate_user($username, $password)`  
15 Mengembalikan sebuah *boolean* yang menyatakan bahwa `$password` dan `$username`  
16 – `selected_assignment($username)`  
17 Mengembalikan *assignment* yang dipilih oleh `$username`.  
18 – `get_names()`  
19 Mengembalikan semua *display name* pada tabel *users*.  
20 – `update_profile($user_id)`  
21 Memperbaharui nama, email, password, atau role sebuah *user*.  
22 – `send_password_reset_mail($email)`  
23 Mengirimkan *link reset password* ke email *user* yang dapat dipakai selama 1 jam.  
24 – `passchange_is_valid($passchange_key)`  
25 Mengembalikan sebuah *boolean* yang menyatakan bahwa *link reset password* masih dapat  
26 dipakai.  
27 – `reset_password($passchange_key, $newpassword)`  
28 Memperbaharui *password* dengan divalidasinya *password change key*.  
29 – `get_all_users()`  
30 Mengembalikan seluruh *user* pada tabel *users*.  
31 – `get_user($user_id)`  
32 Mengembalikan sebuah *user* yang memiliki id `$user_id`.  
33 – `update_login_time($username)`  
34 Memperbaharui catatan *login* untuk sebuah *user*.

35 **View**

36 *View* merupakan tampilan yang menjadi perantara antara pengguna dan *sistem*. Pada SharIF Judge,  
37 *View* disimpan pada direktori `application/views` dan dibagi menjadi 3 direktori terpisah yaitu  
38 `errors`, `pages`, dan `template`. Gambar 3.3 merupakan struktur direktori *view* beserta *view* yang  
39 terdapat pada direktorinya dalam SharIF Judge.



Gambar 3.3: Struktur Direktori View pada SharIF Judge

Berikut merupakan penjelasan mengenai direktori penyimpanan untuk *view* pada SharIF Judge.

- **errors**

Pada direktori *errors*, berisi tampilan halaman *error* jika terjadi error pada penggunaan SharIF Judge. Berikut merupakan *views* yang terdapat pada direktori **errors**:

- **error\_404**
- **error\_db**
- **error\_expection**
- **error\_general**
- **error\_php**

- **pages**

Pada direktori *pages*, berisi tampilan halaman-halaman utama. *pages* juga memiliki dua direktori selain halaman-halama. Berikut merupakan *views* dan direktori yang terdapat pada direktori *pages*:

1       – `pages/admin`

2       Direktori *admin* berisi tampilan halaman khusus untuk *role admin*. Berikut merupakan  
3       *views* yang terdapat pada direktori *admin*:

4           \* `add_assignment.twig`  
5           \* `add_notification.twig`  
6           \* `add_user.twig`  
7           \* `add_user_result.twig`  
8           \* `delete_assignment.twig`  
9           \* `edit_problem_html.twig`  
10          \* `edit_problem_md.twig`  
11          \* `edit_problem_plain.twig`  
12          \* `install.twig`  
13          \* `logs.twig`  
14          \* `moss.twig`  
15          \* `queue.twig`  
16          \* `rejudge.twig`  
17          \* `settings.twig`  
18          \* `users.twig`

19       – `pages/authentication`

20       Direktori *authentication* berisi tampilan halaman khusus untuk *authentication* seperti  
21       halaman direktori *Login*. Berikut merupakan *views* yang terdapat pada direktori *admin*:

22           \* `login.twig`  
23          \* `lost.twig`  
24          \* `register.twig`  
25          \* `register_success.twig`  
26          \* `reset_password.twig`  
27          – `assignments.twig`  
28          – `dashboard.twig`  
29          – `halloffame.twig`  
30          – `notification.twig`  
31          – `problems.twig`  
32          – `profile.twig`  
33          – `scoreboard.twig`  
34          – `scoreboard_tabel.twig`  
35          – `submissions.twig`  
36          – `submit.twig`

37       • `templates`

38       Pada direktori *templates*, berisikan tampilan yang digunakan berulang oleh halaman utama  
39       seperti *header*, *side bar*, dan *base*. Berikut merupakan *views* yang terdapat pada direktori  
40       *templates*:

41           – `base.twig`  
42           – `side_bar.twig`

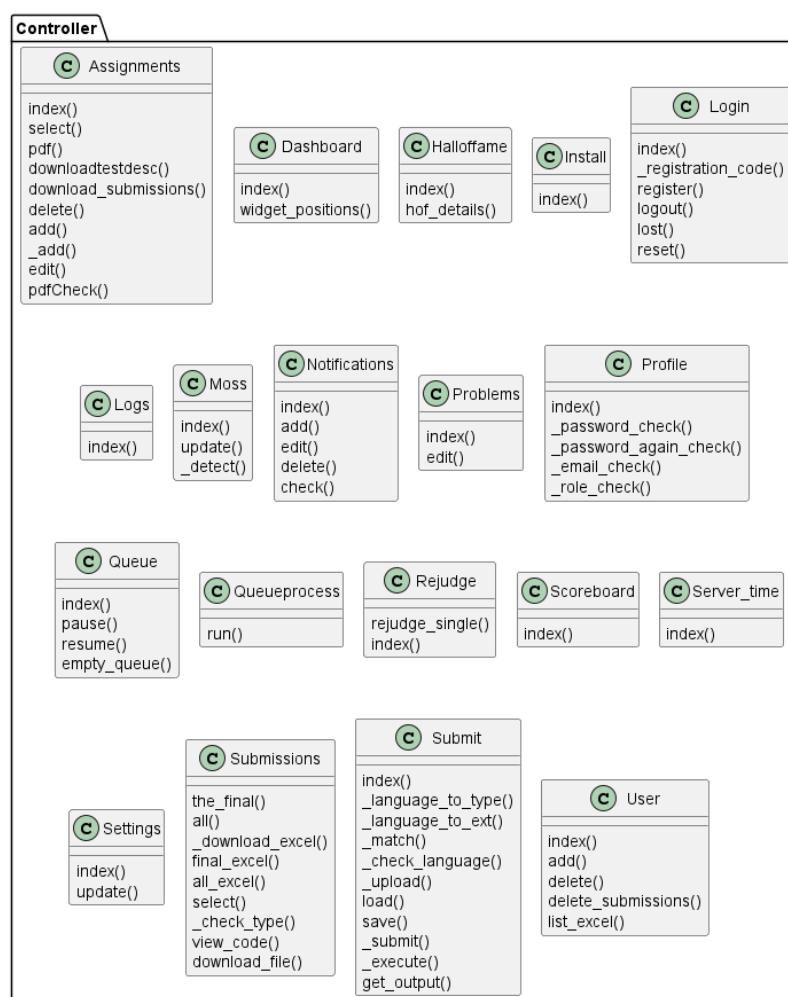
```

1     - simple_header.twig
2     - top_bar.twig

```

### 3 Controller

4 Pada bagian analisis MVC terakhir, terdapat *controller* yang berada pada direktori  
5 `application/controller`. Seperti yang dijelaskan pada subbab 2.2.1, *Controller* digunakan sebagai  
6 perantara antara *model*, *view*, dan *resources* lainnya yang dibutuhkan saat membuat sebuah web  
7 page. Direktori controller berisi kelas-kelas yang akan mengolah data yang didapat pada *model*  
8 dan menyatukan data tersebut ke dalam *views* yang akan ditampilkan kepada pengguna. Pada  
9 setiap kelas *controller*, terdapat fungsi `index()` yang menjadi fungsi utama saat kelas di akses oleh  
10 pengguna. Gambar 3.4 merupakan struktur kelas *controller* yang terdapat pada SharIF Judge.



Gambar 3.4: Struktur Kelas Controller pada SharIF Judge

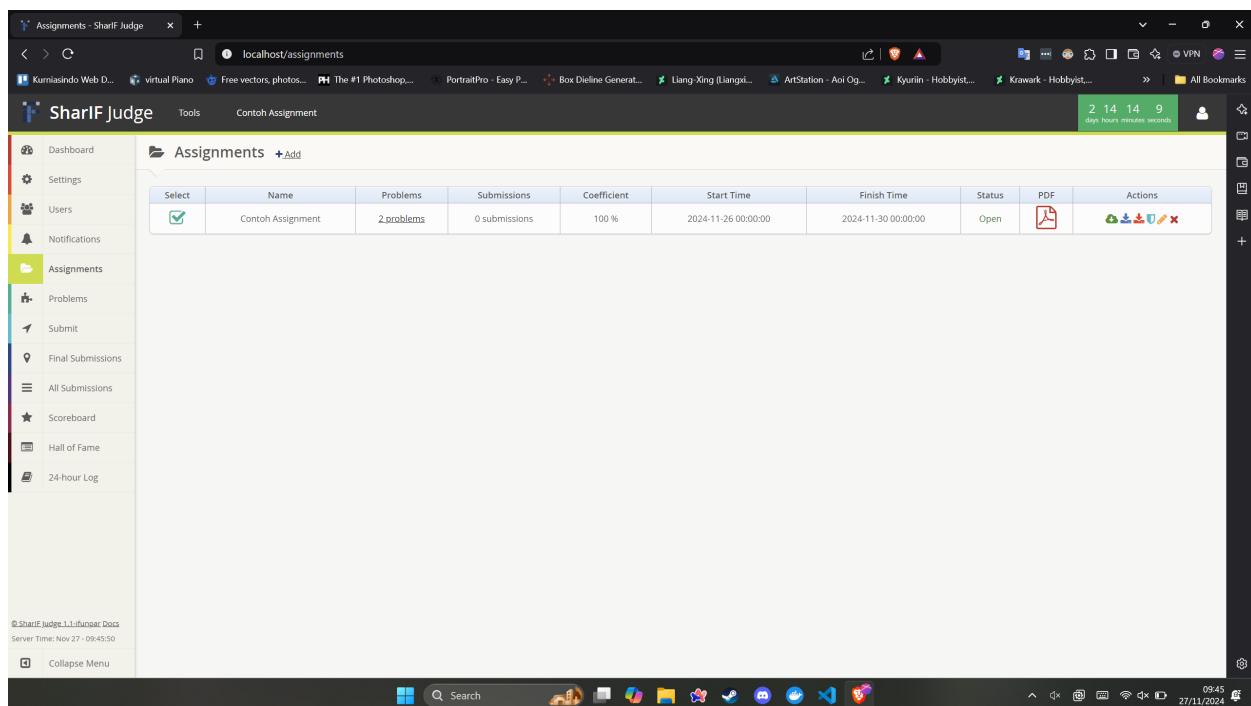
11 Berikut merupakan file *controller* dan penjelasan fungsinya yang terdapat pada SharIF  
12 Judge:

- `Assignments.php`

14 Berikut fungsi dengan penjelasannya pada controller `Assignments.php`:

- `select()`

1            Memilih *assignment* yang ditampilkan pada *top bar* menggunakan *ajax request*.  
 2    – `pdf($assignment_id, $problem_id, $no_download)`  
       Mengunduh *assignment* atau *problem* dalam bentuk *pdf file* ke browser.  
 3    – `downloadtestsdesc($assignment_id)`  
       Mengunduh dan mencompress data uji dan deskripsi sebuah *assignment*.  
 4    – `download_submissions($type, $assignment_id)`  
       Mengunduh semua *final submission* pada semua *assignment*.  
 5    – `delete($assignment_id)`  
       Menghapus sebuah *assignment*.  
 6    – `add()`  
       Mendapatkan *input* dari pengguna untuk menambah atau memperbarui sebuah *assignment*.  
 7    – `_add()`  
       Menambahkan atau memperbarui sebuah *assignment*.  
 8    – `edit($assignment_id)`  
       Menandai *assignment* yang akan di *edit* dan memanggil fungsi *add*.  
 9    – `pdfCheck($assignment_id, $problem_id)`  
       Melakukan validasi ketersediaan pdf pada sebuah *assignment* atau pada sebuah *problem*.  
 10   – `index()`  
       Mengambil data dari `Assignment_model` dan menaruh data dan mengembalikan `views assignments.twig`. Gambar 3.5 menunjukkan hasil halaman Assignment.

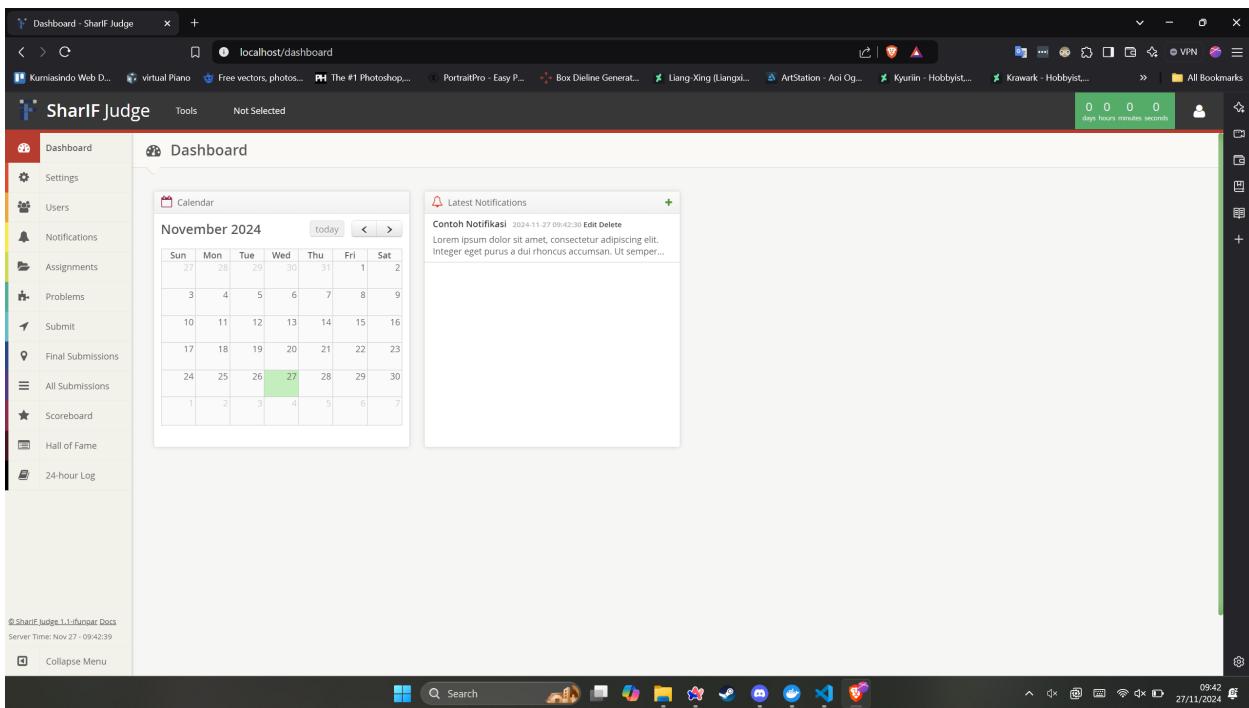


Gambar 3.5: Halaman Assignments

22    • `Dashboard.php`

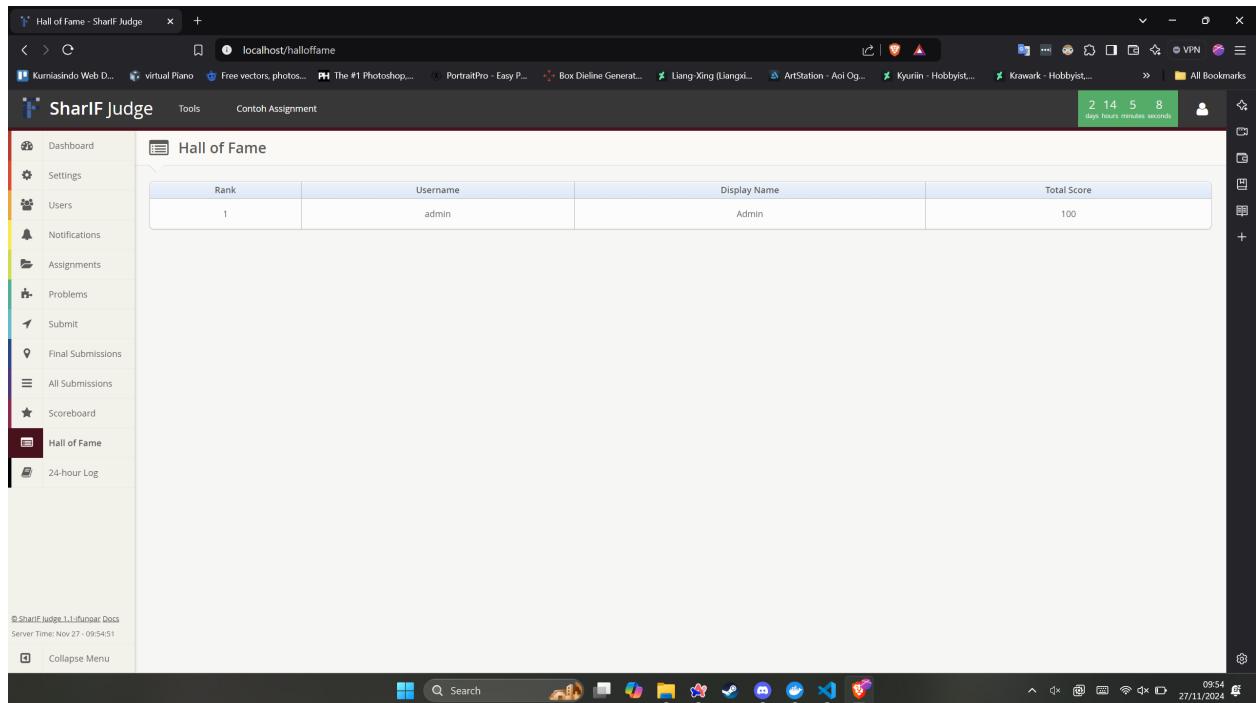
23    Berikut fungsi dengan penjelasannya pada *controller Dashboard.php*:

- 1   – `widget_positions()`
- 2   Menggunakan *ajax request* untuk menyimpan posisi *widget*.
- 3   – `index()`
- 4   Mendapatkan data dari beberapa model yaitu `Assignment_model`, `Settings_model`,
- 5   User, dan `Notifications_model`. Data akan dimasukkan ke dalam `dashboard.twig`
- 6   yang akan dikembalikan ke pengguna. Gambar 3.6 menunjukkan hasil halaman Dashboard
- 7   yang dapat diakses oleh semua *role*.



Gambar 3.6: Halaman Dashboard

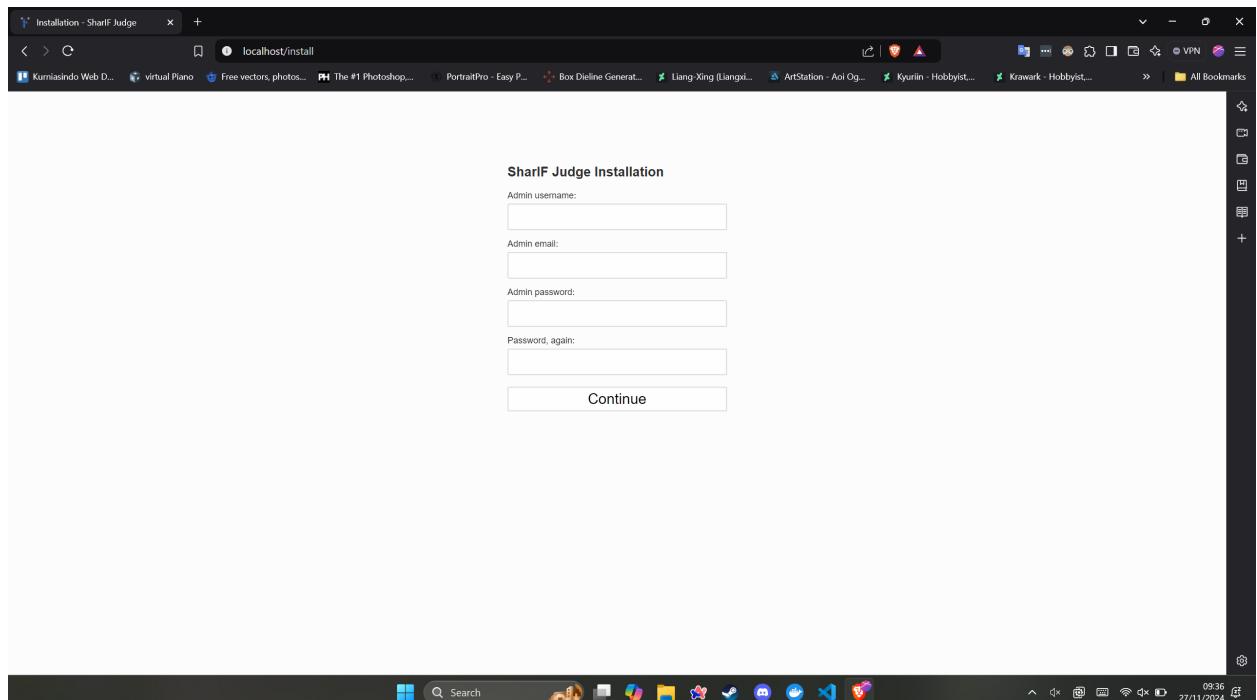
- 8   • `Halloffame.php`
- 9   Berikut fungsi dengan penjelasannya pada controller `Halloffame.php`:
- 10   – `hof_details()`
- 11   Menampilkan nilai akhir semua *problem* dan *assignments* pada sebuah *user*.
- 12   – `index()`
- 13   Mendapatkan data dari `Hof_model` dan mengembalikan *view halloffame.twig*. Gambar
- 14   3.7 menunjukkan hasil halaman Hall of Fame yang dapat diakses oleh semua *role*.



Gambar 3.7: Halaman Hall of Fame

1     • **Install.php**

2         Pada *controller* *Install.php* hanya ada satu fungsi yang menangani pembuatan seluruh  
3         tabel pada *database* yang dibutuhkan oleh SharIF Judge. Setelah membuat *database* akan  
4         mengembalikan *view install.twig* yang dapat diisi oleh pengguna tentang data *user* dengan  
5         role *admin* saat *form* di kirim. Gambar 3.8 menunjukkan hasil halaman Install.



Gambar 3.8: Halaman Install

1     • `Login.php`

2     Berikut fungsi dengan penjelasannya pada *controller Login.php*:

3       – `_registration_code($code)`

4       Melakukan validasi kode registrasi.

5       – `register()`

6       Menunjukkan halaman `register.twig` dan membuat *user* baru.

7       – `logout()`

8       Melakukan *Log out* dan mengalihkan ke halaman `login`.

9       – `lost()`

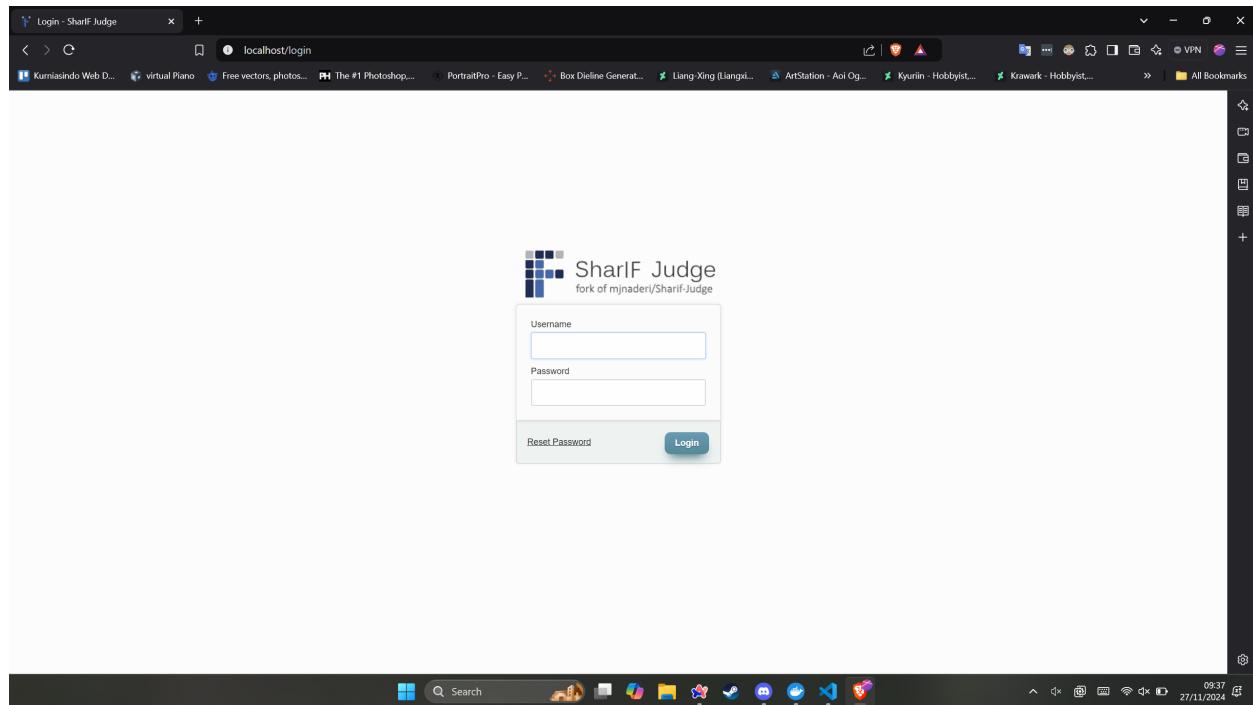
10      Mengirimkan email *reset password*.

11      – `reset($passchange_key)`

12      Melakukan *reset password* dengan halaman `reset_password.twig`.

13      – `index()`

14      Mengembalikan *view login.twig* dan memeriksa username dan password pada *form* saat di kirim. Gambar 3.9 menunjukkan hasil halaman Login.



Gambar 3.9: Halaman Login

16     • `Logs.php`

17     Pada *controller Logs.php* hanya memiliki satu fungsi yaitu `index()`, dimana fungsi tersebut akan mendapatkan data dari `Logs_model` dan memunculkan halaman `logs.twig`. Gambar 3.10 menunjukkan halaman Log yang dinamakan halaman 24-Hour Log.

#	Login ID	Username	IP Address	Login Time	Log from different IP (< 24 hours)
1	2	admin	172.20.0.1	2024-11-27 02:48:17	
2	1	admin	172.20.0.1	2024-11-27 02:38:45	

Gambar 3.10: Halaman 24-Hour Log

1     • Moss.php

2     Berikut fungsi dengan penjelasannya pada *controller Moss.php*:

3        – update(\$assignment\_id)

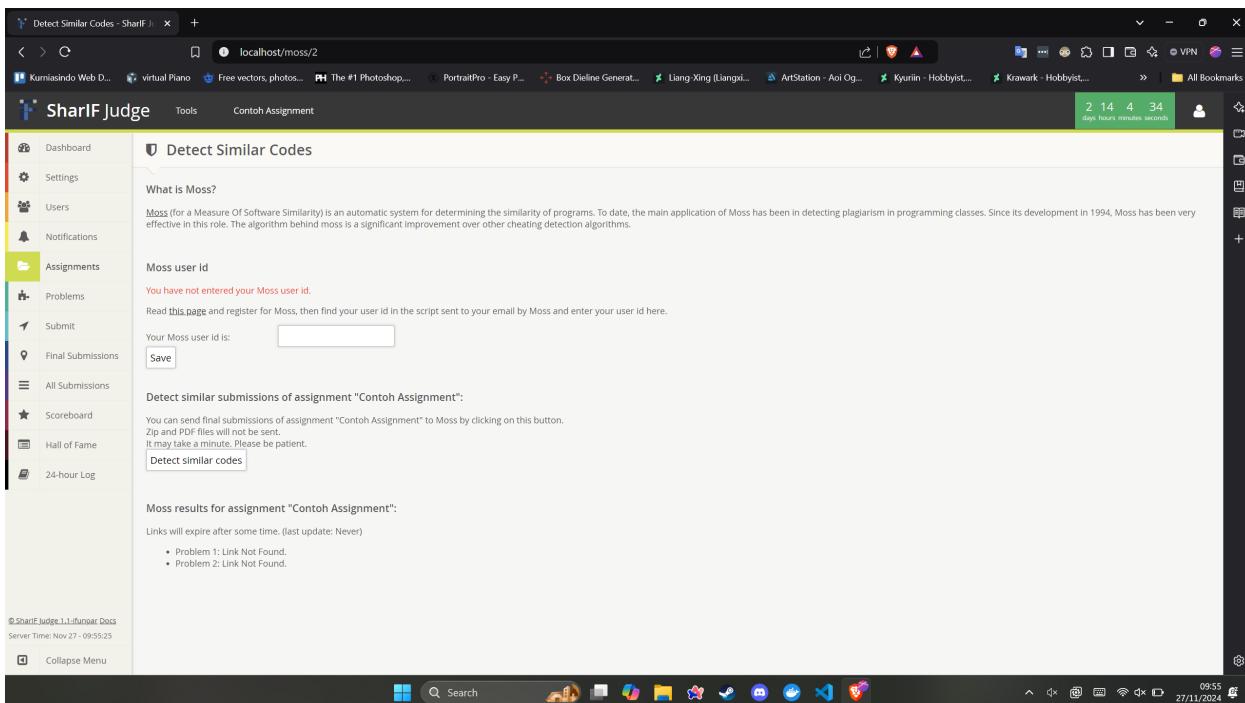
4           Memperbaharui *settings* dari masukkan moss\_userid pengguna.

5        – \_detect(\$assignment\_id)

6           Melakukan pemeriksaan kesamaan kode dengan Moss.

7        – index()

8           Mengambil data dan memasukkannya ke dalam *view moss.twig*. Gambar 3.11 merupakan hasil halaman moss. Fungsi *\_detect* juga akan dijalankan saat *form* terkirim.



Gambar 3.11: Halaman Moss

1     • **Notifications.php**

2     Berikut fungsi dengan penjelasannya pada *controller Notifications.php*:

3       – **add()**

4           Menambahkan atau memperbarui sebuah *notification*.

5       – **edit(\$notif\_id)**

6           Menandai *notification* yang akan di *edit* dan memanggil fungsi *add*.

7       – **delete()**

8           Menghapus sebuah *notification*.

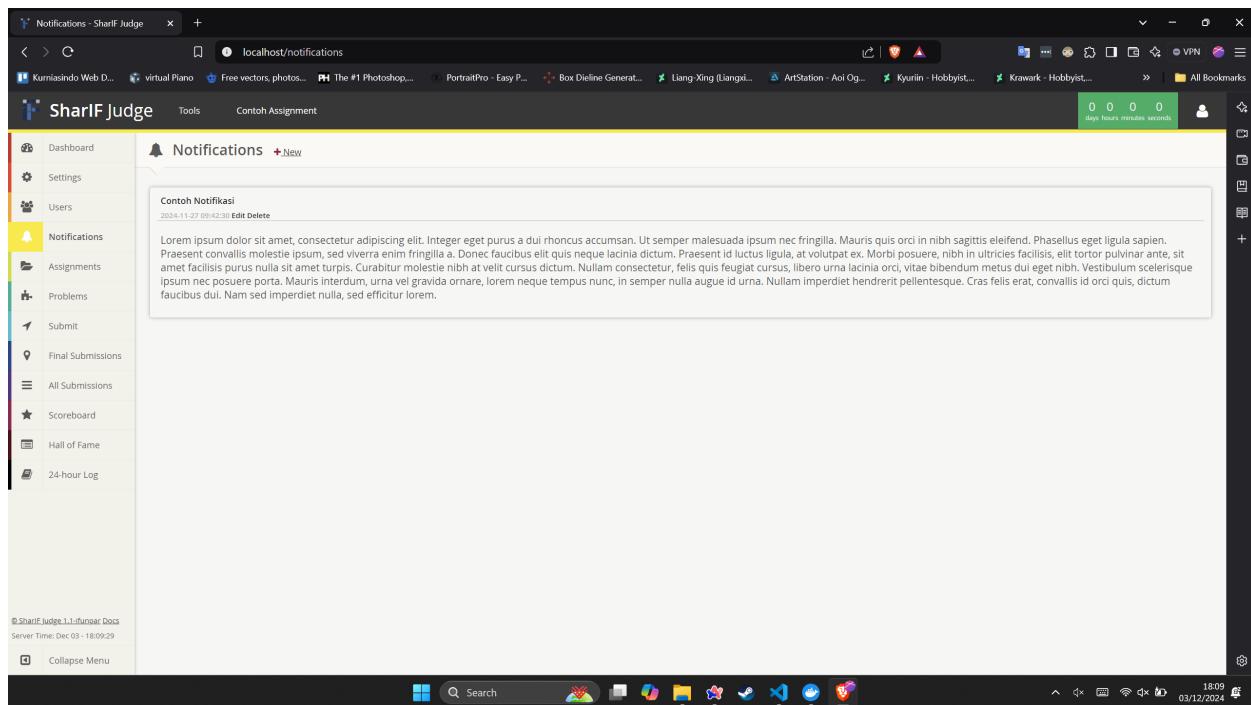
9       – **check()**

10           Menggunakan *ajax request* untuk mengetahui ketersediaan *notification* baru.

11       – **index()**

12           Mendapatkan data dari dua model yaitu **Assignment\_model** dan **Notifications\_model**.

13           Data akan dimasukkan ke dalam *view notifications.twig* yang akan dikembalikan ke pengguna. Gambar 3.12 menunjukkan hasil halaman *Notifications*.



Gambar 3.12: Halaman Notifications

1     • **Problems.php**

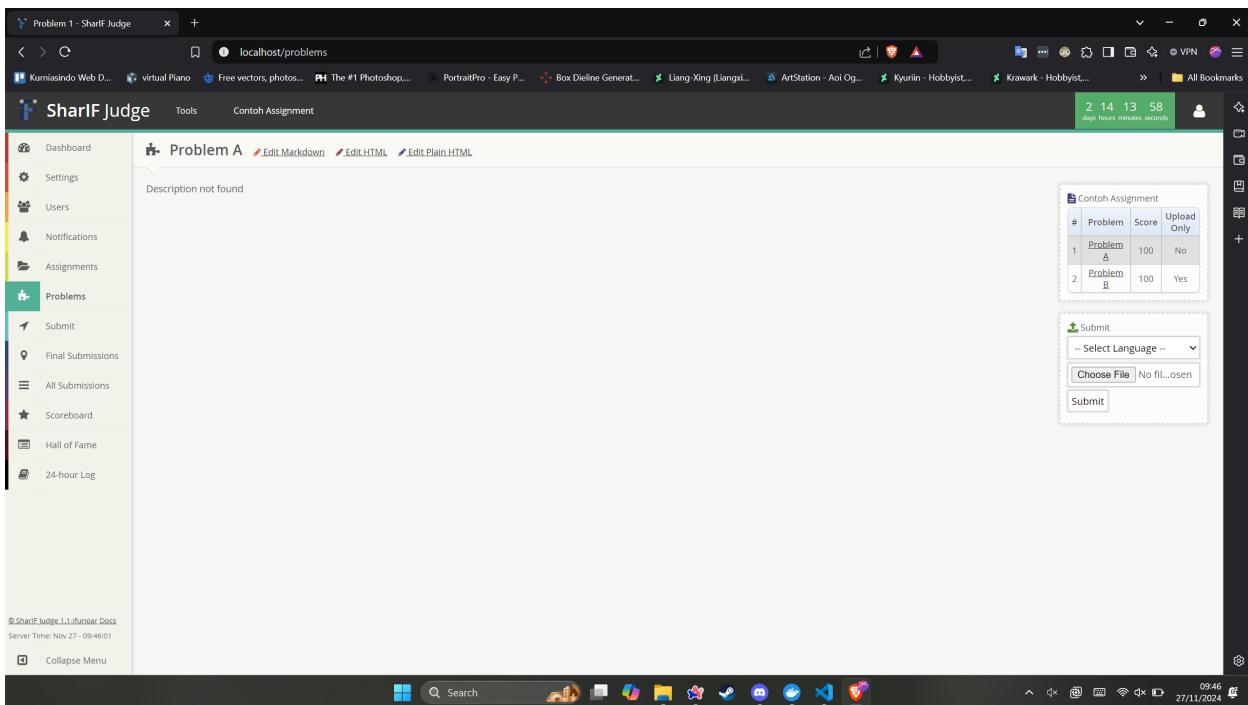
2       Berikut fungsi dengan penjelasannya pada *controller Notifications.php*:

3       – **edit()**

4           Memperbaharui deskripsi sebuah *problem* dalam bentuk **html** atau **markdown**.

5       – **index()**

6           Mendapatkan data *problem* dari berbagai *model* sesuai dengan *assignment* yang dipilih  
7           dan menaruh data tersebut pada halaman **problems.twig** yang akan ditampilkan ke  
8           pengguna. Gambar 3.13 menunjukkan hasil halaman Problems.



Gambar 3.13: Halaman Problems

1     • **Profile.php**

2     Berikut fungsi dengan penjelasannya pada *controller Profile.php*:

3       – *\_password\_check(\$str)*

4           Melakukan validasi *input password*.

5       – *\_password\_again\_check(\$str)*

6           Melakukan validasi *input tulisan pengulangan password*.

7       – *\_email\_check(\$str)*

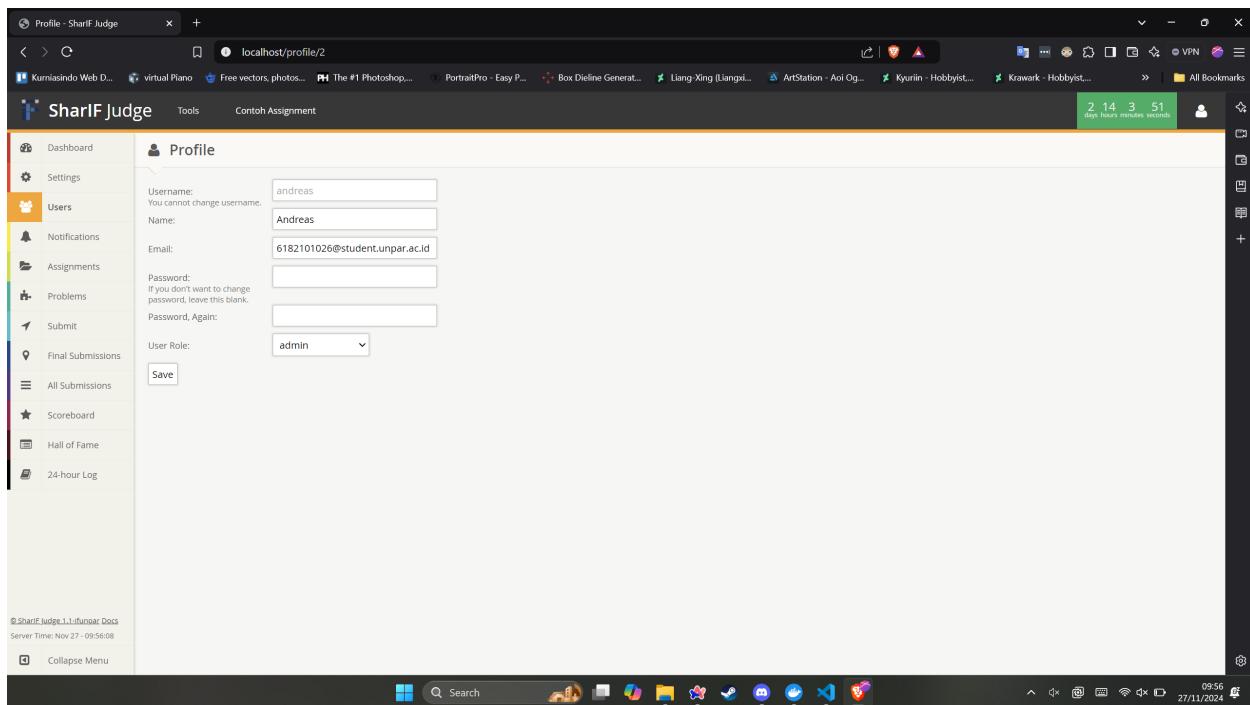
8           Melakukan validasi ketersediaan email pada *database*.

9       – *\_role\_check(\$str)*

10           Melakukan validasi *role* pengguna saat ingin mengubah *role user*.

11       – *index()*

12           Mendapatkan data dari berbagai *model* terutama dari *User* yang akan dimasukkan ke dalam *view profile.twig*. Fungsi ini juga menangani pengiriman *form* pembaharuan data *user* pengguna. Gambar 3.14 menunjukkan hasil halaman Profile.



Gambar 3.14: Halaman Profile

1     • **Queue.php**

2       Berikut fungsi dengan penjelasannya pada *controller Profile.php*:

3       – **pause()**

4           Memberhentikan proses *queue*.

5       – **resume()**

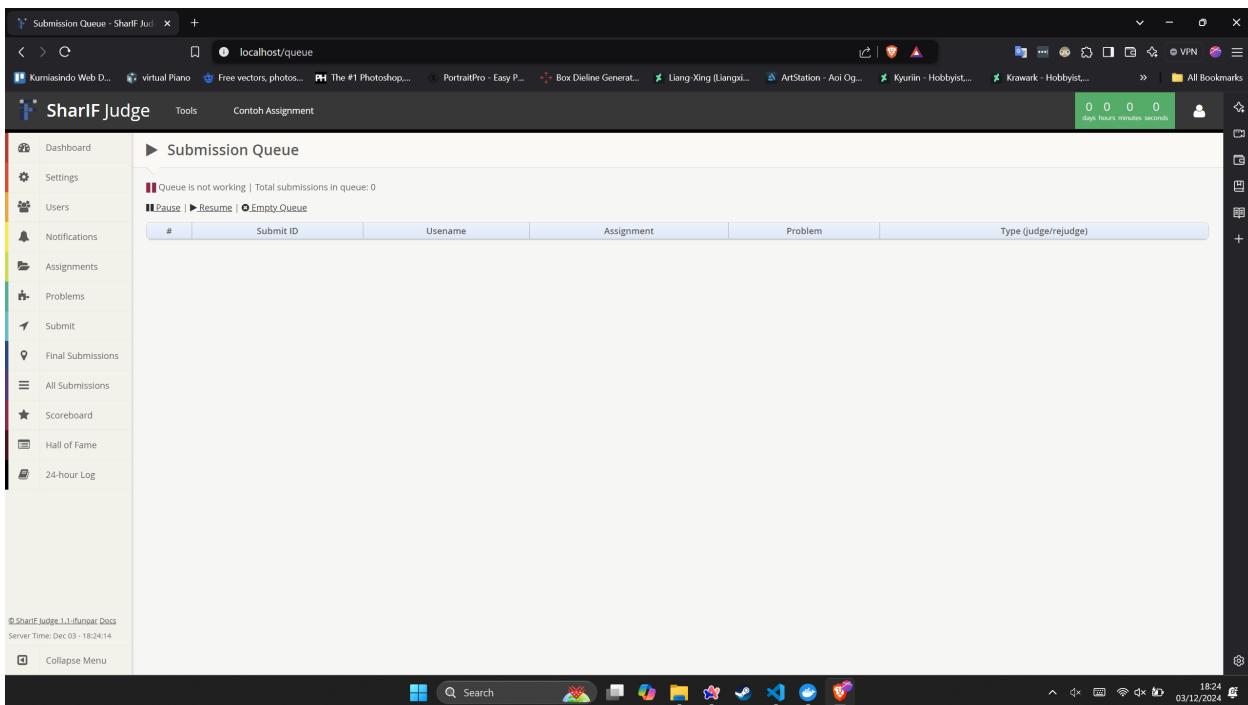
6           Melanjutkan proses *queue*.

7       – **empty\_queue()**

8           Menghapus semua *queue* yang ada.

9       – **index()**

10           Mendapatkan data dari *model Queue*, *Assignments\_model*, dan *Settings\_model* yang dipakai dalam *view queue.twig* dan ditampilkan kepada pengguna. Gambar 3.15 menunjukkan hasil halaman Queue.



Gambar 3.15: Halaman Queue

1     • **Queueprocess.php**

2       Controller *Queueprocess.php* hanya memiliki satu fungsi yaitu *run()* yang akan menjalankan  
3       *queue* satu per satu menggunakan *bash*.

4     • **Rejudge.php**

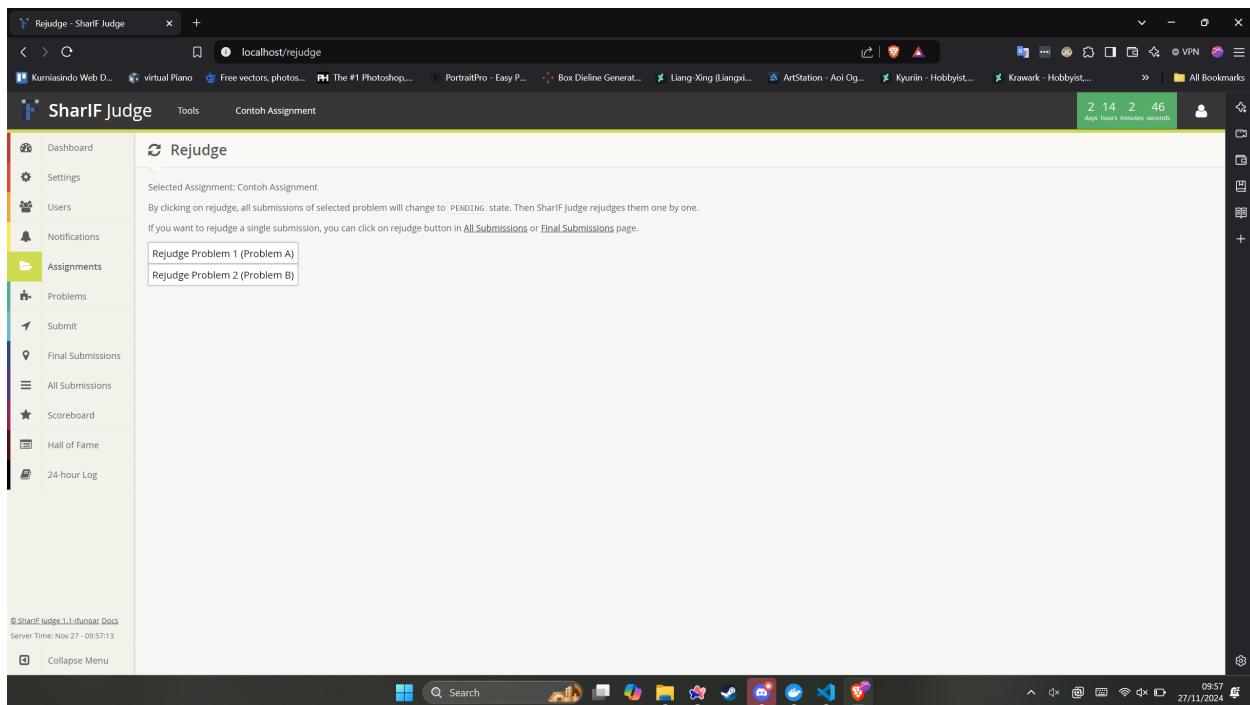
5       Berikut fungsi dengan penjelasannya pada *controller Profile.php*:

6       – *rejudge\_single()*

7           Melakukan *rejudge* untuk satu buah *submission*.

8       – *index()*

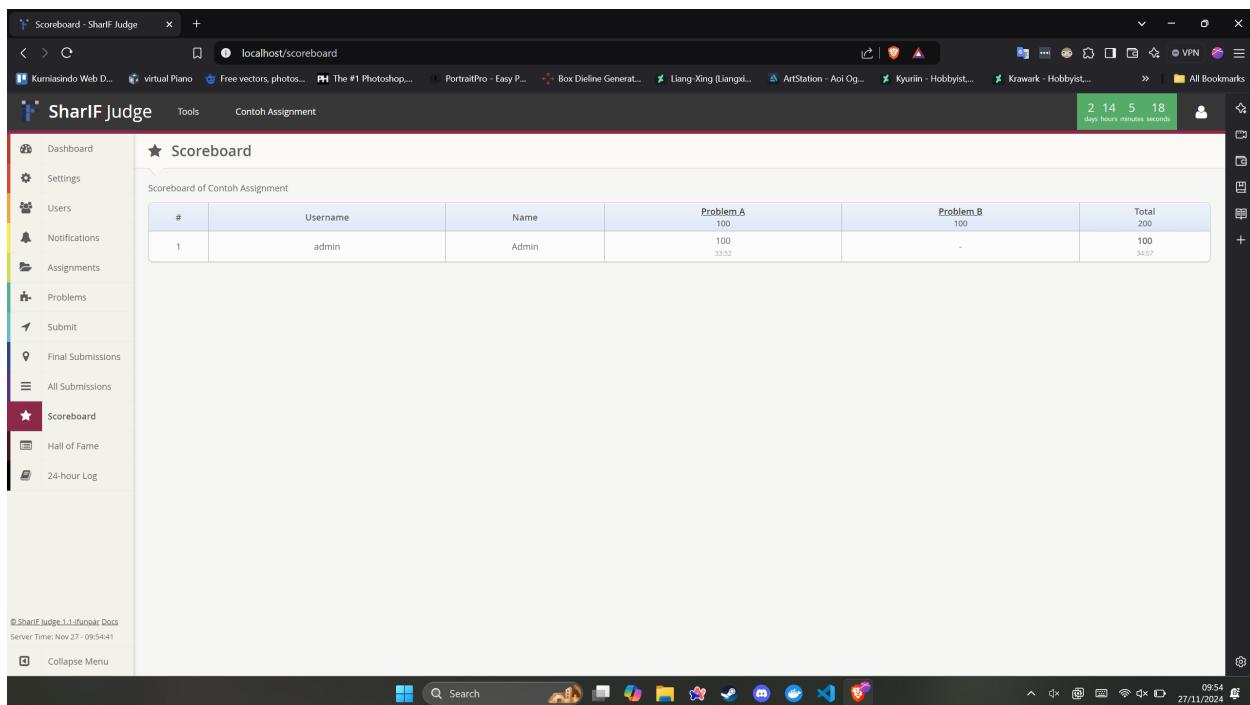
9           Mendapatkan data dan menampilkan *view rejudge.twig*. Fungsi ini juga dapat me-  
10          lakukan *rejudge* pada sebuah *problem* tertentu. Gambar 3.16 menunjukkan halaman  
11          Rejudge.



Gambar 3.16: Halaman Rejudge

1     • **Scoreboard.php**

2     *Controller Queueprocess.php* hanya memiliki satu fungsi yaitu `index()` yang akan menampilkan `view scoreboard.twig` dengan data dari `Scoreboard_model`. Gambar 3.17 menunjukkan hasil halaman Scoreboard.



Gambar 3.17: Halaman Scoreboard

5     • **Server\_time.php**

Controller Queueprocess.php hanya memiliki satu fungsi yaitu index() yang akan mencetak waktu pada server, waktu akan digunakan untuk sinkronisasi waktu.

- **Settings.php**

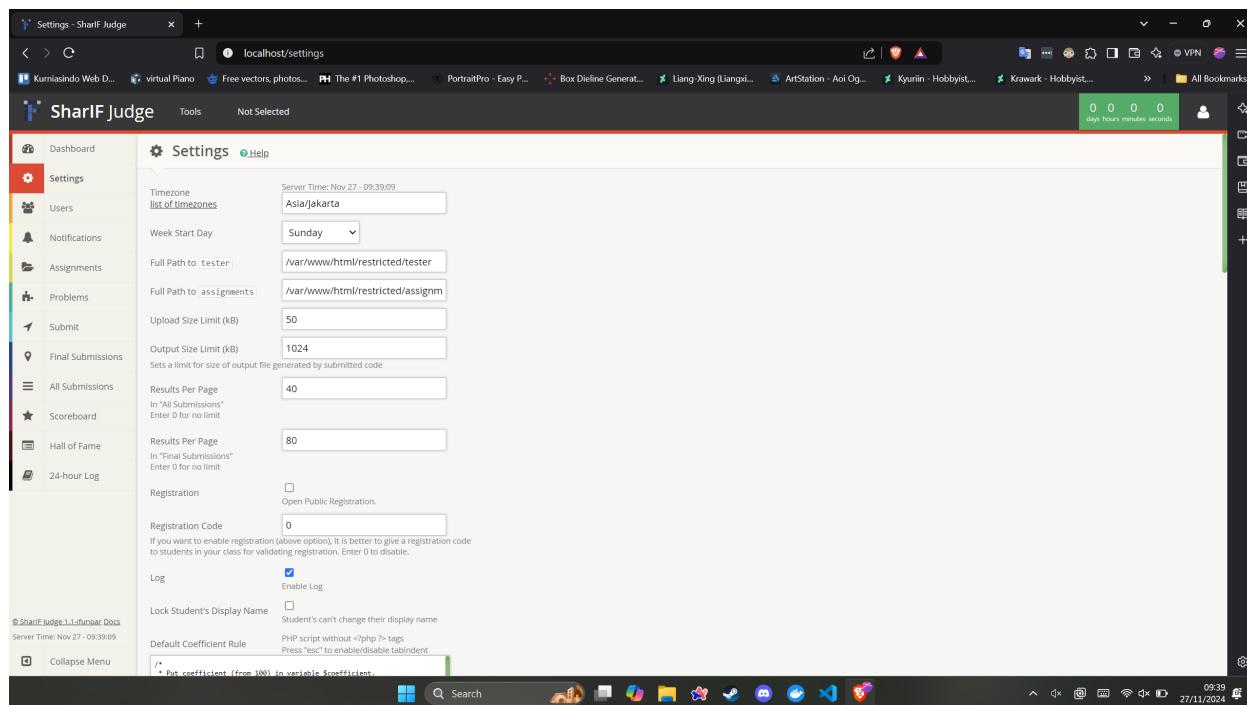
Berikut fungsi dengan penjelasannya pada controller Settings.php:

- update()

Memperbarui settings dari masukkan pengguna.

- index()

Mendapatkan data dari Settings\_model dan menampilkan view settings.twig. Jika terdapat error setting pada sistem, akan ditampilkan juga pada view tersebut. Gambar 3.18 menunjukkan hasil halaman Users.



Gambar 3.18: Halaman Settings

- **Submissions.php**

Berikut fungsi dengan penjelasannya pada controller Submissions.php:

- \_download\_excel(\$view)

Menggunakan library PHPExcel untuk membuat sebuah file excel dari submissions yang akan diunduh pengguna.

- final\_excel()

Menggunakan fungsi \_download\_excel untuk mendownload final submission.

- all\_excel()

Menggunakan fungsi \_download\_excel untuk mendownload seluruh submission.

- select()

Menggunakan ajax request untuk memilih submission yang akan dikumpulkan atau menjadi final.

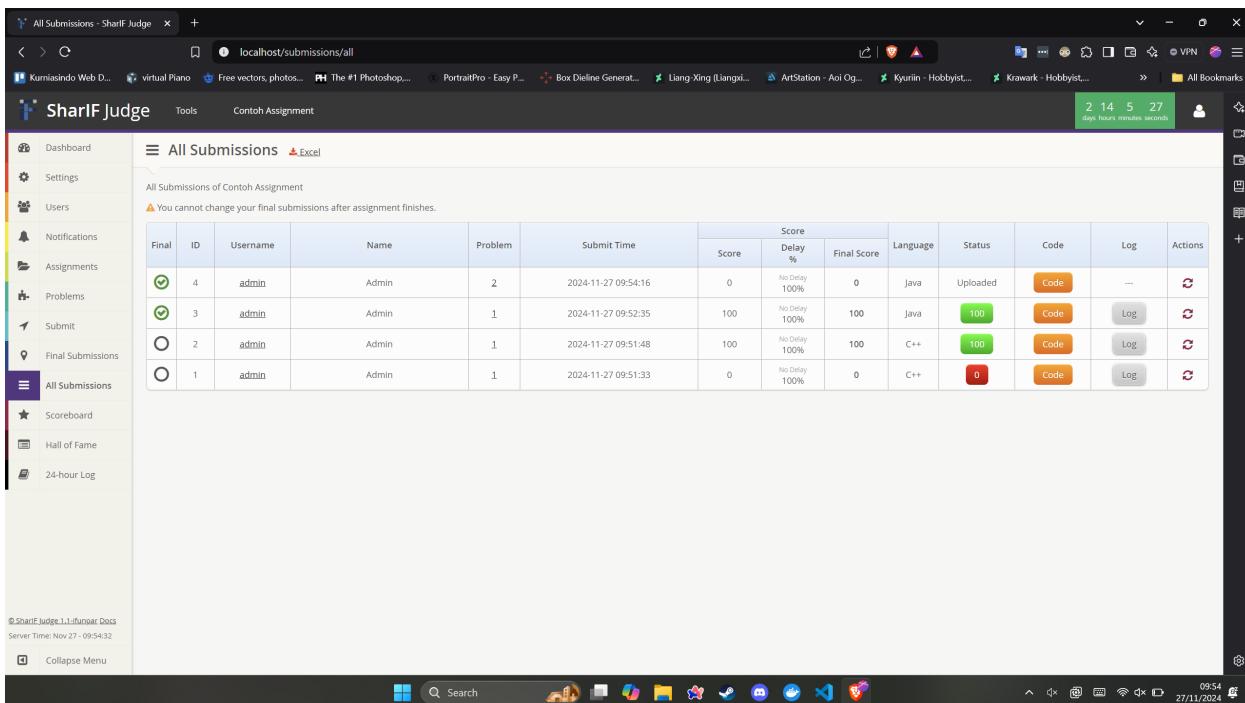
- \_check\_type(\$type)

- 1        Melakukan validasi tipe *submission* yang dikumpulkan.
- 2     – **`view_code()`**  
   Digunakan untuk melihat kode, melihat hasil kode, atau melihat *log* sebuah *submission*.
- 3     – **`download_file()`**  
   Mengunduh *file* kode sebuah *submission*.
- 4     – **`the_final()`**  
   Mendapatkan data dari `Submit_model` untuk mendapatkan *final submission* dan menampilkan halaman `submission.twig` berisi *final submission*. Gambar 3.19 menunjukkan halaman Final Submissions .

#	ID	Username	Name	Problem	Submit Time	Score					Status	Code	Log	Actions
						Score	Delay %	Final Score	Language					
1	3	admin	Admin	1	2024-11-27 09:52:35	100	No Delay 100%	100	java	<span>100</span>	<span>Code</span>	<span>Log</span>	<span>Edit</span>	
2	4	admin	Admin	2	2024-11-27 09:54:16	0	No Delay 100%	0	java	Uploaded	<span>Code</span>	...	<span>Edit</span>	

Gambar 3.19: Halaman Final Submissions

- 10    – **`all()`**  
 11    Mendapatkan data dari `Submit_model` untuk mendapatkan seluruh *submission* dan  
 12    menampilkan halaman `submission.twig` berisi semua *submission*. Gambar 3.20 menunjukkan halaman All Submissions .



Gambar 3.20: Halaman All Submissions

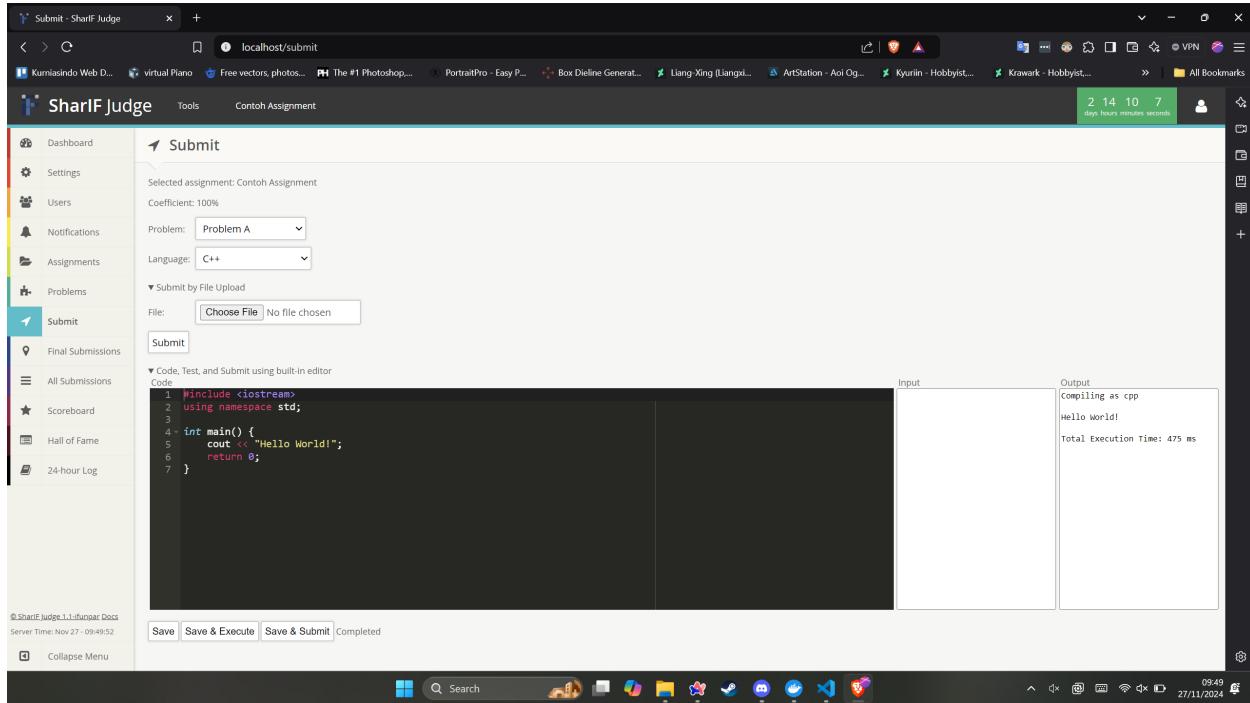
1     • **Submit.php**

2     Berikut fungsi dengan penjelasannya pada *controller* *Submit.php*:

- 3       – *\_language\_to\_type(\$language)*  
4        Mengembalikan kode singkat dari *\$language* dipilih.
- 5       – *\_language\_to\_ext(\$language)*  
6       Mengembalikan extensi file dari *\$language* yang dipilih.
- 7       – *\_match(\$type, \$extension)*  
8       Melakukan validasi untuk *\$type* dan *\$extension* agar sesuai.
- 9       – *\_check\_language(\$str)*  
10      Melakukan validasi sudah dipilihannya bahasa.
- 11      – *\_upload()*  
12      Menyimpan jawaban pengguna yang dikirim dan menambahkannya ke dalam *queue*  
13      untuk dinilai jika bukan *upload only problem*.
- 14      – *load(\$problem\_id)*  
15      Mendapatkan isi file dan menaruh isi file ke editor kode.
- 16      – *save(\$type)*  
17      Menyimpan isi editor kode ke dalam *server* dan menjalankan atau mengumpulkan jika  
18      diinginkan.
- 19      – *\_submit(\$data, \$problem\_id, \$language, \$user\_dir)*  
20      Menambahkan kode ke dalam *submission* untuk dinilai.
- 21      – *\_execute(\$data, \$problem\_id, \$language, \$user\_dir)*  
22      Menambahkan kode ke dalam *queue* untuk di jalankan saja.
- 23      – *get\_output(\$problem\_id)*  
24      Mendapatkan keluaran dari kode yang telah dijalankan sebagai hasil eksekusi.

1     – **index()**

2       Mendapatkan data dari *model Assignments\_model* untuk mendapatkan *problem* dan  
 3       data lainnya. Semua data akan dimasukkan dalam *view submit.twig*. Gambar 3.21  
 4       menunjukkan hasil halaman Submit. Halaman ini terdapat editor kode yang sudah di  
 5       implementasikan [7].



Gambar 3.21: Halaman Submit

6     • **User.php**

7       Berikut fungsi dengan penjelasannya pada *controller User.php*:

8       – **add()**

9           Menambahkan *user* baru ke dalam *sistem*.

10      – **delete()**

11           Menghapus sebuah *user*.

12      – **delete\_submissions()**

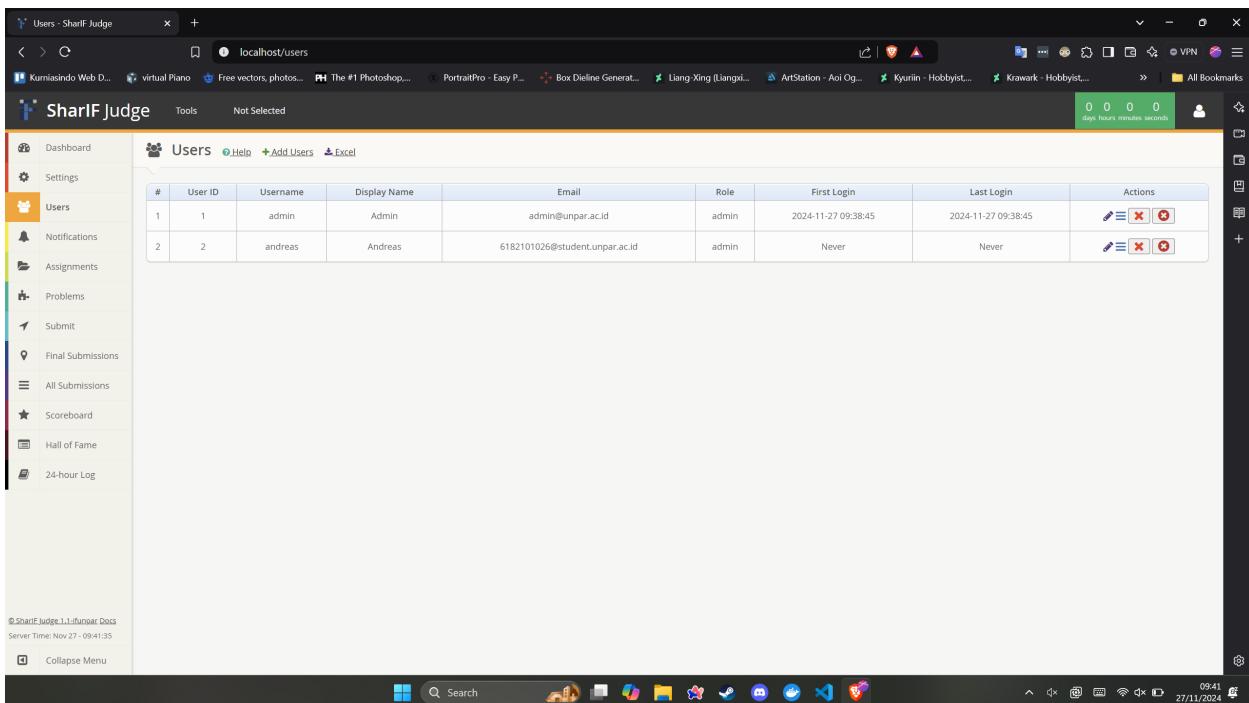
13           Menghapus seluruh *submissions* dari sebuah *user*.

14      – **list\_excel()**

15           Menggunakan *library PHPExcel* untuk membuat sebuah file excel dari seluruh daftar  
 16           *user* yang akan diunduh pengguna.

17      – **index()**

18       Mendapatkan data dari *User\_model* dan menunjukkan *view users.twig*. Gambar 3.22  
 19       menunjukkan hasil halaman Users. Pada halaman ini terdapat daftar seluruh *user*  
 20       yang terdaftar pada SharIF Judge. Pengguna dapat membuat, memperbarui, dan  
 21       menghapus *user*.



Gambar 3.22: Halaman Users

### 1 3.1.2 Assets

2 Pada SharIF Judge terdapat direktori bernama *Assets* yang menjadi tempat untuk menyimpan  
 3 seluruh kebutuhan dari sisi pengguna seperti gambar logo dan *library javascript*. Berikut merupakan  
 4 isi dari direktori *assets* beserta dengan kegunaannya dalam aplikasi SharIF Judge:

- 5 • Direktori **ace**

6 Direktori ini berisikan hasil *build library* Ace, menghasilkan file *javascript* yang berfungsi  
 7 untuk menambahkan editor kode pada aplikasi.

- 8 • Direktori **font**

9 Direktori ini berisikan font khusus dan juga memiliki *library javascript* bernama font-awesome  
 10 yang digunakan untuk menaruh icon berformat svg dalam tampilan SharIF Judge.

- 11 • Direktori **fullcalendar**

12 Direktori ini berisikan hasil *build library* FullCalendar, menghasilkan file *javascript* dan *css*  
 13 yang berfungsi untuk memasukkan kalender dalam tampilan SharIF Judge.

- 14 • Direktori **gridster**

15 Direktori ini berisikan hasil *build plugin* Gridster untuk *library* jQuery, menghasilkan file  
 16 *javascript* dan *css* yang berfungsi untuk memasukkan *layout grid* yang dapat di ubah dengan  
 17 menggunakan sifat mouse *drag and drop*.

- 18 • Direktori **images**

19 Direktori ini berisikan gambar kustom yang digunakan oleh SharIF Judge seperti logo dan  
 20 banner.

- 21 • Direktori **js**

22 Direktori ini berisikan hasil *build library* jQuery, taboverride, moment, *plugins* jQuery berfungsi  
 23 untuk membantu membangun *javascript* khusus yang dipakai dalam SharIF Judge. Direktori

1 ini juga memiliki file *javascript* khusus yang dipakai dalam halaman SharIF Judge yaitu  
2 sebagai berikut:

3 – **shj\_functions.js**

4 File *javascript shj\_functions* digunakan untuk seluruh sistem umum dalam SharIF  
5 Judge seperti waktu server, sidebar toogle, loading dan berbagai macam fungsi yang  
6 digunakan dalam SharIF Judge.

7 – **shj\_submissions.js**

8 File *javascript shj\_submissions* digunakan untuk menangani berbagai fitur untuk  
9 halaman all submissions dan final submissions seperti menampilkan kode program dan  
10 memeriksa ulang hasil kode dalam judge.

11 – **shj\_submit.js**

12 File *javascript shj\_submit* digunakan untuk menangani berbagai fitur untuk halaman  
13 submit terutama fungsi aksi pada IDE yaitu aksi *save*, *execute*, *submit*, dan memuat  
14 kode lama ke dalam editor kode.

15 • Direktori **nano\_scroller**

16 Direktori ini berisikan hasil *build plugin* nanoScrollerJS untuk *library* jQuery, menghasilkan  
17 file *javascript* dan *css* yang berfungsi untuk membangun *scrollbar* pada SharIF Judge.

18 • Direktori **noty**

19 Direktori ini berisikan hasil *build plugin* noty untuk *library* jQuery, menghasilkan file *javascript*  
20 dan *css* yang berfungsi untuk menampilkan pesan atau notifikasi kepada pengguna pada  
21 SharIF Judge.

22 • Direktori **pdfjs**

23 Direktori ini berisikan hasil *build library* pdfjs, menghasilkan file *javascript* yang berfungsi  
24 untuk menampilkan file pdf pada SharIF Judge.

25 • Direktori **reveal**

26 Direktori ini berisikan hasil *build plugin* Reveal yang sudah dimodifikasi untuk *library* jQuery,  
27 menghasilkan file *javascript* dan *css* yang berfungsi untuk menampilkan *modal* konfirmasi.

28 • Direktori **snippet**

29 Direktori ini berisikan hasil *build plugin* Snippet yang sudah dimodifikasi untuk *library* jQuery,  
30 berfungsi untuk sebagai *Syntax Highlighter* untuk teks kode dalam halaman Hall of Fame,  
31 Problems, dan Submissions.

32 • Direktori **styles**

33 Direktori ini berisikan css khusus yang digunakan untuk memperindah halaman dalam SharIF  
34 Judge.

35 • Direktori **tinymce**

36 Direktori ini berisikan hasil *build library* tinymce, berfungsi untuk memasukkan editor teks  
37 dalam SharIF Judge.

38 **3.1.3 Penyimpanan Kode Submission**

39 Pada SharIF Judge, Kode akan disimpan pada lokasi **Assignment** yang dapat di ubah pada halaman  
40 **Settings**. Berikut merupakan format penyimpanan sebuah kode:

41 `assignment_<a_id>/p<p_id>/<nama user>/<nama file>-<s_id>. <file ext>`

Penjelasan untuk format di atas adalah sebagai berikut:

- <a\_id>  
id pada *assignment*.
- <p\_id>  
id pada *problem*.
- <nama user>  
Nama dari pengguna yang mengumpulkan kode/file.
- <nama file>  
Nama file yang dikumpulkan, **editor** jika mengumpulkan menggunakan editor kode.
- <s\_id>  
id pada *submission*.
- <file ext>  
Extensi file kode yang dikumpulkan.

Sebagai contoh, pengguna bernama **kenzhi** mengumpulkan kode dengan nama file **probA.java** ke dalam *problem* pertama dari *assignment* dengan id 5. **kenzhi** sudah melakukan pengumpulan pada *problem* yang sama sebanyak 5 kali dan *submission* kali ini akan menjadi nomor 6, sehingga *submission id* adalah 6. Maka kode pengguna akan disimpan pada alamat:

```
assignment_5/p1/kenzhi/probA-6.java
```

### 3.1.4 Antrean Penilaian Kode

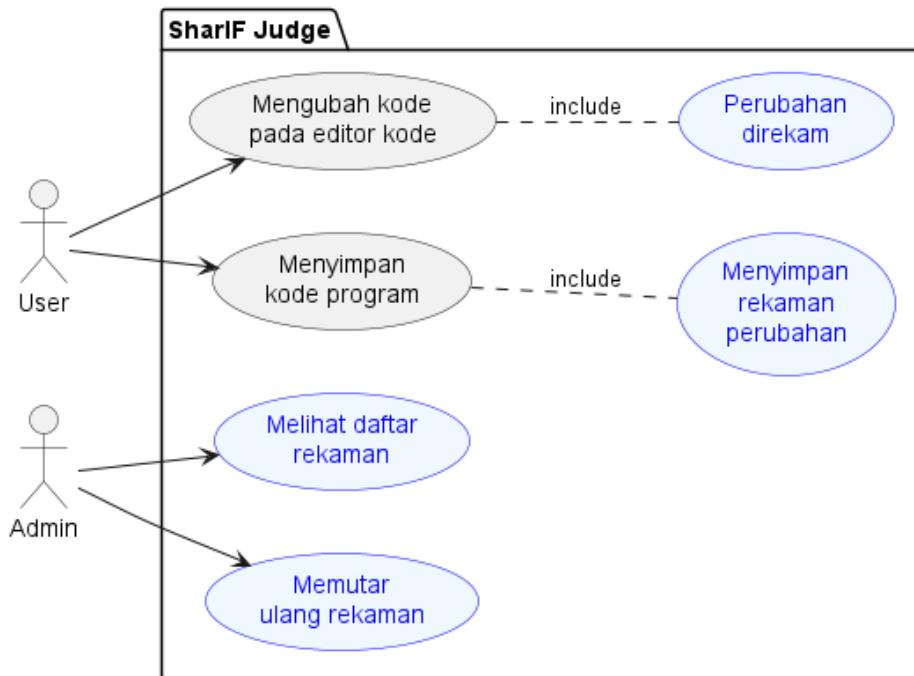
Pada SharIF Judge, Kode yang dikumpulkan akan di jalankan satu per satu pada antrean menggunakan **bash**. Berikut merupakan cara SharIF Judge menilai kode dari awal pengumpulan pada sistem:

1. *Controller Submit* akan menyimpan kode ke dalam file pada folder sesuai pada subbab 3.1.3.
2. *Controller Submit* akan memasukkan data *submission* ke dalam *model Queue\_model*.
3. *Model Queue\_model* akan menyimpan data *submission* pada *database submission* dan menambahkan data *queue*.
4. Selanjutnya *Controller Submit* akan memanggil fungsi *process\_the\_queue()* yang akan menjalankan fungsi *run()* pada *controller Queueprocess*.
5. *Controller Queueprocess* akan menjalankan **tester.sh** pada folder **tester** dengan data dari *queue*.
6. **tester.sh** akan menilai kode yang akan dibaca oleh *controller Queueprocess* yang akan menyimpan hasil penilaian.
7. Terakhir *Queueprocess* akan menyimpan hasil penilaian pada *database submission* dan menghapus data *queue* menggunakan *Queue\_model*.

## 3.2 Analisis Sistem Usulan

Pembuatan sistem pemutaran ulang ketikan membutuhkan 4 fitur baru yaitu fitur perekaman perubahan, fitur penyimpanan rekaman perubahan, fitur melihat daftar rekaman yang ada, dan fitur untuk memutar ulang rekaman. Gambar 3.23 menunjukkan bagaimana fitur baru akan berinteraksi

- 1 dengan sistem SharIF Judge dan *user*. Berikut merupakan penjelasan mengenai fitur-fitur yang  
2 akan ditambahkan pada SharIF Judge untuk membangun sistem perekaman ketikan.



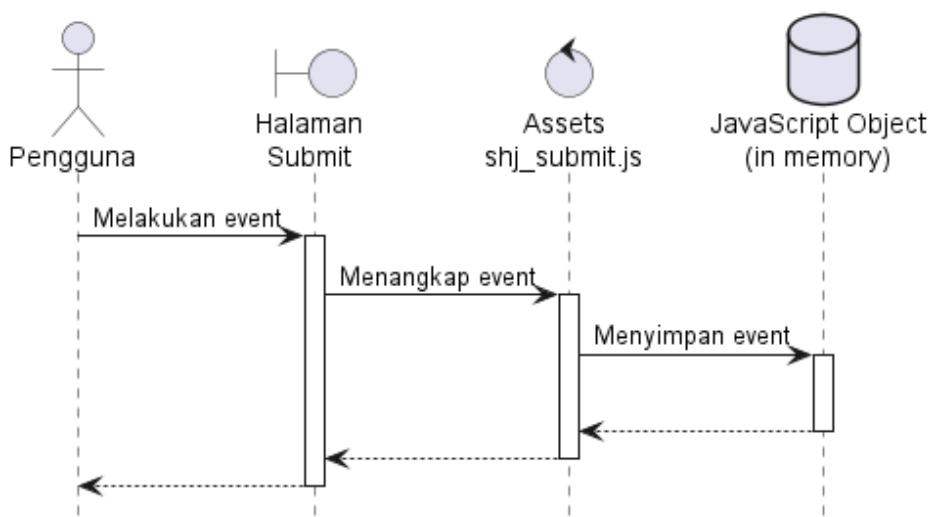
Gambar 3.23: Usecase analisis sistem usulan

### 3 3.2.1 Fitur perekaman perubahan atau event

- 4 Fitur perekaman perubahan pada editor kode bukan hanya perubahan text melainkan pada seluruh  
5 kejadian atau *event* yang terjadi pada editor kode seperti contohnya adalah perubahan posisi  
6 kursor maupun pilihan pada kode. Fitur perekaman perubahan akan otomatis oleh browser dengan  
7 bantuan *javascript* yang ada pada browser pengguna dan akan dijalankan saat sebuah *event* terjadi  
8 pada editor kode.

### 9 Sequence Diagram

- 10 Gambar 3.24 merupakan sebuah *sequence diagram* yang menunjukkan bagaimana fitur perekaman  
11 perubahan atau event akan berintegrasi dengan sistem IDE akan bekerja pada SharIF Judge.



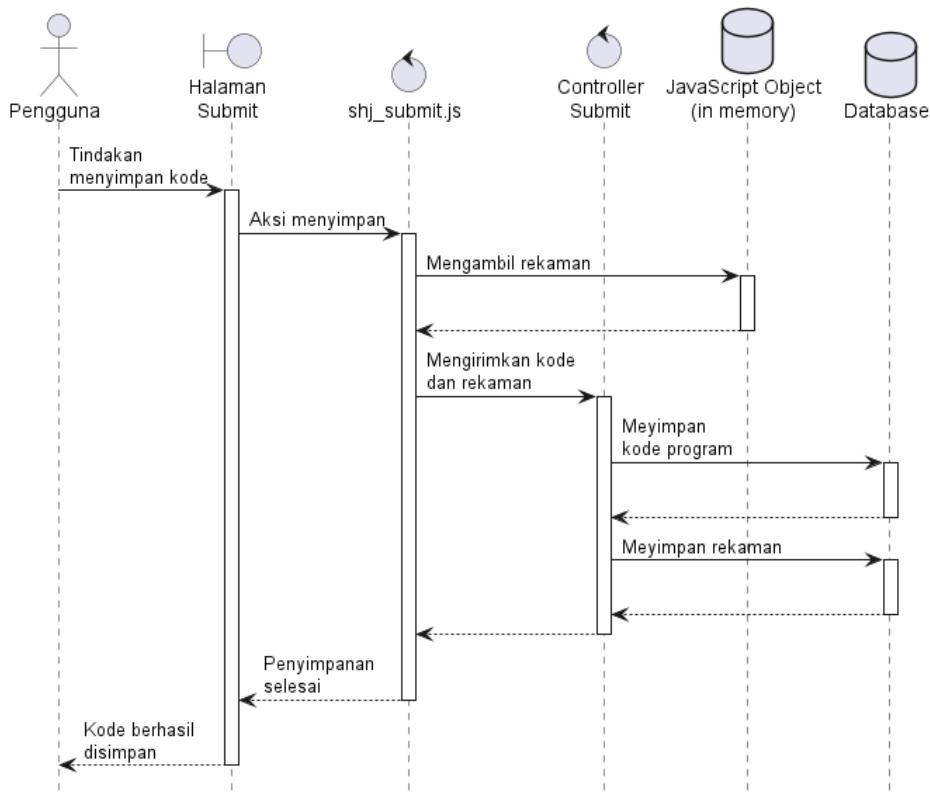
Gambar 3.24: Sequence Diagram Fitur Perekaman Perubahan

### <sup>1</sup> 3.2.2 Fitur penyimpanan rekaman perubahan

- <sup>2</sup> Fitur penyimpanan rekaman perubahan akan dilakukan secara otomatis saat pengguna melakukan tindakan menyimpan kode program. Fitur penyimpanan rekaman akan berintegrasi dengan fitur penyimpanan kode program yang sudah ada. Pada fitur ini daftar rekaman akan diperbaharui dengan adanya rekaman baru atau perubahan pada file rekaman.

### <sup>6</sup> Sequence Diagram

- <sup>7</sup> Gambar 3.25 merupakan sebuah *sequence diagram* yang menunjukkan bagaimana fitur penyimpanan rekaman perubahan akan berintegrasi dengan sistem IDE akan bekerja pada SharIF Judge.



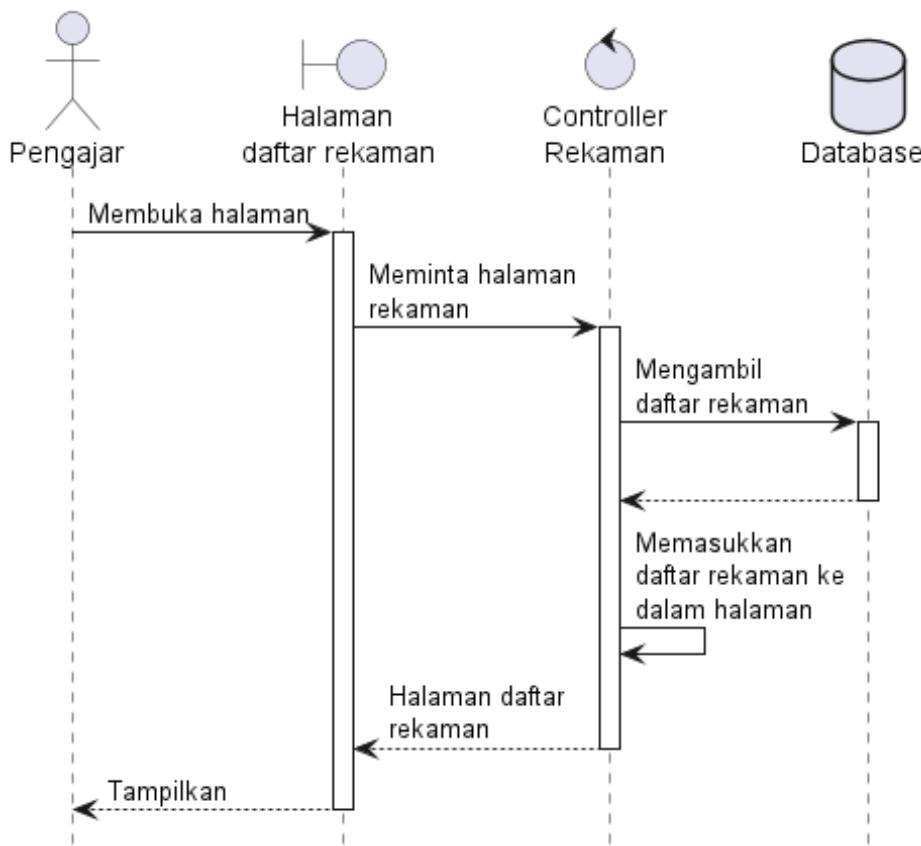
Gambar 3.25: Sequence Diagram Fitur Penyimpanan Rekaman

### <sup>1</sup> 3.2.3 Fitur melihat daftar rekaman

- 2 Pada sistem pemutaran ulang ketikan dibutuhkannya sebuah halaman baru yang akan dinamakan
- 3 halaman rekaman. Pada halaman ini akan dimunculkannya daftar rekaman untuk *assignment* yang
- 4 dipilih pada halaman *Assignment*. Fitur ini akan dijalankan pada saat halaman rekaman dimuat
- 5 ke dalam browser oleh SharIF Judge, dimana data yang dimasukkan ke dalam halaman rekaman
- 6 adalah daftar rekaman tersebut dan beberapa data yang dibutuhkan.

### <sup>7</sup> Sequence Diagram

- 8 Gambar 3.26 merupakan *sequence diagram* yang menunjukkan bagaimana sistem akan bekerja saat
- 9 user akan membuka halaman rekaman pada SharIF Judge.



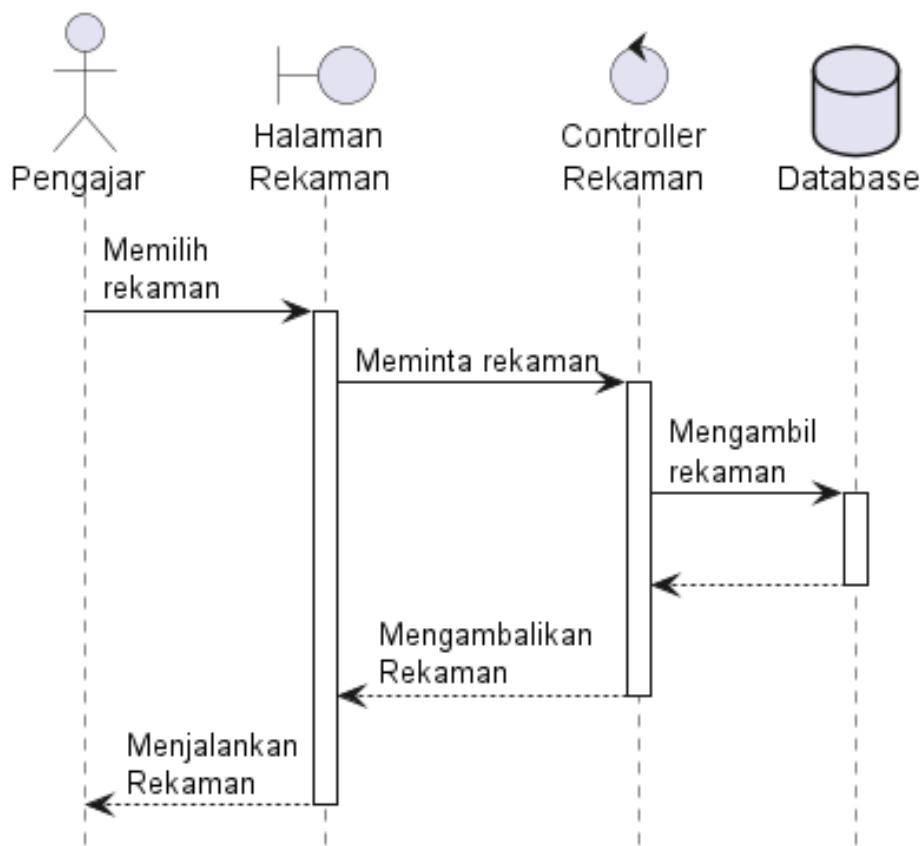
Gambar 3.26: Sequence Diagram Membuka Halaman Rekaman

#### <sup>1</sup> 3.2.4 Fitur pemutaran ulang rekaman

- <sup>2</sup> Fitur pemutaran ulang rekaman akan membuat satu buah rekaman dalam daftar rekaman dalam halaman rekaman dapat ditekan oleh pengguna untuk menandakan bahwa sebuah rekaman dipilih untuk dijalankan. Saat sebuah rekaman dipilih, browser akan meminta data rekaman kepada SharIF Judge dan dengan bantuan *javascript* akan melakukan pemutaran ulang rekaman pada sebuah editor kode yang tidak dapat diubah dalam halaman rekaman.

#### <sup>7</sup> Sequence Diagram

- <sup>8</sup> Gambar 3.27 merupakan *sequence diagram* yang menunjukkan bagaimana sistem SharIF Judge bekerja dari pemilihan rekaman hingga pemutaran ulang rekaman akan terjadi.



Gambar 3.27: Sequence Diagram Membuka Halaman Rekaman



<sup>1</sup>

## BAB 4

<sup>2</sup>

### PERANCANGAN

- <sup>3</sup> Bab ini membahas tentang perancangan untuk implementasi sistem perekaman ulang dalam SharIF-  
<sup>4</sup> Judge. Perancangan akan dilakukan

#### <sup>5</sup> 4.1 Rancangan Antarmuka

##### <sup>6</sup> 4.1.1 Sistem Rekaman

- <sup>7</sup> Seluruh sistem rekaman akan diimplementasikan dalam halaman Submit tidak memerlukan per-  
<sup>8</sup> ubahan pada antarmuka. Gambar XX menunjukkan halaman Submit dan IDE yang sudah di-  
<sup>9</sup> implementasikan oleh Nicholas Aditya Halim [7].

##### <sup>10</sup> 4.1.2 Sistem Pemutaran ulang

- <sup>11</sup> Pada sistem pemutaran ulang dibutuhkan dua halaman baru yaitu halaman untuk menunjukkan  
<sup>12</sup> daftar rekaman dalam sistem dan halaman untuk sistem pemutaran ulang sebuah rekaman. Gambar  
<sup>13</sup> XX merupakan rancangan antarmuka untuk halaman daftar rekaman dan Gambar XX merupakan  
<sup>14</sup> rancangan antarmuka untuk halaman pemutaran ulang.

#### <sup>15</sup> 4.2 Rancangan Penyimpanan Rekaman

- <sup>16</sup> Rekaman yang akan disimpan akan berupa sebuah daftar *event* yang terjadi. Dalam javascript,  
<sup>17</sup> daftar tersebut akan menjadi sebuah *array* yang berisi *event-event* yang terjadi. Rekaman juga  
<sup>18</sup> akan menyimpan waktu dimana rekaman dimulai, awal kode yang ada dalam editor kode dan juga  
<sup>19</sup> awal posisi cursor dalam IDE. Maka dari itu, Rekaman akan menjadi sebuah *object* javascript yang  
<sup>20</sup> berisi sebagai berikut:

- <sup>21</sup> 1. *timestart*: Waktu awal rekaman dimulai.
- <sup>22</sup> 2. *start\_value*: Isi awal dalam editor kode.
- <sup>23</sup> 3. *start\_cursor*: Posisi awal cursor dalam editor kode.
- <sup>24</sup> 4. *events*: Daftar *event* yang terjadi.

- <sup>25</sup> *Event* yang akan direkam juga membutuhkan beberapa data yang harus disimpan yaitu: waktu  
<sup>26</sup> *event* terjadi, *event* yang terjadi, dan muatan *event* yang terjadi. Maka *event* juga akan disimpan  
<sup>27</sup> dalam bentuk *object* javascript. Berikut merupakan format sebuah *event*:

<sup>28</sup> {time: <time>, event: <event>, payload: <payload>}

1 Berikut merupakan penjelasan tentang format penyimpanan perekaman.

- 2 • <time> akan menunjukkan pada milidetik berapa event terjadi setelah waktu awal perekaman  
3 dimulai. sedangkan <event> dan <payload> merupakan data event yang terjadi.
- 4 • <event> merupakan *event* yang terjadi pada waktu tersebut, pada contohnya adalah pengguna  
5 melakukan perubahan pada editor kode dengan menambahkan huruf ‘a’, maka *event* yang  
6 terjadi merupakan *insert*. Semua event yang ditanggap oleh sistem akan dijelaskan pada sub  
7 Bagian 4.3.1.
- 8 • <payload> merupakan muatan *event* yang terjadi. Muatan akan disesuaikan dengan *event*  
9 yang terjadi. Sebagai contoh untuk event *insert* di atas, maka isi dari *event* tersebut adalah  
10 huruf ‘a’, dan posisi cursor dalam editor kode dimana huruf tersebut dimasukkan.

11 Berikut contoh hasil untuk sebuah *event insert* pada penjelasan di atas:

```
12 {time: 1203, event: "insert", payload: {data: "a", start: [10, 9]}}
```

13 Penyimpanan rekaman juga akan disimpan pada folder yang sama dengan penyimpanan kode  
14 submission seperti yang dijelaskan pada Bagian 3.1.3 dengan nama file **record**.

### 15 4.3 Rancangan Perubahan Kode

16 Untuk mengimplementasikan fitur yang diusulkan pada Bagian 3.2, diperlukannya perubahan kode  
17 berikut ini pada SharIF-Judge.

#### 18 4.3.1 Merekam perubahan atau event

19 Untuk menambahkan fitur ini, diperlukannya perubahan pada bagian javascript yaitu **assets/js/**  
20 **shj\_submit.js** dalam halaman Submit. Dimana javascript tersebut akan menjalankan perekaman  
21 secara otomatis saat pengguna memilih *problem* yang ada dalam *assignment* yang dipilih. Berikut  
22 merupakan *event* yang akan ditangkap oleh *javascript* dalam halaman Submit:

- 23 • Perubahan isi kode pada editor kode.
- 24 • Perubahan posisi cursor pada editor kode.
- 25 • Perubahan fokus pada web page.
- 26 • Pergantian *tab* dalam browser.
- 27 • Perubahan fokus pada PDF Viewer.
- 28 • Perubahan isi pada editor *input* dalam IDE.
- 29 • Perubahan isi pada editor *output* dalam IDE.
- 30 • Aksi men-*Save*.
- 31 • Aksi men-*Save & Execute*.
- 32 • Aksi men-*Save & Submit*.

#### 33 4.3.2 Menyimpan rekaman

34 Untuk setiap aksi menyimpan kode, menjalankan kode dengan tes kasus, dan mengumpulkan kode  
35 melalui IDE, rekaman juga akan disimpan ke dalam sistem.

36 Untuk menyimpan rekaman, perlu dilakukan perubahan sebagai berikut:

- 37 • *Controller* Submit:

- 1     – Fungsi `save($type)`:  
2         Fungsi ini akan diubah agar dapat menangani data rekaman yang dikirim oleh *user*.  
3         Data tersebut akan disimpan dalam folder yang sama dengan kode program.
- 4     – Fungsi `_submit($data, $problem_id, $language, $user_dir)`:  
5         Fungsi ini akan diubah agar dapat menangani data rekaman yang dikirim oleh fungsi  
6         `save($type)` dengan menambahkan parameter `$rec` berisi rekaman oleh *user*. Untuk  
7         setiap submit rekaman dari *save-save* sebelumnya akan diubah menjadi rekaman untuk  
8         *submit* tersebut.
- 9     • *Assets shj\_submit.js*:  
10         Menambahkan data rekaman yang dimuat ke dalam fungsi aksi menyimpan kode, menjalankan  
11         kode dengan tes kasus, dan mengumpulkan kode melalui IDE, rekaman juga akan disimpan  
12         ke dalam sistem.

#### 13 4.3.3 Melihat daftar rekaman

14 Untuk melihat semua daftar rekaman yang terjadi, maka dibutuhkannya database untuk menyimpan  
15 daftar dan mendapatkan daftar rekaman yang sudah disimpan dalam sistem. Setelah itu dibutuh-  
16 kannya juga halaman baru dalam SharIF-Judge, maka perubahan *Controller*, *Model*, dan *View*  
17 dalam SharIF-Judge.

18 Dikarenakan itu, perlu dilakukan perubahan kode sebagai berikut:

- 19     • *Controller Submit*:
  - 20         – Fungsi `save($type)`:  
21             Fungsi ini akan menambahkan *metadata* rekaman *user* ke dalam *database*. *Metadata* yang  
22             dimaksud adalah *id problem*, *id assignment*, dan *user* rekaman ini direkam. *Metadata*  
23             dalam *database* digunakan untuk mendapatkan daftar rekaman yang belum disubmit  
24             pada sebuah *problem* dalam *assignment* beserta dengan nama *user* rekaman.
  - 25         – Fungsi `_submit($data, $problem_id, $language, $user_dir)`:  
26             Fungsi ini akan menambahkan *metadata* ke dalam *database* sebagai daftar rekaman yang  
27             sudah di submit. *Metadata* yang dimaksud adalah *id problem*, *id assignment*, dan *user*  
28             rekaman ini direkam. *Metadata* dalam *database* digunakan untuk mendapatkan daftar  
29             rekaman yang sudah disubmit pada sebuah *problem* dalam *assignment* beserta dengan  
30             nama *user* rekaman.
- 31     • *Controller Recording*:  
32         Sebuah *controller* baru yang menangani segala hal mengenai sistem pemutaran ulang dalam  
33         SharIF-Judge. Fungsi yang dibutuhkan agar fitur ini berjalan hanyalah fitur untuk menun-  
34         jukkan daftar rekaman dalam sistem dengan mengambil data dari *model Recording* dan  
35         menaruhnya dalam *view recording*. Setalah itu, *controller* akan menunjukkan *view* tersebut  
36         kepada *user* yang sudah login dan memiliki akses *instructor* atau lebih tinggi.
- 37     • *Model Recording*:  
38         Sebuah *model* baru yang menangani segala hal mengenai penyimpanan dan pengambilan  
39         data rekaman dalam *database*. Fungsi-fungsi yang direncanakan dalam *model* adalah sebagai  
40         berikut:
  - 41             – Fungsi `get_recording()`:

1 Fungsi ini digunakan untuk mendapatkan seluruh daftar rekaman dalam database, fungsi  
2 ini juga dapat menyaring daftar rekaman berdasarkan *assignment*, *problem*, dan *user*.

- 3 – Fungsi `add_recording()`:

4 Fungsi ini digunakan untuk menaruh sebuah rekaman ke dalam database.

- 5 • *View Recording\_list*:

6 Sebuah *view* baru yang menampilkan daftar rekaman yang ada dalam sistem. *View* ini akan  
7 digunakan oleh *Controller Recording*.

8 **4.3.4 Pemutaran ulang rekaman**

9 Untuk dapat memutar ulang rekaman diperlukan beberapa perubahan kode dalam SharIF-Judge.

10 Berikut merupakan rencana perubahan kode dalam SharIF-Judge:

- 11 • *Controller Recording*:

12 Sebuah *controller* baru yang menangani segala hal mengenai sistem pemutaran ulang dalam  
13 SharIF-Judge. Untuk fitur pemutaran ulang rekaman, diperlukan dua fungsi pada *controller*  
14 yaitu sebagai berikut:

- 15 – Fungsi `index()`:

16 Fungsi ini digunakan untuk menunjukkan *view Recording* kepada *user*.

- 17 – Fungsi `download_record()`:

18 Fungsi ini digunakan untuk mendapatkan file rekaman dalam sistem. Fungsi ini akan  
19 dipanggil menggunakan AJAX dalam *assets Recording.js*.

- 20 • *Model Recording*:

21 Sebuah *model* baru yang menangani segala hal mengenai penyimpanan dan pengambilan  
22 data rekaman dalam *database*. Fungsi yang dibutuhkan oleh fitur pemutaran ulang rekaman  
23 adalah fungsi `get_recording()` untuk mendapatkan seluruh daftar rekaman sebuah *user*  
24 dalam database berdasarkan *problem* dan *assignment*.

- 25 • *View Recording*:

26 Sebuah *view* baru yang menampilkan rekaman sebuah *user* yang ada dalam sistem. *View* ini  
27 akan digunakan oleh *Controller Recording*.

- 28 • *Assets Recording.js*:

29 Sebuah *assets* javascript yang digunakan oleh *view Recording* sebagai *script* yang akan  
30 dijalankan oleh *browser user*.

1

## BAB 5

2

### IMPLEMENTASI DAN PENGUJIAN

- 3 Bab ini membahas mengenai implementasi dan pengujian sistem perekaman ulang dalam SharIF-  
4 Judge.

5 **5.1 Implementasi**

- 6 Bagian ini menjelaskan hasil implementasi sistem pemutaran ulang pada SharIF-Judge berdasarkan  
7 perancangan pada Bab 3. Pada saat implementasi juga dilakukan penyesuaian pada perancangan  
8 yang sudah dibuat untuk mengatasi kendala yang dialami pada saat implementasi.

9 **5.1.1 Merekam Peristiwa pada IDE**

- 10 Fitur merekam peristiwa pada editor kode diimplementasikan untuk menangkap seluruh interaksi  
11 pengguna terhadap IDE dan SharIF-Judge pada saat pengguna menyelesaikan sebuah masalah  
12 dalam *assignment*. Data yang direkam akan digunakan untuk memutar ulang penyelesaian yang  
13 dilakukan oleh pengguna. Fitur ini diimplementasikan dengan memanfaatkan *Library Ace*, *event*  
14 *hooks* pada javascript. Implementasi ini akan memerlukan penyesuaian pada bagian *javascript* yaitu  
15 file `assets/js/shj_submit.js` yang dimuat pada halaman *Submit*.

16 Berikut merupakan alur sistem perekaman peristiwa:

17 1. Inisialisasi Perekam

18 Pada alur ini, semua perekaman akan diinisialisasi dengan menjalankan sebuah fungsi di-  
19 namakan `recordStart`. Fungsi ini akan memanggil seluruh *event hooks* dan *event listener*  
20 dalam *Library ace* agar dinyalakan.

21 Dalam menjalankan sebuah fungsi *event listener* dibutuhkannya dua argumen yaitu event  
22 yang akan dipanggil dan sebuah *callback function* yang akan dipanggil pada saat terjadinya  
23 sebuah event tersebut.

24 Dalam implementasi akan dibuat 2 buah *object javascript* yang menjadi fungsi *event listener*  
25 yaitu *object* pemanggilan *event listener* dan *object* menyimpan *callback function* yang dipa-  
26 kai oleh *event listener* masing-masing. Kode 5.1 merupakan *object callback function* yang  
27 diimplementasikan.

Kode 5.1: *object callback function*

```
28
29     const handlers = {
30         editor_change: (e) =>
31             recordEvent(e.action, {
32                 4                     data: e.lines,
33                 5                     start: e.start,
```

```
1           6|           end: e.end,  
2           7|           },  
3           8| }
```

5 Kode 5.2 merupakan *object* pemanggilan *event listener* yang dipanggil pada saat inisialisasi  
6 dan mendapatkan fungsi *callback function* dari *object handlers* pada Kode 5.1.

Kode 5.2: *object event listener*

```
7 const addListener = {  
8   editor_change: () => editor.session.on("change", handlers.editor_change),  
9 }  
10 }
```

Setelah itu *object addListener* akan dipanggil oleh fungsi *recordStart*. Kode 5.3 menunjukkan fungsi yang dipanggil saat inisialisasi.

Kode 5.3: Beberapa *event listener* yang dipanggil

```
14  
15     1 addListener.editor_change();  
16
```

<sup>17</sup> Fungsi `recordStart` akan dilakukan pada saat pengguna mengubah *problem* yang dipilih  
<sup>18</sup> dalam halaman Submit.

## 2. Penyimpan sebuah rekaman

Untuk setiap perekaman yang dibutuhkan, dijalankan sebuah fungsi yang mendeteksi perubahan tersebut dan menjalankan sebuah *callback function* saat terjadi perubahan tersebut. Fungsi ini dipanggil pada saat inisialisasi. *Callback function* tersebut akan mendapatkan argumen sesuai dengan perubahan yang dideteksi. Pada contohnya untuk perubahan teks pada editor kode yaitu fungsi `onchange`, argumen yang diberikan merupakan teks yang dimasukan yaitu contohnya adalah ‘A’, dan juga posisi dimana teks tersebut dimasukkan dalam editor kode. Kode 5.4 merupakan contoh argumen yang diberikan.

Kode 5.4: Contoh argumen yang diberikan oleh fungsi onchange

```
27
28  1 {
29    2   data: ["A"],
30    3   start: {row: 0, column: 1},
31    4   end: { row: 0, column: 2}
32  5 }
```

Setelah itu data akan disimpan dalam sebuah *object javascript* seperti yang sudah dijelaskan pada Bab 4.2. data argumen akan disimpan menggunakan key ‘args’ atau ‘payload’.

### 3. Penyimpanan data rekaman

Selanjutnya sebuah *event* atau rekaman yang sudah dicatat dan menjadi sebuah *object javascript* bernama **recording**. seluruh event rekaman akan simpan dalam sebuah *array* dalam **recording** dengan key ‘events’ seperti yang sudah dijelaskan pada Bab 4.2. **recording** juga memiliki waktu dimulainya rekaman, isi awal editor kode, posisi awal cursor dalam editor kode. **recording** juga memiliki fungsi **init** untuk meninisialisasi seluruh *object recording*.

Kode 5.5: Contoh argumen yang diberikan oleh fungsi onchange

```
42
43     1 const recording = {
44         2     events: [],
45         3     startTime: -1,
46         4     startValue: "",
47         5     startSelection: [],
48
49         6
50         7     init: () => {
51             8         recording.events = [];
52             9         recording.startTime = Date.now();
```

```

1  10     recording.startValue = editor.getValue();
2  11     recording.startSelection = getSelection(editor);
3  12   },
4  13 };

```

Kode 5.5 merupakan `recording` pada saat keadaan kosong. Pada *object recording*, `events` merupakan sebuah array dengan isi sebuah rekaman event, `startTime` merupakan waktu awal rekaman dimulai, `startValue` merupakan isi awal dalam editor kode, dan `startSelection` merupakan posisi awal cursor dalam editor kode.

Berikut merupakan peristiwa yang akan direkam oleh sistem perekaman:

- `editor.change`: Peristiwa ini akan menangkap perubahan isi teks dalam editor kode.
  - `editor.changeCursor`: Peristiwa ini akan menangkap perubahan cursor dalam editor kode.
  - `editor.changeSelection`: Peristiwa ini akan menangkap perubahan selection cursor dalam editor kode.
  - `window.focus`: Peristiwa ini akan menangkap pengguna pada saat pengguna *focus* pada SharIF Judge web page.
  - `window.blur`: Peristiwa ini akan menangkap pengguna pada saat pengguna tidak *focus* pada SharIF Judge web page atau *focus* pada aplikasi lain atau web page lain.
  - `window.visibilitychange`: Peristiwa ini akan menangkap pengguna yang mengubah *tab* dari SharIF Judge ke *tab* lain dalam browser.
  - `pdf_viewer.focusin`: Peristiwa ini akan menangkap pengguna pada saat pengguna *focus* pada PDF Viewer dalam IDE.
  - `pdf_viewer.focusout`: Peristiwa ini akan menangkap pengguna pada saat pengguna tidak *focus* pada PDF Viewer dalam IDE.
  - `editor_input.input`: Peristiwa ini akan menangkap perubahan isi editor input dalam IDE.
  - `editor_output.change`: Peristiwa ini akan menangkap hasil output saat terjadinya aksi *Save & Execute* dalam IDE.
  - `save.click`: Peristiwa ini akan menangkap aksi *Save* yang dilakukan pengguna.
  - `execute.click`: Peristiwa ini akan menangkap aksi *Save & Execute* yang dilakukan pengguna.
  - `submit.click`: Peristiwa ini akan menangkap aksi *Save & Submit* yang dilakukan pengguna.
- Seluruh alur sistem perekaman peristiwa dalam SharIF-Judge akan ditambahkan ke dalam file `assets/js/shj_submit.js`. Kode perubahan terdapat pada Lampiran ??.

### 33 Perbaikan Implementasi

Pada saat mengujian fungsi penyimpanan rekaman, dalam tahap alur penyimpanan data rekaman dan juga bagaimana inisialisasi perekaman akan diubah dari perancangan Bab 4.3.1. Hal ini dikarenakan saat pengguna memilih ulang masalah atau memuat ulang halaman Submit, maka semua events yang sudah direkam akan hilang. Maka dari itu berikut merupakan perubahan pada alur fitur sistem perekaman ketikan:

#### 1. Inisialisasi Perekam

Alur inisialisasi akan diubah agar dapat memuat events yang sebelumnya sudah disimpan dalam *object javascript* bernama `befRecording`. Oleh karena itu, dibutuhkannya penambahan fungsi pada `Controller Submit.php` yaitu fungsi untuk mengambil data rekaman sebelumnya bernama `load_rec` yang mengambil argumen pengenal masalah yang dipilih oleh pengguna.

- 1 Kode penambahan terdapat di Lampiran ??.
2. Penyimpanan data rekaman
- 3 Pada *object recording*, *startValue* dan *startSelection* tidak dibutuhkan karena isi awal  
4 dari editor kode dan juga posisi awal cursor dalam editor kode akan selalu berisi dengan nilai  
5 kosong yaitu teks kosong dan posisi di baris dan kolom pertama.

### 6 5.1.2 Menyimpan Rekaman pada Sistem

7 Fitur penyimpanan rekaman pada sistem bertujuan untuk menyimpan data secara permanen ke  
8 dalam database dan *server* agar dapat diputar kembali di lain waktu. Berikut merupakan alur fitur  
9 penyimpanan rekaman pada sistem:

10 1. Mengirimkan Data ke *Server*

11 Dalam halaman Submit, pengguna memiliki 3 aksi penting dalam IDE SharIF-Judge yaitu:  
12 *save*, *execute*, dan *submit*. Alur ini akan mengirimkan data rekaman pada saat pengguna  
13 melakukan aksi tersebut. Data yang dikirim merupakan *object recording* yang sudah jadikan  
14 sebagai teks JSON dengan menggunakan fungsi `JSON.stringify`.

15 2. Menyimpan Data Dalam File Sistem

16 File akan disimpan dalam *folder* yang sama dengan penyimpanan kode *submission* yang  
17 dijelaskan pada Bab 3.1.3. File akan diisi secara langsung oleh data rekaman dan tidak  
18 diubah oleh *server*. Penamaan file dapat dibagi berdasarkan aksi yang membuat pengguna  
19 mengirimkan data rekaman. Untuk aksi *save* dan *execute*, file dengan data rekaman akan  
20 disimpan dengan nama *recording*. Untuk aksi *submit*, file akan disimpan dengan nama  
21 *recording* dilanjutkan dengan sebuah ‘-’ dan *submit id* yang dibuat. File tersebut akan  
22 memiliki tipe data yang sama yaitu JSON dikarenakan itu extensi file yang digunakan adalah  
23 `.json`.

24 3. Menyimpan Data Dalam Database

25 Saat penyimpanan data ke dalam file sistem berhasil, maka penyimpanan kedalam database  
26 juga akan dilakukan. Data yang akan disimpan kedalam database akan digunakan untuk  
27 mendaftar rekaman yang ada dalam sistem, maka data yang akan disimpan bukan data  
28 rekaman melainkan data statistik. Berikut merupakan data yang akan disimpan ke dalam  
29 Database:

- 30 • **`rec_id`**: pengenal rekaman yang sama dengan *submit id*.
- 31 • **`username`**: nama pengguna yang mengirimkan data rekaman.
- 32 • **`problem_id`**: pengenal masalah yang pengguna kerjakan.
- 33 • **`assignment_id`**: pengenal tugas yang pengguna kerjakan.
- 34 • **`upload_at`**: waktu sistem menyimpan data rekaman.

35 Untuk membuat databasenya sendiri, dibutuhkan penambahan tabel bernama tabel **recording**  
36 yang memiliki lima atribut diatas. Kode 5.6 menunjukkan pembuatan tabel baru menggunakan  
37 *CodeIgniter* dalam SharIF-Judge.

38 Kode 5.6: Kode membuat database pada SharIF-Judge

```
39
40 // create table 'recording'
41 $fields = array(
42     'rec_id'      => array('type' => 'INT', 'constraint' => 11, 'unsigned' => TRUE),
43     'upload_at'   => array('type' => $DATETIME),
44     'assignment'  => array('type' => 'SMALLINT', 'constraint' => 4, 'unsigned' => TRUE),
```

```

1      'problem'      => array('type' => 'SMALLINT', 'constraint' => 4, 'unsigned' => TRUE),
2      'username'     => array('type' => 'VARCHAR', 'constraint' => 20),
3  );
4  $this->dbforge->add_field($fields);
5  if (! $this->dbforge->create_table('recording', TRUE))
6      show_error("Error creating database table " . $this->db->dbprefix('recording'));
7 // ADD Unique constraint
8 $this->db->query(
9     "ALTER TABLE {$this->db->dbprefix('recording')}
10    ADD CONSTRAINT {$this->db->dbprefix('sruap_unique')} UNIQUE (rec_id, username, assignment, problem);"
11 );
12

```

Mengikuti arsitektural *CodeIgniter*, untuk menambahkan sebuah data ke dalam database perlu menggunakan sebuah *Model*. Oleh karena itu, dibutuhkannya *model* baru bernama **Recording\_model.php** yang ditambahkan fungsi **add\_recording()** yang memiliki argumen yaitu seluruh data yang ingin disimpan dalam database.

Untuk aksi *save* dan *execute* dimana tidak adanya *submit id* maka akan dibuat menjadi angka nol ('0') pada pengenal rekamannya atau **rec\_id** jika aksinya merupakan *submit*.

Untuk alur pengiriman data ke *server*, dibutuhkannya penambahan kode ke dalam **assets/js/shj\_submit.js**. Kode pembahaman terdapat pada Lampiran ???. Agar dapat menyimpan dibutuhkannya perubahan dalam kode *Controller Submit.php* pada fungsi **save(\$type)** dan fungsi **\_submit()**. Kode perubahan *Submit.php* terdapat pada Lampiran ???.

### 23 Perbaikan Implementasi

Pada saat pengujian yang sama dengan Bab 5.1.1, dibutuhkannya perubahan pada alur pengirimkan data ke server. Dikarenakan *events* yang sudah di *save* dapat terhapus karena pengguna memilih ulang masalah dan memuat ulang halaman *Submit*, yang membuat rekaman inisialisasi dan menghapus rekaman lama. Maka dari itu pada saat mengirimkan data **recording**, akan disertakan juga data **befRecording** yang sudah diambil pada saat inisialisasi. Format pengiriman data juga akan berubah dikarenakan adanya rekaman yang lama menjadi sebuah *key* dan *value* karena hanya dua *value* yang harus disimpan yaitu *events* sebagai *value* dan *startTime* sebagai *key*. Maka format ini menjadi format keseluruhan *events* yang terjadi dan dapat disatukan dengan format yang sama menggunakan *spread operator* agar seluruh rekaman lama digabungkan. Kode 5.7 merupakan kode untuk mengirimkan teks JSON dengan mengabungkan kedua rekaman menggunakan *spread operator*.

Kode 5.7: *object callback function*

```

35
36 1 JSON.stringify({
37 2   ...befRecording,
38 3   [recording.startTime]: recording.events
39 4 }),
40

```

### 41 5.1.3 Melihat Daftar Rekaman

Fitur ini digunakan untuk melihat daftar rekaman mahasiswa yang tersimpan dalam sistem, pengguna juga dapat melihat isi rekaman yang terdapat dalam daftar rekaman tersebut. Fitur ini dibutuhkan dua tahap untuk diimplementasikan yaitu implementasi pengambilan data dan implementasi menampilkan data dan antarmuka.

1 **Pengambilan Data Rekaman**

2 Fitur pengambilan data rekaman digunakan agar bagian depan SharIF-Judge dapat meminta daftar  
3 rekaman yang ada pada bagian belakang SharIF-Judge. Oleh karena ini merupakan sebuah halaman  
4 baru dalam SharIF-Judge, maka dibutuhkannya sebuah *Controller* baru bernama *Recording.php*  
5 yang menampilkan sebuah halaman baru yang dapat diakses melalui rute */recording/all/* yang  
6 menggunakan fungsi baru dalam *Recording\_model.php* yaitu fungsi untuk mendapatkan daftar re-  
7 kaman dinamakan *all\_user\_recordings*. Kode pertambahan pada *Model Recording.php* berada  
8 pada Lampiran ??.

9 Berikut fungsi yang akan diimplementasikan dalam *Controller Recording.php*:

10 1. *\_\_construct()*

11 Fungsi ini akan memuat seluruh kebutuhan *Model* dan *Helper* ke dalam *Recording.php*,  
12 fungsi *construct* juga akan membatasi akses oleh pengguna dibawah *instructor*. Fungsi ini  
13 juga mendapatkan *params url* yang dikirim oleh pengguna.

14 2. *all()*

15 Fungsi ini akan mengambil beberapa data yang dibutuhkan oleh antarmuka SharIF-Judge  
16 yaitu *assignment* yang dipilih oleh pengguna menggunakan *assignment\_model*, *problem*  
17 yang ada dalam *assignment* tersebut menggunakan *assignment\_model*, dan daftar *recording*  
18 yang tersimpan dalam sistem menggunakan *recording\_model*. Setelah itu, server akan  
19 menempatkan seluruh data tersebut ke dalam *view* baru bernama *recording\_list.twig*.

20 Seluruh alur untuk mengimplementasikan fitur pengambilan data rekaman dalam *Controller*  
21 *Recording.php* terdapat dalam Lampiran ??.

22 **Antarmuka dan Tampilan Data**

23 Antarmuka yang akan dibuat serupa dengan perancangan pada Bab 4.1.2 yang diimplementasikan  
24 ke dalam SharIF-Judge. Data yang dikirim oleh *Controller Recording.php* juga dapat ditampilkan  
25 menggunakan *Library twig* tanpa menbutuhkan *javascript* maupun *php* dalam antarmukanya.  
26 Gambar ?? menunjukkan implementasi antarmuka beserta data yang terdapat dalam sistem. Untuk  
27 kode keseluruhan antarmuka menggunakan *Library twig* dapat dilihat pada Lampiran ??.

28 **5.1.4 Pemutaran Ulang Rekaman**

29 Fungsi pemutaran ulang rekaman menggunakan data rekaman yang sudah disimpan dalam sistem  
30 untuk menvisualisasikan proses penyelesaian masalah pengguna secara kronologis. Fitur pemutaran  
31 ulang ini membutuhkan tiga implementasi antarmuka, implementasi memuat data rekaman, dan  
32 implementasi menjalankan rekaman.

33 **Implementasi Antarmuka**

34 Untuk menambahkan sebuah halaman baru dalam SharIF-Judge dibutuhkannya juga fungsi baru  
35 pada *Controller Recording.php*. Fungsi baru akan dinamakan *index* untuk menampilkan sebuah  
36 *view* baru bernama *recording.twig*. Fungsi *index* akan menampilkan halaman baru itu melalui  
37 rute */recording/*. Penamaan *index* itu agar rute tidak memerlukan */index* pada akhir rute karena  
38 jika rute *function* (Bab 2.2.2) akan otomatis mengarah pada fungsi *index* dalam kelas tersebut.

- 1 Fungsi `index` akan mengirim data daftar rekaman pengguna lainnya dalam masalah yang dipilih.
- 2 Data tersebut akan dipakai oleh `recording.twig` untuk menambahkan daftar rekaman pengguna lainnya pada *assignment* dan *problem* yang sama.
- 4 Gambar ?? merupakan antarmuka yang diimplementasikan serupa dengan perancangan pada Bab 4.1.2.

## 6 Implementasi Memuat Data Rekaman

- 7 Data rekaman yang akan diambil sudah disimpan (Bab 5.1.2) dalam sistem menggunakan *Controller*.
- 8 Tetapi data rekaman tidak akan dikirim oleh *Controller* pada saat halaman *recording* dimuat, melainkan menggunakan AJAX pada halaman *recording*. Maka dari itu, butuh fungsi baru pada *Controller Recording.php* dan sebuah assets *javascript* baru bernama `shj_function.js`.
- 11 Fungsi baru dalam *Controller Recording.php* akan dinamakan `download_record` yang memiliki argumen `assignment_id`, `problem_id`, dan `rec_id`. Fungsi tersebut akan mengambil file rekaman dalam file sistem dengan menggunakan argumen untuk mendapatkan lokasi dan nama file rekaman (Bab 4.2) SharIF-Judge dan mengirimkan file tersebut secara langsung. File juga akan dirikim dengan header `Content-Type: application/json` dan `Content-Disposition: attachment; filename= "rec.json"`.

- 17 Fungsi `download_record` akan dipanggil pada saat halaman *recording* dimuat oleh *javascript shj\_function.js* menggunakan fungsi baru yaitu `getRecording`. Fungsi `getRecording` akan meninisialisasi editor kode dalam antarmuka dan menformat data rekaman agar lebih mudah untuk diputar ulang. Berikut merupakan format data tambahan yang akan diubah oleh fungsi `getRecording`:

- `events`: Data rekaman
- `eventsIndex`: sebuah map dengan *key* waktu saat sebuah *events* terjadi dan *value* *index* waktu saat sebuah *events* terjadi.
- `indexEvents`: sebuah map dengan *key* *index* waktu saat sebuah *events* terjadi dan *value* waktu saat sebuah *events* terjadi.
- `presumIndexDuration`: menkalkulasikan panjang rekaman sebelum rekaman selesai.
- `length`: panjang data rekaman
- `duration`: durasi dari seluruh rekaman yang ada pada data rekaman

- 29 Data tersebut akan disimpan dalam sebuah *object javascript* yang dinamakan `recording` dan akan dipakai pada saat menjalankan rekaman dan untuk menampilkan histogram *events* yang terjadi.

## 32 Implementasi Menjalankan Rekaman

- 33 Fitur menjalankan rekaman akan menggunakan data berdasarkan data yang didapatkan oleh AJAX yang dijelaskan pada bagian 5.1.4. Fitur menjalankan rekaman akan membutuhkan penambahan kode pada *javascript shj\_function.js* yang akan menjalankan fungsi `play` atau `stop` untuk menjalankan atau memberhentikan rekaman oleh pengguna.

- 37 Fungsi menjalankan atau mematikan rekaman dibagi menjadi dua yaitu fungsi rekaman dalam IDE dengan data rekaman dinamakan `Recording` dan fungsi timer yang digunakan untuk memberitahu kepada pengguna progress waktu pemutaran rekaman dinamakan `Timer`. Fungsi `Recording` menggunakan fungsi dalam *Library Ace* dan fungsi *javascript* untuk memperbarui IDE

1 antarmuka berdasarkan *event* yang dipanggil, fungsi `setTimeout` dalam *javascript* akan digunakan  
 2 untuk menjalankan *event* selanjutnya bedasarkan perbedaan waktu antara event sekarang dan event  
 3 selanjutnya dengan memanggil fungsi `playRecording` dengan *event* selanjutnya. Sedangkan fungsi  
 4 `Timer` menggunakan fungsi `setInterval` yang akan dijalankan berulang untuk setiap detiknya dan  
 5 memperbarui progress waktu dalam antarmuka berdasarkan waktu yang sudah lewat.

## 6 Menampilkan Histogram Events yang Terjadi

7 Pada fungsi `getRecording` setelah memuat data rekaman dan menformat data rekaman tersebut,  
 8 fungsi `setUpChart` akan dipanggil dan membuat data grafik histogram. Data histogram akan  
 9 dimuat menggunakan *Library Chart.js*.

## 10 5.2 Pengujian Fungsional

## 11 5.3 Pengujian Eksperimental

12 Pada Bagian ini dilakukannya pengujian terhadap sistem Perekaman Ulang pada SharIF Judge.

### 13 5.3.1 Lingkungan pengujian

14 Pengujian sistem perekaman ulang akan dilakukan pada sebuah server VPS atau *Virtual Private*  
 15 *Server*. Pada server akan dijalankannya *docker* agar sistem aplikasi akan identik dengan pada  
 16 saat sistem sedang diimplementasikan. Kode 5.8 dan Kode 5.9 merupakan file *docker-compose* dan  
 17 *Dockerfile* yang digunakan membangun sistem SharIF Judge dalam server VPS.

Kode 5.8: File *docker-compose* yang digunakan

```

18
19 1 version: "3"
20 2 services:
21 3   codeigniter-3:
22 4     build: .
23 5     ports:
24 6       - "81:80"
25 7     volumes:
26 8       - ./var/www/html
27 9     depends_on:
28 0       - db
29 1
30 2
31 3   db:
32 4     image: mysql:5.7
33 5     ports:
34 6       - "3306:3306"
35 7     environment:
36 8       MYSQL_TABLE: judge
37 9       MYSQL_USER: sharif
38 0       MYSQL_PASSWORD: judge
39 1       MYSQL_ROOT_PASSWORD: root
39 2     command: --sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
39 3     volumes:
39 4       - ./mysql:/var/lib/mysql
39 5
39 6
39 7   phpmyadmin:
39 8     image: phpmyadmin/phpmyadmin
39 9     ports:
40 0       - "8080:80"
40 1     environment:
40 2       PMA_HOST: db
40 3       MYSQL_ROOT_PASSWORD: freehost
40 4     depends_on:
40 5       - db
40 6
40 7
40 8
40 9
41 0
41 1
41 2
41 3
41 4
41 5
41 6
41 7
41 8
41 9
41 10
41 11
41 12
41 13
41 14
41 15
41 16
41 17
41 18
41 19
41 20
41 21
41 22
41 23
41 24
41 25
41 26
41 27
41 28
41 29
41 30
41 31
41 32
41 33
41 34
41 35
41 36
41 37
41 38
41 39
41 40
41 41
41 42
41 43
41 44
41 45
41 46
41 47
41 48
41 49
41 50
41 51
41 52
41 53
41 54
41 55
41 56
41 57
41 58
41 59
41 60
41 61
41 62
41 63
41 64
41 65
41 66
41 67
41 68
41 69
41 70
41 71
41 72
41 73
41 74
41 75
41 76
41 77
41 78
41 79
41 80
41 81
41 82
41 83
41 84
41 85
41 86
41 87
41 88
41 89
41 90
41 91
41 92
41 93
41 94
41 95
41 96
41 97
41 98
41 99
41 100
41 101
41 102
41 103
41 104
41 105
41 106
41 107
41 108
41 109
41 110
41 111
41 112
41 113
41 114
41 115
41 116
41 117
41 118
41 119
41 120
41 121
41 122
41 123
41 124
41 125
41 126
41 127
41 128
41 129
41 130
41 131
41 132
41 133
41 134
41 135
41 136
41 137
41 138
41 139
41 140
41 141
41 142
41 143
41 144
41 145
41 146
41 147
41 148
41 149
41 150
41 151
41 152
41 153
41 154
41 155
41 156
41 157
41 158
41 159
41 160
41 161
41 162
41 163
41 164
41 165
41 166
41 167
41 168
41 169
41 170
41 171
41 172
41 173
41 174
41 175
41 176
41 177
41 178
41 179
41 180
41 181
41 182
41 183
41 184
41 185
41 186
41 187
41 188
41 189
41 190
41 191
41 192
41 193
41 194
41 195
41 196
41 197
41 198
41 199
41 200
41 201
41 202
41 203
41 204
41 205
41 206
41 207
41 208
41 209
41 210
41 211
41 212
41 213
41 214
41 215
41 216
41 217
41 218
41 219
41 220
41 221
41 222
41 223
41 224
41 225
41 226
41 227
41 228
41 229
41 230
41 231
41 232
41 233
41 234
41 235
41 236
41 237
41 238
41 239
41 240
41 241
41 242
41 243
41 244
41 245
41 246
41 247
41 248
41 249
41 250
41 251
41 252
41 253
41 254
41 255
41 256
41 257
41 258
41 259
41 260
41 261
41 262
41 263
41 264
41 265
41 266
41 267
41 268
41 269
41 270
41 271
41 272
41 273
41 274
41 275
41 276
41 277
41 278
41 279
41 280
41 281
41 282
41 283
41 284
41 285
41 286
41 287
41 288
41 289
41 290
41 291
41 292
41 293
41 294
41 295
41 296
41 297
41 298
41 299
41 300
41 301
41 302
41 303
41 304
41 305
41 306
41 307
41 308
41 309
41 310
41 311
41 312
41 313
41 314
41 315
41 316
41 317
41 318
41 319
41 320
41 321
41 322
41 323
41 324
41 325
41 326
41 327
41 328
41 329
41 330
41 331
41 332
41 333
41 334
41 335
41 336
41 337
41 338
41 339
41 340
41 341
41 342
41 343
41 344
41 345
41 346
41 347
41 348
41 349
41 350
41 351
41 352
41 353
41 354
41 355
41 356
41 357
41 358
41 359
41 360
41 361
41 362
41 363
41 364
41 365
41 366
41 367
41 368
41 369
41 370
41 371
41 372
41 373
41 374
41 375
41 376
41 377
41 378
41 379
41 380
41 381
41 382
41 383
41 384
41 385
41 386
41 387
41 388
41 389
41 390
41 391
41 392
41 393
41 394
41 395
41 396
41 397
41 398
41 399
41 400
41 401
41 402
41 403
41 404
41 405
41 406
41 407
41 408
41 409
41 410
41 411
41 412
41 413
41 414
41 415
41 416
41 417
41 418
41 419
41 420
41 421
41 422
41 423
41 424
41 425
41 426
41 427
41 428
41 429
41 430
41 431
41 432
41 433
41 434
41 435
41 436
41 437
41 438
41 439
41 440
41 441
41 442
41 443
41 444
41 445
41 446
41 447
41 448
41 449
41 450
41 451
41 452
41 453
41 454
41 455
41 456
41 457
41 458
41 459
41 460
41 461
41 462
41 463
41 464
41 465
41 466
41 467
41 468
41 469
41 470
41 471
41 472
41 473
41 474
41 475
41 476
41 477
41 478
41 479
41 480
41 481
41 482
41 483
41 484
41 485
41 486
41 487
41 488
41 489
41 490
41 491
41 492
41 493
41 494
41 495
41 496
41 497
41 498
41 499
41 500
41 501
41 502
41 503
41 504
41 505
41 506
41 507
41 508
41 509
41 510
41 511
41 512
41 513
41 514
41 515
41 516
41 517
41 518
41 519
41 520
41 521
41 522
41 523
41 524
41 525
41 526
41 527
41 528
41 529
41 530
41 531
41 532
41 533
41 534
41 535
41 536
41 537
41 538
41 539
41 540
41 541
41 542
41 543
41 544
41 545
41 546
41 547
41 548
41 549
41 550
41 551
41 552
41 553
41 554
41 555
41 556
41 557
41 558
41 559
41 560
41 561
41 562
41 563
41 564
41 565
41 566
41 567
41 568
41 569
41 570
41 571
41 572
41 573
41 574
41 575
41 576
41 577
41 578
41 579
41 580
41 581
41 582
41 583
41 584
41 585
41 586
41 587
41 588
41 589
41 590
41 591
41 592
41 593
41 594
41 595
41 596
41 597
41 598
41 599
41 600
41 601
41 602
41 603
41 604
41 605
41 606
41 607
41 608
41 609
41 610
41 611
41 612
41 613
41 614
41 615
41 616
41 617
41 618
41 619
41 620
41 621
41 622
41 623
41 624
41 625
41 626
41 627
41 628
41 629
41 630
41 631
41 632
41 633
41 634
41 635
41 636
41 637
41 638
41 639
41 640
41 641
41 642
41 643
41 644
41 645
41 646
41 647
41 648
41 649
41 650
41 651
41 652
41 653
41 654
41 655
41 656
41 657
41 658
41 659
41 660
41 661
41 662
41 663
41 664
41 665
41 666
41 667
41 668
41 669
41 670
41 671
41 672
41 673
41 674
41 675
41 676
41 677
41 678
41 679
41 680
41 681
41 682
41 683
41 684
41 685
41 686
41 687
41 688
41 689
41 690
41 691
41 692
41 693
41 694
41 695
41 696
41 697
41 698
41 699
41 700
41 701
41 702
41 703
41 704
41 705
41 706
41 707
41 708
41 709
41 710
41 711
41 712
41 713
41 714
41 715
41 716
41 717
41 718
41 719
41 720
41 721
41 722
41 723
41 724
41 725
41 726
41 727
41 728
41 729
41 730
41 731
41 732
41 733
41 734
41 735
41 736
41 737
41 738
41 739
41 740
41 741
41 742
41 743
41 744
41 745
41 746
41 747
41 748
41 749
41 750
41 751
41 752
41 753
41 754
41 755
41 756
41 757
41 758
41 759
41 760
41 761
41 762
41 763
41 764
41 765
41 766
41 767
41 768
41 769
41 770
41 771
41 772
41 773
41 774
41 775
41 776
41 777
41 778
41 779
41 780
41 781
41 782
41 783
41 784
41 785
41 786
41 787
41 788
41 789
41 790
41 791
41 792
41 793
41 794
41 795
41 796
41 797
41 798
41 799
41 800
41 801
41 802
41 803
41 804
41 805
41 806
41 807
41 808
41 809
41 810
41 811
41 812
41 813
41 814
41 815
41 816
41 817
41 818
41 819
41 820
41 821
41 822
41 823
41 824
41 825
41 826
41 827
41 828
41 829
41 830
41 831
41 832
41 833
41 834
41 835
41 836
41 837
41 838
41 839
41 840
41 841
41 842
41 843
41 844
41 845
41 846
41 847
41 848
41 849
41 850
41 851
41 852
41 853
41 854
41 855
41 856
41 857
41 858
41 859
41 860
41 861
41 862
41 863
41 864
41 865
41 866
41 867
41 868
41 869
41 870
41 871
41 872
41 873
41 874
41 875
41 876
41 877
41 878
41 879
41 880
41 881
41 882
41 883
41 884
41 885
41 886
41 887
41 888
41 889
41 890
41 891
41 892
41 893
41 894
41 895
41 896
41 897
41 898
41 899
41 900
41 901
41 902
41 903
41 904
41 905
41 906
41 907
41 908
41 909
41 910
41 911
41 912
41 913
41 914
41 915
41 916
41 917
41 918
41 919
41 920
41 921
41 922
41 923
41 924
41 925
41 926
41 927
41 928
41 929
41 930
41 931
41 932
41 933
41 934
41 935
41 936
41 937
41 938
41 939
41 940
41 941
41 942
41 943
41 944
41 945
41 946
41 947
41 948
41 949
41 950
41 951
41 952
41 953
41 954
41 955
41 956
41 957
41 958
41 959
41 960
41 961
41 962
41 963
41 964
41 965
41 966
41 967
41 968
41 969
41 970
41 971
41 972
41 973
41 974
41 975
41 976
41 977
41 978
41 979
41 980
41 981
41 982
41 983
41 984
41 985
41 986
41 987
41 988
41 989
41 990
41 991
41 992
41 993
41 994
41 995
41 996
41 997
41 998
41 999
41 1000
41 1001
41 1002
41 1003
41 1004
41 1005
41 1006
41 1007
41 1008
41 1009
41 1010
41 1011
41 1012
41 1013
41 1014
41 1015
41 1016
41 1017
41 1018
41 1019
41 1020
41 1021
41 1022
41 1023
41 1024
41 1025
41 1026
41 1027
41 1028
41 1029
41 1030
41 1031
41 1032
41 1033
41 1034
41 1035
41 1036
41 1037
41 1038
41 1039
41 1040
41 1041
41 1042
41 1043
41 1044
41 1045
41 1046
41 1047
41 1048
41 1049
41 1050
41 1051
41 1052
41 1053
41 1054
41 1055
41 1056
41 1057
41 1058
41 1059
41 1060
41 1061
41 1062
41 1063
41 1064
41 1065
41 1066
41 1067
41 1068
41 1069
41 1070
41 1071
41 1072
41 1073
41 1074
41 1075
41 1076
41 1077
41 1078
41 1079
41 1080
41 1081
41 1082
41 1083
41 1084
41 1085
41 1086
41 1087
41 1088
41 1089
41 1090
41 1091
41 1092
41 1093
41 1094
41 1095
41 1096
41 1097
41 1098
41 1099
41 1100
41 1101
41 1102
41 1103
41 1104
41 1105
41 1106
41 1107
41 1108
41 1109
41 1110
41 1111
41 1112
41 1113
41 1114
41 1115
41 1116
41 1117
41 1118
41 1119
41 1120
41 1121
41 1122
41 1123
41 1124
41 1125
41 1126
41 1127
41 1128
41 1129
41 1130
41 1131
41 1132
41 1133
41 1134
41 1135
41 1136
41 1137
41 1138
41 1139
41 1140
41 1141
41 1142
41 1143
41 1144
41 1145
41 1146
41 1147
41 1148
41 1149
41 1150
41 1151
41 1152
41 1153
41 1154
41 1155
41 1156
41 1157
41 1158
41 1159
41 1160
41 1161
41 1162
41 1163
41 1164
41 1165
41 1166
41 1167
41 1168
41 1169
41 1170
41 1171
41 1172
41 1173
41 1174
41 1175
41 1176
41 1177
41 1178
41 1179
41 1180
41 1181
41 1182
41 1183
41 1184
41 1185
41 1186
41 1187
41 1188
41 1189
41 1190
41 1191
41 1192
41 1193
41 1194
41 1195
41 1196
41 1197
41 1198
41 1199
41 1200
41 1201
41 1202
41 1203
41 1204
41 1205
41 1206
41 1207
41 1208
41 1209
41 1210
41 1211
41 1212
41 1213
41 1214
41 1215
41 1216
41 1217
41 1218
41 1219
41 1220
41 1221
41 1222
41 1223
41 1224
41 1225
41 1226
41 1227
41 1228
41 1229
41 1230
41 1231
41 1232
41 1233
41 1234
41 1235
41 1236
41 1237
41 1238
41 1239
41 1240
41 1241
41 1242
41 1243
41 1244
41 1245
41 1246
41 1247
41 1248
41 1249
41 1250
41 1251
41 1252
41 1253
41 1254
41 1255
41 1256
41 1257
41 1258
41 1259
41 1260
41 1261
41 1262
41 1263
41 1264
41 1265
41 1266
41 1267
41 1268
41 1269
41 1270
41 1271
41 1272
41 1273
41 1274
41 1275
41 1276
41 1277
41 1278
41 1279
41 1280
41 1281
41 1282
41 1283
41 1284
41 1285
41 1286
41 1287
41 1288
41 1289
41 1290
41 1291
41 1292
41 1293
41 1294
41 1295
41 1296
41 1297
41 1298
41 1299
41 1300
41 1301
41 1302
41 1303
41 1304
41 1305
41 1306
41 1307
41 1308
41 1309
41 1310
41 1311
41 1312
41 1313
41 1314
41 1315
41 1316
41 1317
41 1318
41 1319
41 1320
41 1321
41 1322
41 1323
41 1324
41 1325
41 1326
41 1327
41 1328
41 1329
41 1330
41 1331
41 1332
41 1333
41 1334
41 1335
41 1336
41 1337
41 1338
41 1339
41 1340
41 1341
41 1342
41 1343
41 1344
41 1345
41 1346
41 1347
41 1348
41 1349
41 1350
41 1351
41 1352
41 1353
41 1354
41 1355
41 1356
41 1357
41 1358
41 1359
41 1360
41 1361
41 1362
41 1363
41 1364
41 1365
41 1366
41 1367
41 1368
41 1369
41 1370
41 1371
41 1372
41 1373
41 1374
41 1375
41 1376
41 1377
41 1378
41 1
```

Kode 5.9: File *Dockerfile* yang digunakan

```

1 # Menggunakan image PHP 7.3 sebagai base image
2 FROM php:7.3-apache
3
4 # Install dependensi dan ekstensi PHP yang dibutuhkan untuk CodeIgniter
5 RUN apt-get update && apt-get install -y \
6     libpng-dev \
7     libjpeg-dev \
8     libldap2-dev \
9     libcurl4 \
10    libcurl4-openssl-dev \
11    libzip-dev \
12    libfreetype6-dev \
13    zip \
14    unzip \
15    default-jdk \
16    g++ \
17    python2 \
18    python3
19
20 # Install ekstensi GD dan mysqli
21 RUN docker-php-ext-configure gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ \
22     && docker-php-ext-install gd mysqli
23
24 RUN docker-php-ext-install curl
25
26 RUN docker-php-ext-configure ldap --with-libdir=lib/x86_64-linux-gnu/ \
27     && docker-php-ext-install ldap
28
29 RUN docker-php-ext-install fileinfo
30 RUN docker-php-ext-install mbstring
31 RUN docker-php-ext-install zip
32
33 RUN cp /usr/local/etc/php/php.ini-production /usr/local/etc/php/php.ini && \
34     sed -i -e "s/^ *memory_limit.*memory_limit = 4G/g" /usr/local/etc/php/php.ini && \
35     sed -i -e "s/^ *max_input_vars.*max_input_vars = 3000000/g" /usr/local/etc/php/php.ini && \
36     sed -i -e "s/^ *post_max_size.*post_max_size = 50M/g" /usr/local/etc/php/php.ini && \
37     sed -i -e "s/^ *upload_max_filesize.*upload_max_filesize = 50M/g" /usr/local/etc/php/php.ini
38
39 # Aktifkan mod_rewrite untuk Apache
40 RUN a2enmod rewrite
41
42 # Copy kode CodeIgniter ke dalam container
43 COPY . /var/www/html/
44
45 # Set direktori kerja
46 WORKDIR /var/www/html/
47
48 # Make Folder tester writeable by PHP
49 RUN chmod 777 /var/www/html/restricted/tester
50 RUN chmod 777 /var/www/html/application/cache/Twig
51
52 # Expose port 80
53 EXPOSE 80
54
55 # Jalankan Apache server
56 CMD ["apache2-foreground"]

```

59 Tabel 5.1 menunjukkan spesifikasi perangkat keras yang digunakan saat pengujian.

Tabel 5.1: Perangkat Keras Lingkungan Pembangunan

Parameter	Nilai
<i>Processor</i>	<i>AMD EPYC 9354P 2 vCPU</i>
<i>Random Access Memory (RAM)</i>	8 GB
<i>Storage</i>	100 GB SSD

60 Tabel 5.2 menunjukkan spesifikasi perangkat lunak yang digunakan saat pengujian.

Tabel 5.2: Perangkat Lunak Lingkungan Pembangunan

Parameter	Nilai
Sistem Operasi	<i>Debian Version 12</i>
Bahasa Pemrograman	<i>PHP, JavaScript, CSS, dan HTML</i>
<i>Framework</i>	<i>CodeIgniter 3.1.13</i>
<i>Code Editor</i>	<i>Visual Studio Code 1.99.3</i>
Perangkat Lunak Pendukung	<i>Docker Version 20.10.24+dfsg1</i> <i>Debian 11-slim</i> <i>MySQL 5.7</i> <i>phpMyAdmin 5.2.1</i> <i>PHP 7.3.33</i>

### 5.3.2 Pengujian

Pengujian dilakukan untuk mengetahui permasalahan yang terjadi jika IDE dalam SharIF Judge dimasukkan sistem perekaman. Hasil Pengujian juga akan dianalisis secara sederhana agar dapat dibuatnya sebuah sistem untuk mendeteksi tindakan kecurangan.

Pengujian pada sistem pemutaran ulang dilakukan dengan cara mengajak beberapa peserta yang masih menempuh kuliah maupun yang sudah lulus dengan mengerjakan tiga permasalahan dengan tingkat kesusahan mudah, sedang, dan sulit dalam waktu tempuh satu hari. Para peserta dianjurkan mengerjakan 2 soal yang sudah disediakan dan dapat melihat *syntax* bahasa pemrograman. Pada pengujian peserta juga dapat melakukan kecurangan pada satu buah nomor dengan cara apapun. Kecurangan akan dianggap terjadi pada saat peserta melihat cara pengerjaan permasalahan dengan cara apapun.

Berikut merupakan masalah yang ditemukan pada saat pengujian sistem pemutaran ulang:

- Fitur menampilkan soal pada IDE

Fitur ini tidak dapat dilakukan karena permasalahan dengan *networking* dalam server VPS yang menjadikan tidak bisaanya mengakses PDF permasalahan dalam SharIF Judge dan menghasilkan *status 404*. Fitur ini diabaikan karena fitur ini tidak mengganggu pengujian yang sedang berjalan, tetapi untuk peserta melihat soal dengan cara mendownload soal dan melihatnya pada aplikasi lain.

- Fitur Menyimpan Rekaman pada Sistem

Fitur ini tidak mengubah database pada saat pengguna melakukan aksi *submit*. Pada saat aksi *submit* dilakukan file rekaman diubah menjadi file rekaman yang sudah di-*submit* tetapi pada daftar rekaman dalam database itu sendiri belum dihapus dan pada saat rekaman diambil, tidak ditemukannya rekaman tersebut dan mengembalikan error pada pengguna. Untuk menyelesaikan masalah ini, diperlukannya sebuah fungsi baru dalam *Model Recording\_model*. Kode 5.10 merupakan fungsi yang ditambahkan pada *Model Recording\_model*.

Kode 5.10: Fungsi tambahan pada *Recording model*

```

26
27 public function remove_saveonly_recording($assignment_id, $problem_id, $username) {
28     $this->db->delete('recording', array(
29         'assignment' => $assignment_id,
30         'problem'=>$problem_id,
31         'username'=>$username,
32         'rec_id'=>0)
33     );
34 }
```

1 Penambahan fungsi ini akan digunakan pada *Controller Recording* pada saat fungsi `_submit()`  
 2 dipanggil.

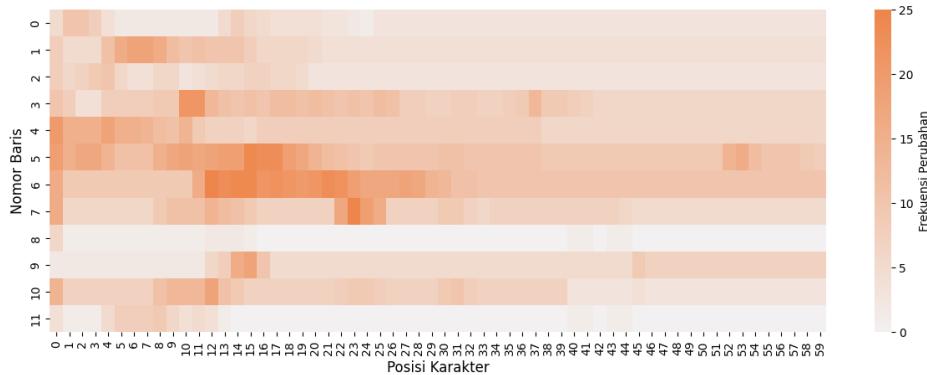
3 Setelah pengujian berakhir, peserta diminta untuk mengisi beberapa pertanyaan mengenai  
 4 permasalahan yang dikerjakan dan juga beberapa pertanyaan mengenai pengalaman mengerjakan  
 5 permasalahan pemrograman.

## 6 Analisa Hasil Pengujian

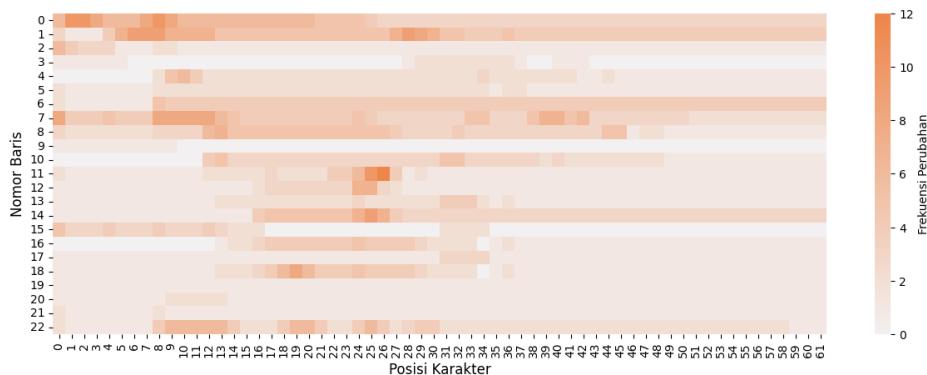
7 Analisa ini dilakukan dengan melihat data yang rekaman yang sudah di proses dan menjadi bagan  
 8 histogram dalam sistem dan bagan lainnya yang dibutuhkan pada proses analisa. Ada beberapa  
 9 pola yang dapat dilihat pada saat melihat histogram hasil rekaman peserta pengujian. Berikut  
 10 merupakan pola yang dapat dilihat pada:

- 11 • Pola Pembuatan Kode
- 12 • Pola *Debugging*

13 Pola ini dapat dilihat pada `events editor.change`, `editor.changeCursor`, `editor.changeSelection`,  
 14 `editor_input.input`, dan `editor_output.change`. Pengguna yang *debugging* akan mengubah kode pada lokasi yang sama. Gambar 5.1 menunjukkan bahwa frekuensi lebih  
 15 tinggi akan berwarna lebih merah dan menandakan pola debugging yang dapat dilihat karena  
 16 terjadinya banyak perubahan pada posisi yang sama dibandingkan dengan Gambar 5.2 yang  
 17 memiliki frekuensi maksimum 12 pada warna termerahnya.

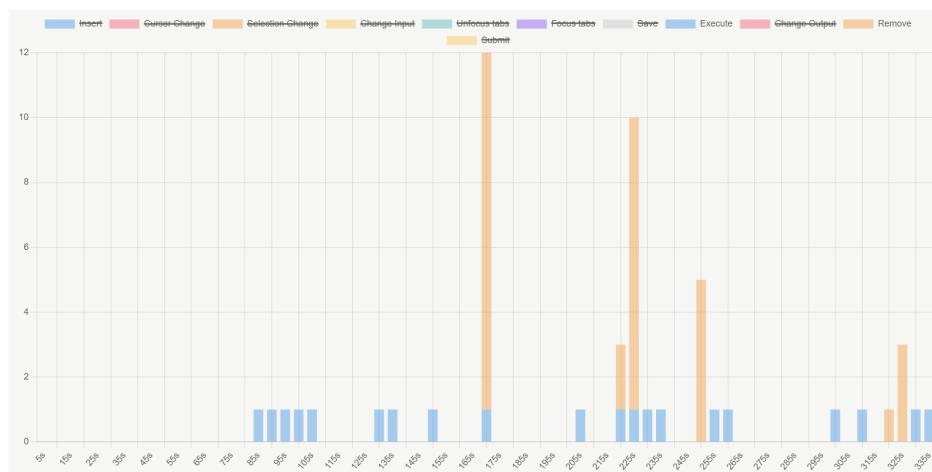


Gambar 5.1: Bagan Heatmap Perubahan Saat *Debugging*

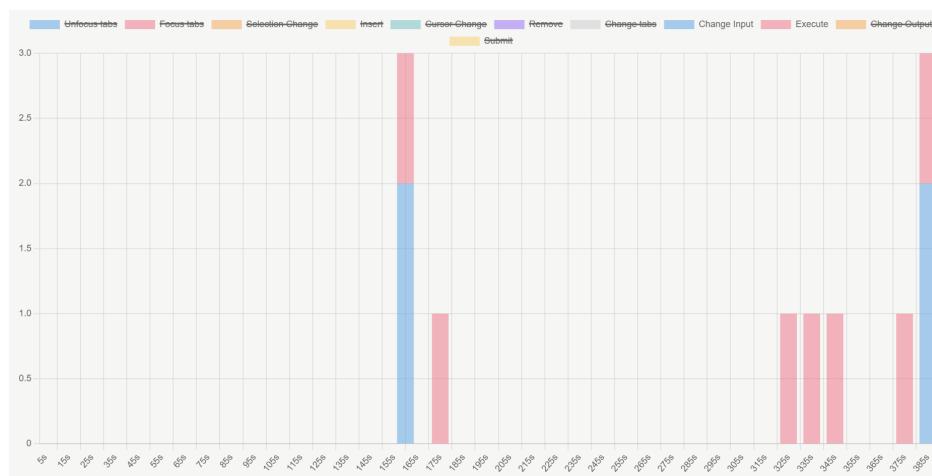


Gambar 5.2: Bagan Heatmap Perubahan Tanpa *Debugging*

Peserta yang *debugging* juga akan mencoba untuk mengubah input dan melakukan aksi *Execute* lebih banyak dibandingkan dengan peserta yang tidak melakukan *debugging*. Gambar 5.3 menunjukkan warna kuning sebagai perubahan input dan warna biru sebagai aksi *Execute* yang dilakukan sedangkan Gambar 5.4 menunjukkan warna biru sebagai perubahan input dan warna merah sebagai aksi *Execute* yang dilakukan. Perbedaan kedua bagan adalah frekuensi dimana aksi *Execute* dilakukan dan juga perubahan input. Maka pola *debugging* juga dapat dilihat melalui frekuensi aksi *Execute* dan juga frekuensi perubahan input.



Gambar 5.3: Bagan Histogram Perubahan Saat *Debugging*



Gambar 5.4: Bagan Histogram Perubahan Tanpa *Debugging*

- Pola Distraksi
- Pola Perubahan Navigasi
- Pola Berpikir
- Pola *Copy-Paste*
- Pola Perubahan Cursor dalam Editor Kode

## DAFTAR REFERENSI

- [1] 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.
- [2] Önder Demir, Aykut Soysal, Ahmet Arslan, Burcu Yürekli, dan Özgür Yilmazel (2010) Automatic grading system for programming homework. *Proceedings of the Annual International Conference on Computer Science Education: Innovation & Technology CSEIT 2010 & Proceedings of the Annual International Conference on Software Engineering SE 2010*, **18**, 68–75.
- [3] Kurnia, A., Lim, A., dan Cheang, B. (2001) Online judge. *Computers & Education*, **18**, 299–315.
- [4] IDCloudHost (2020) Algoritma pemrograman beserta contohnya. <https://idcloudhost.com/blog/algoritma-pemrograman-pengertian-fungsi-cara-kerja-dan-contohnya/>. 6 Desember 2024.
- [5] Vallian, S. (2018) Kustomisasi Sharif Judge Untuk Kebutuhan Program Studi Teknik Informatika. Skripsi. Universitas Katolik Parahyangan, Indonesia.
- [6] Version 1.4 (2014) *Sharif Judge Documentation*. Mohammad Javad Naderi. Tehran, Iran.
- [7] Halim, N. A. (2021) Implementasi Editor Kode pada SharIF Judge. Skripsi. Universitas Katolik Parahyangan, Indonesia.
- [8] Version (2025) *Chart.js Documentation*. Chart.js Contributors. Open Source Project.



# LAMPIRAN A

## KODE PROGRAM

Kode A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[i] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = (--aaa + &dcaa) / (bbb++ - ccc % 2 );
14             strcpy(a,"hello_@?");
15         }
16     count = ~mask | 0x00FF00AA;
17 }
18
19 // Fonts for Displaying Program Code in LATEX
20 // Adrian P. Robson, nepsweb.co.uk
21 // 8 October 2012
22 // http://nepsweb.co.uk/docs/progfonts.pdf
23

```

Kode A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                         //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35}
36

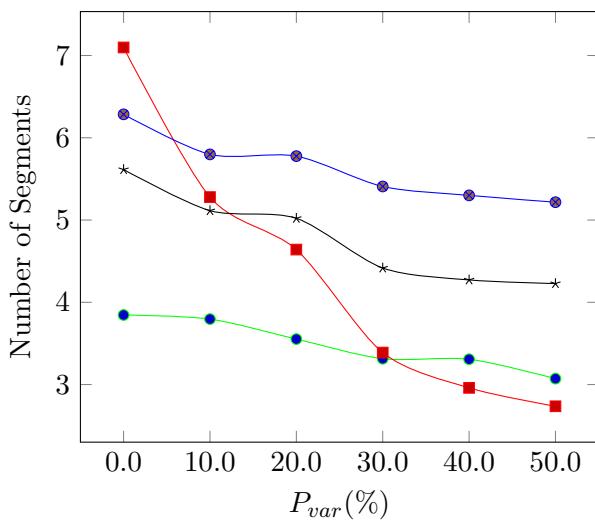
```



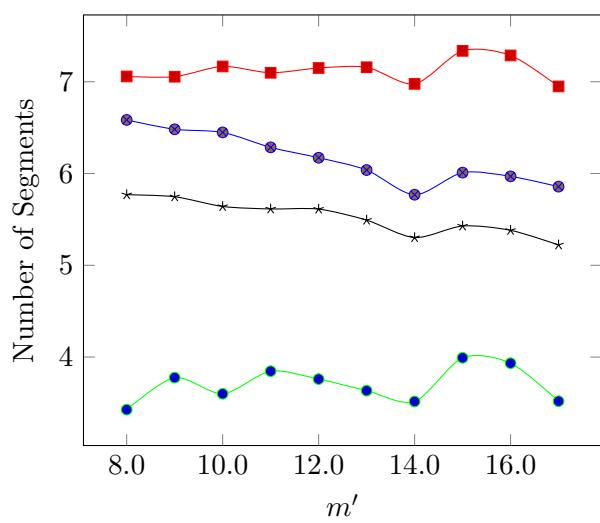
## LAMPIRAN B

### HASIL EKSPERIMENT

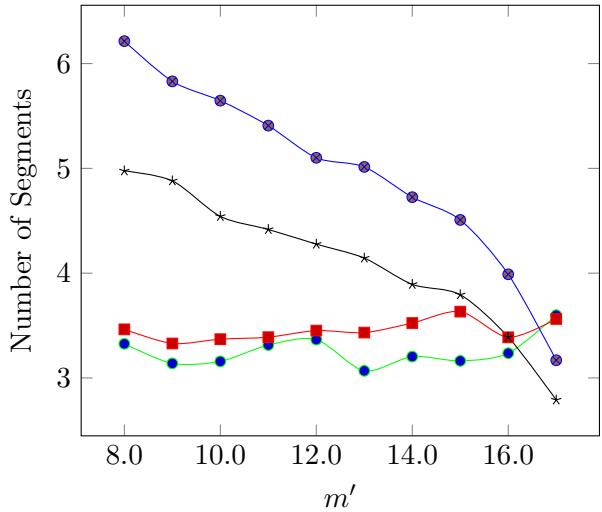
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



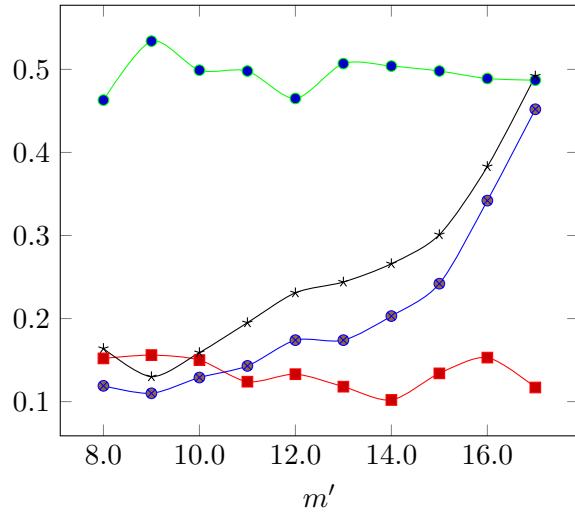
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4