

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

DAFTAR ISI	iii
DAFTAR GAMBAR	v
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 SharIF Judge	5
2.1.1 Instalasi	5
2.1.2 Users	6
2.2 CodeIgniter 3	6
2.2.1 Model-View-Controller	7
2.2.2 CodeIgniter URLs	9
2.3 Twig	9
2.4 Integrated Development Environment	10
2.5 Ace	10
3 ANALISIS	13
3.1 Analisis Sistem Kini	13
3.1.1 Model, View, Controller	13
3.1.2 Penyimpanan Kode Submission	35
3.1.3 Antrean Penilaian Kode	36
3.2 Analisis Sistem Usulan	36
3.2.1 Perekam event pada editor kode	36
3.2.2 Pemutar ulang event	37
DAFTAR REFERENSI	39
A KODE PROGRAM	41
B HASIL EKSPERIMEN	43

DAFTAR GAMBAR

1.1	Tampilan Awal SharIF Judge	1
1.2	Tampilan editor kode pada SharIF-Judge	2
2.1	<i>Flow Chart</i> CodeIgniter	6
3.1	Halaman Assignments	20
3.2	Halaman Dashboard	21
3.3	Halaman Hall of Fame	22
3.4	Halaman Install	22
3.5	Halaman Login	23
3.6	Halaman 24-Hour Log	24
3.7	Halaman Moss	25
3.8	Halaman Notifications	26
3.9	Halaman Problems	27
3.10	Halaman Profile	28
3.11	Halaman Queue	29
3.12	Halaman Rejudge	30
3.13	Halaman Scoreboard	30
3.14	Halaman Settings	31
3.15	Halaman Final Submissions	32
3.16	Halaman All Submissions	33
3.17	Halaman Submit	34
3.18	Halaman Users	35
B.1	Hasil 1	43
B.2	Hasil 2	43
B.3	Hasil 3	43
B.4	Hasil 4	43

1

BAB 1

2

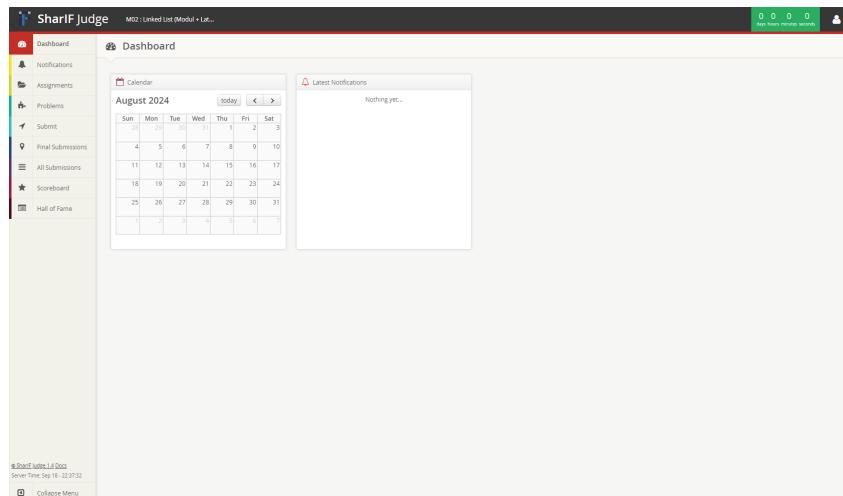
PENDAHULUAN

3 1.1 Latar Belakang

4 Ujian merupakan sebuah alat bantu untuk menilai pemahaman pelajar tentang ilmu yang diberikan
5 oleh pengajar. Salah satu ujian yang diberikan kepada pelajar informatika adalah ujian koding
6 yang biasanya dinilai berdasarkan ketepatan algoritma yang dipakai. Tetapi melakukan penilaian
7 untuk setiap kode merupakan sebuah hal yang sulit untuk dilakukan karena dibutuhkannya waktu
8 yang lama. Maka dari itu website *judge* dibuat untuk memudahkan pekerjaan pengajar.

9 *Judge* merupakan sebuah website yang akan menilai sebuah kode dengan menjalankannya
10 berdasarkan masukkan yang ditentukan dan menyamakan keluaran dari kode dengan keluaran yang
11 sudah ditetapkan oleh pembuat soal dalam kurun waktu yang ditetapkan. Oleh karena itu, kode
12 yang dibuat harus dapat mencakupi waktu yang diberikan dengan menggunakan algoritma yang
13 tepat. Bukan hanya menilai dengan keluaran yang tetap tetapi *judge* juga dapat menggunakan kode
14 yang sudah dibuat oleh pengajar dan membandingkannya dengan keluaran kode yang di kumpulkan.

15 Sudah banyak perguruan tinggi informatika yang menggunakan website *judge* dalam pemeriksaan
16 kode pelajar termasuk perguruan UNPAR untuk penilai kode dari para mahasiswanya. Judge yang
17 digunakan adalah SharIF-Judge [1] yang dimodifikasi oleh Stillmen Vallian terhadap Sharif-Judge [2]
18 buatan Mohammad Javad Naderi dengan *framework* CodeIgniter dan Bash. Gambar 1.1 merupakan
19 halaman utama setelah masuk ke dalam website SharIF-Judge.



Gambar 1.1: Tampilan Awal SharIF Judge



Gambar 1.2: Tampilan editor kode pada SharIF-Judge

1 Tugas akhir ini merupakan sebuah pengembangan lanjutan dari tugas akhir yang bertopik
 2 “Implementasi editor kode pada Sharif Judge” [3] oleh Nicholas Aditya Halim. Tugas akhir tersebut
 3 menceritakan bahwa SharIF-Judge tidak memiliki kemampuan untuk mengawasi proses pembuatan
 4 kode program karena para mahasiswa menggunakan aplikasi eksternal untuk pembuatan kode
 5 program tersebut. Sehingga dibuatnya modifikasi terhadap SharIF-Judge untuk menambahkan
 6 *Intergated Development Enviroment* (IDE), sebuah aplikasi untuk mengedit, mengompilasi, dan
 7 menjalankan kode program pada SharIF-Judge dengan editor kode bernama Ace [4]. Gambar 1.2
 8 merupakan tampilan editor kode yang sudah diimplementasikan pada SharIF-Judge.

9 Tetapi SharIF Judge masih tidak memiliki kemampuan untuk mengawasi proses pembuatan
 10 kode program pada aplikasi eksternal maupun IDE dalam SharIF Judge. Maka dari itu tugas akhir
 11 ini menambahkan fitur pada SharIF Judge dengan merekam ketikan pada IDE yang tersedia di
 12 SharIF-Judge untuk membantu pengawasan dengan merekam dan memutar ulang ketikan mahasiswa.
 13 Tugas akhir ini akan membuat pengawasan terhadap kegiatan kuliah lebih mudah untuk pengawas
 14 dan dapat menjadi bukti kecurangan jika dibutuhkan.

15 1.2 Rumusan Masalah

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

- 17 1. Bagaimana mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada
 18 IDE SharIF-Judge?
- 19 2. Bagaimana cara menyimpan data pemutaran ulang mahasiswa secara rutin dengan otomatis
 20 dan tidak mengambil penyimpanan *database* sangat besar?
- 21 3. Bagaimana tanggapan pengguna terhadap implementasi perekaman dan pemutaran ulang
 22 kode ketikan pada SharIF Judge?

23 1.3 Tujuan

24 Tujuan yang ingin dicapai skripsi ini adalah sebagai berikut:

- 25 1. Mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada IDE SharIF-
 26 Judge.
- 27 2. Mencari cara penyimpanan data efektif dan mengimplementasikannya pada perekaman dan
 28 pemutaran ulang ketikan.
- 29 3. Mendapatkan umpan balik dari tanggapan pengguna terhadap perekaman dan pemutaran
 30 ulang ketikan mahasiswa pada SharIF-Judge.

1 1.4 Batasan Masalah

- 2 Pada pengerojaan tugas akhir ini terhadap batasan sebagai berikut:
- 3 • Perangkat lunak SharIF Judge hanya digunakan pada lingkungan Teknik Informatika Unpar.
- 4 • Perangkat lunak hanya dapat diuji pada mata kuliah pemrograman di mana dosen pembimbing
- 5 terlibat.

6 1.5 Metodologi

- 7 Metodologi pengerojaan tugas akhir ini adalah sebagai berikut:
- 8 1. Melakukan studi mengenai komponen yang diperlukan untuk membuat sistem perekaman
- 9 dan pemutaran ulang ketikan pada IDE berbasis web.
- 10 2. Merancang sistem perekaman dan pemutaran ulang ketikan berbasis web untuk SharIF Judge
- 11 3. Mengimplementasikan IDE berbasis web pada SharIF Judge.
- 12 4. Melakukan pengujian dan eksperimen.
- 13 5. Menulis dokumen tugas akhir.

14 1.6 Sistematika Pembahasan

- 15 Sistematika pembahasan skripsi ini adalah sebagai berikut:

- 16 • **Bab 1:** Pendahuluan
Membahas latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
- 17 • **Bab 2:** Landasan Teori
Membahas teori-teori yang berhubungan dengan penelitian ini, yaitu SharIF Judge, CodeIgniter 3, Twig, IDE, dan Ace.
- 18 • **Bab 3:** Analisis
Membahas analisis terhadap perangkat lunak SharIF Judge dan IDE pada SharIF Judge.
- 19 • **Bab 4:** Perancangan
Membahas perancangan fitur yang diimplementasikan pada SharIF Judge.
- 20 • **Bab 5:** Implementasi dan Pengujian
Membahas implementasi fitur pada SharIF Judge dan pengjuian yang dilakukan.
- 21 • **Bab 6:** Kesimpulan dan Saran
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 [1]. 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

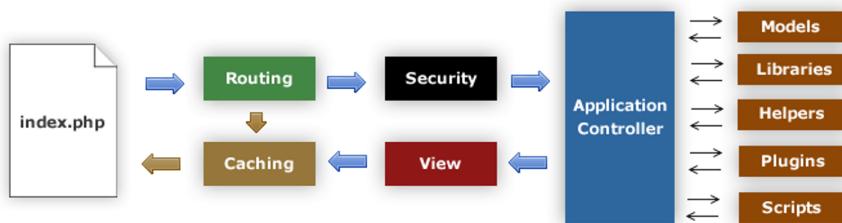
- Pada SharIF Judge, pengguna dibagi menjadi 4 buah *role*. Role yang tersedia adalah sebagai berikut:
1. *admin*
 2. *head instructor*
 3. *instructor*
 4. *student*
- Setiap *role* memiliki akses pada aksi yang berbeda berdasarkan *role*-nya. Tabel 2.1 merupakan aksi-aksi yang dapat dilakukan untuk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

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

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

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

- 7 *Syntax* 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 *Syntax* 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     }
}
```

```

18
29     public function comments()
30     {
31         echo 'Look_at_this!';
32     }
33 }
```

- 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:

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

- 12 URL diatas akan mengembalikan sebuah teks ‘Hello World!’.

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

```

13
14 class Blog_controller extends CI_Controller {
15
16     public function blog()
17     {
18         $this->load->model('blog');
19
20         $data['query'] = $this->blog->get_last_ten_entries();
21
22         $this->load->view('blog', $data);
23     }
24 }
25 }
```

- 27 Pada CodeIgniter, *model* dan *view* hanya dapat dimuat melalui controller. Seperti contoh, Kode
 28 2.4 akan memuat *model* `blog` dan mengambil data dari *database*, lalu menampilkan *view* yang
 29 memuat data tersebut.

30 2.2.2 CodeIgniter URLs

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

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

- 35 Struktur URL pada CodeIgniter juga mengikuti pendekatan MVC (referensi 2.2.1) dan biasanya
 36 memiliki struktur sebagai berikut:

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

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

42 2.3 Twig

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

- 1 • Pewarisan *Template*
- 2 • Struktur Kontrol (mengunaan kondisional, *looping*)
- 3 • Filter
- 4 • Variable pada PHP

5 Pada saat template dievaluasi, semua *variable* atau *expression* akan dibuat menjadi value dan
6 *tag* yang mengontrol logika template.

Kode 2.5: Contoh template Twig

```
7 1  {% extends "base.html" %}  
8 2  {% block navigation %}  
9 3      <ul id="navigation">  
10 4          {% for item in navigation %}  
11 5              <li>  
12 6                  <a href="{{item.href}}>  
13 7                      {% if item.level == 2 %}&nbsp;&nbsp;{% endif %}  
14 8                          {{ item.caption|upper }}  
15 9                      </a>  
16 10            </li>  
17 11        {% endfor %}  
18 12    </ul>  
19 13  {% endblock navigation %}
```

22 Kode 2.5 merupakan contoh sebuah template Twig. Terdapat dua jenis *delimiter*, yaitu `{% ... %}`
23 dan `{{ ... }}`. *Delimiter* `{% ... %}` digunakan untuk Menjalankan sebuah *statement* seperti *for-loops*, sedangkan *delimiter* `{{ ... }}` digunakan untuk mengubah sebuah *variable* atau *expression*
24 menjadi nilai sesungguhnya.
25

26 2.4 Integrated Development Environment

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

- 30 • *Editor*
31 Editor teks sebagai tempat untuk mengetik kode, dapat dilengkapi dengan berbagai fitur
32 seperti *syntax highlighting* (menampilkan teks dengan warna yang berbeda untuk menginkatkan
33 keterbacaan kode) dan *word completion* (menampilkan prediksi kata yang sedang atau yang
34 akan diketik pengguna).
- 35 • *Complier*
36 Digunakan untuk menterjemahkan kode program yang dibuat pada editor teks ke dalam
37 sebuah program yang dapat dijalankan oleh komputer.
- 38 • *Execution*
39 Menjalankan kode program yang sudah dikompilasi, dengan input jika dibutuhkan, dan
40 mengembalikan hasilnya.

41 2.5 Ace

42 Ace merupakan sebuah editor kode yang dapat dimasukkan ke dalam sebuah website yang dibuat
43 menggunakan bahasa *Javasciprt*. Ace memiliki kemampuan dari editor pada umumnya. Berikut
44 merupakan beberapa fitur utama yang dimiliki oleh Ace:

- 45 • *Syntax highlighting* untuk bahasa pemrograman.

- 1 • Automatic indent dan outdent.
- 2 • Kemampuan *cut*, *copy*, dan *paste*.
- 3 • Kemampuan *drag and drop* teks menggunakan mouse.
- 4 • Banyak *Cursors* dan *selections*
- 5 • *Line wrapping*
- 6 • *Code folding*

7 Beberapa kelas penting yang terdapat pada library Ace adalah sebagai berikut:

8 • **Ace**

9 Merupakan kelas utama untuk menyiapkan editor kode Ace pada *browser*

10 • **Anchor**

11 Menangani posisi *pointer* pada dokumen.

12 • **Document**

13 Menyimpan teks dokumen.

14 • **Editor**

15 Entri utama untuk fungsionalitas library Ace.

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/src-noconflict/ace.js" type="text/javascript" charset="utf-8"></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.

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 bab 2.2.1. Berikut merupakan analisis MVC pada SharIF Judge:

10 Model

11 Analisis MVC akan dimulai dengan *model* yang berada pada direktori `application/models`. Di-
12 rektori *Model* berisi kelas-kelas yang digunakan untuk mengelola dan mengembalikan data dari
13 *database*. Berikut merupakan file *model* dan penjelasan fungsi-fungsinya yang terdapat pada SharIF
14 Judge:

- 15 • `Assignment_model.php`

16 Model ini digunakan untuk mengelola tabel `assignments` dan mengembalikan informasi yang
17 digunakan dalam halaman *assignment* dan *problem*. Fungsi yang dimiliki adalah sebagai
18 berikut:

19 – `add_assignment($id, $edit)`

20 Menambahkan atau memperbarui sebuah *assignment*.

21 – `delete_assignment($assignment_id)`

22 Menghapus sebuah *assignment*.

23 – `all_assignments()`

24 Mengembalikan daftar semua *assignment* dan informasinya.

25 – `new_assignment_id()`

26 Mendapatkan nomor terkecil dan dapat digunakan sebagai *id assignment* terbaru.

27 – `all_problems($assignment_id)`

28 Mengembalikan daftar semua *problems* dari sebuah *assignment*.

29 – `problem_info($assignment_id, $problem_id)`

30 Mengembalikan semua informasi sebuah *problem*

```
1   – assignment_info($assignment_id)
2     Mengembalikan semua informasi sebuah assignment
3   – is_participant($participants, $username)
4     Mengembalikan sebuah boolean yang menyatakan bahwa $username terdapat dalam
5     $participants.
6   – increase_total_submits($assignment_id)
7     Menambahkan jumlah total submits sebanyak satu pada sebuah assignment.
8   – set_moss_time($assignment_id)
9     Memperbarui “Moss Update Time” pada sebuah assignment.
10  – get_moss_time($assignment_id)
11    Mengembalikan “Moss Update Time” pada sebuah assignment.
12  – save_problem_description($assignment_id, $problem_id, $text, $type)
13    Menambahkan atau memperbarui deskripsi pada sebuah problem.
14  – _update_coefficients($a_id, $extra_time, $finish_time, $new_late_rule)
15    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:

```
19  – get_all_final_submission()
20    Mengembalikan seluruh total nilai final submission untuk semua user.
21  – get_all_user_assignments($username)
22    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:

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

- **Notifications_model.php**

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

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

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

1 Model ini digunakan untuk mengelola tabel **settings**. Fungsi yang dimiliki adalah sebagai
2 berikut:

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

11 • **Submit_model.php**

12 Model ini digunakan untuk mengelola tabel **submission**. Fungsi yang dimiliki adalah sebagai
13 berikut:

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

30 • **User.php**

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

- 33 – **select_assignment(\$assignment_id)**
34 Menyimpan *assignment* yang dipilih oleh *user*.
- 35 – **save_widget_positions(\$positions)**
36 Menyimpan posisi *widget* sebuah *user*.
- 37 – **get_widget_positions()**
38 Mendapatkan posisi *widget* sebuah *user*.

39 • **User_model.php**

40 Model ini digunakan untuk mengelola tabel **users**. Fungsi yang dimiliki adalah sebagai
41 berikut:

- 42 – **have_user(\$username)**

```

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

```

37 View

38 View merupakan tampilan yang menjadi perantara antara pengguna dan sistem. Pada SharIF Judge,
 39 View disimpan pada direktori application/views dan dibagi menjadi 3 direktori terpisah yaitu:
 40 • errors
 41 Pada direktori errors, berisi tampilan halaman error jika terjadi error pada penggunaan SharIF

1 Judge. Berikut merupakan *view* yang terdapat pada direktori **errors**:

2 – **error_404**
3 – **error_db**
4 – **error_expectation**
5 – **error_general**
6 – **error_php**

7 • **pages**

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

11 – **pages/admin**

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

14 * **add_assignment.twig**
15 * **add_notification.twig**
16 * **add_user.twig**
17 * **add_user_result.twig**
18 * **delete_assignment.twig**
19 * **edit_problem_html.twig**
20 * **edit_problem_md.twig**
21 * **edit_problem_plain.twig**
22 * **install.twig**
23 * **logs.twig**
24 * **moss.twig**
25 * **queue.twig**
26 * **rejudge.twig**
27 * **settings.twig**
28 * **users.twig**

29 – **pages/authentication**

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

32 * **login.twig**
33 * **lost.twig**
34 * **register.twig**
35 * **register_success.twig**
36 * **reset_password.twig**
37 – **assignments.twig**
38 – **dashboard.twig**
39 – **halloffame.twig**
40 – **notification.twig**
41 – **problems.twig**
42 – **profile.twig**

```

1   - scoreboard.twig
2   - scoreboard_tabel.twig
3   - submissions.twig
4   - submit.twig

```

- 5 • templates

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

```

9   - base.twig
10  - side_bar.twig
11  - simple_header.twig
12  - top_bar.twig

```

13 **Controller**

14 Pada bagian analisis MVC terakhir, terdapat *controller* yang berada pada direktori
15 *application/controller*. Seperti yang dijelaskan pada bab 2.2.1, *Controller* digunakan sebagai
16 perantara antara *model*, *view*, dan *resources* lainnya yang dibutuhkan saat membuat sebuah web
17 page. Direktori controller berisi kelas-kelas yang akan mengolah data yang didapat pada *model*
18 dan menyatukan data tersebut ke dalam *views* yang akan ditampilkan kepada pengguna. Pada
19 setiap kelas *controller*, terdapat fungsi *index()* yang menjadi fungsi utama saat kelas di akses oleh
20 pengguna. Berikut merupakan file *controller* dan penjelasan fungsi-fungsinya yang terdapat pada
21 SharIF Judge:

- 22 • *Assignments.php*

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

```

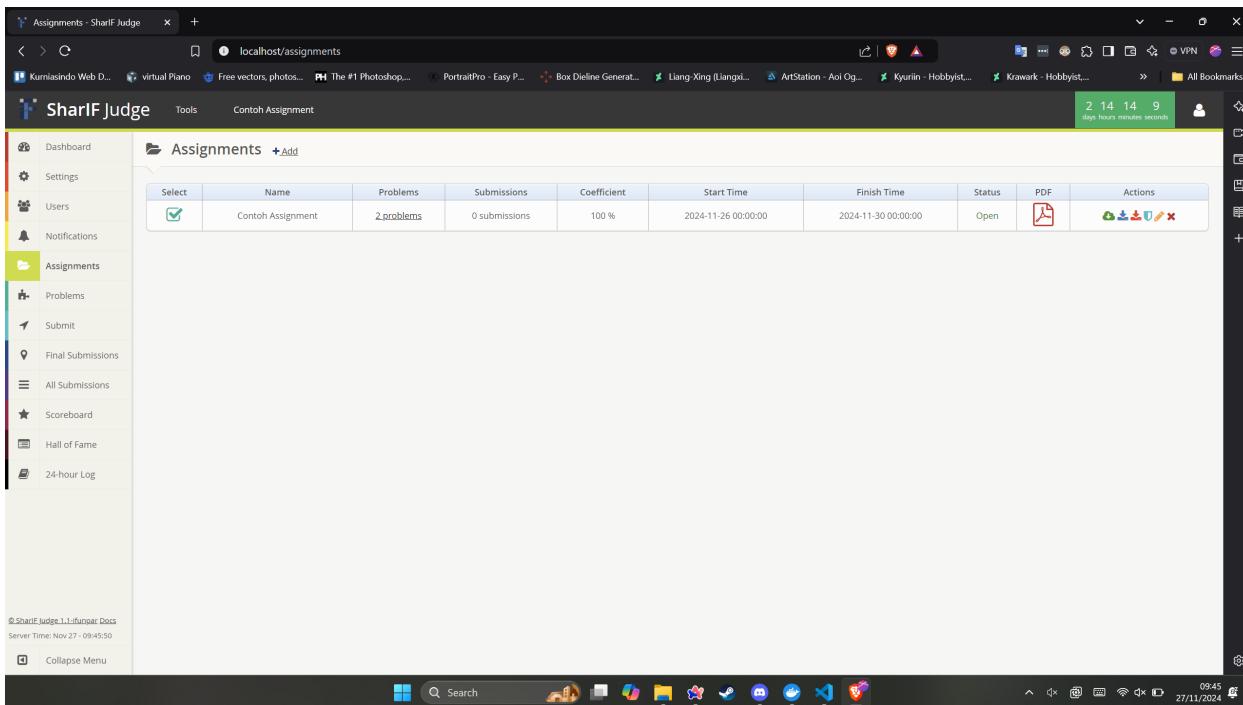
24   - select()
25     Memilih assignment yang ditampilkan pada top bar menggunakan ajax request.
26   - pdf($assignment_id, $problem_id, $no_download)
27     Mengunduh assignment atau problem dalam bentuk pdf file ke browser.
28   - downloadtestsdesc($assignment_id)
29     Mengunduh dan mencompress data uji dan deskripsi sebuah assignment.
30   - download_submissions($type, $assignment_id)
31     Mengunduh semua final submission pada semua assignment.
32   - delete($assignment_id)
33     Menghapus sebuah assignment.
34   - add()
35     Mendapatkan input dari pengguna untuk menambah atau memperbarui sebuah assig-
36     nment.
37   - _add()
38     Menambahkan atau memperbarui sebuah assignment.
39   - edit($assignment_id)
40     Menandai assignment yang akan di edit dan memanggil fungsi add.
41   - pdfCheck($assignment_id, $problem_id)

```

1 Melakukan validasi ketersediaan pdf pada sebuah *assignment* atau pada sebuah *problem*.

2 – **index()**

3 Mengambil data dari **Assignment_model** dan menaruh data dan mengembalikan *views assignments.twig*. Gambar 3.1 menunjukkan hasil halaman Assignment.



Gambar 3.1: Halaman Assignments

5 • **Dashboard.php**

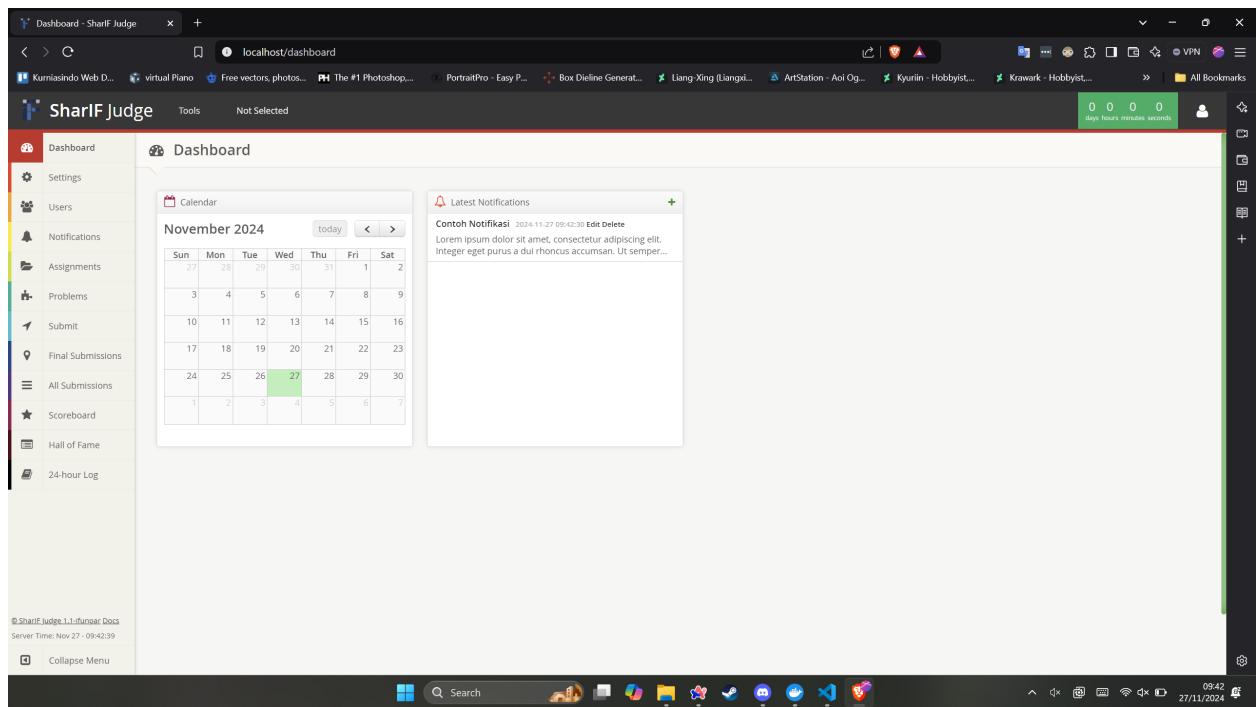
6 Berikut fungsi dengan penjelasannya pada *controller Dashboard.php*:

7 – **widget_positions()**

8 Menggunakan *ajax request* untuk menyimpan posisi *widget*.

9 – **index()**

10 Mendapatkan data dari beberapa model yaitu **Assignment_model**, **Settings_model**,
11 **User**, dan **Notifications_model**. Data akan dimasukkan ke dalam **dashboard.twig**
12 yang akan dikembalikan ke pengguna. Gambar 3.2 menunjukkan hasil halaman Dashboard
13 yang dapat diakses oleh semua *role*.



Gambar 3.2: Halaman Dashboard

1 • **Halloffame.php**

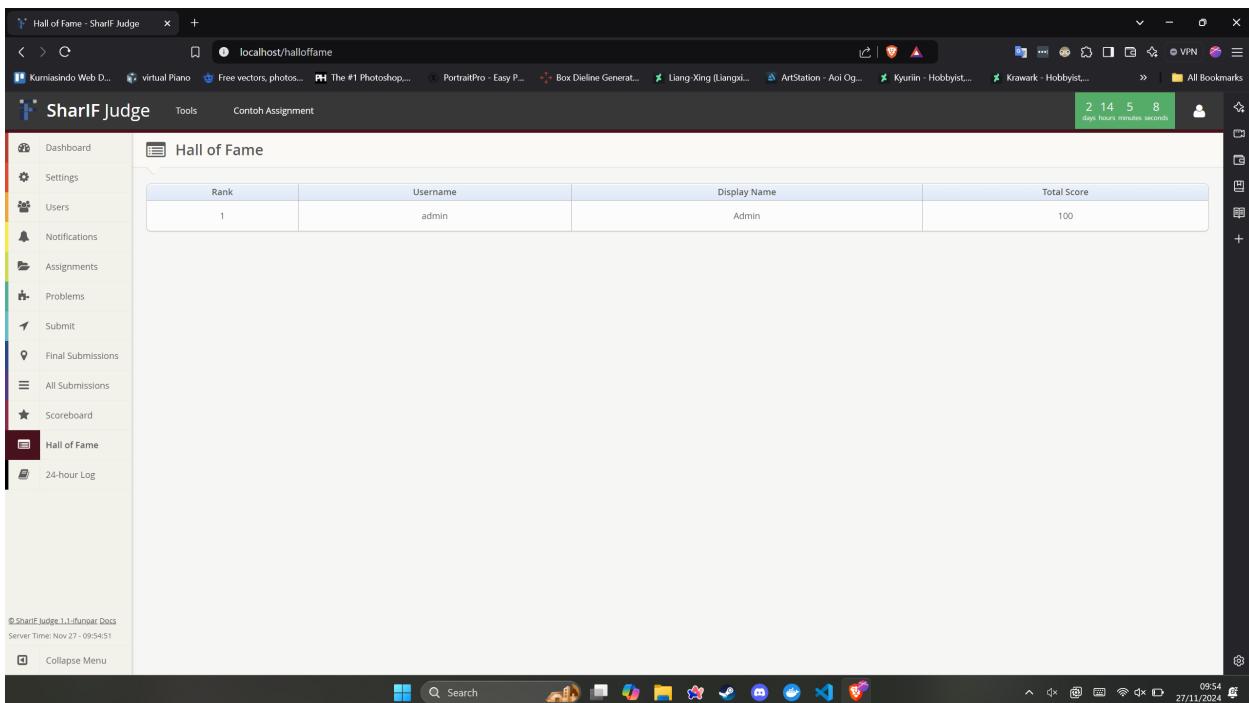
2 Berikut fungsi dengan penjelasannya pada *controller Halloffame.php*:

3 – **hof_details()**

4 Menampilkan nilai akhir semua *problem* dan *assignments* pada sebuah *user*.

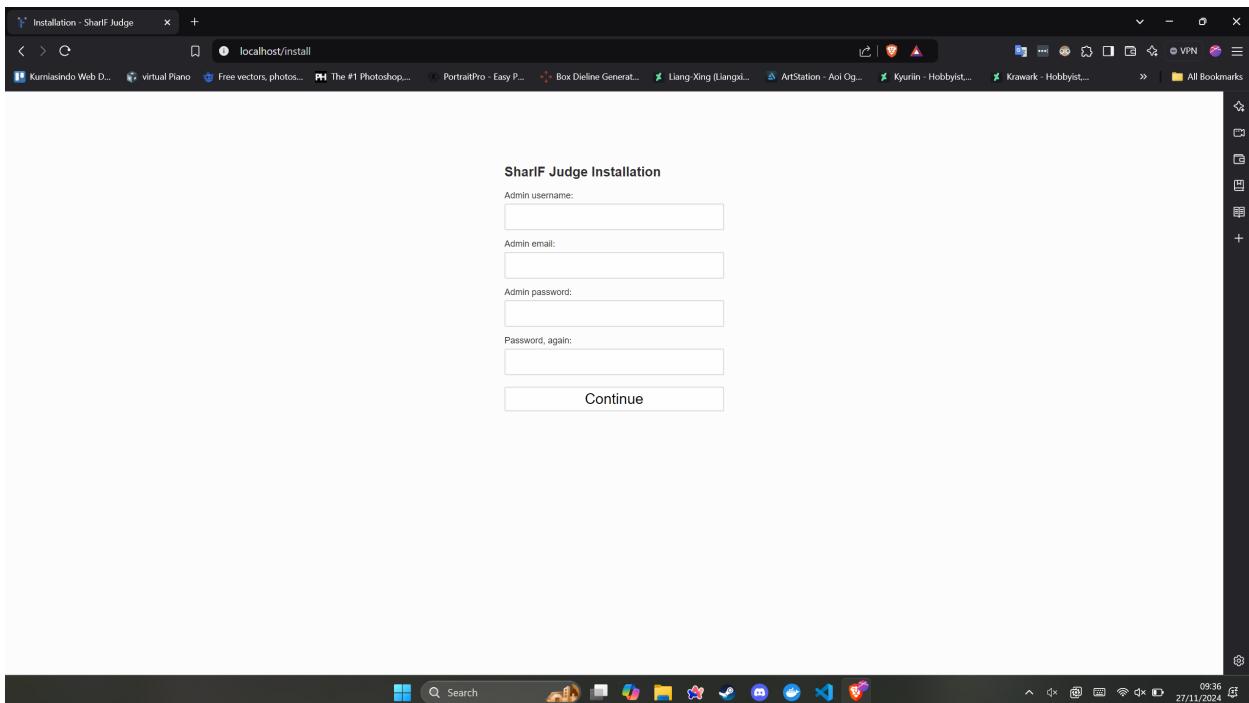
5 – **index()**

6 Mendapatkan data dari *Hof_model* dan mengembalikan *view halloffame.twig*. Gambar
7 3.3 menunjukkan hasil halaman Hall of Fame yang dapat diakses oleh semua *role*.



Gambar 3.3: Halaman Hall of Fame

- **Install.php**
- Pada *controller Install.php* hanya ada satu fungsi yang menangani pembuatan seluruh tabel pada *database* yang dibutuhkan oleh SharIF Judge. Setelah membuat *database* akan mengembalikan *view install.twig* yang dapat diisi oleh pengguna tentang data *user* dengan role *admin* saat *form* di kirim. Gambar 3.4 menunjukkan hasil halaman Install.



Gambar 3.4: 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()**

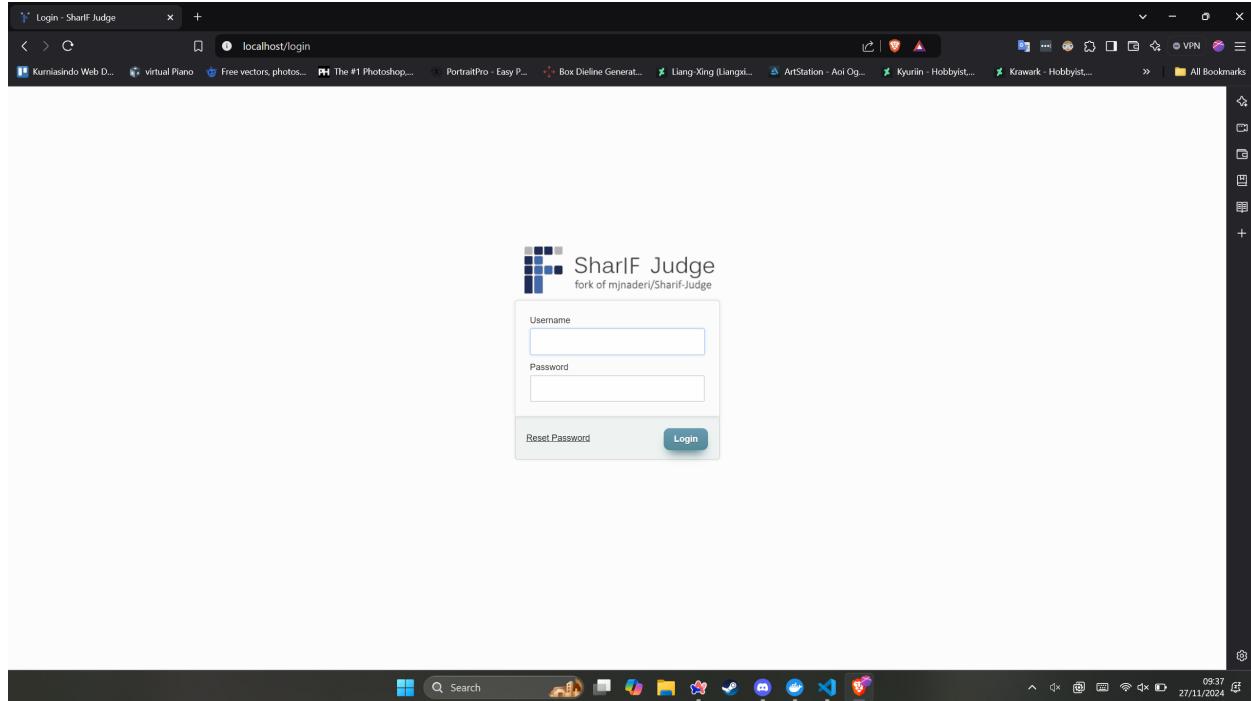
10 Mengirimkan email *reset password*.

11 – **reset(\$passchange_key)**

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

13 – **index()**

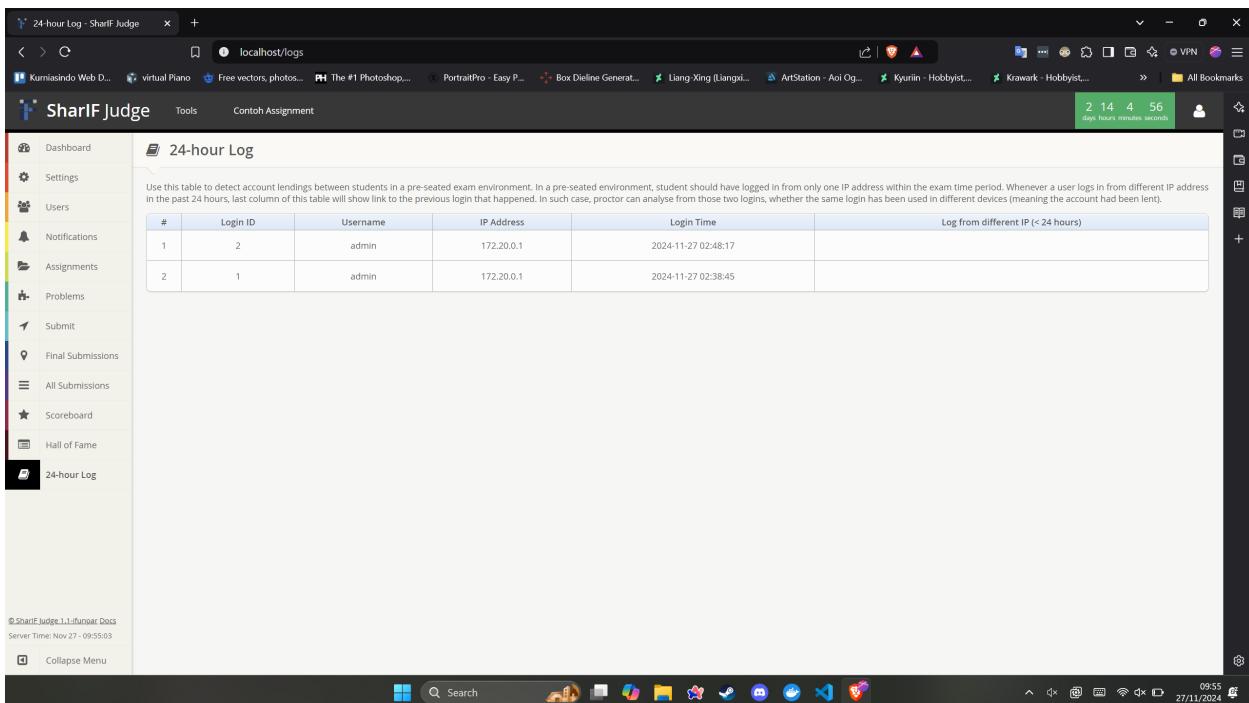
14 Mengembalikan *view login.twig* dan memeriksa username dan password pada *form* saat di kirim. Gambar 3.5 menunjukkan hasil halaman Login.



Gambar 3.5: 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 3.6 menunjukkan halaman Log yang dinamakan 24-Hour Log.



Gambar 3.6: Halaman 24-Hour Log

1 • **Moss.php**

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

3 – `update($assignment_id)`

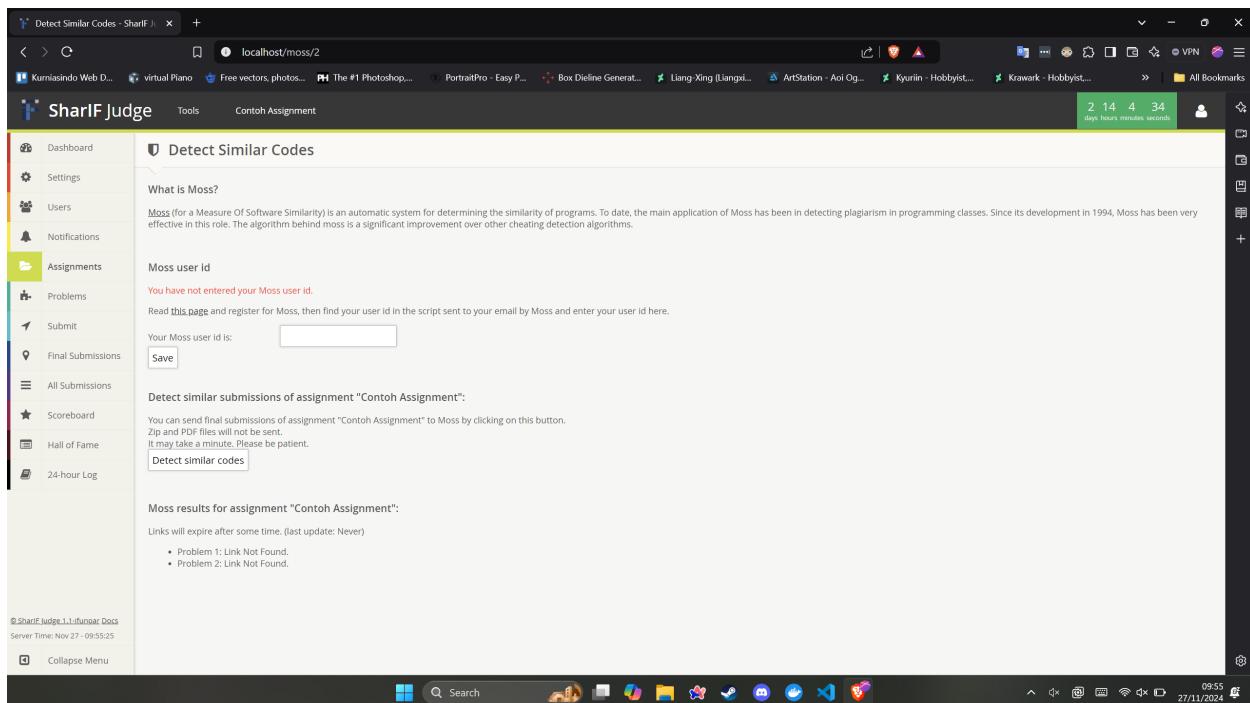
4 Memperbaharui *settings* dari masukkan `moss_userid` pengguna.

5 – `_detect($assignment_id)`

6 Melakukan pemeriksaan kesamaan kode dengan Moss.

7 – `index()`

8 Mengambil data dan memasukkannya ke dalam *view moss.twig*. Gambar 3.7 merupakan hasil halaman moss. Fungsi `_detect` juga akan dijalankan saat *form* terkirim.



Gambar 3.7: Halaman Moss

1 • **Notifications.php**

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

3 – **add()**

4 Menambahkan atau memperbaharui sebuah *notification*.

5 – **edit(\$notif_id)**

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

7 – **delete()**

8 Menghapus sebuah *notification*.

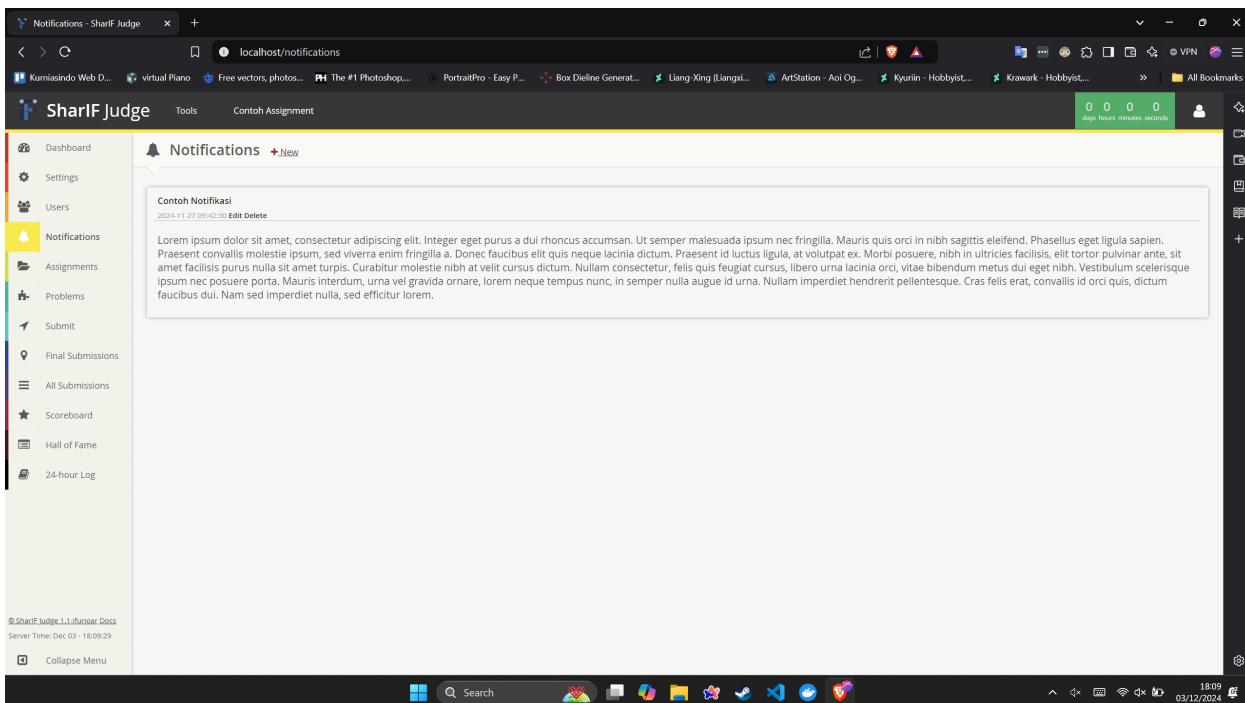
9 – **check()**

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 3.8 menunjukkan hasil halaman *Notifications*.



Gambar 3.8: Halaman Notifications

1 • **Problems.php**

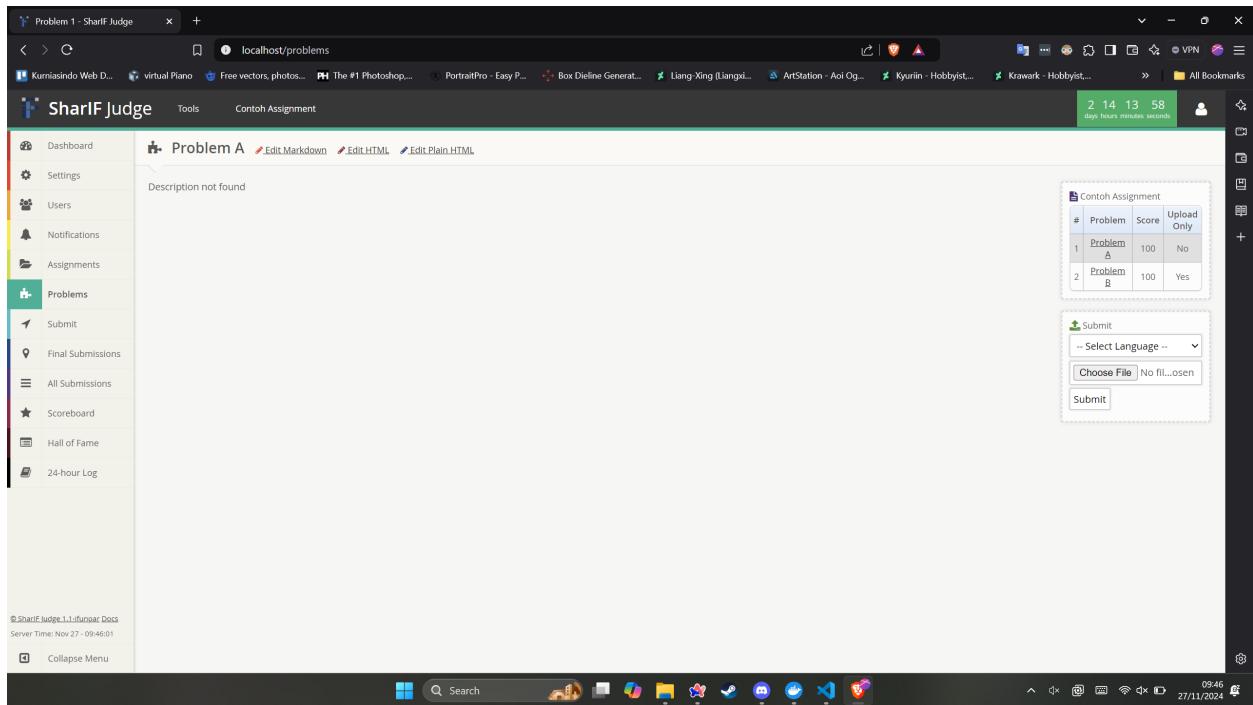
2 Berikut fungsi dengan penjelasannya pada *controller Notifications.php*:

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 3.9 menunjukkan hasil halaman Problems.



Gambar 3.9: 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)`

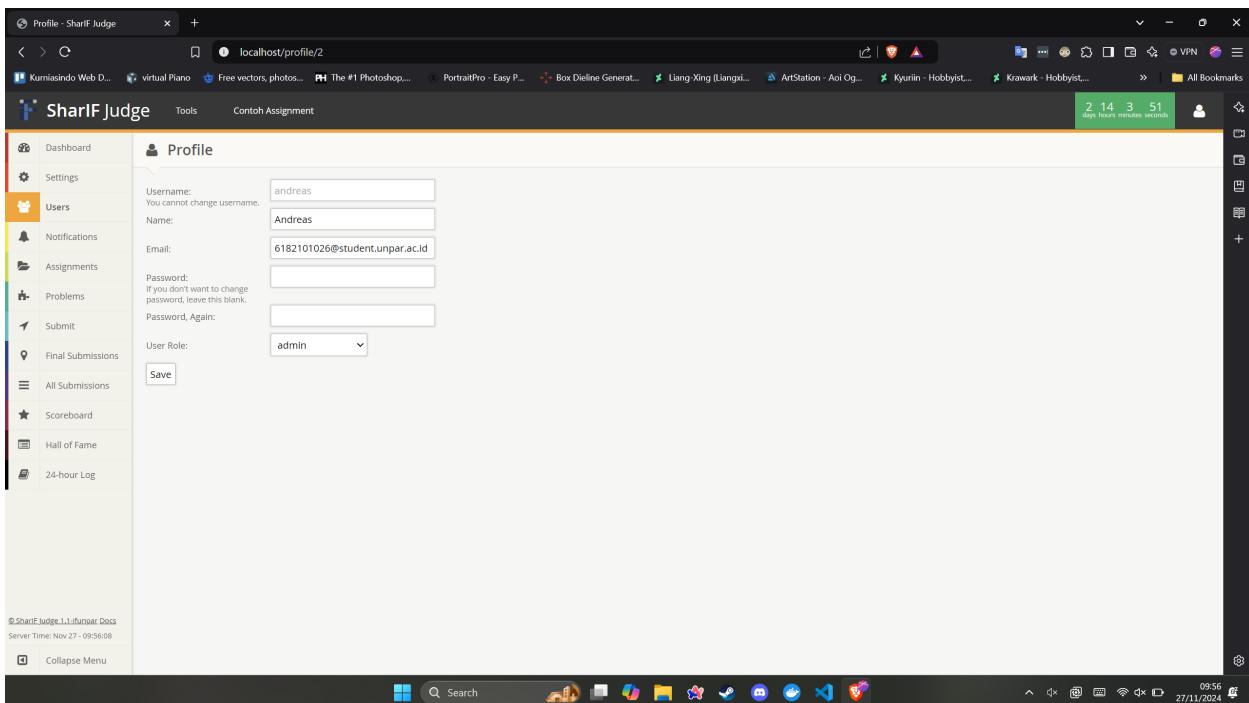
8 Melakukan validasi ketersediaan email pada *database*.

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 3.10 menunjukkan hasil halaman Profile.



Gambar 3.10: Halaman Profile

1 • **Queue.php**

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

3 – **pause()**

4 Memberhentikan proses *queue*.

5 – **resume()**

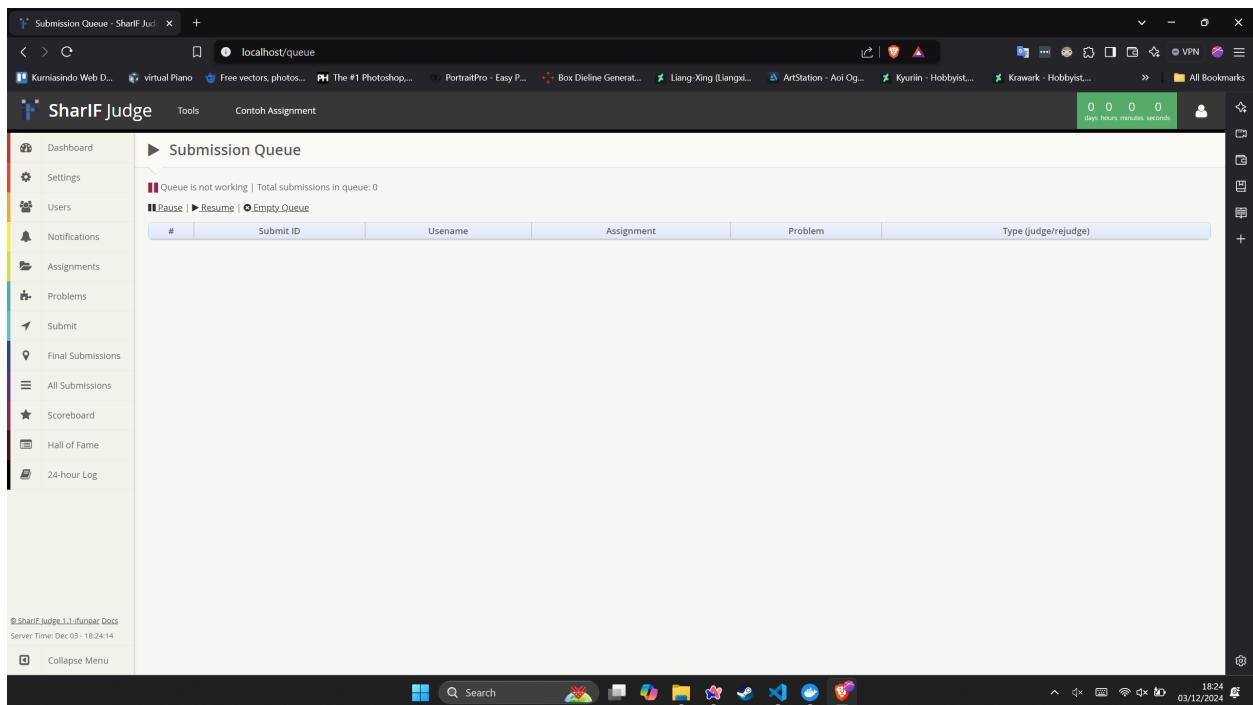
6 Melanjutkan proses *queue*.

7 – **empty_queue()**

8 Menghapus semua *queue* yang ada.

9 – **index()**

10 Mendapatkan data dari *model Queue*, *Assignments_model*, dan *Settings_model* yang
11 dipakai dalam *view queue.twig* dan ditampilkan kepada pengguna. Gambar 3.11 me-
12 nunjukkan hasil halaman Queue.



Gambar 3.11: 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**

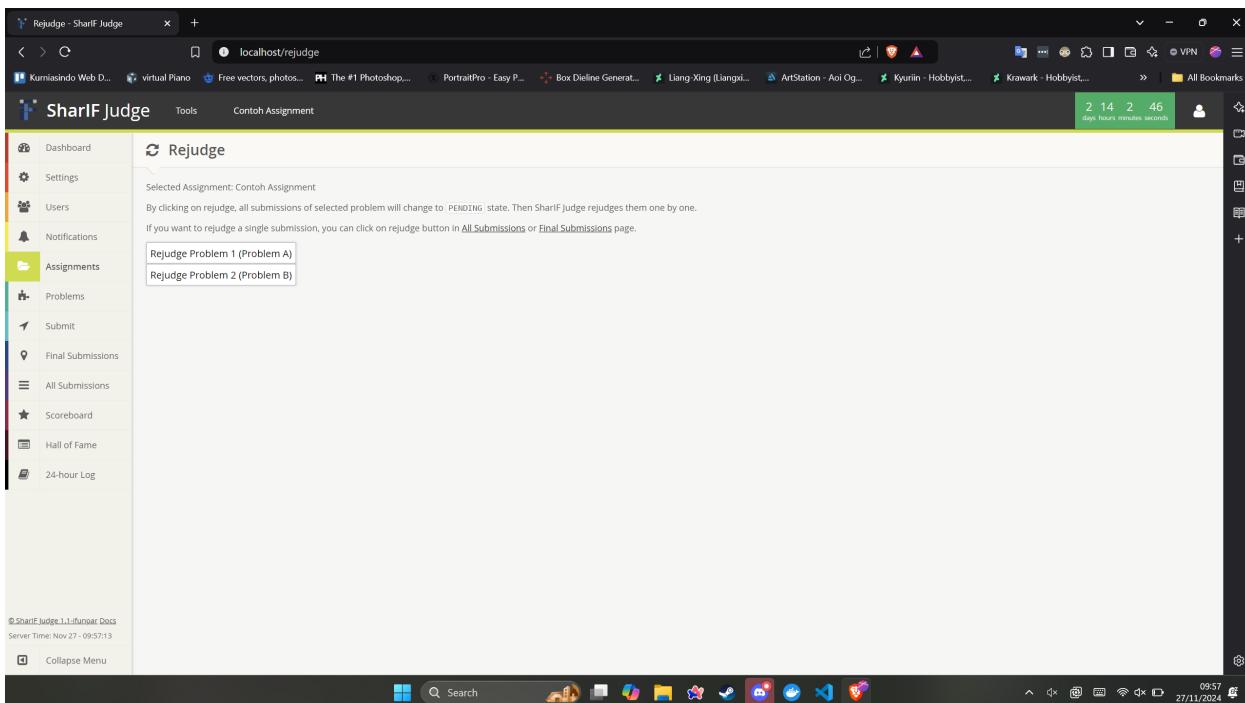
5 Berikut fungsi dengan penjelasannya pada controller Profile.php:

6 – `rejudge_single()`

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

8 – `index()`

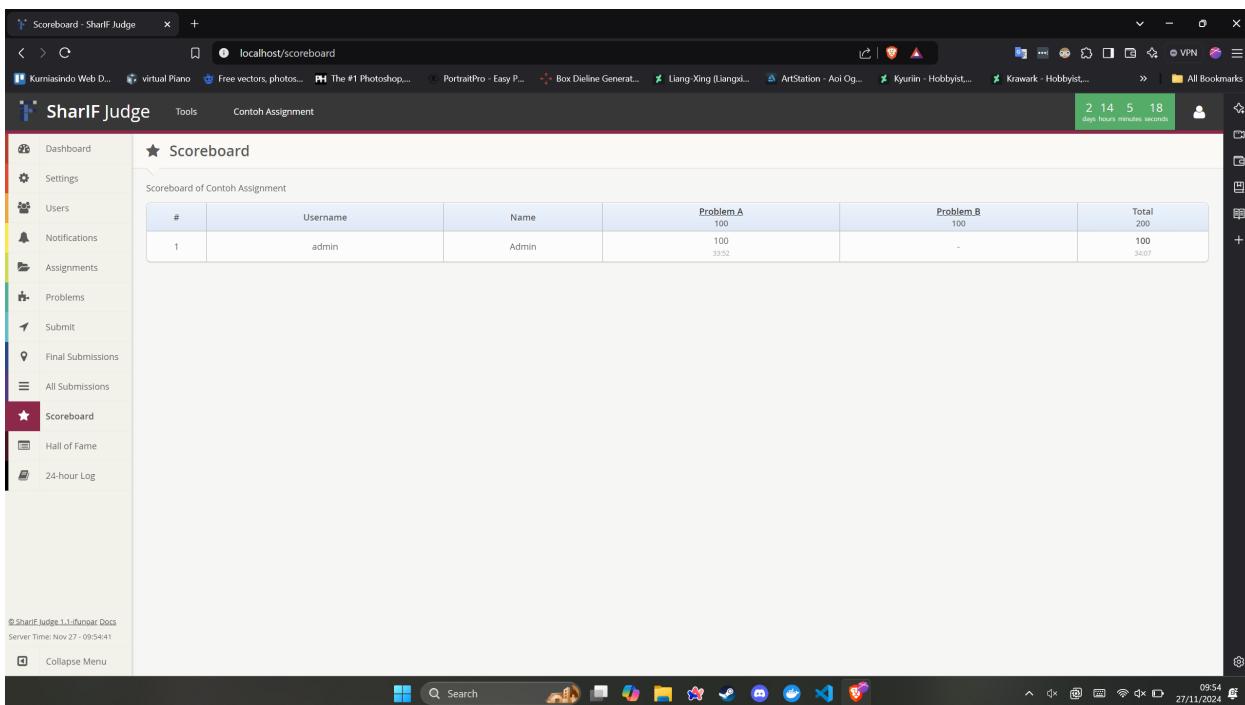
9 Mendapatkan data dan menampilkan *view rejuge.twig*. Fungsi ini juga dapat me-
10 lakukan *rejudge* pada sebuah *problem* tertentu. Gambar 3.12 menunjukkan halaman
11 Rejudge.



Gambar 3.12: 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 3.13 menunjukkan hasil halaman Scoreboard.



Gambar 3.13: Halaman Scoreboard

5 • **Server_time.php**

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

3 • **Settings.php**

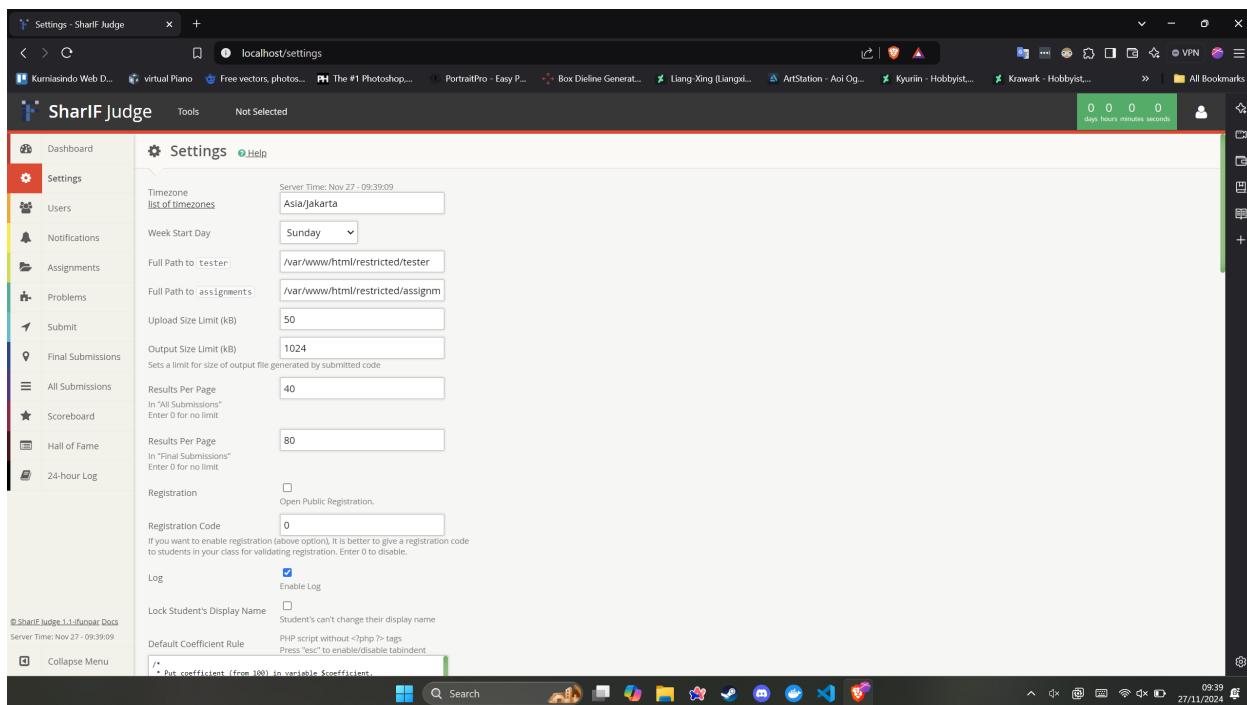
4 Berikut fungsi dengan penjelasannya pada controller `Settings.php`:

5 – `update()`

6 Memperbarui *settings* dari masukkan pengguna.

7 – `index()`

8 Mendapatkan data dari `Settings_model` dan menampilkan *view settings.twig*. Jika
 9 terdapat *error setting* pada sistem, akan ditampilkan juga pada *view* tersebut. Gambar
 10 3.14 menunjukkan hasil halaman Users.



Gambar 3.14: Halaman Settings

11 • **Submissions.php**

12 Berikut fungsi dengan penjelasannya pada controller `Submissions.php`:

13 – `_download_excel($view)`

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

16 – `final_excel()`

17 Menggunakan fungsi `_download_excel` untuk mendownload *final submission*.

18 – `all_excel()`

19 Menggunakan fungsi `_download_excel` untuk mendownload seluruh *submission*.

20 – `select()`

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

23 – `_check_type($type)`

- 1 Melakukan validasi tipe *submission* yang dikumpulkan.
- 2 – **view_code()**
- 3 Digunakan untuk melihat kode, melihat hasil kode, atau melihat *log* sebuah *submission*.
- 4 – **download_file()**
- 5 Mengunduh *file* kode sebuah *submission*.
- 6 – **the_final()**
- 7 Mendapatkan data dari `Submit_model` untuk mendapatkan *final submission* dan menam-
- 8 pilkan halaman `submission.twig` berisi *final submission*. Gambar 3.15 menunjukkan
- 9 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	100	Code	Log	...	
2	4	admin	Admin	2	2024-11-27 09:54:16	0	No Delay 100%	0	Java	Uploaded	Code	Log	...	

Gambar 3.15: 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 3.16 menun-
- 13 jukkan halaman All Submissions .

Final	ID	Username	Name	Problem	Submit Time	Score			Language	Status	Code	Log	Actions
						Score	Delay %	Final Score					
✓	4	admin	Admin	2	2024-11-27 09:54:16	0	No Delay 100%	0	java	Uploaded	Code	...	Log
✓	3	admin	Admin	1	2024-11-27 09:52:35	100	No Delay 100%	100	java	100	Code	Log	Log
○	2	admin	Admin	1	2024-11-27 09:51:48	100	No Delay 100%	100	C++	100	Code	Log	Log
○	1	admin	Admin	1	2024-11-27 09:51:33	0	No Delay 100%	0	C++	0	Code	Log	Log

Gambar 3.16: 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* untuk dinilai jika bukan *upload only problem*.

13 – `load($problem_id)`

14 Mendapatkan isi file dan menaruh isi file ke editor kode.

15 – `save($type)`

16 Menyimpan isi editor kode ke dalam *server* dan menjalankan atau mengumpulkan jika diinginkan.

17 – `_submit($data, $problem_id, $language, $user_dir)`

18 Menambahkan kode ke dalam *submission* untuk dinilai.

19 – `_execute($data, $problem_id, $language, $user_dir)`

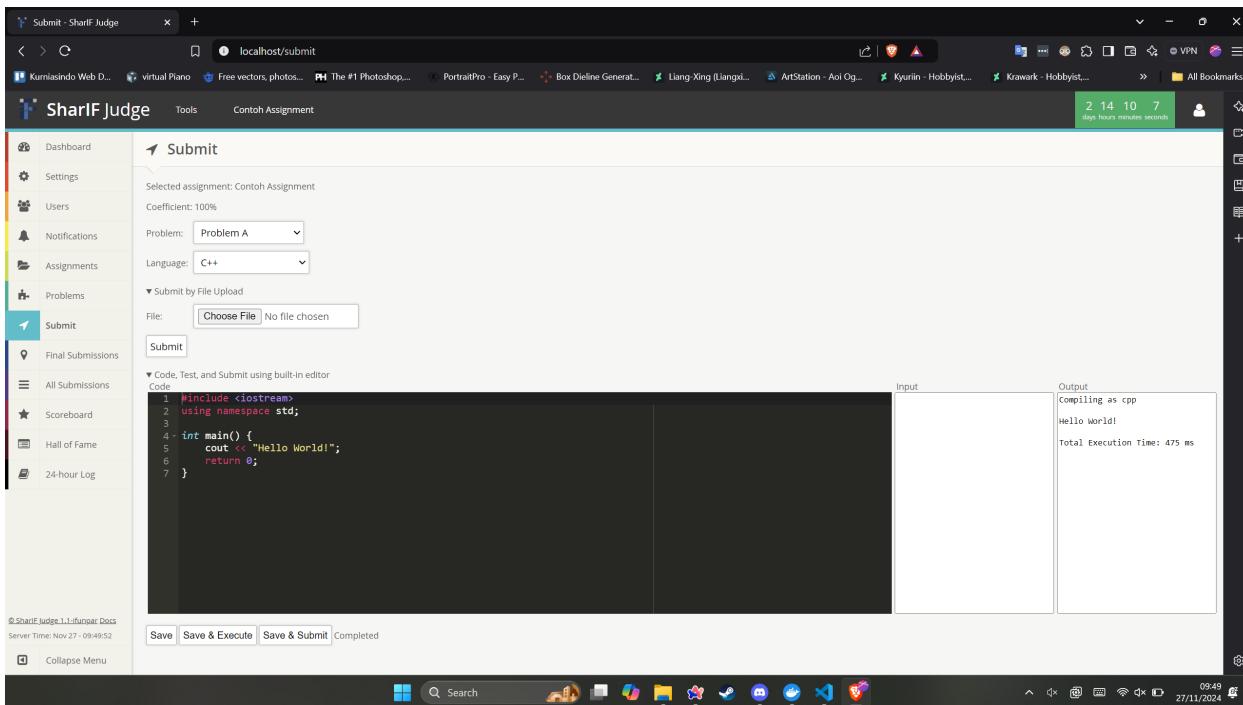
20 Menambahkan kode ke dalam *queue* untuk di jalankan saja.

21 – `get_output($problem_id)`

22 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 3.17
 4 menunjukkan hasil halaman Submit. Halaman ini terdapat editor kode yang sudah di
 5 implementasikan [3].



Gambar 3.17: 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 3.18
 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*.

#	User ID	Username	Display Name	Email	Role	First Login	Last Login	Actions
1	1	admin	Admin	admin@unpar.ac.id	admin	2024-11-27 09:38:45	2024-11-27 09:38:45	
2	2	andreas	Andreas	6182101026@student.unpar.ac.id	admin	Never	Never	

Gambar 3.18: Halaman Users

1 3.1.2 Penyimpanan Kode Submission

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

```
4     assignment_<a_id>/p_<p_id>/<nama user>/<nama file>-<s_id>.<file ext>
```

5 Penjelasan untuk format di atas adalah sebagai berikut:

- 6 • <a_id>
7 id pada *assignment*.
- 8 • <p_id>
9 id pada *problem*.
- 10 • <nama user>
11 Nama dari pengguna yang mengumpulkan kode/file.
- 12 • <nama file>
13 Nama file yang dikumpulkan, *editor* jika mengumpulkan menggunakan editor kode.
- 14 • <s_id>
15 id pada *submission*.
- 16 • <file ext>
17 Extensi file kode yang dikumpulkan.

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

```
22         assignment_5/p_1/kenzhi/probA-6.java
```

1 3.1.3 Antrean Penilaian Kode

- 2 Pada SharIF Judge, Kode yang dikumpulkan akan di jalankan satu per satu pada antrean meng-
3 gunakan `bash`. Berikut merupakan cara SharIF Judge menilai kode dari awal pengumpulan pada
4 sistem:
- 5 1. *Controller Submit* akan menyimpan file pada folder sesuai pada [3.1.2](#).
6 2. *Controller Submit* akan memasukkan data *submission* kedalam *model Queue_model*.
7 3. *Model Queue_model* akan menyimpan data *submission* pada *database submission* dan me-
8 nambahkan data *queue*.
9 4. Selanjutnya *Controller Submit* akan memanggil fungsi *process_the_queue()* yang akan
10 menjalankan fungsi *run()* pada *controller Queueprocess*.
11 5. *Controller Queueprocess* akan menjalankan *tester.sh* pada folder *tester* dengan data dari
12 *queue*.
13 6. *tester.sh* akan menilai kode yang akan dibaca oleh *controller Queueprocess* yang akan
14 menyimpan hasil penilaian.
15 7. Terakhir *Queueprocess* akan menyimpan hasil penilaian pada *database submission* dan
16 menghapus data *queue* menggunakan *Queue_model*.

17 3.2 Analisis Sistem Usulan

- 18 Pembuatan sistem pemutaran ulang ketikan membutuhkan 2 fitur penting yaitu perekaman ketikan
19 pada sebuah masukkan dan pemutaran ketikan pada masukkan tersebut. Masukkan yang dimaksud
20 pada tugas akhir ini adalah IDE SharIF Judge. IDE SharIF Judge memiliki berbagai masukkan
21 penting yaitu editor kode, *input*, dan *output*. Berikut merupakan fitur-fitur yang akan ditambahkan
22 pada SharIF Judge untuk membangun sistem perekaman ketikan.

23 3.2.1 Perekam event pada editor kode

- 24 Sebelum ketikan dapat di putar kembali, dibutuhkannya fitur untuk merekam segala *event* yang
25 terjadi pada masukkan tersebut. Berikut merupakan masukkan yang akan ditanggap oleh sistem:

- 26 • Editor Kode

27 Editor Kode pada SharIF Judge dibuat dengan *framework Ace*. Pada *framework Ace* sudah
28 menyediakan fitur untuk penangkapan *event* seperti yang sudah dijelaskan pada subbab [2.5](#).
29 Berikut *event* pada *Ace* yang akan ditanggap oleh sistem:

- 30 – `editor.commands.on("afterExec")`
31 *afterExec* akan merekam semua *command* yang dijalankan pada editor kode seperti
32 *paste*, *copy*, *insert* dan banyak lagi.
- 33 – `editor.on("mouseup")`
34 *mouseup* akan merekam *event* saat dilepaskannya tekanan pada *mouse*. Pada *event* ini
35 akan dipakai untuk merekam posisi *cursor* dan *selection*.
- 36 – `editor.selection.on("beforeEndOperation")`
37 *beforeEndOperation* akan menangkap *event* yang sama seperti *mouseup* yaitu merekam
38 posisi *cursor* dan *selection* saat setelah operasi command selesai dijalankan.

1 • **Input**

2 Perubahan *input* pada IDE SharIF Judge akan di tanggap dan dicatat.

3 • **Execute**

4 Pada IDE SharIF Judge terdapat fungsi untuk menjalankan file dan mendapatkan keluaran
5 program dari *input*. Maka *event* ini akan ditanggap dan juga keluarannya.

6 *Event* yang sudah direkam akan disimpan bersama dengan waktu saat di terjadinya *event*
7 tersebut. Penyimpanan rekaman juga akan disimpan pada folder yang sama dengan penyimpanan
8 kode submission seperti yang dijelaskan pada subbab 3.1.2 dengan nama file **recording** dan
9 menggunakan format file *JavaScript Object Notation* atau JSON. Pemimpanan perekaman akan
10 memiliki format sebagai berikut:

11 <timeline>: {event: <event>, data: <payload>}

12 <timeline> akan menunjukkan pada milidetik berapa *event* terjadi dengan menggunakan fungsi
13 **Date.now()** pada *Javascript*. sedangkan <event> dan <payload> merupakan data *event* yang
14 terjadi. <payload> akan disesuaikan dengan *event* yang terjadi. Sebagai contoh untuk *event insert*
15 huruf ‘a’ akan di tuliskan menjadi sebagai berikut:

16 1733335637486: {event: "insert", data: "a"}

17 Penyimpanan *event* akan di lakukan oleh *Controller Submit* yang akan menyimpan *file* pada
18 saat kode disimpan. Maka fungsi yang akan dimodifikasi adalah fungsi **save()**.

19 **3.2.2 Pemutar ulang event**

20 Fitur kedua yang akan ditambahkan adalah pemutaran ulang *event* pada SharIF Judge. Fitur ini
21 membutuhkan sebuah halaman baru yang akan dinamakan **Recording**. Maka dari itu dibutuh-
22 kannya beberapa hal baru yaitu *Controller Recording.php*, *View recording.twig* dan *Javascript*
23 **shj_recording.js**.

24 *Controller Recording.php* akan memiliki beberapa fungsi yaitu:

25 • **index()**

26 fungsi ini akan mengambil list recording sesuai dengan *assignment problem* yang dipilih dan
27 mengembalikan *view recording.twig*

28 • **load_recording()**

29 fungsi ini akan digunakan untuk mengambil file JSON yang sudah di simpan dan mengembalikan-
30 kannya.

31 *Javasciprt* yang akan dibuat pada **shj_recording.js** akan memiliki berbagai fungsi yaitu
32 sebagai berikut:

33 • Fungsi untuk memilih *user* dan *problem* yang ditampilkan.

34 • Fungsi ajax untuk meminta *file recording* sesuai dengan pilihan.

35 • Fungsi untuk menjalankan, memberhentikan *recording* dari *file JSON*.

DAFTAR REFERENSI

- [1] Version 1.4 (2014) *Sharif Judge Documentation*. Mohammad Javad Naderi. Tehran, Iran.
- [2] Vallian, S. (2018) Kustomisasi Sharif Judge Untuk Kebutuhan Program Studi Teknik Informatika. Skripsi. Universitas Katolik Parahyangan, Indonesia.
- [3] Halim, N. A. (2021) Implementasi Editor Kode pada SharIF Judge. Skripsi. Universitas Katolik Parahyangan, Indonesia.
- [4] Version 1.4.13 (2021) *Ace API Reference*. Ajax.org B.V. Amsterdam, The Netherlands.

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

```

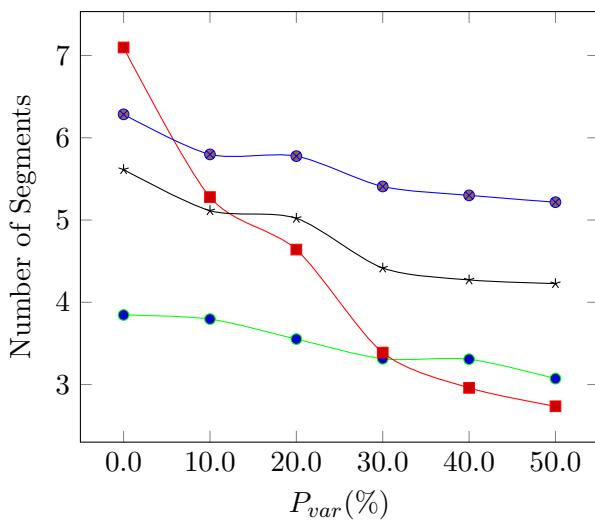
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the 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

```

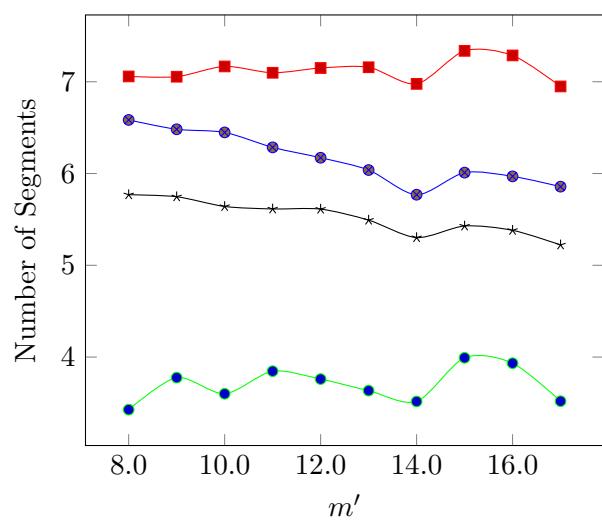

LAMPIRAN B

HASIL EKSPERIMENT

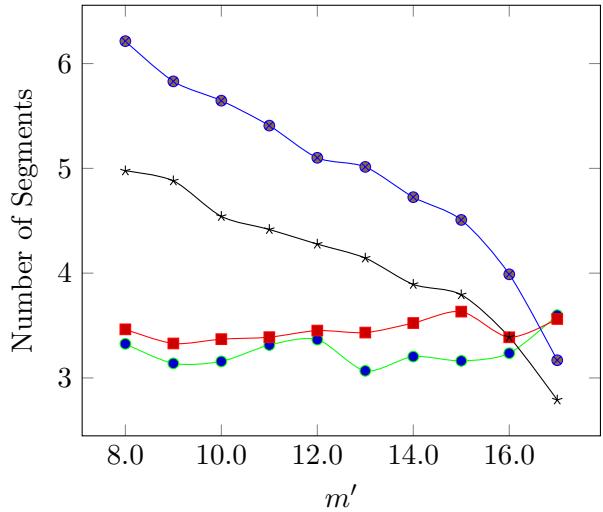
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



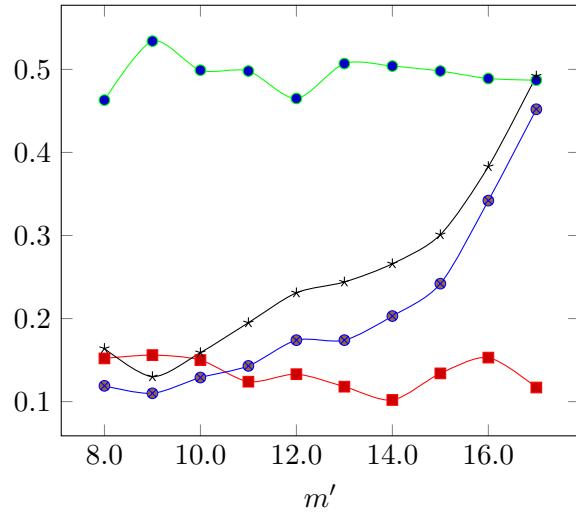
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4