

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	4
1.3 Tujuan	4
1.4 Batasan Masalah	4
1.5 Metodologi	4
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	7
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
2.3 Twig	11
2.4 Integrated Development Environment	12
2.5 Ace	12
2.5.1 Perekaman Event	14
3 ANALISIS	17
3.1 Analisis Sistem Kini	17
3.1.1 Model, View, Controller	17
3.1.2 Penyimpanan Kode Submission	42
3.1.3 Antrean Penilaian Kode	43
3.2 Analisis Sistem Usulan	43
3.2.1 Fitur perekaman perubahan atau event	44
3.2.2 Fitur penyimpanan rekaman perubahan	44
3.2.3 Fitur melihat daftar rekaman	45
3.2.4 Fitur pemutaran ulang rekaman	46
DAFTAR REFERENSI	49
A KODE PROGRAM	51
B HASIL EKSPERIMEN	53

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
3.1	Struktur MVC pada SharIF Judge	18
3.2	Struktur MVC pada SharIF Judge	19
3.3	Struktur Direktori View pada SharIF Judge	24
3.4	Struktur Kelas Controller pada SharIF Judge	26
3.5	Halaman Assignments	27
3.6	Halaman Dashboard	28
3.7	Halaman Hall of Fame	29
3.8	Halaman Install	29
3.9	Halaman Login	30
3.10	Halaman 24-Hour Log	31
3.11	Halaman Moss	32
3.12	Halaman Notifications	33
3.13	Halaman Problems	34
3.14	Halaman Profile	35
3.15	Halaman Queue	36
3.16	Halaman Rejudge	37
3.17	Halaman Scoreboard	37
3.18	Halaman Settings	38
3.19	Halaman Final Submissions	39
3.20	Halaman All Submissions	40
3.21	Halaman Submit	41
3.22	Halaman Users	42
3.23	Use-case analisis sistem usulan	43
3.24	Sequence Diagram Fitur Perekaman Perubahan	44
3.25	Sequence Diagram Fitur Penyimpanan Rekaman	45
3.26	Sequence Diagram Membuka Halaman Rekaman	46
3.27	Sequence Diagram Membuka Halaman Rekaman	47
B.1	Hasil 1	53
B.2	Hasil 2	53
B.3	Hasil 3	53
B.4	Hasil 4	53

1

BAB 1

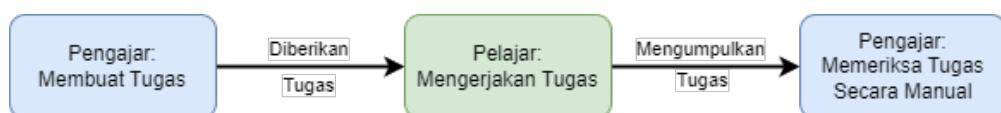
2

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. 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.

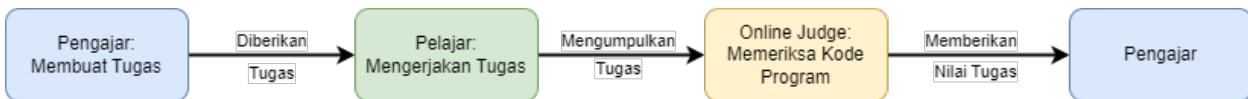


Gambar 1.1: Sistem Tradisional Pemberian Tugas

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 [2]. 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

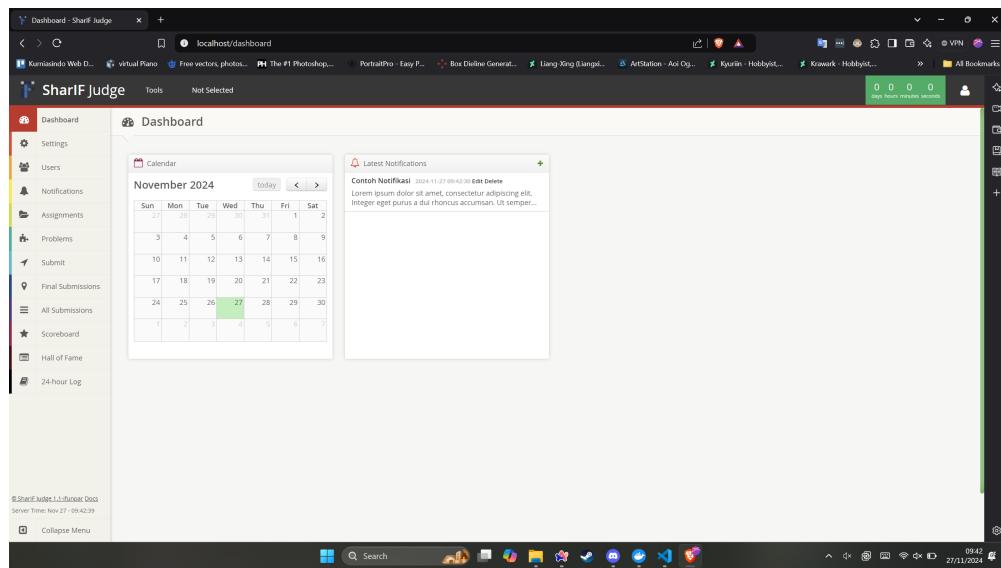
oleh banyak pelajar, biasanya program yang dibuat memiliki antarmuka dan harus diperiksa oleh pengguna khusus yang mengetahui fitur-fitur yang dibutuhkan. Sedangkan tugas individu merupakan sebuah tugas yang diberikan untuk satu individu, biasanya program yang dibuat bersifat algoritmik dan tidak memerlukan 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 [3]. 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 [2]. 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 [4] yang merupakan hasil dimodifikasi oleh Stillmen Vallian terhadap Sharif-Judge [5] 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” [6], 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 mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada
4 IDE SharIF-Judge?
- 5 2. Bagaimana cara menyimpan data pemutaran ulang mahasiswa secara rutin dengan otomatis
6 dan tidak mengambil penyimpanan *database* 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. Mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada IDE SharIF-
12 Judge.
- 13 2. Mencari cara penyimpanan data efektif dan mengimplementasikannya pada perekaman dan
14 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

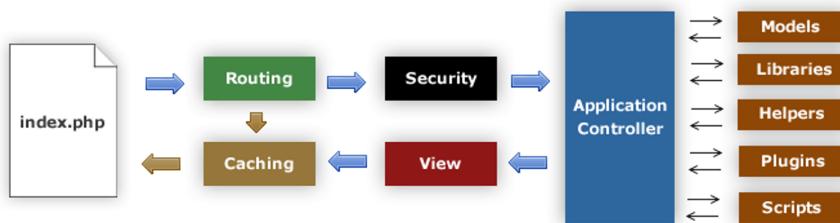
- Pada SharIF Judge, pengguna dibagi menjadi 4 buah *role*. Role yang tersedia adalah sebagai berikut:
1. *admin*
 2. *head instructor*
 3. *instructor*
 4. *student*
- Setiap *role* memiliki akses pada aksi yang berbeda berdasarkan *role*-nya. Tabel 2.1 merupakan aksi-aksi yang dapat dilakukan untuk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

```

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

- 25 Pada CodeIgniter, *model* dan *view* hanya dapat dimuat melalui controller. Seperti contoh, Kode
 26 2.4 akan memuat *model* `blog` dan mengambil data dari *database*, lalu menampilkan *view* yang
 27 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:

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

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

35 `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 2.3 Twig

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

- Struktur Kontrol (menggunakan kondisional, *looping*)
 - Filter
 - Variable pada PHP
- Pada saat template dievaluasi, semua *variable* atau *expression* akan dibuat menjadi value dan *tag* yang mengontrol logika template.

Kode 2.5: Contoh template Twig

```

71  {% extends "base.html" %} 
82  {% block navigation %} 
93      <ul id="navigation">
104     {% for item in navigation %} 
115         <li>
126             <a href="{{item.href}}>
137                 {% if item.level == 2 %}&nbsp;&nbsp;{% endif %}
148                 {{ item.caption|upper }}
159             </a>
160         </li>
171     {% endfor %} 
182 </ul>
383 {% endblock navigation %}
```

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

2.4 Integrated Development Environment

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

- *Editor*

Editor teks sebagai tempat untuk mengetik kode, dapat dilengkapi dengan berbagai fitur seperti *syntax highlighting* (menampilkan teks dengan warna yang berbeda untuk mengindikasikan keterbacaan kode) dan *word completion* (menampilkan prediksi kata yang sedang atau yang akan diketik pengguna).

- *Complier*

Digunakan untuk menterjemahkan kode program yang dibuat pada editor teks ke dalam sebuah program yang dapat dijalankan oleh komputer.

- *Execution*

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

2.5 Ace

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

- *Syntax highlighting* untuk bahasa pemrograman.
- Automatic indent dan outdent.

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

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

7 • **Ace**

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

9 • **Editor**

10 Entri utama untuk fungsionalitas library Ace. Editor sendiri merepresentasikan editor kode
11 yang dibuat pada web. Editor juge menjadi kelas utama untuk mengakses kelas-kelas yang
12 berhubungan dengan editor kode.

13 • **EditSession**

14 Sebuah kelas yang menyimpan semua status dalam editor seperti isi editor, *selection*, dan
15 lain-lain. Kelas ini dinamakan **EditSession**, tetapi untuk mengakses dari editor dinamakan
16 *session*.

17 • **Anchor**

18 Menangani posisi *pointer* pada dokumen. Saat teks dimasukkan atau dihapus, posisi *anchor*
19 akan diperbaharui.

20 • **Document**

21 Menyimpan teks dokumen.

22 • **Range**

23 Kelas ini digunakan di berbagai tempat untuk mengindikasikan suatu wilayah di dalam editor.
24 Kelas ini menyimpan posisi baris awal dan kolom awal, serta baris akhir dan kolom akhir.

25 • **Selection**

26 Kelas ini menyimpan posisi yang di pilih oleh pengguna dalam editor.

27 • **Commands**

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

Kode 2.6: Contoh kode penggunaan Ace

```

30<!DOCTYPE html>
31<html lang="en">
32<head>
33<title>ACE in Action</title>
34<style type="text/css" media="screen">
35  #editor {
36    position: absolute;
37    top: 0;
38    right: 0;
39    bottom: 0;
40    left: 0;
41  }
42</style>
43</head>
44<body>
45<div id="editor">function foo(items) {
46  var x = "All this is syntax highlighted";
47  return x;
48}</div>
49<script src="/ace-builds/src-noconflict/ace.js" type="text/javascript" charset="utf-8"></script>
50<script>
```

```

24 var editor = ace.edit("editor");
25 editor.setTheme("ace/theme/monokai");
26 editor.session.setMode("ace/mode/javascript");
27 </script>
28 </body>
29 </html>

```

8 Kode 2.6 merupakan cara penggunaan Ace pada sebuah div dengan id `editor`. Ace juga memiliki
 9 beberapa konfigurasi, seperti contoh ini yaitu menggunakan tema *monokai* dan menggunakan *syntax*
 10 *highlighting* untuk bahasa pemrograman JavaScript.

11 2.5.1 Perekaman Event

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

- 16 • **Editor**

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

- 19 • **EditSession**

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

- 24 • **Selection**

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

- 26 – `changeCursor` : Mendengarkan perubahan pada kursor atau *anchor* dalam editor kode.
- 27 – `changeSelection` : Mendengarkan perubahan pemilihan isi kode dalam editor kode.

- 28 • **Commands**

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

33 Untuk menggunakan fungsi *event listener* pada kelas yang diinginkan, dibutuhkan fungsi `on`
 34 pada kelas tersebut. Fungsi `on` memiliki dua parameter yaitu nama *event* ingin didengar (`exec` atau
 35 `change`) dan sebuah fungsi *callback* yang akan dijalankan saat *event* terjadi. Kode 2.7 merupakan
 36 perubahan kode yang dilakukan dalam tag `<script>` pada Kode 2.6 agar perubahan isi editor
 37 dapat didengar.

Kode 2.7: Contoh kode event listener

```

38
39 1 <script>
40 2   var editor = ace.edit("editor");
41 3   editor.setTheme("ace/theme/monokai");
42 4   editor.session.setMode("ace/mode/javascript");
43 5
44 6   editor.session.on("change", (delta) => {
45 7     console.log(delta);
46 8     // Contoh Keluaran :
47 9     // {
48 0     //   action: "insert"
49 1     //   end: {row: 3, column: 5}

```

```
12      //      id: 1
13      //      lines: ['a']
14      //      start: {row: 3, column: 4}
15      // }
16  });
17 </script>
```

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

1

BAB 3

2

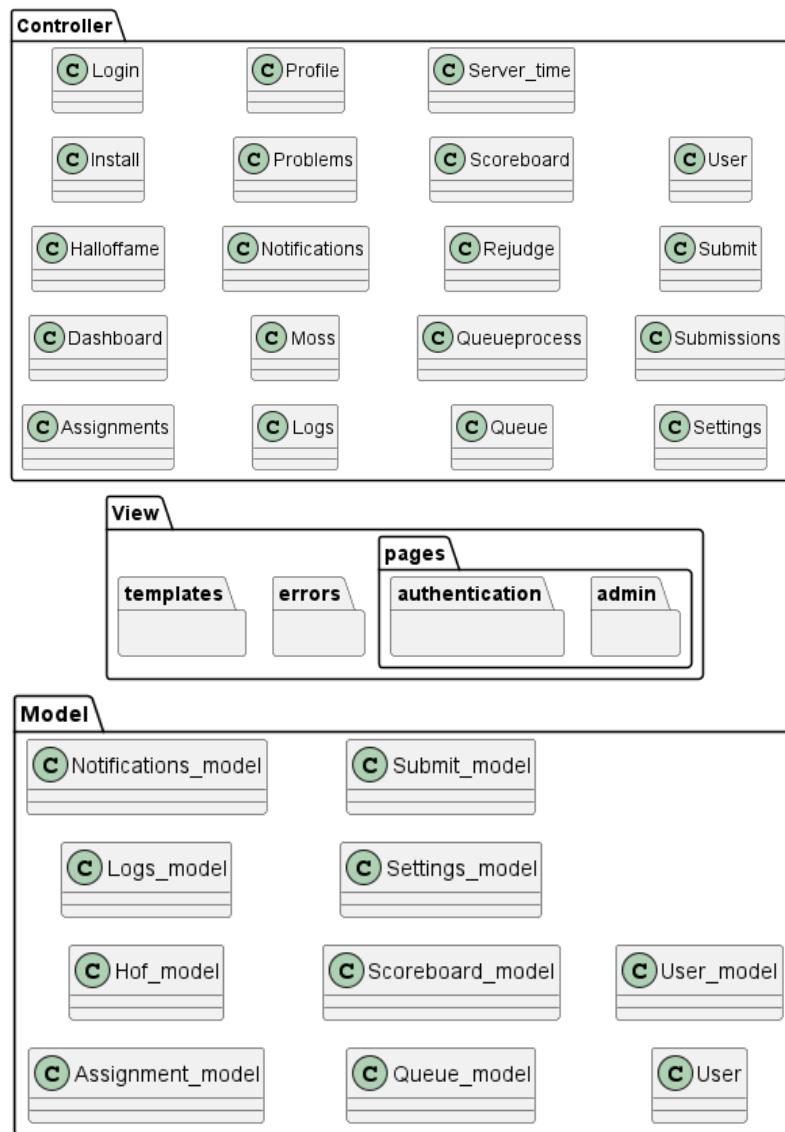
ANALISIS

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 gambar kelas diagram struktur MVC pada
10 SharIF Judge.

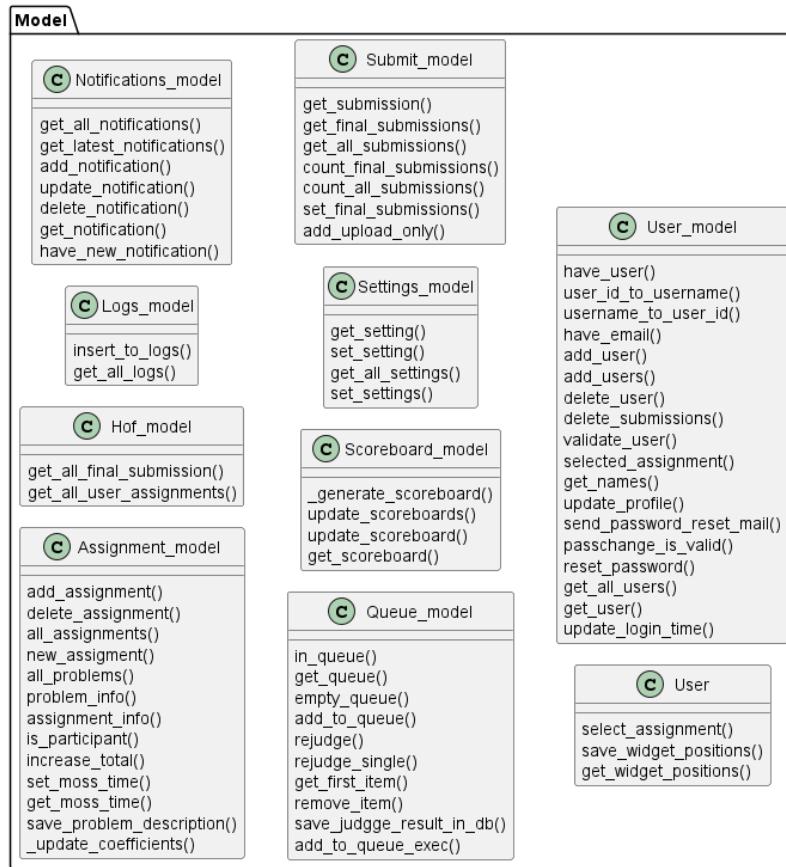


Gambar 3.1: Struktur MVC pada SharIF Judge

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*.
- 29 • **Scoreboard_model.php**
30 Model ini digunakan untuk mengelola tabel **scoreboard**. Fungsi yang dimiliki adalah sebagai
31 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*.
- 40 • **Settings_model.php**
41 Model ini digunakan untuk mengelola tabel **settings**. Fungsi yang dimiliki adalah sebagai
42 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**

10 Model ini digunakan untuk mengelola tabel `submission`. Fungsi yang dimiliki adalah sebagai
11 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**

29 Model ini digunakan untuk menyimpan `settings` sebuah `user`. Fungsi yang dimiliki adalah
30 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**

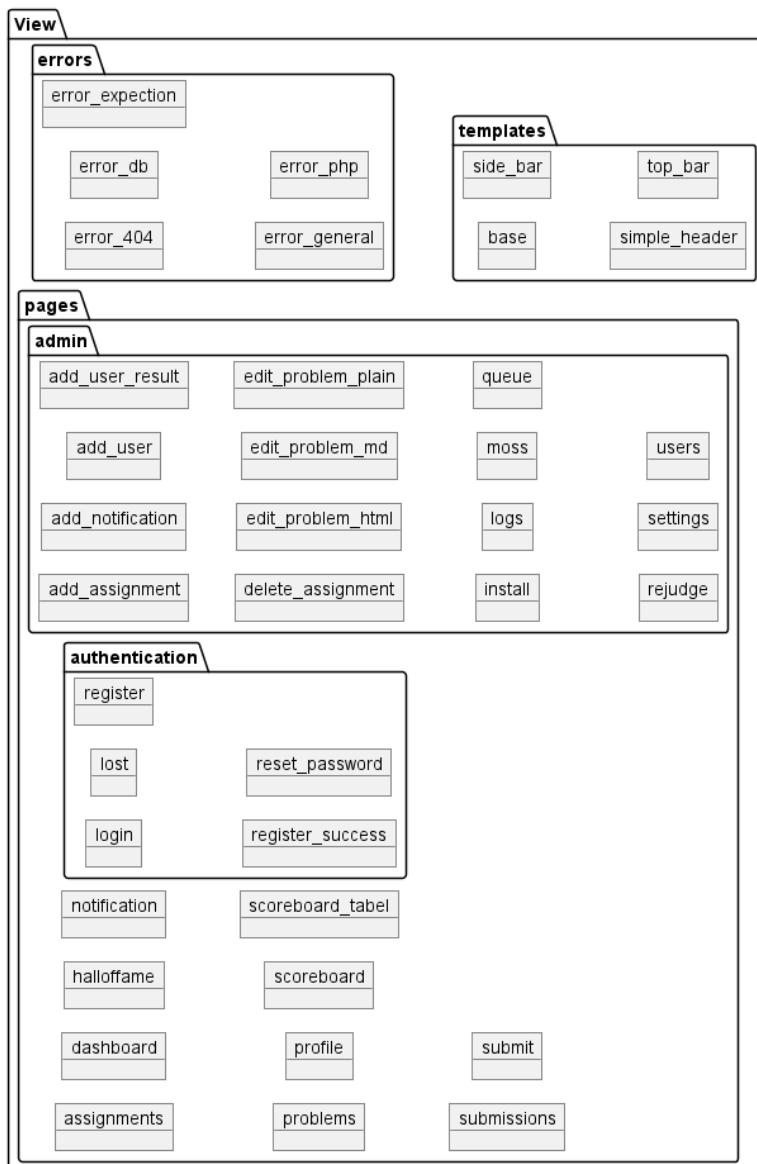
38 Model ini digunakan untuk mengelola tabel `users`. Fungsi yang dimiliki adalah sebagai
39 berikut:

- 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_exception**
- **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`

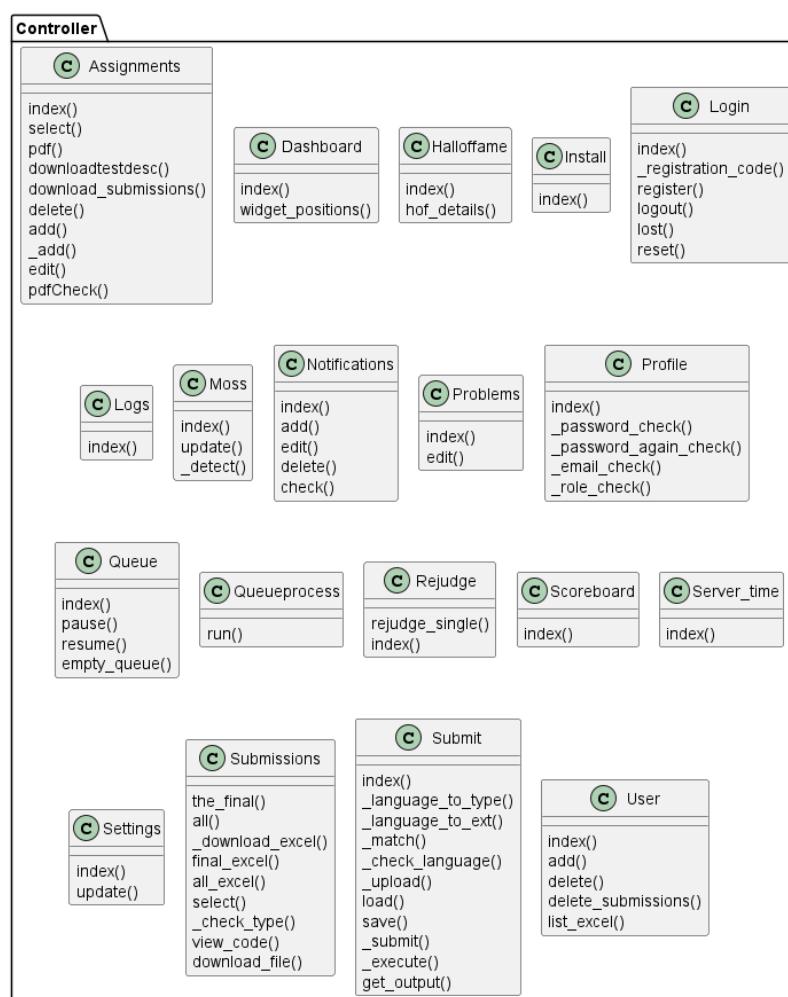
```

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 bab 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`:

15 – `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*.

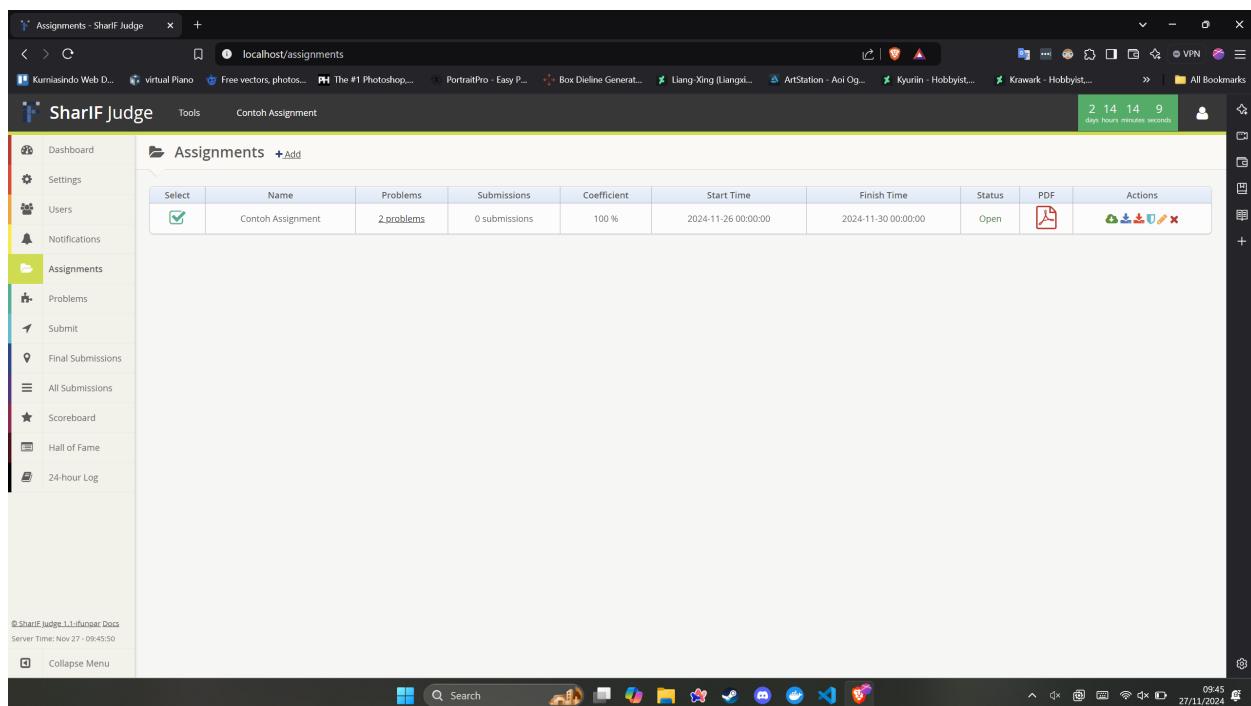
6 – `add()`
 Mendapatkan *input* dari pengguna untuk menambah atau memperbarui sebuah *assignment*.

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.

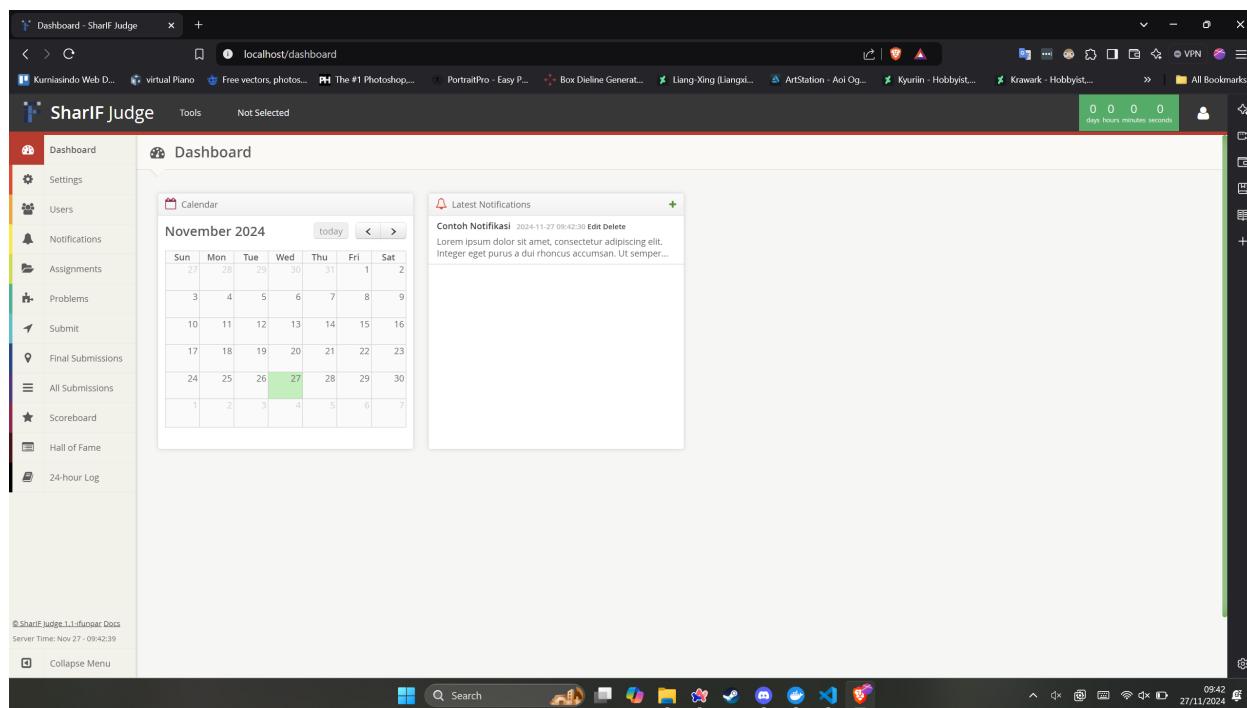


Gambar 3.5: Halaman Assignments

22 • `Dashboard.php`

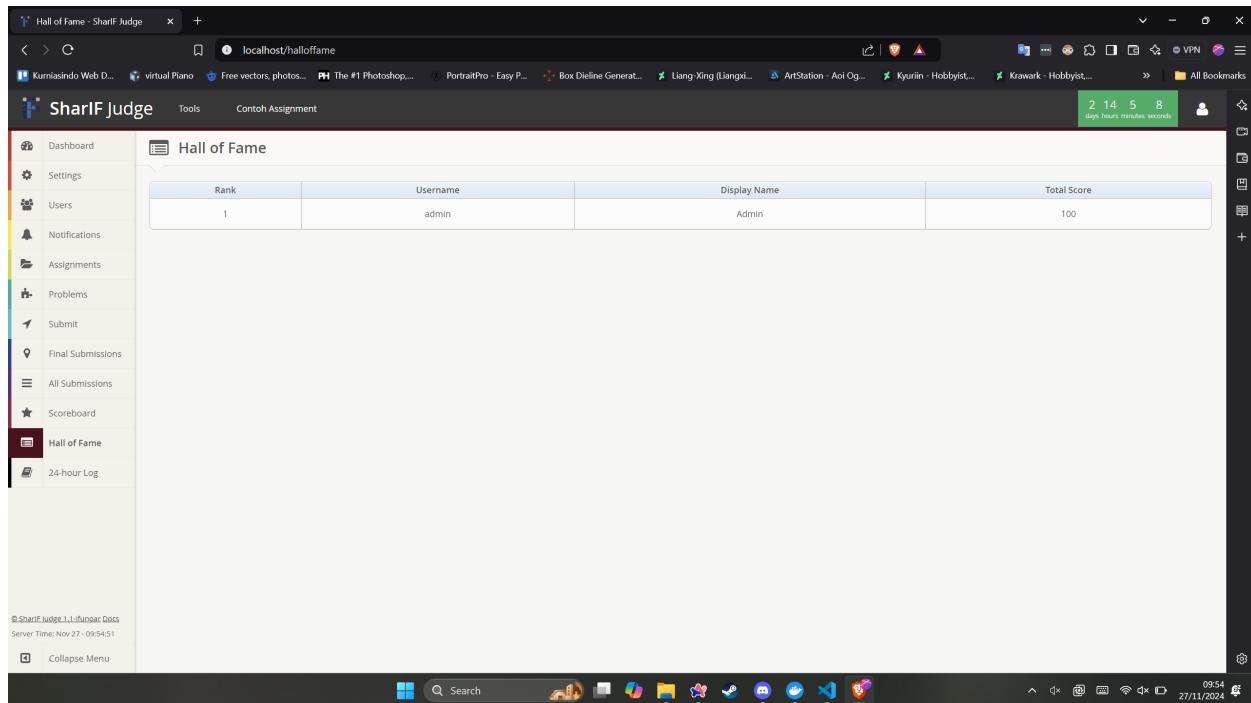
23 Berikut fungsi dengan penjelasannya pada *controller Dashboard.php*:

- 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

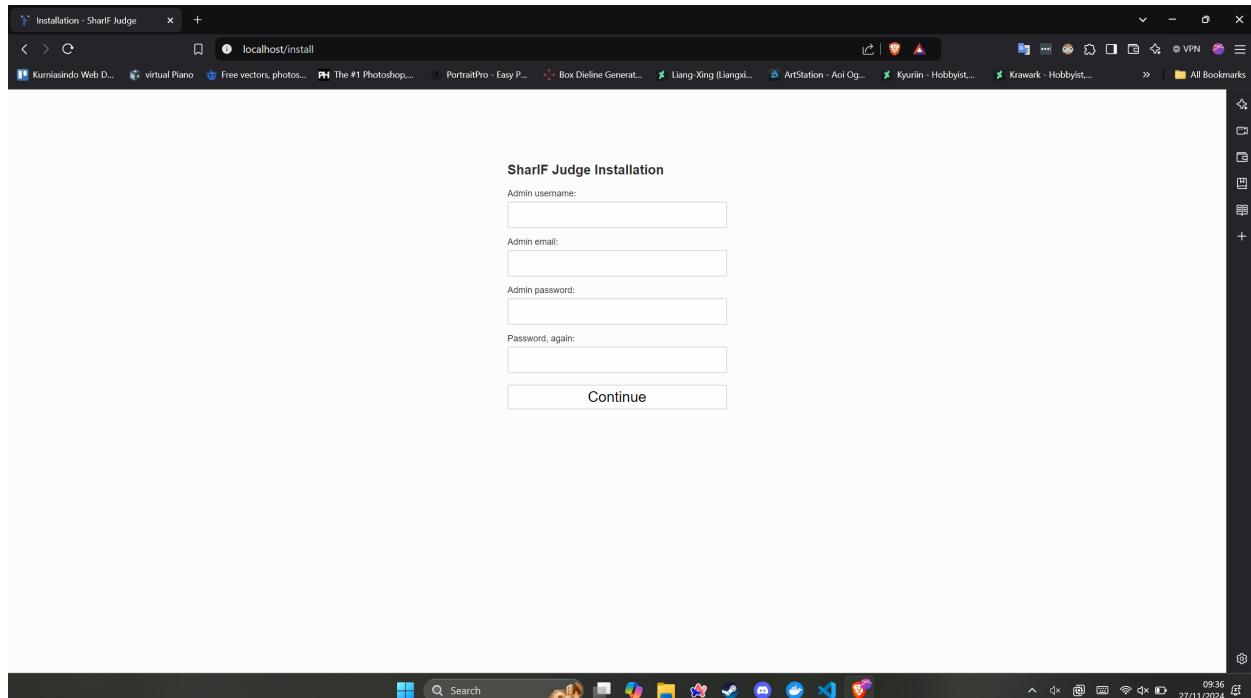
- 8 • `Halloffame.php`
- 9 Berikut fungsi dengan penjelasannya pada controller `Halloffame.php`:
- 10 – `hof_details()`
- 11 Menampilkan nilai akhir semua *problem* dan *assignments* pada sebuah *user*.
- 12 – `index()`
- 13 Mendapatkan data dari `Hof_model` dan mengembalikan *view halloffame.twig*. 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()`

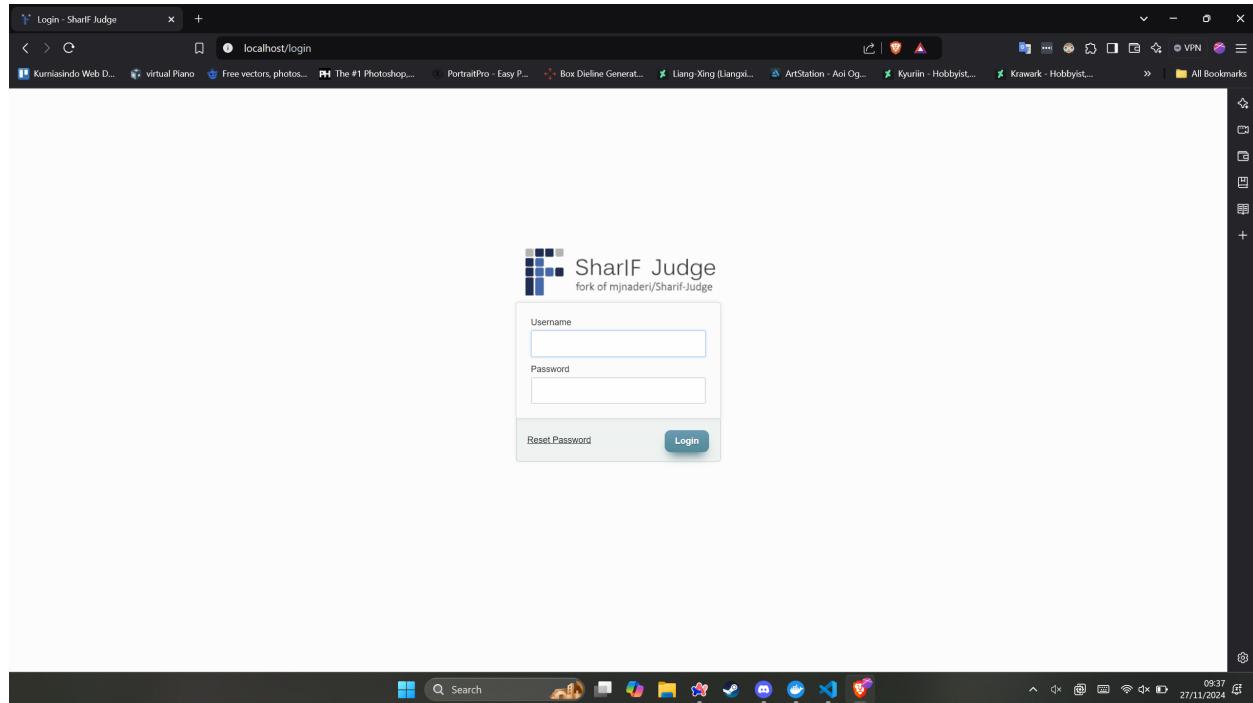
10 Mengirimkan email *reset password*.

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)

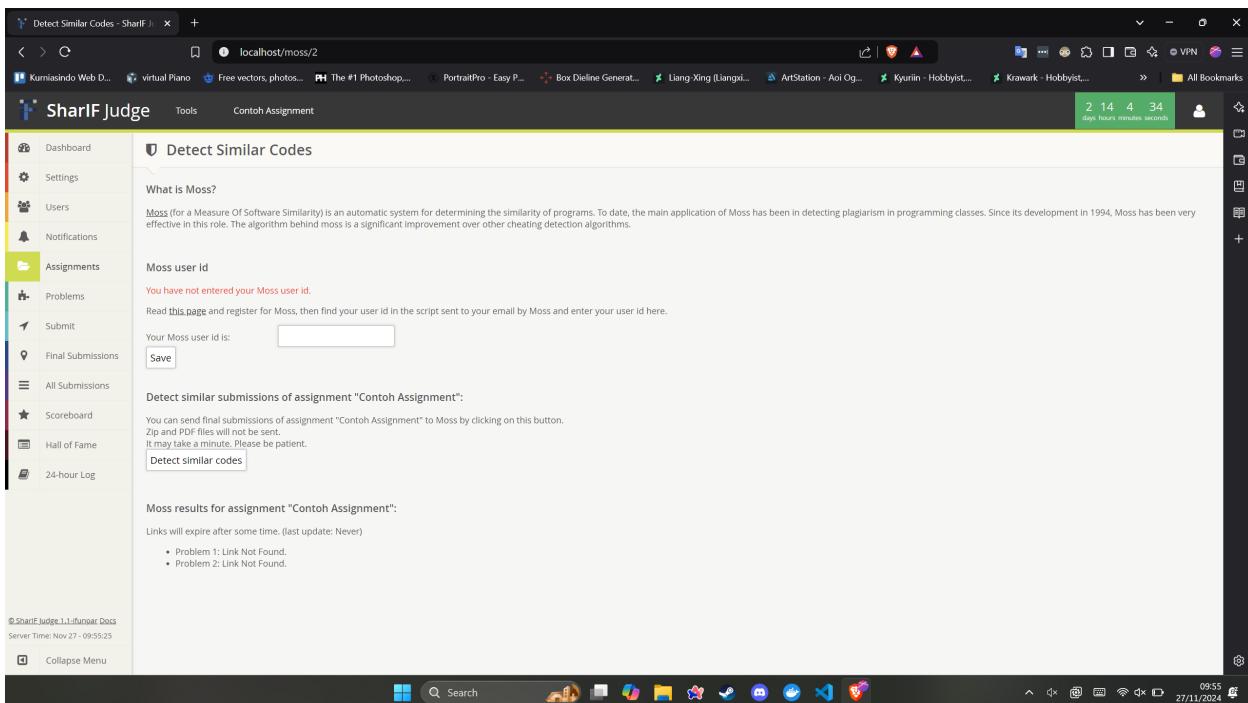
4 Memperbaharui *settings* dari masukkan moss_userid pengguna.

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*.

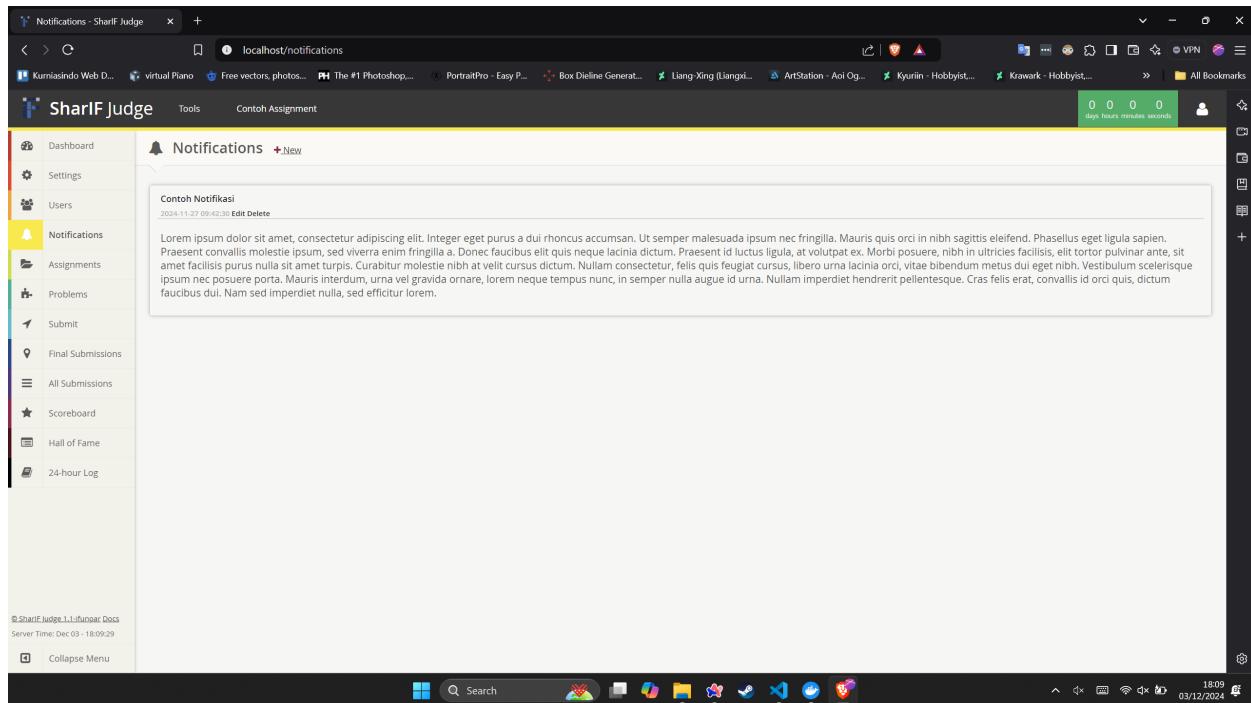
9 – **check()**

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**

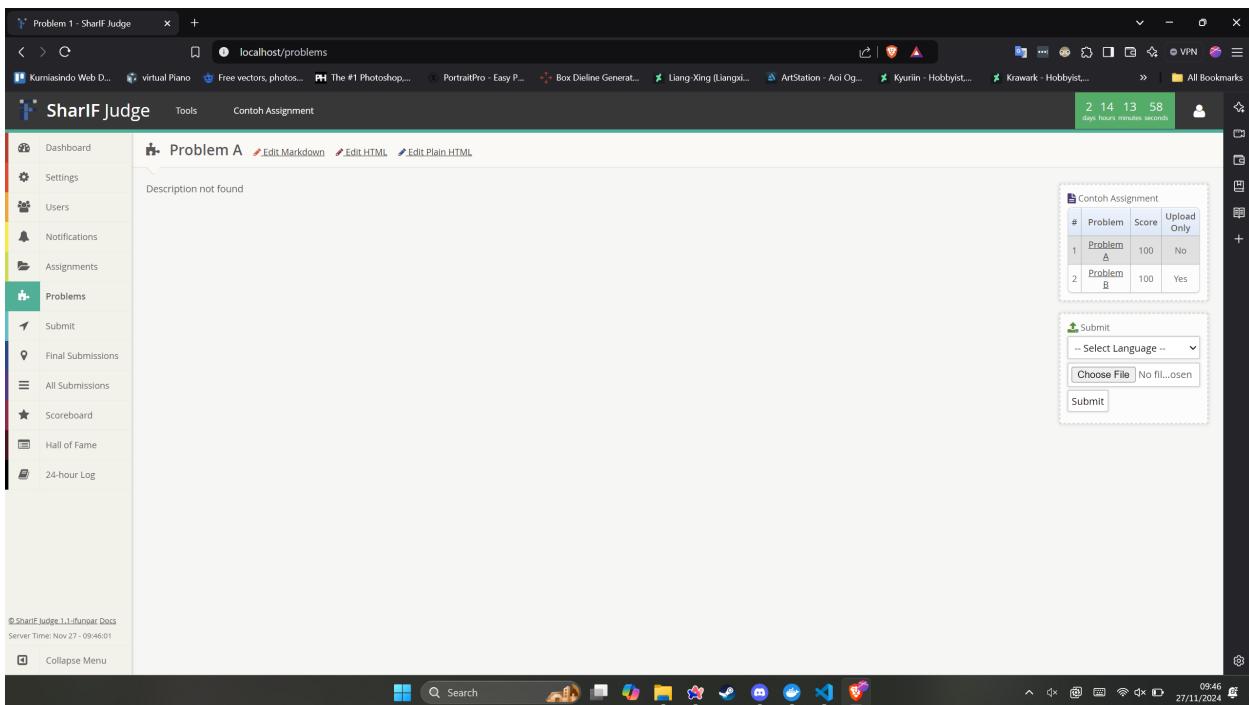
2 Berikut fungsi dengan penjelasannya pada *controller Notifications.php*:

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)*

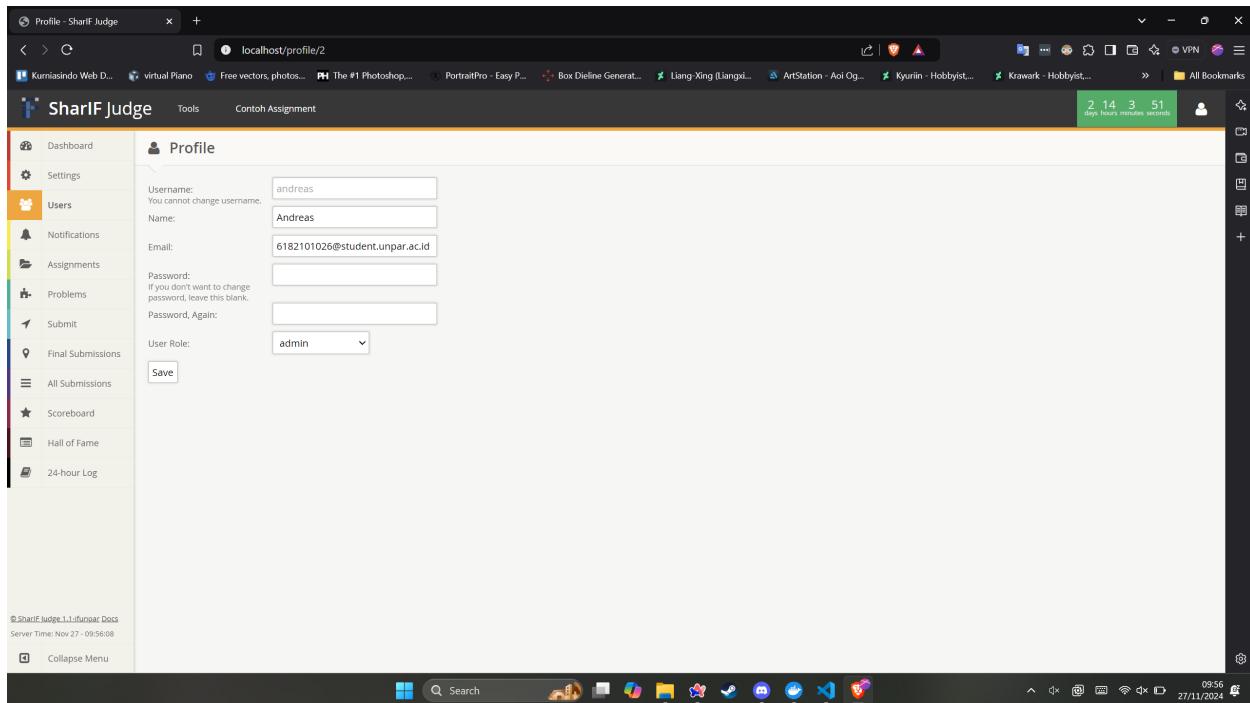
8 Melakukan validasi ketersediaan email pada *database*.

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()**

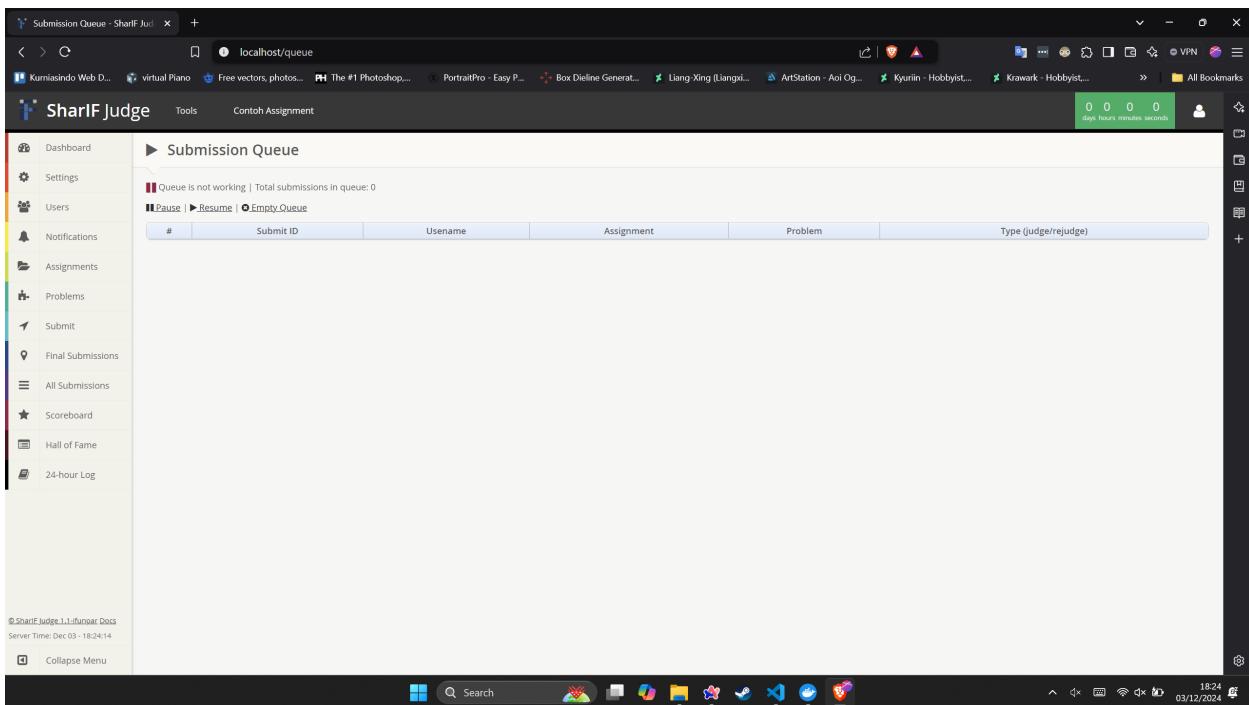
6 Melanjutkan proses *queue*.

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**

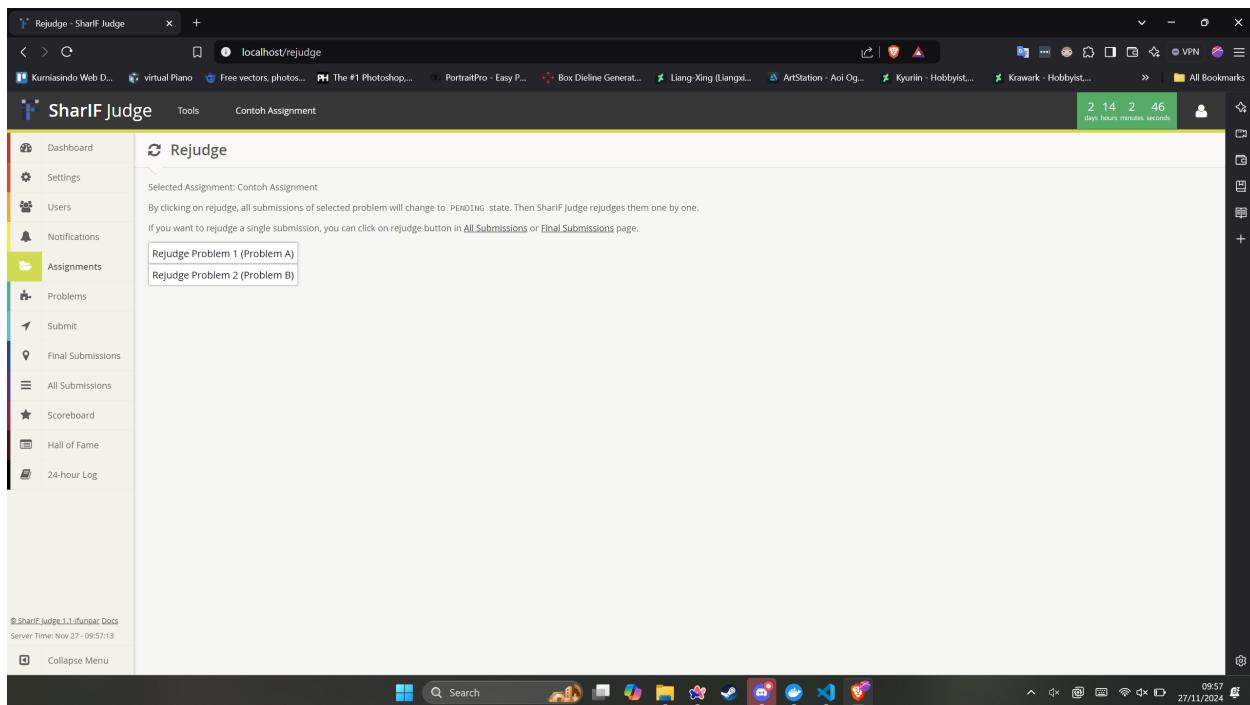
5 Berikut fungsi dengan penjelasannya pada *controller Profile.php*:

6 – *rejudge_single()*

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

8 – *index()*

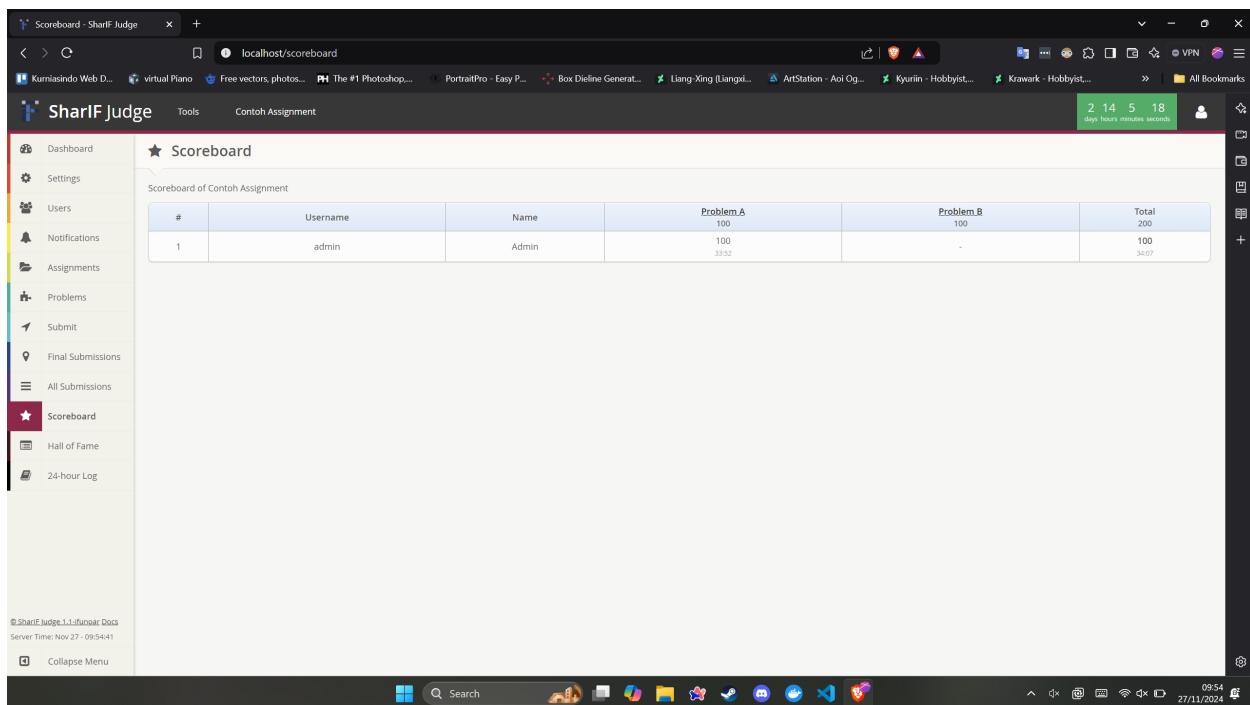
9 Mendapatkan data dan menampilkan *view rejudge.twig*. Fungsi ini juga dapat me-
10 lakukan *rejudge* pada sebuah *problem* tertentu. Gambar 3.16 menunjukkan halaman
11 Rejudge.



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**

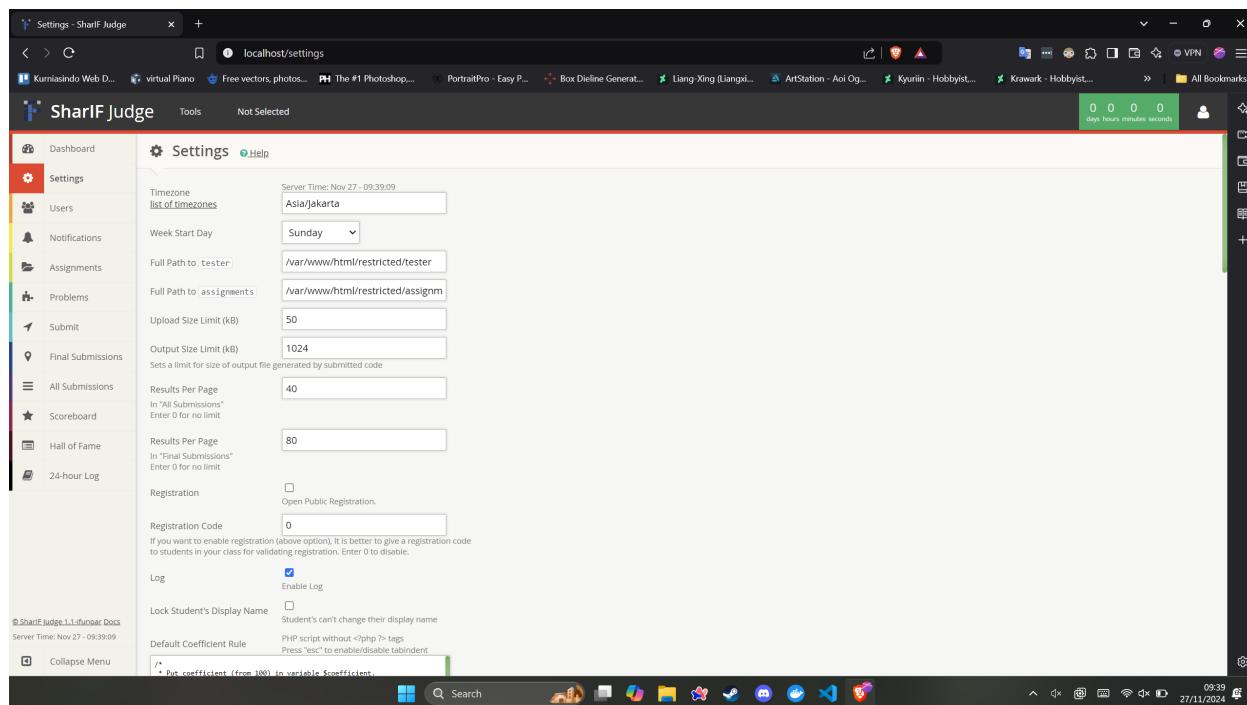
Berikut fungsi dengan penjelasannya pada controller Settings.php:

- 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.

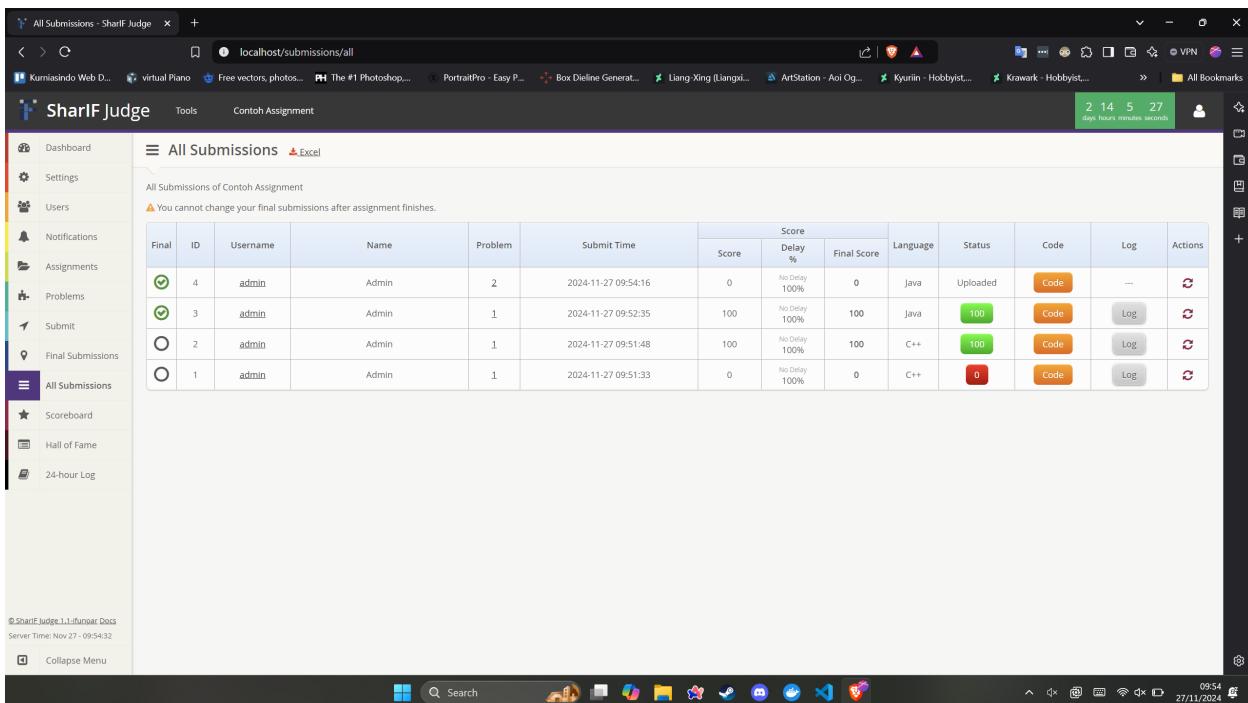
- _check_type(\$type)

- 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	100	Code	Log	↻	
2	4	admin	Admin	2	2024-11-27 09:54:16	0	No Delay 100%	0	java	Uploaded	Code	...	↻	

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

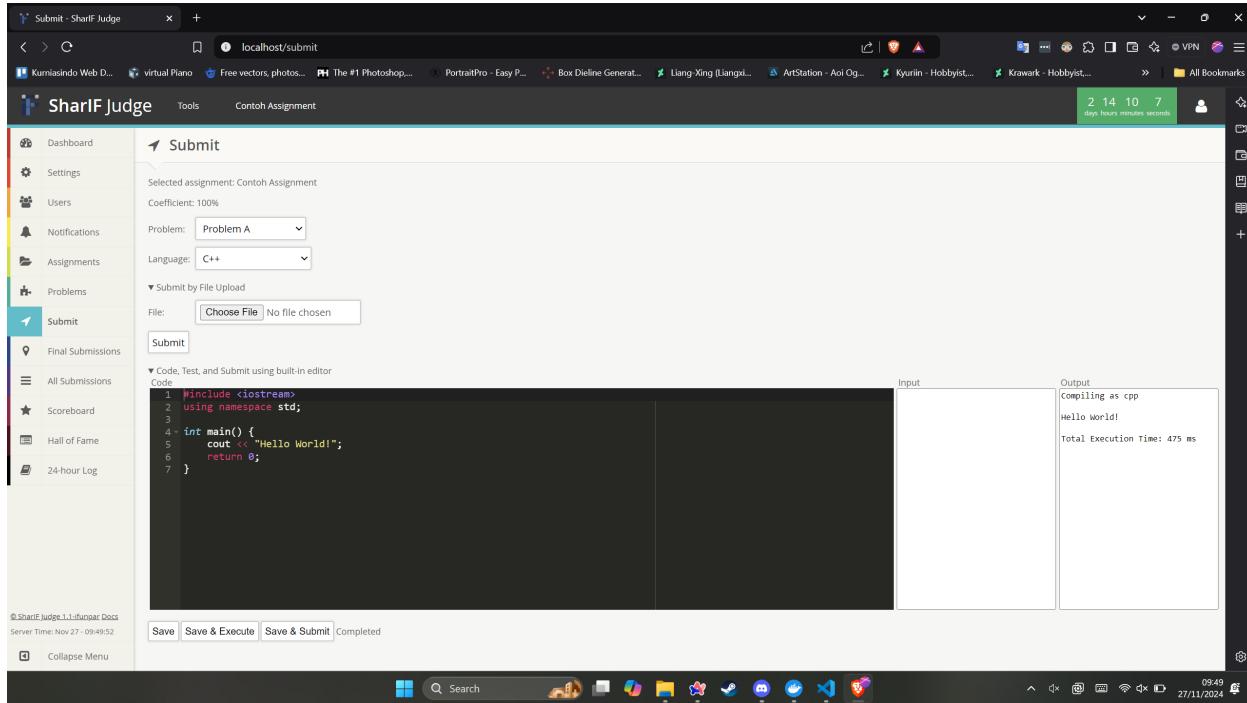
1 • **Submit.php**

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 [6].



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()**

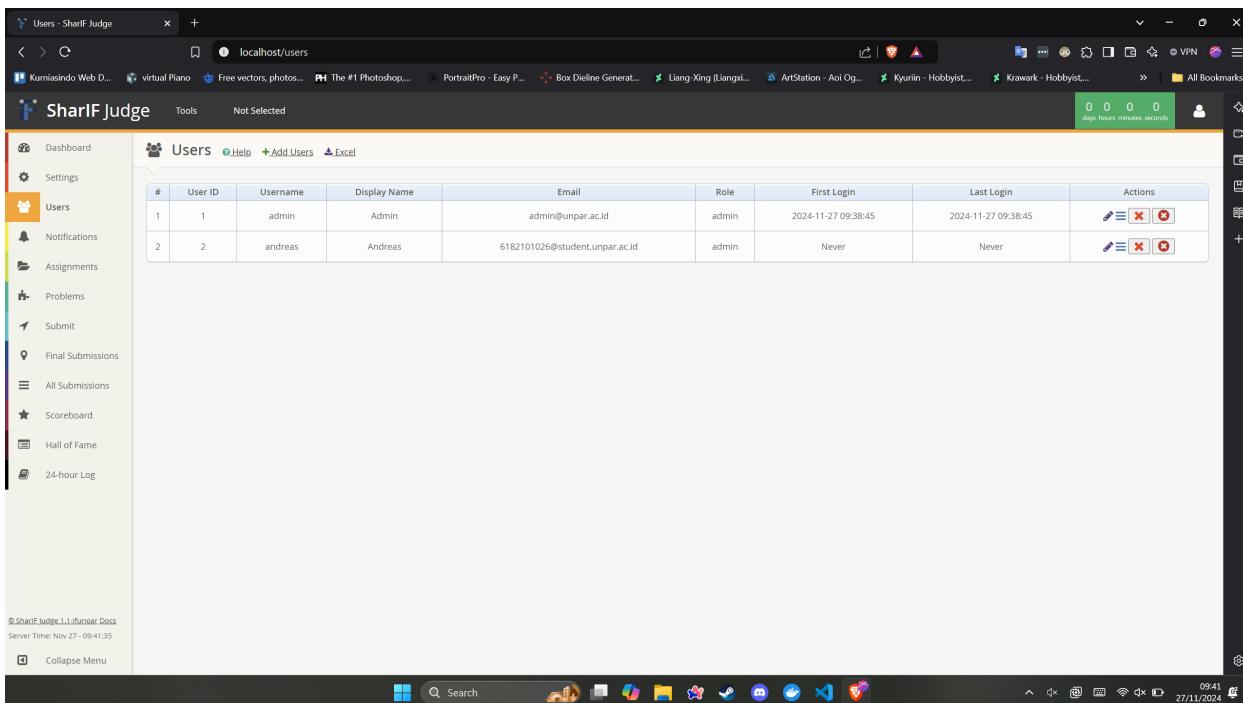
13 Menghapus seluruh *submissions* dari sebuah *user*.

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 Penyimpanan Kode Submission

2 Pada SharIF Judge, Kode akan disimpan pada lokasi **Assignment** yang dapat di ubah pada halaman
3 **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:

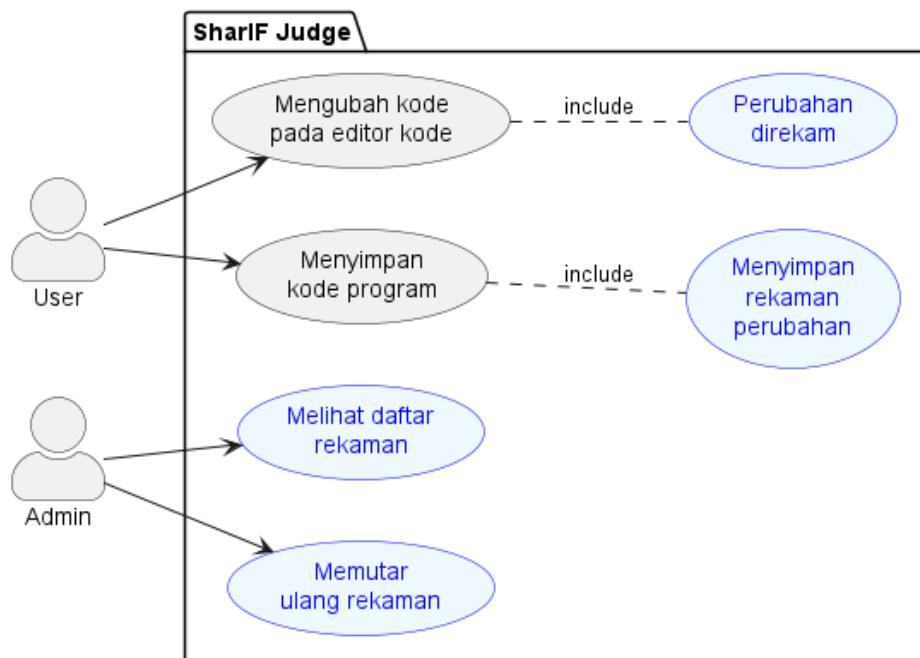
22 `assignment_5/p_1/kenzhi/probA-6.java`

1 3.1.3 Antrean Penilaian Kode

- 2 Pada SharIF Judge, Kode yang dikumpulkan akan di jalankan satu per satu pada antrean menggunakan bash. Berikut merupakan cara SharIF Judge menilai kode dari awal pengumpulan pada sistem:
- 5 1. *Controller Submit* akan menyimpan kode ke dalam 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 menambahkan data *queue*.
- 9 4. Selanjutnya *Controller Submit* akan memanggil fungsi *process_the_queue()* yang akan menjalankan fungsi *run()* pada *controller Queueprocess*.
- 11 5. *Controller Queueprocess* akan menjalankan *tester.sh* pada folder *tester* dengan data dari *queue*.
- 13 6. *tester.sh* akan menilai kode yang akan dibaca oleh *controller Queueprocess* yang akan menyimpan hasil penilaian.
- 15 7. Terakhir *Queueprocess* akan menyimpan hasil penilaian pada *database submission* dan menghapus data *queue* menggunakan *Queue_model*.

17 3.2 Analisis Sistem Usulan

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



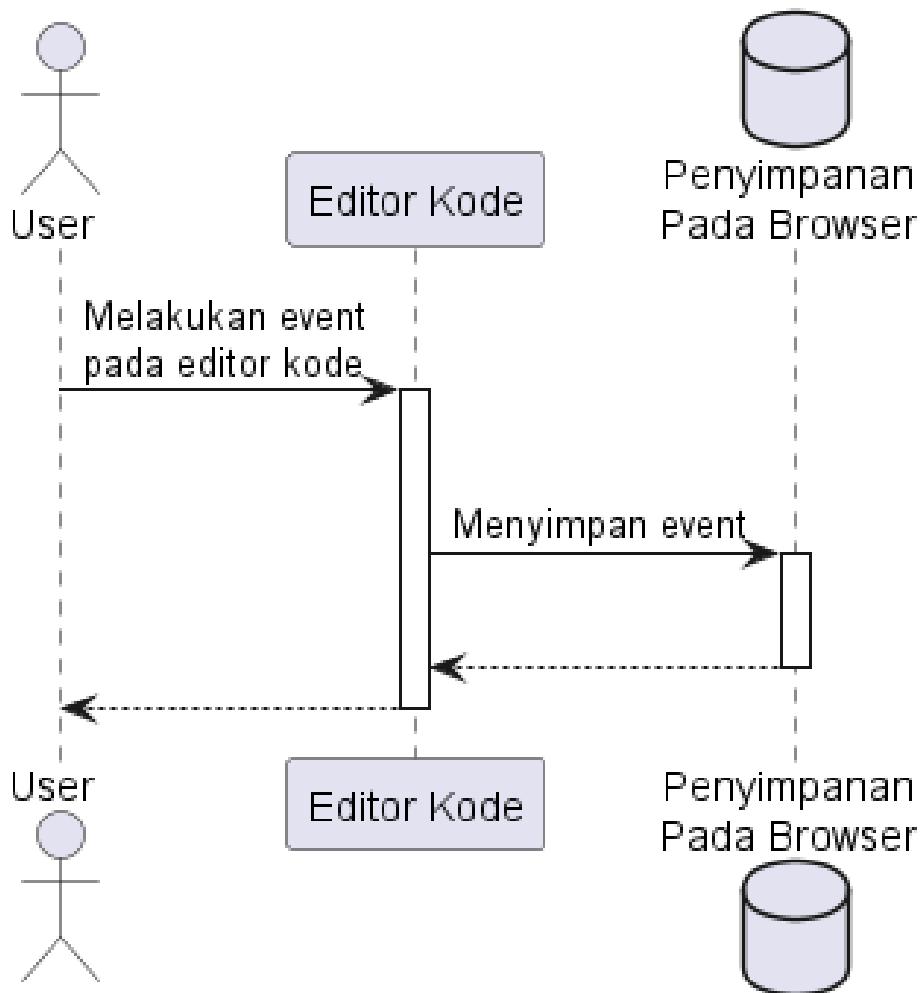
Gambar 3.23: Usecase analisis sistem usulan

3.2.1 Fitur perekaman perubahan atau event

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

Sequence Diagram

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



Gambar 3.24: Sequence Diagram Fitur Perekaman Perubahan

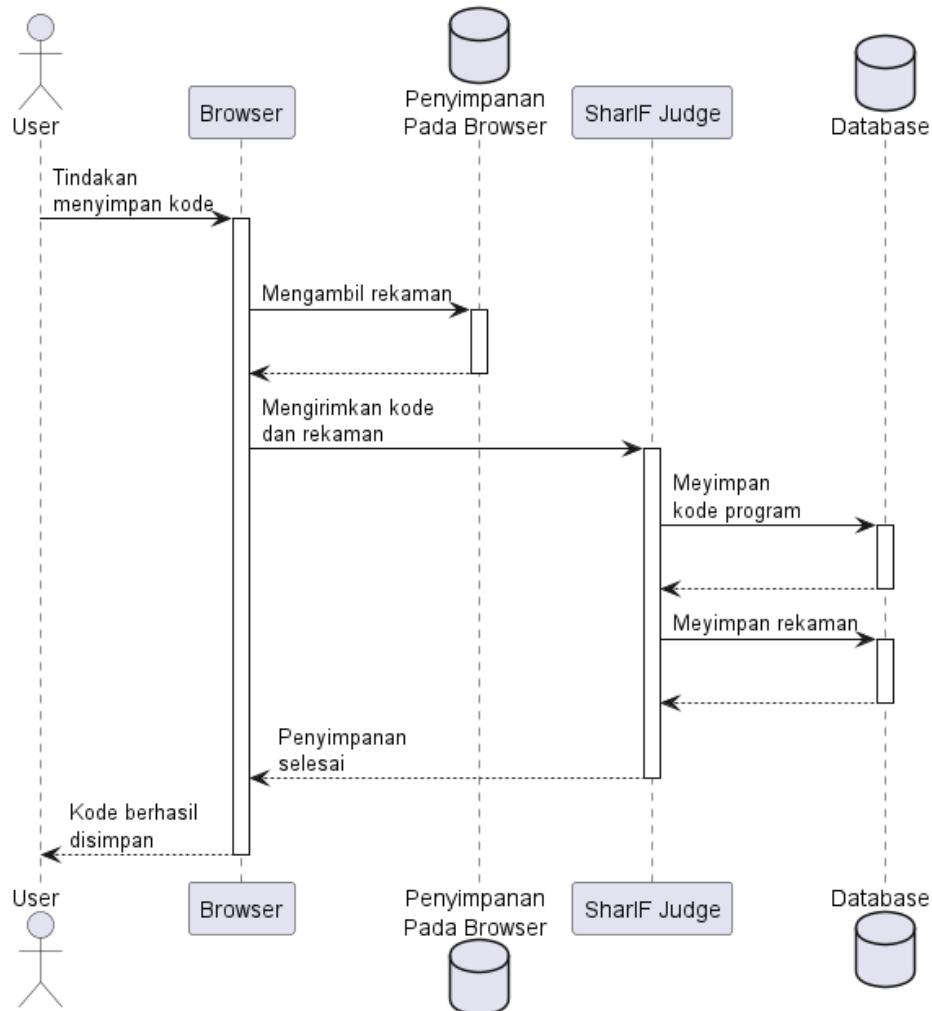
3.2.2 Fitur penyimpanan rekaman perubahan

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

- 1 dengan adanya rekaman baru atau perubahan pada *file* rekaman.

2 Sequence Diagram

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



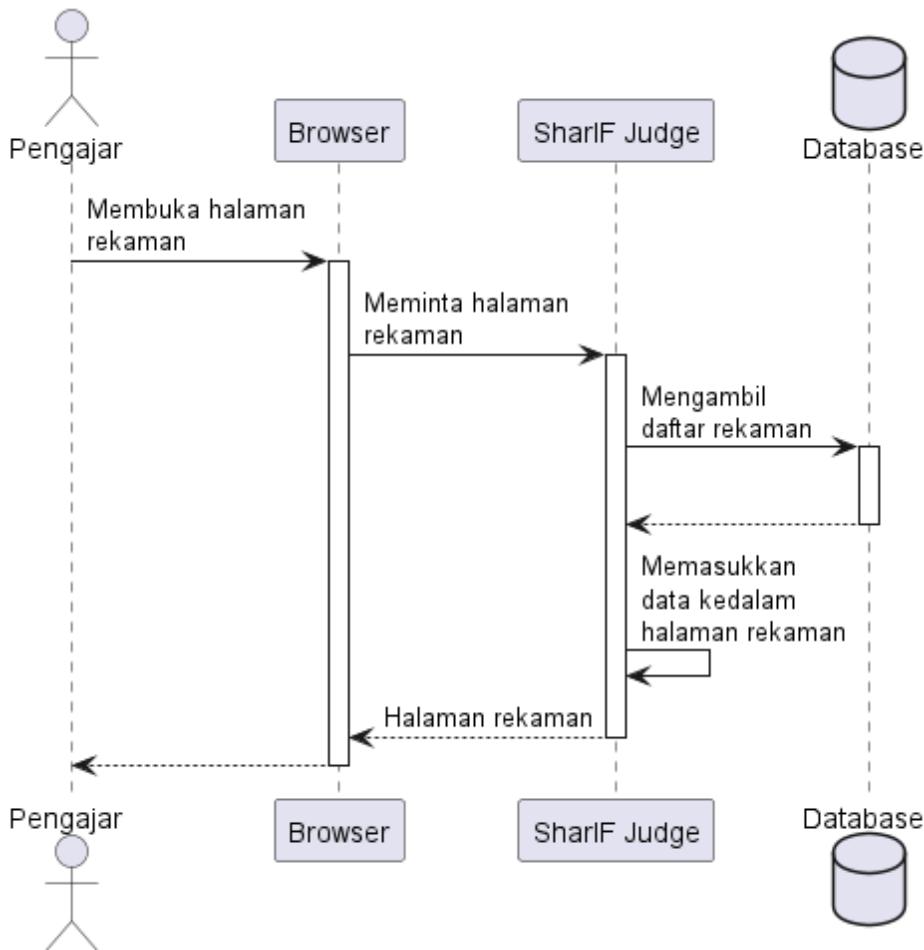
Gambar 3.25: Sequence Diagram Fitur Penyimpanan Rekaman

5 3.2.3 Fitur melihat daftar rekaman

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

1 Sequence Diagram

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



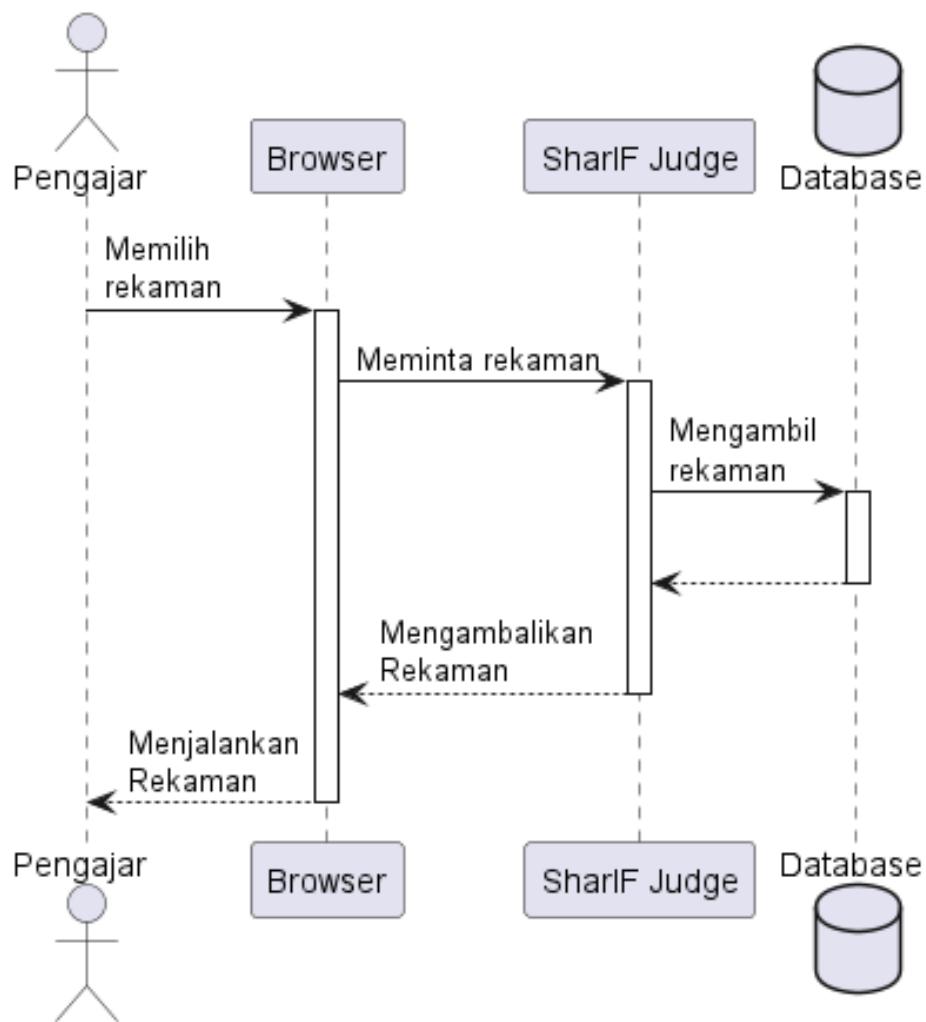
Gambar 3.26: Sequence Diagram Membuka Halaman Rekaman

4 3.2.4 Fitur pemutaran ulang rekaman

- 5 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.

10 Sequence Diagram

- 11 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

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] Kurnia, A., Lim, A., dan Cheang, B. (2001) Online judge. *Computers & Education*, **18**, 299–315.
- [3] IDCloudHost (2020) Algoritma pemrograman beserta contohnya. <https://idcloudhost.com/blog/algoritma-pemrograman-pengertian-fungsi-cara-kerja-dan-contohnya/>. 6 Desember 2024.
- [4] Vallian, S. (2018) Kustomisasi Sharif Judge Untuk Kebutuhan Program Studi Teknik Informatika. Skripsi. Universitas Katolik Parahyangan, Indonesia.
- [5] Version 1.4 (2014) *Sharif Judge Documentation*. Mohammad Javad Naderi. Tehran, Iran.
- [6] Halim, N. A. (2021) Implementasi Editor Kode pada SharIF Judge. Skripsi. Universitas Katolik Parahyangan, Indonesia.

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

```

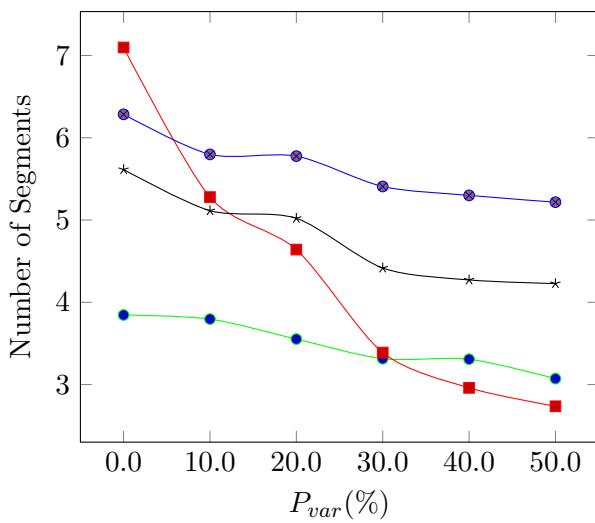
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the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

```

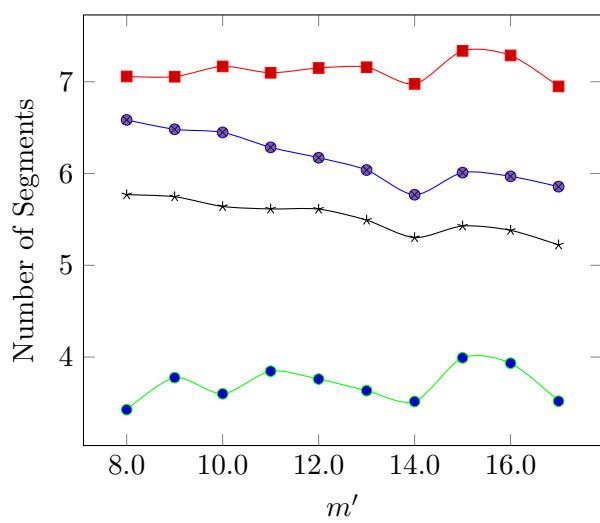

LAMPIRAN B

HASIL EKSPERIMENT

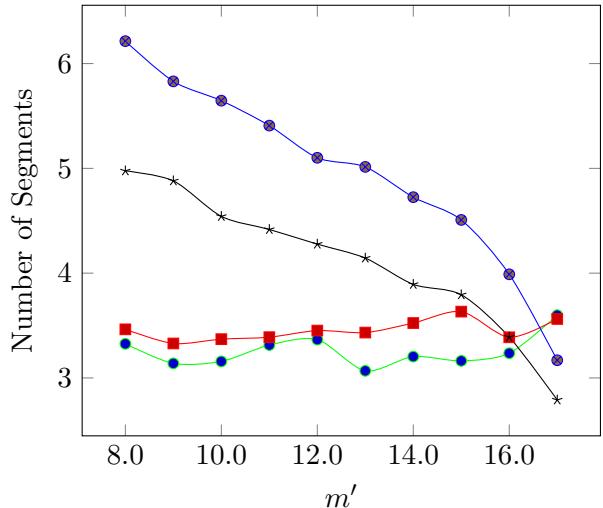
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



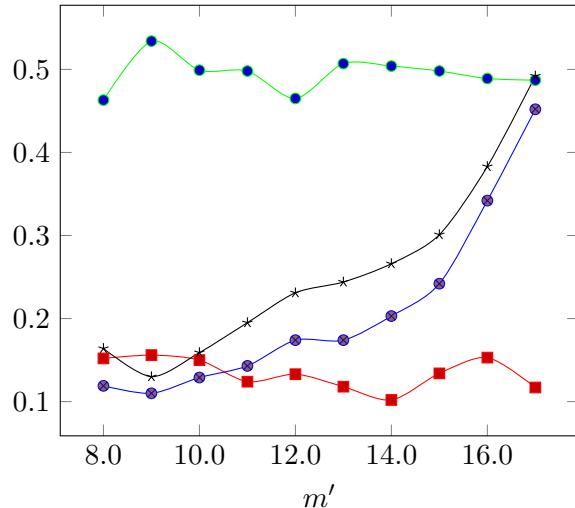
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4