

# *Porting* PHP MENJADI JAVA/PLAY FRAMEWORK (STUDI KASUS KIRI *Dashboard Server Side*)

TOMMY ADHITYA THE-2012730031

## 1 Data Skripsi

Pembimbing utama/tunggal: **Pascal Alfadian, M.Com.**

Pembimbing pendamping: -

Kode Topik : **PAS3901**

Topik ini sudah dikerjakan selama : **1 semester**

Pengambilan pertama kali topik ini pada : Semester **39 - Ganjil 15/16**

Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**

Tipe Laporan : **B** - Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

## 2 Detail Perkembangan Pengerjaan Skripsi

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

1. Mempelajari isi kode situs web KIRI secara umum, isi kode KIRI *Dashboard*, dan isi kode KIRI *Dashboard Server Side*.

**status :** Ada sejak rencana kerja skripsi, kecuali isi kode situs web KIRI secara umum dan isi kode KIRI *Dashboard*

**hasil :**

Berdasarkan hasil analisa serta wawancara dengan kontributor kode KIRI, berikut adalah penjelasan secara umum mengenai isi kode KIRI (gambar 1):

- (a) *Folder “settings”* merupakan *folder* yang menyimpan *file-file* pengaturan sistem KIRI untuk Eclipse. Eclipse adalah sebuah *open source* IDE (*Integrated Development Environment*) yang membantu programmer dalam membangun suatu perangkat lunak[1].
- (b) *Folder “etc”* merupakan *folder* yang menyimpan *file-file* untuk membangun konstanta-konstanta dan fungsi-fungsi yang digunakan oleh sistem KIRI.
- (c) *Folder “log”* merupakan *folder* yang berisi *file-file* untuk mencatat setiap kinerja sistem KIRI.
- (d) *Folder “public\_html”* merupakan *folder* yang berisi *file-file* untuk membangun KIRI *Front End*.
- (e) *Folder “public\_html\_dev”* merupakan *folder* yang berisi *file-file* untuk membangun KIRI *Dashboard*.
- (f) *Folder “res”* merupakan *folder* yang berisi *file-file* yang bersifat publik seperti gambar, data XML, dll yang digunakan sebagai pendukung sistem KIRI.
- (g) *Folder “sql”* merupakan *folder* yang menyimpan *file-file* untuk membangun *database* sistem KIRI.
- (h) *File “.buildpath”* dan *“.project”* merupakan *file* yang menyimpan konfigurasi untuk Eclipse.
- (i) *File “.gitignore”* merupakan *file* yang menyimpan daftar *file* yang tidak perlu dikirimkan ke tempat penyimpanan GitHub karena *file-file* tersebut dibangkitkan oleh sistem. GitHub adalah sebuah tempat penyimpanan *online* yang dikhususkan untuk menyimpan isi kode suatu perangkat lunak[2].
- (j) *File “.gitmodules”* merupakan *file* yang menyimpan alamat kode eksternal yang digunakan dalam sistem KIRI.

- (k) *File* “README.md” merupakan *file* yang berisi keterangan singkat mengenai proyek KIRI. Isi keterangan singkat tersebut digunakan untuk mencegah sembarang orang agar tidak membuka proyek KIRI di GitHub.
- (l) *File* “build.properties” dan “build.xml” merupakan *file* yang digunakan Ant untuk mendistribusikan program. Ant adalah sebuah perangkat yang dapat digunakan untuk menjalankan tes dan menjalankan aplikasi dalam bahasa Java[3].

pascalalfadian First commit		Latest commit b650bfa on Mar 20
📁 .settings	First commit	8 months ago
📁 etc	First commit	8 months ago
📁 log	First commit	8 months ago
📁 public_html	First commit	8 months ago
📁 public_html_dev	First commit	8 months ago
📁 res	First commit	8 months ago
📁 sql	First commit	8 months ago
📄 .buildpath	First commit	8 months ago
📄 .gitignore	First commit	8 months ago
📄 .gitmodules	First commit	8 months ago
📄 .project	First commit	8 months ago
📄 README.md	First commit	8 months ago
📄 build.properties	First commit	8 months ago
📄 build.xml	First commit	8 months ago

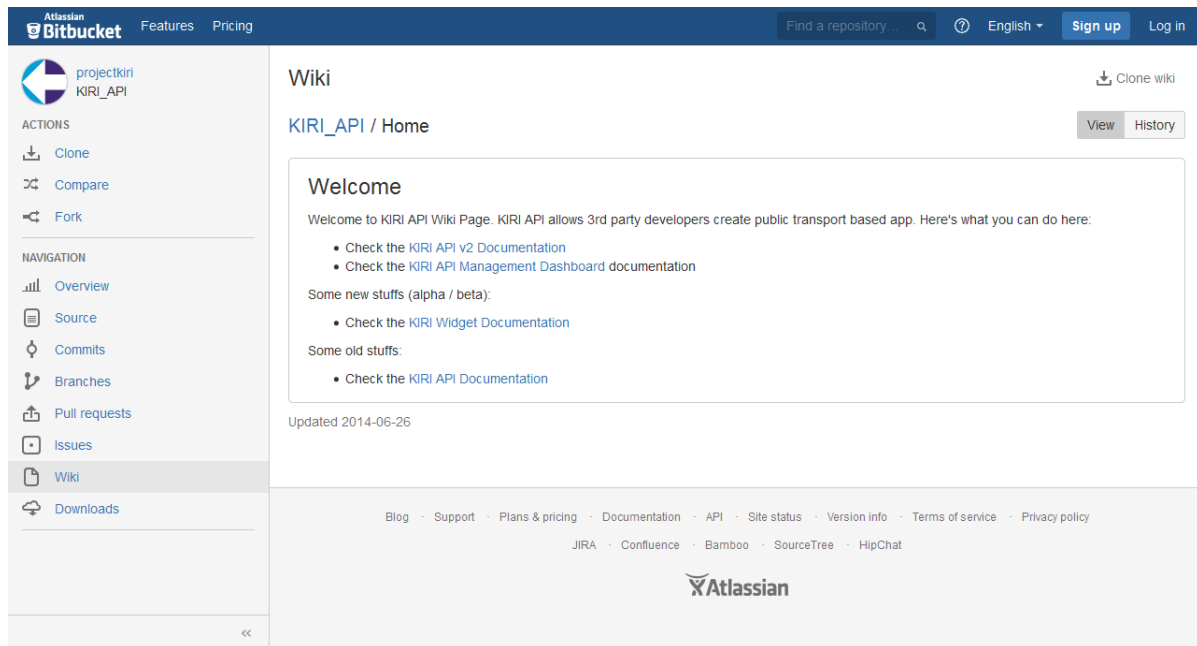
Gambar 1: Struktur Kode KIRI

pascalalfadian First commit		Latest commit b650bfa on Mar 20
..		
📁 bukitjarian	First commit	8 months ago
📁 docs	First commit	8 months ago
📄 index.php	First commit	8 months ago

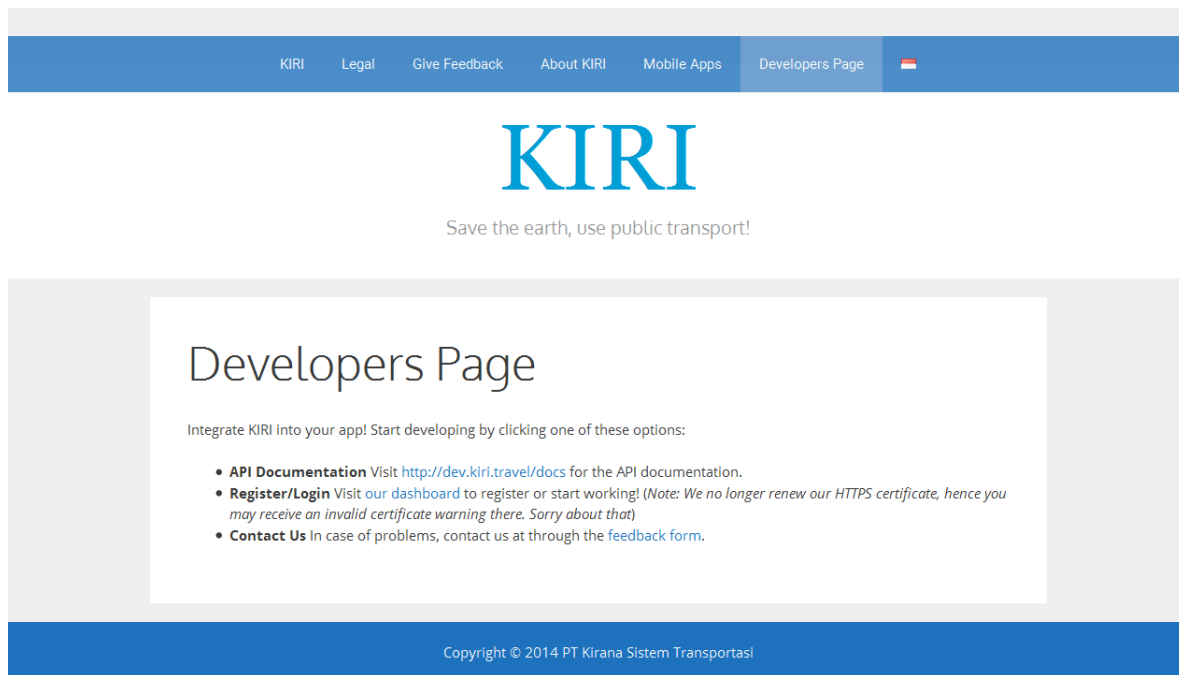
Gambar 2: Struktur *folder* “public\_html\_dev”

Penelitian ini berfokus pada pembahasan KIRI *Dashboard*, untuk itu bagian yang akan dianalisa lebih mendalam adalah pada bagian “public\_html\_dev” (gambar 2) dan beberapa bagian-bagian pendukung untuk membangun KIRI *Dashboard*. *Folder* “public\_html\_dev” memiliki struktur sebagai berikut:









- (a) *Folder* “bukitjarian”, berisi mengenai *file-file* untuk membangun KIRI *Dashboard* (gambar 5).
- (b) *Folder* “docs”, berisi sebuah *file* “index.php” yang bila dijalankan akan mengarahkan pengguna ke alamat [https://bitbucket.org/projectkiri/kiri\\_api/wiki/Home](https://bitbucket.org/projectkiri/kiri_api/wiki/Home) (gambar 3). Halaman tersebut berisi mengenai dokumentasi sistem KIRI.
- (c) *File* “index.php”, bila *file* ini dieksekusi maka akan mengarahkan pengguna ke alamat <http://static.kiri.travel/developer> (gambar 4). Halaman tersebut berisi informasi mengenai alamat KIRI *Dashboard*, dokumentasi KIRI, dan halaman untuk memberikan masukan kepada kontributor sistem KIRI.



Gambar 3: Halaman dokumentasi KIRI



Gambar 4: Halaman developer KIRI

 <b>pascalalfadian</b> First commit	Latest commit b650bfa on Mar 20
..	
 <a href="#">bukitjariangwt</a>	First commit 8 months ago
 <a href="#">images</a>	First commit 8 months ago
 <a href="#">BukitJarianGWT.css</a>	First commit 8 months ago
 <a href="#">getplaces.php</a>	First commit 8 months ago
 <a href="#">gettracks.php</a>	First commit 8 months ago
 <a href="#">handle.php</a>	First commit 8 months ago
 <a href="#">index.html</a>	First commit 8 months ago

Gambar 5: Struktur *folder* “bukitjarian”

Pada direktori “public\_html\_dev/bukitjarian” (gambar 5) terdapat bermacam-macam *file* dan *folder*. Berdasarkan wawancara dengan kontributor kode, terdapat dua bagian *file* dan *folder* yang berperan penting dalam membangun KIRI *Dashboard*:

- “index.html”, “bukitjariangwt/”, dan “images/”. *File* dan *folder* ini dibangun dengan menggunakan perangkat lunak GWT. GWT (*Google Web Toolkit*) adalah perangkat lunak yang digunakan untuk membangun dan melakukan optimasi suatu aplikasi berbasis web[4]. Dengan GWT, pengguna tidak perlu ahli dalam menggunakan bahasa pemrograman berbasis web seperti JavaScript. GWT menggunakan bahasa Java dalam pembuatannya dan akan melakukan konversi kode Java tersebut menjadi HTML dan JavaScript. Hasil konversi menggunakan GWT akan mengalami *obfuscate* (pengacakan) sehingga sulit untuk dianalisa. Namun demikian, *file-file* ini bersifat statis dan tidak memerlukan operasi khusus di *server*, sehingga dapat disalin (*copy*) apa adanya.
- “handle.php” merupakan kode pada sisi *server* yang bertugas untuk melayani permintaan-permintaan dari *browser* yang dieksekusi oleh “index.html”.

Seperti disebutkan sebelumnya, *file* “handle.php” berfungsi untuk melayani berbagai jenis permintaan yang dikirimkan oleh “index.html”. *File* ini dapat dibagi menjadi 16 bagian yang masing-masing melayani sebuah permintaan tertentu.

Berikut adalah kode situs web KIRI *Dashboard Server Side*:

Listing 1: handle.php

```

1  <?php
2  require_once '../etc/Utils.php';
3  require_once '../etc/constants.php';
4  require_once '../etc/PasswordHash.php';
5
6  start_working();
7
8  $mode = retrieve_from_post($proto_mode);
9
10 // Initializes MySQL and check for session
11 init_mysql();
12 if ($mode != $proto_mode_login && $mode != $proto_mode_logout && $mode != $proto_mode_register) {
13     $sessionid = addslashes(retrieve_from_post($proto_sessionid));
14     // Clear expired sessions
15     mysqli_query($global_mysql_link, "DELETE FROM sessions WHERE lastSeen < (NOW() - INTERVAL_
        $session_expiry_interval_mysql)"); or
16     die_nice('Failed to clean expired sessions: ' . mysqli_error($global_mysql_link), true);
17     $result = mysqli_query($global_mysql_link, "SELECT users.email, users.privilegeRoute, users.
        privilegeApiUsage FROM users LEFT JOIN sessions ON users.email = sessions.email WHERE sessions.
        sessionid = '$sessionid'"); or
18     die_nice('Failed to get user session information: ' . mysqli_error($global_mysql_link), true);
19     if (mysqli_num_rows($result) == 0) {
20         deinit_mysql();
21         // Construct json - session expired.
22         $json = array(
23             $proto_status => $proto_status_sessionexpired,
24         );
25         print(json_encode($json));
26         exit(0);

```

```

27     }
28     $columns = mysqli_fetch_row($result);
29     $active_userid = $columns[0];
30     $privilege_route = $columns[1] != '0';
31     $privilege_apiUsage = $columns[2] != '0';
32 }
33
34 if ($mode == $proto_mode_login) {
35     $userid = addslashes(retrieve_from_post($proto_userid));
36     $plain_password = addslashes(retrieve_from_post($proto_password));
37     if (strlen($userid) > $maximum_userid_length) {
38         return_invalid_credentials("User_ID_length_is_more_than_allowed_( ". strlen($userid) . " )");
39     }
40     if (strlen($plain_password) > $maximum_password_length) {
41         return_invalid_credentials("Password_length_is_more_than_allowed_( ". strlen($password) . " )");
42     }
43
44     // Retrieve the user information
45     $result = mysqli_query($global_mysql_link, "SELECT * FROM users WHERE email='$userid'" ) or
46         die_nice('Failed_to_verify_userid: ' . mysqli_error($global_mysql_link), true);
47     if (mysqli_num_rows($result) == 0) {
48         deinit_mysql();
49         return_invalid_credentials("User_id_not_found: '$userid'");
50     }
51     $userdata = mysqli_fetch_assoc($result);
52
53     // Check against the stored hash.
54     $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
55     if (!$hasher->CheckPassword($plain_password, $userdata['password'])) {
56         log_statistic("$apikey_kiri", 'LOGIN', $userid . '/FAIL');
57         deinit_mysql();
58         return_invalid_credentials("Password_mismatch_for_$userid");
59     }
60
61     log_statistic("$apikey_kiri", 'LOGIN', $userid . '/SUCCESS');
62
63     // Create session id
64     $sessionid = generate_sessionid();
65     mysqli_query($global_mysql_link, "INSERT INTO sessions (sessionid, email) VALUES ('$sessionid', '$userid' )") or
66         die_nice('Failed_to_generate_session: ' . mysqli_error($global_mysql_link), true);
67
68     // Construct privilege lists
69     $privileges = '';
70     if ($userdata['privilegeRoute'] != 0) {
71         $privileges .= ", $proto_privilege_route";
72     }
73     if ($userdata['privilegeApiUsage'] != 0) {
74         $privileges .= ", $proto_privilege_apiUsage";
75     }
76     if (strlen($privileges) > 0) {
77         $privileges = substr($privileges, 1);
78     }
79
80     // Construct json.
81     $json = array(
82         $proto_status => $proto_status_ok,
83         $proto_sessionid => $sessionid,
84         $proto_privileges => $privileges
85     );
86
87     deinit_mysql();
88     print(json_encode($json));
89 } elseif ($mode == $proto_mode_logout) {
90     $sessionid = addslashes(retrieve_from_post($proto_sessionid));
91
92     // Remove the session information
93     $result = mysqli_query($global_mysql_link, "DELETE FROM sessions WHERE sessionid='$sessionid'" ) or
94         die_nice('Failed_to_logout_sessionid: '$sessionid: ' . mysqli_error($global_mysql_link), true);
95     deinit_mysql();
96     well_done();
97 } elseif ($mode == $proto_mode_add_track) {
98     check_privilege($privilege_route);
99     $trackid = addslashes(retrieve_from_post($proto_trackid));
100    $trackname = addslashes(retrieve_from_post($proto_trackname));
101    $tracktype = addslashes(retrieve_from_post($proto_tracktype));
102    $penalty = addslashes(retrieve_from_post($proto_penalty));
103    $internalinfo = addslashes(retrieve_from_post($proto_internalinfo, false)) or $internalinfo = '';
104
105    // Check if the id is already existed
106    $result = mysqli_query($global_mysql_link, "SELECT trackId FROM tracks WHERE trackId='$trackid'" ) or
107        die_nice('Failed_to_check_trackid_existence: ' . mysqli_error($global_mysql_link), true);
108    if (mysqli_num_rows($result) == 0) {
109        mysqli_query($global_mysql_link, "INSERT INTO tracks (trackId, trackTypeId, trackName, penalty, internalInfo) VALUES ('$trackid', '$tracktype', '$trackname', '$penalty', '$internalinfo' )") or
110            die_nice('Failed_to_add_a_new_track: ' . mysqli_error($global_mysql_link), true);
111        update_trackversion();
112    } else {
113        die_nice("The trackId '$trackid' already existed.", true);
114    }
115    deinit_mysql();
116    well_done();
117 } elseif ($mode == $proto_mode_update_track) {
118     check_privilege($privilege_route);
119     $trackid = addslashes(retrieve_from_post($proto_trackid));

```

```

120 $newtrackid = addslashes(retrieve_from_post($proto_new_trackid));
121 $tracktype = addslashes(retrieve_from_post($proto_tracktype));
122 $trackname = addslashes(retrieve_from_post($proto_trackname));
123 $internalinfo = addslashes(retrieve_from_post($proto_internalinfo, false)) or $internalinfo = '';
124 $pathloop = retrieve_from_post($proto_pathloop) == 'true' ? 1 : 0;
125 $penalty = addslashes(retrieve_from_post($proto_penalty));
126 $transfernodes = retrieve_from_post($proto_transfernodes, false);
127
128 // When changed, check if the id is already existed
129 if ($newtrackid != $trackid) {
130     $result = mysqli_query($global_mysql_link, "SELECT trackId FROM tracks WHERE trackId='$newtrackid'")
131     ) or
132     die_nice('Failed to check trackid existence: ' . mysqli_error($global_mysql_link), true);
133     if (mysqli_num_rows($result) != 0) {
134         die_nice("The new trackId '$newtrackid' already existed.", true);
135     }
136 }
137 mysqli_query($global_mysql_link, "UPDATE tracks SET trackTypeId='$tracktype', trackId='$newtrackid',
138 trackName='$trackname', internalInfo='$internalinfo', pathloop='$pathloop', penalty='$penalty'
139 WHERE trackId='$trackid'") or
140 die_nice('Failed to update the track: ' . mysqli_error($global_mysql_link));
141 if (!is_null($transfernodes)) {
142     $transfernodes = addslashes($transfernodes);
143     mysqli_query($global_mysql_link, "UPDATE tracks SET transferNodes='$transfernodes' WHERE trackId=
144     '$trackid'") or
145     die_nice('Failed to update the track: ' . mysqli_error($global_mysql_link));
146 }
147 update_trackversion();
148 deinit_mysql();
149 well_done();
150 } elseif ($mode == $proto_mode_list_tracks) {
151     check_privilege($privilege_route);
152     // Retrieve track list from database
153     $result = mysqli_query($global_mysql_link, 'SELECT trackTypeId, trackId, trackName FROM tracks ORDER BY
154     trackTypeId, trackId') or
155     die('Cannot retrieve the track names from database');
156     $track_list = array();
157     while ($row = mysqli_fetch_row($result)) {
158         $track_list[] = array($row[1], htmlspecialchars($row[0] . '/' . $row[2]));
159     }
160     // Retrieve track types list result from database
161     $result = mysqli_query($global_mysql_link, 'SELECT trackTypeId, name FROM tracktypes ORDER BY
162     trackTypeId') or
163     die('Cannot retrieve the track types from database');
164     $tracktype_list = array();
165     while ($row = mysqli_fetch_row($result)) {
166         $tracktype_list[] = array($row[0], htmlspecialchars($row[1]));
167     }
168     // Construct json.
169     $json = array(
170         $proto_status => $proto_status_ok,
171         $proto_trackslist => $track_list,
172         $proto_tracktypeslist => $tracktype_list
173     );
174     deinit_mysql();
175     print(json_encode($json));
176 } elseif ($mode == $proto_mode_getdetails_track) {
177     check_privilege($privilege_route);
178     $trackid = addslashes(retrieve_from_post($proto_trackid));
179
180     // Retrieve result from database and construct in XML format
181     $result = mysqli_query($global_mysql_link, "SELECT trackTypeId, trackName, internalInfo, AsText(geodata
182     ), pathloop, penalty, transferNodes FROM tracks WHERE trackId='$trackid'") or
183     die_nice("Can't retrieve the track details from database: " . mysqli_error($global_mysql_link),
184     true);
185     $i = 0;
186     $row = mysqli_fetch_row($result);
187     if ($row == FALSE) {
188         die_nice("Can't find track information for '$trackid'", true);
189     }
190     $geodata = lineStringToLatLngArray($row[3]);
191     // Construct json.
192     $json = array(
193         $proto_status => $proto_status_ok,
194         $proto_trackid => $trackid,
195         $proto_tracktype => $row[0],
196         $proto_trackname => $row[1],
197         $proto_internalinfo => $row[2],
198         $proto_geodata => $geodata,
199         $proto_pathloop => ($row[4] > 0 ? true : false),
200         $proto_penalty => doubleval($row[5]),
201         $proto_transfernodes => is_null($row[6]) ? array('0-' . (count($geodata) - 1)) : split(',', $row[6])
202     );
203     deinit_mysql();
204     print(json_encode($json));
205 } elseif ($mode == $proto_mode_cleargeodata) {
206     check_privilege($privilege_route);
207     $trackid = addslashes(retrieve_from_post($proto_trackid));
208
209     mysqli_query($global_mysql_link, "UPDATE tracks SET geodata=NULL, transferNodes=NULL WHERE trackId=
210     '$trackid'") or

```

```

205     die_nice('Failed_to_clear_the_geodata:~' . mysqli_error($global_mysql_link), true);
206
207     deinit_mysql();
208     well_done();
209 } elseif ($mode == $proto_mode_importkml) {
210     check_privilege($privilege_route);
211     $trackid = addslashes(retrieve_from_post($proto_trackid));
212     // Import KML file into a geodata in database
213     if ($FILES[$proto_uploadedfile][ 'error' ] != UPLOAD_ERR_OK) {
214         die_nice("Server_script_is_unable_to_retrieve_the_file ,_with_PHP's_UPLOAD_ERR_XXX_code:~" . $FILES[
                $proto_uploadedfile][ 'error' ], true);
215     }
216     if ($FILES[$proto_uploadedfile][ 'size' ] > $max_filesize) {
217         die_nice("Uploaded_file_size_is_greater_than_maximum_size_allowed_($max_filesize)", true);
218     }
219     $file = fopen($FILES[$proto_uploadedfile][ 'tmp_name' ], "r") or die_nice('Unable_to_open_uploaded_file ',
                true);
220     $haystack = '';
221     while ($line = fgets($file)) {
222         $haystack .= trim($line);
223     }
224     $num_matches = preg_match_all("/<LineString>.*<coordinates>(.*?)</coordinates>.*</LineString>/i",
                $haystack, $matches, PREG_PATTERN_ORDER);
225     if ($num_matches != 1) {
226         die_nice("The_KML_file_must_contain_exactly_one_<coordinate>_tag_inside_one_<LineString>_tag._But_I_
                found_$num_matches_occurrences", true);
227     }
228     fclose($file);
229
230     // Start constructing output
231     $output = 'LINESTRING(';
232     $points = preg_split('/\s+/', $matches[1][0]);
233     for ($i = 0, $size = sizeof($points); $i < $size; $i++) {
234         list($x, $y, $z) = preg_split('/\s*,\s*/', $points[$i]);
235         if ($i > 0) {
236             $output .= ',';
237         }
238         $output .= "$x,$y";
239     }
240     $output .= ')';
241     mysqli_query($global_mysql_link, "UPDATE_tracks_SET_geodata=GeomFromText('$output'),_transferNodes=NULL
                _WHERE_trackId='$trackid'") or
242         die_nice("Error_updating_the_geodata:~" . mysqli_error($global_mysql_link), true);
243     update_trackversion();
244     deinit_mysql();
245     well_done();
246 } elseif ($mode == $proto_mode_delete_track) {
247     check_privilege($privilege_route);
248     $trackid = addslashes(retrieve_from_post($proto_trackid));
249
250     init_mysql();
251
252     // Check if the id is already existed
253     mysqli_query($global_mysql_link, "DELETE_FROM_tracks_WHERE_trackId='$trackid'") or
254         die_nice('Failed_to_delete_track_$trackid:~' . mysqli_error($global_mysql_link), true);
255     if (mysqli_affected_rows($global_mysql_link) == 0) {
256         die_nice("The_track_$trackid_was_not_found_in_the_database", true);
257     }
258     update_trackversion();
259     deinit_mysql();
260     well_done();
261 } elseif ($mode == $proto_mode_list_apikeys) {
262     check_privilege($privilege_apiUsage);
263     // Retrieve api key list from database
264     $result = mysqli_query($global_mysql_link, "SELECT_verifier ,_domainFilter ,_description_FROM_apikeys_
                WHERE_email='$active_userid'_ORDER_BY_verifier") or
265         die_nice('Cannot_retrieve_the_API_keys_list_from_database:~' . mysqli_error($global_mysql_link));
266     $apikey_list = array();
267     while ($row = mysqli_fetch_row($result)) {
268         $apikey_list[] = array($row[0], $row[1], $row[2]);
269     }
270
271     // Construct json.
272     $json = array(
273         $proto_status => $proto_status_ok ,
274         $proto_apikeys_list => $apikey_list ,
275     );
276
277     deinit_mysql();
278     print(json_encode($json));
279 } elseif ($mode == $proto_mode_add_apikey) {
280     check_privilege($privilege_apiUsage);
281     $domainfilter = addslashes(retrieve_from_post($proto_domainfilter));
282     $description = addslashes(retrieve_from_post($proto_description));
283     $apikey = generate_apikey();
284
285     // Retrieve api key list from database
286     $result = mysqli_query($global_mysql_link, "INSERT INTO_apikeys(verifier ,_email ,_domainFilter ,_
                description)_VALUES('$apikey ','$active_userid ','$domainfilter ','$description')") or
287         die_nice('Cannot_insert_a_new_api_key:~' . mysqli_error($global_mysql_link));
288
289     log_statistic("$apikey_kiri", 'ADDAPIKEY', $userid . $apikey);
290
291     // Construct json.
292     $json = array(

```

```

293         $proto_status => $proto_status_ok,
294         $proto_verifier => $apikey,
295     );
296
297     deinit_mysql();
298     print(json_encode($json));
299 } elseif ($mode == $proto_mode_update_apikey) {
300     check_privilege($privilege_apiUsage);
301     $apikey = addslashes(retrieve_from_post($proto_verifier));
302     $domainfilter = addslashes(retrieve_from_post($proto_domainfilter));
303     $description = addslashes(retrieve_from_post($proto_description));
304     // Ensure that this user has access to the apikey
305     $result = mysqli_query($global_mysql_link, "SELECT_email_FROM_apikeys_WHERE_verifier='$apikey'" ) or
306         die_nice('Cannot_check_API_key_owner:_' . mysqli_error($global_mysql_link));
307     while ($row = mysqli_fetch_row($result)) {
308         if ($row[0] != $active_userid) {
309             die_nice("User_$active_userid_does_not_have_privilege_to_update_API_Key_$apikey");
310         }
311     }
312     mysqli_query($global_mysql_link, "UPDATE_apikeys_SET_domainFilter='$domainfilter',_description='
313         $description '_WHERE_verifier='$apikey'" ) or
314         die_nice('Failed_to_update_API_Key:_' . mysqli_error($global_mysql_link));
315
316     deinit_mysql();
317     well_done();
318 } elseif ($mode == $proto_mode_register) {
319     $email = addslashes(retrieve_from_post($proto_userid));
320     $fullname = addslashes(retrieve_from_post($proto_fullname));
321     $company = addslashes(retrieve_from_post($proto_company));
322
323     // Check if the email has already been registered.
324     $result = mysqli_query($global_mysql_link, "SELECT_email_FROM_users_WHERE_email='$email'" ) or
325         die_nice('Cannot_check_user_id_existence:_' . mysqli_error($global_mysql_link));
326     if (mysqli_num_rows($result) > 0) {
327         die_nice("Oops!_Email_$email_has_already_registered._Please_check_your_mailbox_or_contact_
328             hello@kiri.travel");
329     }
330
331     // Generate and send password
332     $password = generate_password();
333     $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
334     $passwordHash = $hasher->HashPassword($password);
335     mysqli_query($global_mysql_link, "INSERT INTO users(email,_password,_privilegeApiUsage,_fullName,_
336         company)_VALUES('$email',_'$passwordHash',_1,_'$fullname',_'$company')") or
337         die_nice('Cannot_add_new_user_$email:_' . mysqli_error($global_mysql_link));
338     sendPassword($email, $password);
339
340     log_statistic("$apikey_kiri", 'REGISTER', "$email/$fullname/$company");
341
342     deinit_mysql();
343     well_done();
344 } elseif ($mode == $proto_mode_getprofile) {
345     $email = $active_userid;
346
347     $result = mysqli_query($global_mysql_link, "SELECT_fullName,_company_FROM_users_WHERE_email='$email
348         '" ) or
349         die_nice('Cannot_retrieve_user_details:_' . mysqli_error($global_mysql_link));
350     if ($row = mysqli_fetch_row($result)) {
351         $fullname = $row[0];
352         $company = $row[1];
353     } else {
354         die_nice("User_$email_not_found_in_database.");
355     }
356
357     deinit_mysql();
358     // Construct json.
359     $json = array(
360         $proto_status => $proto_status_ok,
361         $proto_fullname => $fullname,
362         $proto_company => $company
363     );
364
365     print(json_encode($json));
366 } elseif ($mode == $proto_mode_update_profile) {
367     $email = $active_userid;
368     $password = addslashes(retrieve_from_post($proto_password, false));
369     $fullname = addslashes(retrieve_from_post($proto_fullname));
370     $company = addslashes(retrieve_from_post($proto_company));
371
372     // Updates password if necessary
373     if (!is_null($password) && $password != "") {
374         $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
375         $passwordHash = $hasher->HashPassword($password);
376         mysqli_query($global_mysql_link, "UPDATE_users_SET_password='$passwordHash'_WHERE_email='$email'" )
377             or
378             die_nice('Cannot_update_password_for_$email:_' . mysqli_error($global_mysql_link));
379     }
380     mysqli_query($global_mysql_link, "UPDATE_users_SET_fullName='$fullname',_company='$company'_WHERE_email
381         ='$email'" ) or
382         die_nice('Cannot_update_profile_for_$email:_' . mysqli_error($global_mysql_link));
383
384     deinit_mysql();
385     well_done();
386 } else {
387     die_nice("Mode_not_understood:_" . $mode . " ", true);

```



```

382 }
383
384 /**
385  * Return invalid credential error, close mysql connection, and exit.
386  * @param string $logmessage the message to record in the log file.
387  */
388 function return_invalid_credentials($logmessage) {
389     global $proto_status, $proto_status_credentialfail, $errorlog_file, $global_mysqli_link;
390     $ip_address = $_SERVER['REMOTE_ADDR'];
391     log_error("Login failed (IP=$ip_address): $logmessage", '...' . $errorlog_file);
392     $json = array(
393         $proto_status => $proto_status_credentialfail);
394     print(json_encode($json));
395     mysqli_close($global_mysqli_link);
396     exit(0);
397 }
398
399 /**
400  *
401  * Simply checks the input parameter, when false do default action
402  * to return "user does not have privilege"
403  * @param boolean $privilege if false will return error
404  */
405 function check_privilege($privilege) {
406     if (!$privilege) {
407         die_nice("User doesn't have enough privilege to perform the action.", true);
408     }
409 }
410
411 /**
412  * Scans a directory and remove files that have not been modified for max_age
413  * @param string $path the path to the directory to clean
414  * @param int $max_age maximum age of the file in seconds
415  * @return boolean true if okay, false if there's an error.
416  */
417 function clean_temporary_files($path, $max_age) {
418     $currenttime = time();
419     if ($dirhandle = opendir($path)) {
420         while (($file = readdir($dirhandle)) != FALSE) {
421             $fullpath = "$path/$file";
422             if (is_file($fullpath) && $currenttime - filemtime($fullpath) > $max_age) {
423                 if (!unlink($fullpath)) {
424                     return FALSE;
425                 }
426             }
427         }
428         return TRUE;
429     } else {
430         return FALSE;
431     }
432 }
433
434 ?>

```

### • Bagian Pemeriksaan *Login*

Bagian ini terletak di baris 12-32 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi untuk semua “mode” pada permintaan POST kecuali “mode=login”, “mode=logout”, dan “mode=register”. Bagian ini berfungsi untuk memeriksa apakah pengguna sudah melakukan *login* terlebih dahulu untuk melakukan aksi-aksi tertentu.

Bagian ini diawali dengan memeriksa apakah pengguna memberikan *session id* pada permintaan atau tidak (baris 13). Setelah itu, program akan membersihkan sesi-sesi di *database* yang sudah kadaluwarsa (baris 14-16). Baris 17-18 memeriksa apakah *session* yang dikirimkan dari permintaan masih *valid* di *database* atau tidak. Jika tidak, maka bagian ini akan mengembalikan respon yang menyatakan bahwa sesi tidak *valid* dan permintaan tidak dapat dilanjutkan (baris 19-27). Jika *valid*, maka bagian ini akan menginisialisasi beberapa variabel yang menampung *privilege* dari pengguna yang aktif (baris 28-31).

### • Bagian *Login*

Bagian ini terletak di baris 34-89 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=login” pada permintaan POST. Bagian ini berfungsi untuk melakukan otentikasi pengguna terhadap *server KIRI Dashboard*. Bagian ini akan menentukan apakah pengguna memiliki hak akses terhadap *KIRI Dashboard* apakah tidak.

Bagian ini diawali dengan memeriksa apakah pengguna mengirimkan *userid* dan *password* dengan ukuran yang sesuai apa tidak (baris 35-42). Setelah itu, program akan mengambil data informasi pengguna (berdasarkan *userid*) ke *database* sistem (baris 45-51). Bila data pengguna tidak

ditemukan maka program akan mengembalikan pesan kesalahan (baris 49). Jika informasi pengguna ditemukan, maka selanjutnya *password* yang dikirimkan pengguna akan dicek kecocokannya dengan *password* yang tersimpan dalam *database* (baris 54-55). Hasil kecocokan tersebut akan dicatat ke dalam data statistik *server* (baris 56 atau 61). Bila *password* cocok, maka server akan membangun sebuah *session id* (baris 64-66) dan memberikan hak akses tertentu kepada pengguna (baris 68-78). Terakhir, *server* akan membangun data JSON (baris 81-85) untuk dikirimkan ke pengguna (baris 88) sebagai pesan keberhasilan pengguna dalam melakukan otentikasi terhadap *server*.

- **Bagian Logout**

Bagian ini terletak di baris 89-97 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=logout” pada permintaan POST. Bagian ini berfungsi untuk menghentikan hubungan otentikasi dengan *server* (menghilangkan hak akses). Hal tersebut bertujuan agar hak akses yang dimiliki pengguna tidak digunakan sembarangan oleh pengguna lain yang tidak berwenang.

Bagian ini diawali dengan memeriksa apakah pengguna memberikan *session id* pada permintaan atau tidak (baris 90). Setelah itu, program akan membersihkan sesi-sesi (sesuai dengan *session id* pengguna) yang terdapat dalam *database* (baris 93-95). Terakhir, *server* akan mengirimkan pesan dalam format JSON (baris 96) sebagai penanda bahwa pengguna berhasil melakukan *logout*.

- **Bagian Menambahkan Rute**

Bagian ini terletak di baris 97-117 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=addtrack” pada permintaan POST. Bagian ini berfungsi untuk menambahkan sebuah rute jalan yang dapat ditempuh oleh kendaraan umum tertentu (contoh: angkot).

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses untuk menambahkan rute atau tidak (baris 98). Lalu memeriksa apakah pengguna mengirimkan data *trackid*, *trackname*, *tracktype*, *penalty*, dan *internalinfo* pada permintaan atau tidak (baris 99-103). Setelah itu, program akan mengecek apakah rute jalan yang ingin ditambahkan pengguna sudah ada atau belum di *database* (106-114). Bila rute jalan belum ada, maka rute jalan akan ditambahkan ke dalam *database* (baris 109) dan *server* akan mengirimkan pesan dalam format JSON (baris 116) sebagai penanda bahwa pengguna berhasil menambahkan rute jalan.

- **Bagian Mengubah Rute**

Bagian ini terletak di baris 117-146 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=updatetrack” pada permintaan POST. Bagian ini berfungsi untuk mengubah data sebuah rute jalan yang dapat ditempuh oleh kendaraan umum tertentu (contoh: angkot).

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses untuk mengubah rute atau tidak (baris 118). Lalu memeriksa apakah pengguna mengirimkan data *trackid*, *newtrackid*, *trackname*, *tracktype*, *penalty*, *pathloop*, *transferrnodes* dan *internalinfo* pada permintaan atau tidak (baris 119-126). Setelah itu, *server* akan mengecek apakah rute yang ingin diubah pengguna sudah memenuhi aturan (*trackid* harus sama dengan *newtrackid*) atau tidak (129-143). Bila rute yang ingin diubah maka *server* akan mengirimkan pesan dalam format JSON sebagai penanda bahwa pengguna berhasil mengubah rute jalan.

- **Bagian Melihat Daftar Rute**

Bagian ini terletak di baris 146-172 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=listtracks” pada permintaan POST. Bagian ini berfungsi untuk memberikan daftar rute jalan yang terdapat dalam *database* sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap rute jalan atau tidak (baris 147). Setelah itu program akan mengambil data daftar rute jalan yang terdapat pada *database* sistem KIRI (baris 149-154). Lalu program juga akan mengambil data daftar tipe rute jalan dari *database* (baris 156-161). Data-data yang diperoleh program (rute jalan dan tipe rute jalan) akan diubah formatnya menjadi sebuah data JSON (baris 164-168). Terakhir, program akan mengirimkan data dalam format JSON tersebut ke pengguna (baris 171).

- **Bagian Melihat Informasi Rute secara Detail**

Bagian ini terletak di baris 172-200 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=getdetailstrack” pada permintaan POST. Bagian ini berfungsi untuk memberikan informasi detail tentang suatu rute jalan.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap rute jalan atau tidak (baris 173). Lalu program akan memeriksa apakah pengguna memberikan *trackid* pada permintaan atau tidak (baris 174). Selanjutnya program akan mengambil data dari *database* sistem KIRI (baris 177-184). Data yang diperoleh dari *database* tersebut akan diubah formatnya ke dalam format JSON (baris 186-196). Terakhir, program akan mengirimkan data dalam format JSON tersebut ke pengguna (baris 199).

- **Bagian Menghapus Data Geografis suatu Rute**

Bagian ini terletak di baris 200-209 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=cleargeodata” pada permintaan POST. Bagian ini berfungsi untuk menghapus data geografis suatu rute jalan yang terdapat dalam *database* sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap rute jalan atau tidak (baris 201). Lalu program akan memeriksa apakah pengguna memberikan *trackid* pada permintaan atau tidak (baris 202). Program akan langsung menghapus data geografis rute jalan sesuai dengan *trackid* permintaan pengguna jika *trackid* tersebut terdapat dalam *database* sistem KIRI (baris 204-205). Terakhir, program akan mengirimkan pesan keberhasilan dalam format JSON kepada pengguna (baris 208).

- **Bagian Impor Data KML**

Bagian ini terletak di baris 209-246 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=importkml” pada permintaan POST. Bagian ini berfungsi untuk menambahkan data geografis suatu rute dimana data yang ditambahkan berasal dari sebuah *file* dengan format KML (*Keyhole Markup Language*). KML adalah format *file* yang digunakan untuk menampilkan data geografis dalam aplikasi pemetaan[5].

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap rute jalan atau tidak (baris 210). Lalu program akan memeriksa apakah pengguna memberikan *trackid* pada permintaan atau tidak (baris 211). Selanjutnya program akan memeriksa apakah *file* pengguna memberikan *file* dengan format sesuai atau tidak (baris 213-218). Pada baris 219-228 program akan mengambil data *LineString* yang terdapat pada *file* dengan menggunakan *regular expression* (baris 224). *Regular expression* adalah karakter atau kata spesial yang digunakan untuk menjelaskan pola pencarian[6]. Baris 231-239 program akan membangun data *LineString* yang semula dalam format KML menjadi format WKT. Program akan menambahkan data *LineString* dalam WKT tersebut ke dalam *database* sesuai dengan *trackid* yang diberikan pengguna (baris 241-243). Terakhir, program akan mengirimkan pesan dalam format JSON sebagai penanda bahwa pengguna berhasil melakukan import data KML (baris 245).

- **Bagian Menghapus Rute**

Bagian ini terletak di baris 246-261 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi

hanya jika terdapat parameter “*mode=deletetrack*” pada permintaan POST. Bagian ini berfungsi untuk menghapus suatu rute jalan yang terdapat dalam sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap rute jalan atau tidak (baris 247). Lalu program akan memeriksa apakah pengguna memberikan *trackid* pada permintaan atau tidak (baris 248). Program akan memeriksa apakah terdapat *trackid* yang sesuai dengan *trackid* yang ada pada *database* KIRI (baris 250-259). Bila terdapat *trackid* yang sesuai, maka program akan menghapus rute jalan tersebut. Terakhir, program akan mengirimkan pesan keberhasilan dalam format JSON kepada pengguna (baris 260).

- **Bagian Melihat Daftar API *Keys***

Bagian ini terletak di baris 261-279 dari “*handle.php*” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “*mode=listapikeys*” pada permintaan POST. Bagian ini berfungsi untuk memberikan daftar API *keys* yang terdapat dalam database sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap penggunaan API atau tidak (baris 262). Setelah itu program akan mengambil data daftar API *keys* yang terdapat pada *database* sistem KIRI (baris 264-269). Data-data yang diperoleh program akan diubah formatnya menjadi sebuah data JSON (baris 272-275). Terakhir, program akan mengirimkan data dalam format JSON tersebut ke pengguna (baris 278).

- **Bagian Menambahkan API *Key***

Bagian ini terletak di baris 279-299 dari “*handle.php*” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “*mode=addapikey*” pada permintaan POST. Bagian ini berfungsi untuk menambahkan sebuah data API *key* ke dalam sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap API *keys* atau tidak (baris 280). Lalu memeriksa apakah pengguna mengirimkan data *domainfilter* dan *description* pada permintaan atau tidak (baris 281-282). Setelah itu, program akan membangun sebuah API *key* secara acak (baris 283). Program akan menambahkan data API *key* sesuai dengan data yang dikirimkan pengguna ke dalam *database* KIRI (baris 286) dan mencatat proses penambahan tersebut ke dalam *database* (baris 289). Terakhir, program akan membangun sebuah pesan keberhasilan dalam format JSON (baris 292-295) dan mengirimkan pesan tersebut kepada pengguna (baris 298).

- **Bagian Mengubah API *Key***

Bagian ini terletak di baris 299-317 dari “*handle.php*” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “*mode=updateapikey*” pada permintaan POST. Bagian ini berfungsi untuk mengubah data sebuah API *key* pada *database* sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memiliki hak akses terhadap API *keys* atau tidak (baris 300). Lalu memeriksa apakah pengguna mengirimkan data *apikey*, *domainfilter*, dan *description* pada permintaan atau tidak (baris 301-303). Setelah itu, program akan memeriksa apakah pengguna yang bersangkutan adalah pemilik API *key* yang ingin diubah atau bukan (baris 305-311). Lalu program mengubah data API *key* yang terdapat dalam database (baris 312) sesuai dengan permintaan pengguna. Terakhir, program akan mengirimkan pesan dalam format JSON sebagai penanda bahwa pengguna berhasil mengubah rute jalan.

- **Bagian *Register***

Bagian ini terletak di baris 317-341 dari “*handle.php*” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “*mode=register*” pada permintaan POST. Bagian ini berfungsi untuk melakukan pendaftaran sebagai pengguna KIRI *Dashboard*. Pendaftaran ini berguna agar pengguna bisa mendapatkan hak akses terhadap fitur-fitur yang terdapat dalam KIRI *Dashboard*.

Bagian ini diawali dengan memeriksa apakah pengguna memberikan *email*, *fullname*, dan *company* pada permintaan atau tidak (baris 318-320). Setelah itu, program akan memeriksa *database* apakah data pengguna yang ingin dibuat sudah ada atau belum (baris 323-327). Bila belum ada, maka program akan membangun sebuah sandi secara acak untuk pengguna (baris 330-332). Program akan menambahkan data pengguna beserta sandi yang telah dibangun ke dalam *database* sistem KIRI (baris 333). Sandi yang telah dibangun program juga dikirimkan ke alamat *email* pengguna (baris 335). Lalu program mencatat proses tersebut ke dalam statistik *database* sistem KIRI. Terakhir, program akan mengirimkan pesan dalam format JSON sebagai penanda bahwa pengguna berhasil melakukan proses registrasi (baris 340).

- **Bagian Melihat Data Pribadi Pengguna**

Bagian ini terletak di baris 341-362 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=getprofile” pada permintaan POST. Bagian ini berfungsi untuk memberikan informasi mengenai data pribadi pengguna.

Bagian ini diawali dengan memeriksa apakah data pengguna dengan *email* yang dimiliki pengguna pada saat sesi tersebut ada atau tidak (baris 344-345). Jika data pengguna ditemukan maka program akan mengambil dan membangun data pengguna (baris 346-351). Data pengguna yang dibangun tersebut kemudian diubah ke dalam format JSON (baris 355-359). Terakhir, program mengirimkan data dalam format JSON yang telah dibangun kepada pengguna (baris 361).

- **Bagian Mengubah Data Pribadi Pengguna**

Bagian ini terletak di baris 362-380 dari “handle.php” (kode Listing 1). Bagian ini akan dieksekusi hanya jika terdapat parameter “mode=updateprofile” pada permintaan POST. Bagian ini berfungsi untuk mengubah data pribadi pengguna yang sudah terdaftar dalam sistem KIRI.

Bagian ini diawali dengan memeriksa apakah pengguna memberikan *password*, *fullname*, dan *company* pada permintaan atau tidak (baris 364-366). Bila pengguna memberikan *password* dengan nilai NULL maka program akan membangun *password* secara acak dan menambahkan *password* tersebut ke dalam *database* sistem KIRI sesuai dengan *email* pengguna pada saat sesi tersebut (kode 369-374). Lalu program akan mengubah semua data pribadi pengguna sesuai dengan data yang diberikan oleh pengguna (baris 375-376). Terakhir, program mengirimkan pesan keberhasilan dalam format JSON kepada pengguna (baris 379).

## 2. Melakukan studi literatur tentang MySQL Spatial Extensions, JDBC, Play Framework, dan JSON.

**status :** Ada sejak rencana kerja skripsi, kecuali JDBC dan JSON

**hasil :**

- **MySQL Spatial Extensions**

Suatu *geographic feature*[7] adalah sesuatu yang ada di bumi yang memiliki lokasi sebagai penunjuk letak keberadaannya. Geometri adalah cabang ilmu matematika yang digunakan untuk memodelkan suatu *geographic feature*. Dengan geometri, suatu *geographic feature* dapat dinyatakan sebagai sebuah titik, garis, ruang, ataupun bentuk lainnya. Suatu “*feature*” yang dimaksud dalam istilah *geographic feature* dapat berupa:

- (a) **An entity**, contohnya adalah gunung, kolam, kota, dll.
- (b) **A space**, contohnya adalah daerah, cuaca, dll.
- (c) **A definable location**, contohnya adalah persimpangan jalan, yaitu suatu tempat khusus dimana terdapat 2 buah jalan yang saling berpotongan.

MySQL adalah salah satu perangkat lunak yang digunakan untuk mengatur data-data (*database*) suatu situs web. Bentuk MySQL adalah sekumpulan tabel yang umumnya memiliki hubungan antar satu dengan yang lainnya. Setiap tabel pada MySQL memiliki kolom dan baris. Kolom

pada MySQL menyatakan daftar jenis baris yang ingin dibuat dan baris menyatakan banyaknya data yang ada dalam tabel.

Penamaan suatu kolom dalam MySQL membutuhkan penentuan tipe data yang akan digunakan dalam kolom tersebut. Dalam MySQL terdapat tipe-tipe data yang umum digunakan seperti *Varchar* untuk menyimpan karakter atau kata, *Int* untuk menyimpan angka, *Boolean* untuk menyimpan nilai “true” atau “false”, dan tipe data lainnya. MySQL Spatial Extensions adalah perluasan dari tipe-tipe data yang disediakan MySQL untuk menyatakan nilai geometri dari suatu *geographic feature*.

Berdasarkan kemampuan penyimpanan nilai geometri, tipe data *spatial* dapat dikelompokkan ke dalam 2 jenis:

(a) Tipe data yang hanya dapat menyimpan sebuah nilai geometri saja, yaitu:

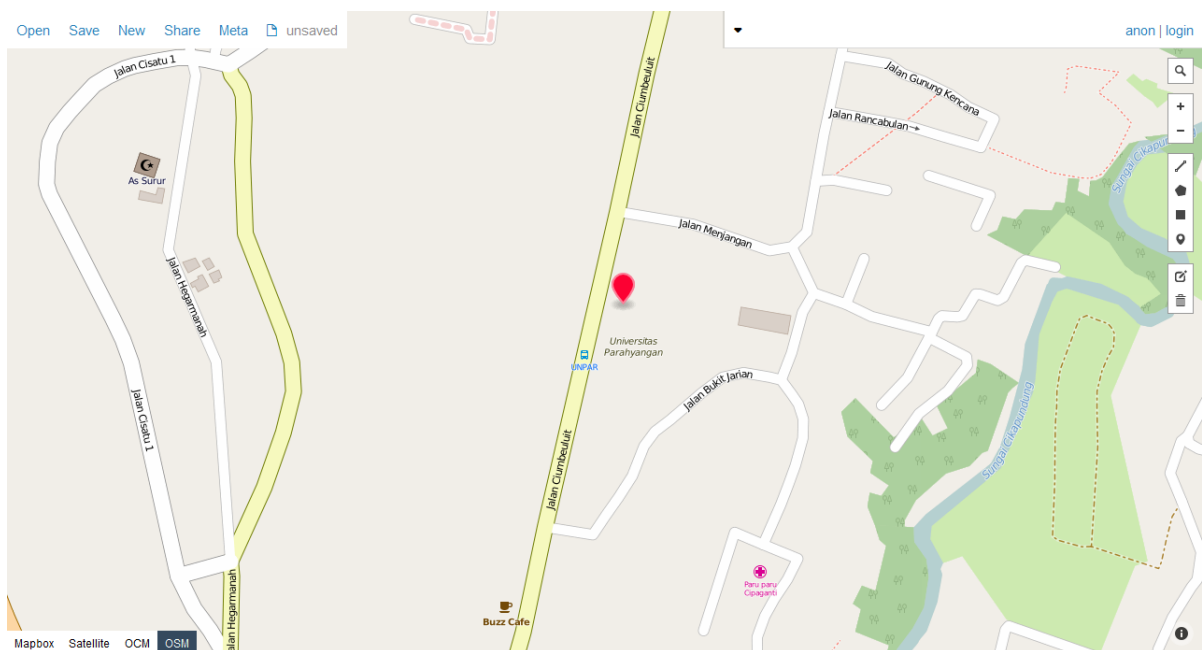
- *Geometry*
- *Point*
- *LineString*
- *Polygon*

(b) Tipe data yang dapat menyimpan sekumpulan nilai geometri, yaitu:

- *MultiPoint*
- *MultiLineString*
- *MultiPolygon*
- *GeometryCollection*

### *Point*

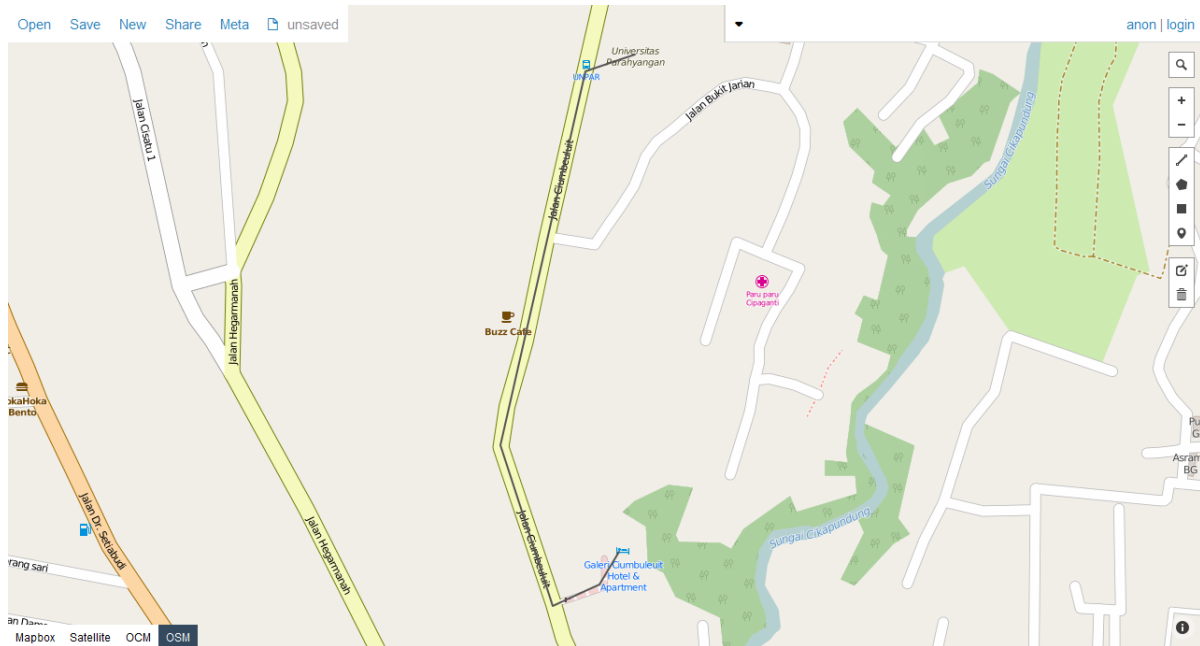
*Point* adalah nilai geometri yang merepresentasikan sebuah lokasi ke dalam suatu koordinat[7]. Koordinat pada *Point* terdiri dari nilai X dan Y dimana X merepresentasikan letak lokasi terhadap garis bujur dan Y merepresentasikan letak lokasi terhadap garis lintang. *Point* tidak memiliki dimensi maupun nilai batasan. Contoh representasi *Point* adalah Universitas Katolik Parahyangan direpresentasikan dalam koordinat X=107.6049079 dan Y=-6.874735 (gambar 6).



Gambar 6: Universitas Katolik Parahyangan dinyatakan dalam *Point*[8]

### *LineString*

*LineString* adalah garis yang terbentuk dari sekumpulan *Point*[7]. Dalam peta dunia, *LineString* dapat merepresentasikan sebuah sungai dan dalam peta perkotaan, *LineString* dapat merepresentasikan sebuah jalan (contoh: gambar 7). Karena *LineString* merupakan sekumpulan *Point*, maka *LineString* menyimpan sekumpulan koordinat dimana setiap koordinat ( $X_1 \dots X_n$  dan  $Y_1 \dots Y_n$ , dimana  $n$  menyatakan banyaknya *Point* dalam *LineString*) terhubung oleh garis dengan koordinat selanjutnya. Contohnya: misal terdapat sebuah *LineString* yang mengandung 3 buah *Point*, maka terdapat garis yang menghubungkan *Point* pertama dengan *Point* kedua dan *Point* kedua dengan *Point* ketiga.



Gambar 7: Rute jalan dari Universitas Katolik Parahyangan menuju Galeri Ciumbuleuit dinyatakan dalam *LineString*[8]

### Format Well-Known Text (WKT)

Format Well-Known Text (WKT) adalah salah satu aturan penulisan tipe data *spatial* untuk merepresentasikan suatu *geographic feature*[7]. WKT merepresentasikan nilai geometri yang dimodelkan untuk pertukaran data geometri dalam ASCII *form*. Berikut adalah contoh format WKT:

```

1 | POINT(107.6049079 -6.874735)
2 |
3 | LINESTRING(107.60502219200134 -6.875194997571583,
4 |             107.60445356369019 -6.875386727913034,
5 |             107.60347723960876 -6.879647382202341,
6 |             107.6040780544281 -6.881479451795388,
7 |             107.60461449623108 -6.8812344661545986,
8 |             107.60483980178833 -6.880861661676069)

```

Contoh di atas menunjukkan format WKT dari *Point* (baris 1) dan format WKT dari *LineString* (baris 3-8).

Berikut adalah contoh penggunaan format WKT dalam MySQL:

```

1 | CREATE TABLE geom (g LINESTRING);
2 |
3 | INSERT INTO geom VALUES (ST_GeomFromText('LINESTRING(107.60502219200134 -6.875194997571583,
4 |             107.60445356369019 -6.875386727913034,
5 |             107.60347723960876 -6.879647382202341,
6 |             107.6040780544281 -6.881479451795388,
7 |             107.60461449623108 -6.8812344661545986,
8 |             107.60483980178833 -6.880861661676069)'));
9 |
10 | SELECT ST_AsText(g) FROM geom;

```

Contoh di atas menunjukkan pembuatan tabel “geom” dengan sebuah kolom “g” dan tipe data “LINESTRING” (baris 1), menambahkan 1 baris data berupa *LineString* ke dalam tabel “geom” (baris 3-8), dan melihat data dari tabel “geom” (baris 10), dimana nilai kembalian “ST\_AsText(g)” berupa data *LineString* dalam format WKT.

## • JDBC

JDBC API adalah bagian dari Java API yang dapat digunakan untuk mengakses semua jenis data yang terstruktur, terutama data yang tersimpan dalam suatu *Relational Database*[9]. JDBC dapat membantu 3 jenis aktivitas *programming* dalam menggunakan bahasa Java, yaitu:

- (a) Menghubungkan aplikasi Java ke suatu sumber data seperti *database*,
- (b) Mengirimkan *queries* dan pembaharuan *statement* ke *database*,
- (c) Menerima dan melakukan proses terhadap hasil yang didapatkan dari pengiriman *queries* tersebut.

Berikut adalah contoh struktur kode yang mewakili 3 jenis aktivitas yang dapat dilakukan JDBC API:

```

1 public void connectToAndQueryDatabase(String username, String password) {
2
3     Connection con = DriverManager.getConnection(
4         "jdbc:mysql:myDatabase",
5         username,
6         password);
7
8     Statement stmt = con.createStatement();
9     ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
10
11     while (rs.next()) {
12         int x = rs.getInt("a");
13         String s = rs.getString("b");
14         float f = rs.getFloat("c");
15     }
16 }

```

Contoh di atas menunjukkan bagaimana JDBC API membantu aplikasi Java membuat koneksi terhadap suatu *database* (baris 3-6), membuat dan mengirimkan suatu *query* ke *database* (baris 8 dan 9), dan menerima dan melakukan proses terhadap hasil yang didapatkan dari pengiriman *query* tersebut (baris 9-15).

## **Interface Connection**

*Interface Connection* adalah sebuah koneksi (*session*) dengan *database* spesifik[9]. Eksekusi SQL *statements* dan penerimaan hasil kembalian dari eksekusi tersebut dapat terjadi karena adanya koneksi dengan *database* yang dibentuk oleh *interface Connection*. Berikut adalah sebagian *method* yang ada pada *interface Connection*:

- `void close()`  
*Method* ini digunakan untuk memutuskan koneksi dengan *database* yang sedang terhubung.
- `Statement createStatement()`  
*Method* ini digunakan untuk membangun objek **Statement** yang dapat digunakan untuk mengirimkan SQL *statements* ke *database* yang sedang terhubung.

## **Kelas DriverManager**

Kelas **DriverManager** adalah cara paling dasar untuk mengatur JDBC *drivers*[9]. Berikut adalah salah satu *method* yang ada di kelas **DriverManager** untuk mengatur JDBC *drivers*:

- `public static Connection getConnection(String url, String user, String password)`  
*Method* ini digunakan untuk membangun sebuah koneksi dengan *database*. Umumnya *method* ini digunakan untuk membangun *interface Connection*.  
 Parameter:
  - (a) `url`, alamat dari *database*, formatnya adalah “*jdbc:subprotocol:subname*”,



(b) *user*, *username* untuk mengakses *database*,

(c) *password*, *password* dari *username*.

Nilai kembalian: sebuah koneksi terhadap *database* yang sesuai dengan alamat *url*.

### Interface Statement

*Interface Statement* adalah objek yang digunakan untuk melakukan eksekusi terhadap suatu *query* dan mengembalikan nilai kembalian dari eksekusi tersebut[9]. Berikut adalah salah satu *method* yang ada di *interface Statement*:

– `ResultSet executeQuery(String sql)`

Parameter: *sql*, sebuah SQL *statement* yang akan dikirimkan ke *database*.

Nilai kembalian: objek *ResultSet* yang berupa data yang dihasilkan dari eksekusi *query sql*.

### Interface ResultSet

*Interface ResultSet* adalah sebuah tabel data yang merepresentasikan hasil dari sebuah eksekusi *query* pada suatu *database*[9]. Cara kerja *interface ResultSet* adalah dengan sistem indeks. Pada awalnya indeks *ResultSet* menunjuk pada data “bayangan” sebelum data pertama. Setiap pemanggilan *method* “*next()*” pada objek *ResultSet* akan menyebabkan nilai indeks semakin meningkat (bertambah 1). Berikut adalah contoh *method interface ResultSet*:

– `boolean next()`

Nilai kembalian: *true* jika terdapat data pada indeks selanjutnya, *false* bila tidak ditemukan data pada indeks selanjutnya.

– `Object getObject(String columnLabel)`

Parameter: *columnLabel*, merupakan nama kolom yang ingin diambil nilainya.

Nilai kembalian: data berupa *Object* pada indeks baris dan kolom yang ditunjuk.

– `String getString(String columnLabel)`

Parameter: *columnLabel*, merupakan nama kolom yang ingin diambil nilainya.

Nilai kembalian: data berupa *String* pada indeks baris dan kolom yang ditunjuk.

– `int getInt(String columnLabel)`

Parameter: *columnLabel*, merupakan nama kolom yang ingin diambil nilainya.

Nilai kembalian: data berupa *int* pada indeks baris dan kolom yang ditunjuk.

– `boolean getBoolean(String columnLabel)`

Parameter: *columnLabel*, merupakan nama kolom yang ingin diambil nilainya.

Nilai kembalian: data berupa *boolean* pada indeks baris dan kolom yang ditunjuk.

## • Play Framework

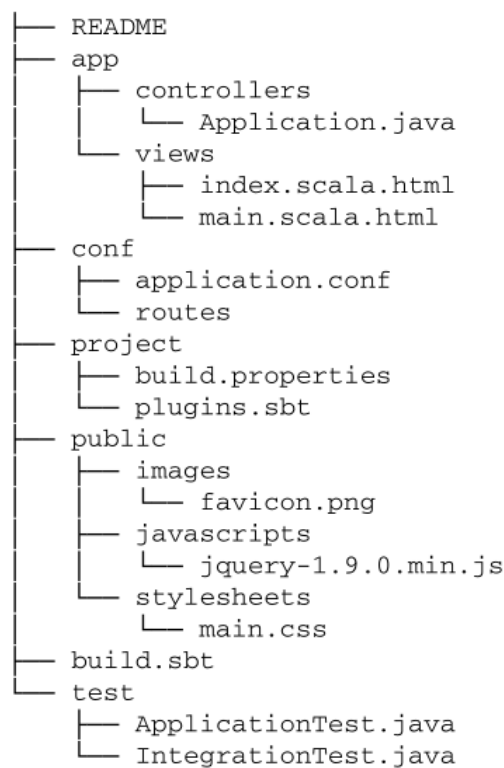
Play Framework adalah sekumpulan kerangka kode yang dapat digunakan untuk membangun suatu situs web[10]. Play Framework tidak hanya menggunakan bahasa Java dalam pembuatannya. Bahasa Scala juga digunakan Play Framework dalam beberapa bagian seperti bagian *view* dan *route*. Play Framework menggunakan konsep MVC (*Model View Controller*) sebagai pola arsitekturnya. Konsep MVC pada suatu kode membuat kode mudah dikembangkan baik secara tampilan maupun pengembangan fitur-fiturnya. Ketika *server* Play Framework dijalankan, secara *default* dapat diakses melalui “localhost:9000”.

### Struktur Aplikasi

Ketika Play Framework pertama kali ter-*install* pada komputer, Play Framework menyediakan *default* direktori dengan struktur minimal (gambar 8). Berikut adalah penjelasan struktur minimal Play Framework:

(a) Folder “*app*” merupakan *folder* yang berisi mengenai pola arsitektur yang dimiliki Play Framework, yaitu “*models*” (tidak dibuat secara *default*), “*views*”, dan “*controllers*”.

- (b) *Folder* “conf” berisi mengenai *file* “application.conf” yang menyimpan pengaturan-pengaturan seperti kumpulan *log*, koneksi ke *database*, jenis *port* tempat *server* bekerja, dll. *Folder* “conf” juga berisi *file* “routes” yang mengatur bagaimana HTTP *requests* nantinya akan diproses lebih lanjut.
- (c) *Folder* “project” terdapat *file* “build.properties” dan “plugins.sbt”, *file* tersebut mendeskripsikan versi Play dan SBT yang digunakan pada aplikasi.
- (d) *Folder* “public” merupakan *folder* yang menyimpan data-data seperti gambar (*folder* “images”), kumpulan Javascript yang digunakan (*folder* “javascripts”, secara *default* berisikan *file* “jquery-1.9.0.min.js”) dan data-data CSS (*folder* “stylesheets”).
- (e) *File* “build.sbt” mengatur *dependencies* yang dibutuhkan dalam pembuatan aplikasi.
- (f) Terakhir adalah *folder* “test” yang merupakan salah satu kelebihan dari Play Framework, bagian ini berisikan *file* “Application.test” dan “Integration.test” yang dapat digunakan untuk melakukan serangkaian *testing* yang diinginkan terhadap aplikasi.



Gambar 8: Struktur minimal Play Framework

### Routes

*Routes* adalah *file* yang mengatur pemetaan dari HTTP URLs menuju kode aplikasi (dalam hal ini menuju ke *controllers*). Secara *default*, *routes* berisikan kode yang dapat memetakan permintaan URL *index* standar seperti “localhost:9000” ketika *server* Play Framework sudah dijalankan.

Berikut adalah isi kode *default routes*:

```

1 | # Home page
2 | GET      /                  controllers.Application.index()
3 |
4 | # Map static resources from the /public folder to the /assets URL path
5 | GET      /assets/*file     controllers.Assets.at(path="/public", file)

```

Contoh di atas menunjukkan bagaimana *routes* memetakan permintaan URL *index* atau “/” (baris ke 2) dan permintaan URL “/assets/\*file” (baris ke 5).

Struktur *routes* terdiri dari 3 bagian (gambar 9), yaitu HTTP *method*, URL *path*, dan *action method*. Struktur *routes* seperti yang dijelaskan pada gambar 9 juga sekaligus menjadi struktur

minimal yang harus ada agar *routes* dapat memetakan suatu HTTP URLs. HTTP *method* berisikan protokol yang ingin dilakukan terhadap suatu HTTP *request*. HTTP *method* dapat berupa “GET”, “POST”, “DELETE”, “PATCH”, “HEAD” atau “PUT”[11]. URL *path* merupakan direktori yang ingin dituju dalam *server* aplikasi. URL *path* dimulai dengan tanda “/” dan diikuti dengan nama direktori yang ingin dituju. Terakhir, *action method* merupakan pemilihan kelas *controller* yang ingin dituju. Struktur *action method* terdiri dari 3 bagian (dipisahkan dengan karakter “.”), yaitu pemilihan *package* “controllers” yang ingin dituju, bagian kedua adalah pemilihan kelas “controllers” yang dipilih (contohnya: “Products” pada gambar 9), dan terakhir adalah pemilihan *method* yang ada pada kelas “controllers” yang dipilih (contohnya: “list()”).



Gambar 9: Struktur kode file “routes”[10]

URL *path* dan *action method* pada *routes* juga dapat berisi sebuah nilai variabel. Berikut adalah contoh penulisan program URL *path* dan *action method* pada *routes* yang berisi sebuah nilai variabel:

```
1 | GET /clients/:id controllers.Clients.show(id: Long)
```

Penulisan sebuah variabel pada URL *path* dimulai dengan tanda “:” lalu diikuti dengan nama variabel yang diinginkan, contohnya: “:id”. Ketika menggunakan variabel pada URL *path*, pada *action method* perlu ditambahkan deklarasi variabel yang diletakkan di dalam bagian *method* yang dipilih. Cara penulisan deklarasi variabel pada *action method* adalah dimulai dengan nama variabel, lalu diikuti karakter “:”, dan diakhiri dengan tipe variabel yang diinginkan. Contoh penulisan deklarasi variabel di dalam *method* suatu kelas pada bagian *action method* adalah “id: Long”.

### Models

Fungsi *models* pada Play Framework sama seperti fungsi *models* pada pola arsitektur MVC secara umum, yaitu untuk memanipulasi dan menyimpan data. Secara *default*, *models* tidak dibuat oleh struktur minimal Play Framework (gambar 8). Untuk itu perlu menambahkan *models* secara manual ke dalam struktur Play Framework. Langkah yang dilakukan untuk menambahkan *models* ke dalam Play Framework adalah:

- (a) Menambahkan folder “models” ke dalam folder “app”,
- (b) Menambahkan file dengan format “.java” ke dalam folder “models”.

Tidak ada aturan khusus yang diharuskan dalam penulisan kode dalam kelas *models*. Selama kelas *models* yang dibuat memenuhi aturan bahasa Java, maka *models* dapat dieksekusi oleh *server* Play Framework.

### Views

Fungsi *views* pada Play Framework adalah mengatur tampilan yang ingin ditampilkan di layar. *Views* menggunakan bahasa HTML dan Scala. Bahasa Scala pada *views* berfungsi sebagai penerima parameter yang dikirimkan dari kelas *models* dimana antara *models* dan *views* dihubungkan oleh *controllers*. Penamaan file di dalam folder *views* (gambar 8) harus dengan format sebagai berikut, “namaFile.scala.html”.

Berikut adalah contoh struktur kode *views*:

```

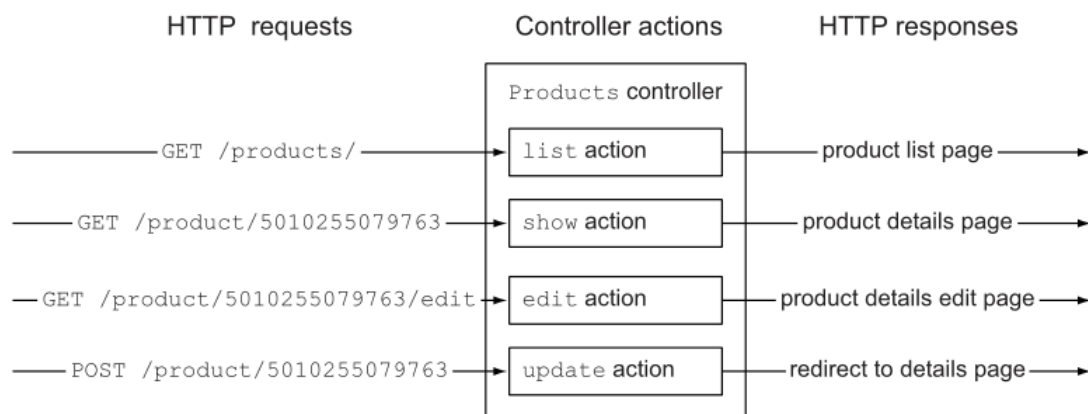
1  @(name: String)
2  <!doctype html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title>Hello</title>
7    </head>
8    <body>
9      <h1>Hello <em>@name</em></h1>
10   </body>
11 </html>

```

Baris 1 pada contoh kode di atas digunakan sebagai parameter penerima input dari *models* yang dihubungkan dengan *controllers*. Format deklarasi variabel pada parameter *views* diawali dengan karakter “@”, lalu diikuti dengan “(namaVariabel<sub>1</sub>: tipeVariabel<sub>1</sub>) (namaVariabel<sub>2</sub>: tipeVariabel<sub>2</sub>) ... (namaVariabel<sub>n</sub>: tipeVariabel<sub>n</sub>)”, dimana *n* adalah jumlah parameter yang ingin digunakan dalam *views*. Variabel pada parameter yang sudah dideklarasikan dapat dipanggil dengan menggunakan format “@namaVariabel” (baris 9).

### Controllers

*Controllers* merupakan bagian pada Play Framework yang terhubung langsung dengan *routes*. Jika *action method* yang dikirimkan oleh *routes* sesuai dengan *method* yang dimiliki suatu kelas *controllers*, maka *controllers* akan mengeksekusi fungsi logika yang terdapat pada *method* dan mengembalikan nilai berupa objek dari kelas *Result* (gambar 10). Fungsi dari *controllers* dalam arsitektur MVC adalah sebagai penghubung antara *models* dan *views*.



Gambar 10: Hubungan *routes* dan *controllers* dalam memproses HTTP requests[10]

Berikut adalah contoh penulisan program suatu kelas *controllers*:

```

1  package controllers;
2
3  import play.mvc.Controller;
4
5  public class Application extends Controller {
6
7      public Result index() {
8          return ok(index.render("Your new application is ready."));
9      }
10
11 }

```

Penulisan kode pada suatu kelas *controllers* menggunakan bahasa Java dan memiliki aturan khusus (contoh kode di atas). Aturan khusus dijelaskan ke dalam poin-poin sebagai berikut:

- Visibility* kelas dan *method* pada kelas tersebut harus *public* (baris 5),
- Kelas yang dibuat harus merupakan turunan dari “*play.mvc.Controller*” (baris 5),
- Nilai kembalian *method* yang dibuat dalam suatu kelas *controllers* harus berupa objek dari kelas *Result* (baris 7 dan 8).

### Database

Play Framework menyediakan sebuah *plugin* yang dapat digunakan untuk mengatur koneksi JDBC ke berbagai jenis aplikasi *database* yang tersedia[11]. Salah satu koneksi *database* yang disediakan oleh Play adalah koneksi ke MySQL. Secara *default plugin* yang disediakan oleh Play masih belum aktif. Perlu dilakukan beberapa langkah agar *plugin* tersebut dapat aktif. Berikut adalah langkah-langkah yang dilakukan agar Play Framework dapat terhubung dengan *database* MySQL:

- (a) Menambahkan kode program ke dalam “build.sbt” (gambar 8), yaitu:

```
1 | libraryDependencies += javaJdbc
2 | libraryDependencies += "mysql" % "mysql-connector-java" % "5.1.18"
```

Baris 1 kode program di atas adalah untuk mengaktifkan plugin JDBC pada Play Framework. Play tidak menyediakan *database driver* apapun, untuk itu perlu menambahkan *database driver* (baris 2) sebagai *dependency* untuk aplikasi Play Framework.

- (b) Menambahkan kode program ke dalam “conf/application.conf” (gambar 8), yaitu:

```
1 | db.default.driver=com.mysql.jdbc.Driver
2 | db.default.url="jdbc:mysql://localhost/playdb"
3 | db.default.username=playdbuser
4 | db.default.password="a strong password"
```

Baris 1 kode program di atas menyatakan jenis *driver* yang digunakan, yaitu MySQL. Baris 2 kode program menyatakan nama *database* yang digunakan, yaitu “playdb”. Baris 3 dan 4 menyatakan *username* dan *password* yang dibutuhkan dalam otentikasi terhadap *server database* untuk mendapatkan hak akses tertentu terhadap *database*.

Salah satu aktivitas programming yang dibantu JDBC adalah menghubungkan aplikasi Java ke suatu sumber data seperti *database*. Play Framework telah menyediakan kelas “DB” yang dapat memudahkan aplikasi Java membuat suatu koneksi dengan *database*. Berikut adalah contoh kode yang diperlukan untuk menggunakan kelas “DB” dari Play Framework:

```
1 | import play.db.*;
2 |
3 | Connection connection = DB.getConnection();
```

Contoh kode di atas menyederhanakan penulisan kode milik JDBC.

### • JSON

JSON (*JavaScript Object Notation*) adalah sebuah format pertukaran data ringan[12]. JSON dapat dibangun dalam 2 buah struktur:

- Sekumpulan pasangan antara nama dengan nilai. Umumnya dikenal dengan sebutan objek. Sebuah objek dalam JSON dimulai dengan karakter “{” dan diakhiri dengan karakter “}”. Diantara karkater “{” dan “}” dapat disisipkan sekumpulan pasangan “**nama:nilai**” yang dipisahkan dengan karakter “,”.
- Sekumpulan data terstruktur. Umumnya dikenal dengan sebutan *array*. Sebuah *array* dalam JSON dimulai dengan karakter “[” dan diakhiri dengan karakter “]”. Diantara karkater “[” dan “]” dapat disisipkan sekumpulan data (dapat berupa nilai, objek atau *array*) yang dipisahkan dengan karakter “,”. Dalam *array*. Setiap data dalam *array* tidak harus sama jenisnya.

Nilai dalam JSON dapat berupa *string* (sekumpulan karakter yang diapit dengan 2 tanda kutip ganda, contoh: “**karakter**”), angka, **true**, **false**, atau **null**.

Berikut adalah contoh sebuah data JSON:

```
1 | {
2 |   "status": "error",
3 |   "message": "Value of userid is expected but not found"
4 | }
```

Contoh kode di atas adalah sebuah objek (baris 1-4) yang memiliki 2 buah pasangan “*nama:nilai*” yang dipisahkan oleh karakter “,” (baris 2-3). Contoh di atas juga menggunakan *string* sebagai nilainya (baris 2 dan 3).

3. Menganalisis teori-teori untuk membangun KIRI *Dashboard Server Side* dalam bahasa Java dengan menggunakan Play Framework.

**status :** Ada sejak rencana kerja skripsi.

**hasil :**

Berdasarkan hasil analisa kode dan wawancara dengan kontributor kode didapatkan poin-poin bahwa:

- (a) KIRI *Dashboard* terdiri dari dua bagian penting, yaitu bagian pertama adalah *file* dan *folder* yang bersifat statis, yaitu: “index.html”, “bukitjariangwt/”, dan “images/”, serta bagian kedua adalah “handle.php” yang bertugas menangani permintaan-permintaan (dari “index.html”) pada sisi *server*.
- (b) KIRI *Dashboard server side* (*file* “handle.php”) terbagi menjadi 16 bagian dalam menangani permintaan.
- (c) Perangkat lunak pengolahan data yang digunakan oleh sistem KIRI adalah MySQL.
- (d) KIRI *Dashboard server side* (*file* “handle.php”) akan mengirimkan data dalam format JSON dalam membalas permintaan dari “index.html”.
- (e) KIRI *Dashboard server side* (*file* “handle.php”) menggunakan *framework* yang didapat dari <http://www.openwall.com/phpass/> untuk melakukan fungsi *hash* terhadap *password* pada bagian *register* dan bagian *login*. *Framework* tersebut terlalu rumit untuk diimplementasikan dan tidak tersedia di Java, maka untuk melakukan fungsi *hash* terhadap *password* akan menggunakan SHA (*Secure Hash Algorithm*) yang disediakan oleh Java.
- (f) KIRI *Dashboard server side* (*file* “handle.php”) menggunakan PHPMailer pada bagian *register* untuk mengirimkan *password* yang telah dibangun oleh sisi *server*. PHPMailer terlalu rumit untuk diimplementasikan dan tidak tersedia di Java, maka untuk mengirimkan *password* yang telah dibangun oleh sisi *server* akan menggunakan JavaMail API Jar yang tersedia untuk Java.

Berdasarkan kebutuhan akan poin-poin di atas, maka dalam memodelkan KIRI *Dashboard* dengan menggunakan Play Framework perlu memperhatikan beberapa hal, yaitu: bagian *models*, *views*, *controllers*, *routes* dan *libraries* yang digunakan.

- **Routes**

Kebutuhan KIRI *Dashboard* adalah pemetaan untuk ke 2 bagian, yaitu:

- (a) Pemetaan URL untuk menangani tampilan KIRI *Dashboard* (*views*). Hanya 1 *route* yang dibutuhkan untuk pemetaan terhadap bagian tampilan (walau fitur KIRI *Dashboard* banyak) karena kode KIRI *Dashboard* sistem kini telah menggunakan kombinasi JavaScript dan AJAX sedemikian rupa yang dapat mengubah tampilan KIRI *Dashboard* sesuai dengan balasan permintaan dari sisi *server*. AJAX (*asynchronous JavaScript and XML*) adalah teknik yang memungkinkan suatu situs web melakukan permintaan ke sisi *server* secara *background*[13].
- (b) Pemetaan URL untuk menangani permintaan-permintaan dari tampilan ke sisi *server*, yaitu KIRI *Dashboard server side* (*controllers*).

- **Views**

Seperti yang telah dijelaskan pada analisa sebelumnya bahwa bagian tampilan KIRI *Dashboard* dibuat dengan menggunakan perangkat GWT dimana kode yang dihasilkan sangat sulit untuk dipelajari. Walau sulit dipelajari, bagian tampilan KIRI *Dashboard* memiliki keuntungan sendiri

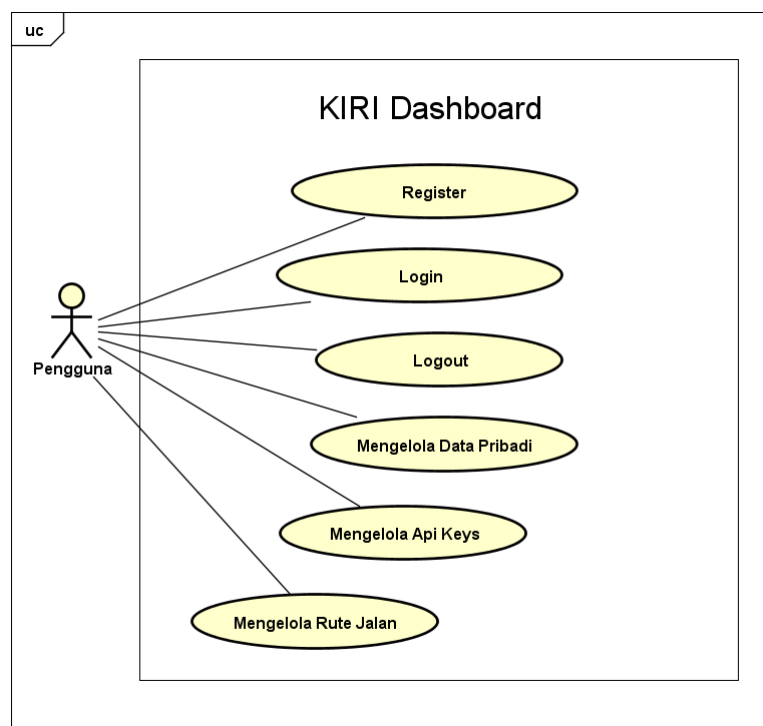
karena bersifat statis (dijelaskan oleh kontributor kode). Sifat statis ini memudahkan pemodelan karena dengan begitu kode dapat disalin apa adanya ke sistem usulan. Pada Play Framework *file-file* yang bersifat statis seperti ini dapat disimpan pada bagian *folder* “public”.

- **Controllers**

Pada bagian ini akan dibuat sebuah metode yang akan menangani permintaan dari *views* sistem usulan. Bagian ini berfungsi seperti “handle.php”, yaitu menangani 16 jenis permintaan seperti yang telah dijelaskan pada subbab sebelumnya.

- **Models**

*Models* adalah bagian yang paling bebas dan tidak memiliki aturan pembuatan khusus pada Play Framework. Pada bagian ini penulis memutuskan untuk membuat *models* sesuai dengan diagram *use case* KIRI Dashboard (gambar 11). Untuk setiap kemampuan pengguna yang dapat dilakukan pada diagram *use case* (gambar 11) akan dibuat kelas Java di bagian *models* pada sistem usulan.



Gambar 11: Diagram *use case* KIRI Dashboard

- **Libraries**

Berikut adalah *libraries* (sumber: <http://mvnrepository.com/>) yang akan digunakan untuk membangun KIRI Dashboard *server side* sistem usulan:

- Jackson Databind, digunakan untuk menangani pengiriman data dalam format JSON.
- JavaMail API, digunakan untuk mengirimkan *email*.
- MySQL Connector/J, digunakan untuk melakukan hubungan koneksi dan eksekusi *query* dengan *database*. *Library* MySQL Connector/J merupakan *library* resmi JDBC driver untuk MySQL.

4. Merancang KIRI Dashboard *Server Side* dalam bahasa Java dengan menggunakan Play Framework.

**status :** Ada sejak rencana kerja skripsi.

**hasil :** belum ada perkembangan.

5. Melakukan *porting* kode situs web KIRI *Dashboard Server Side* yang semula dalam bahasa PHP menjadi bahasa Java dengan menggunakan Play Framework.

**status :** Ada sejak rencana kerja skripsi.

**hasil :**

Berikut adalah hasil kode *porting* yang telah dilakukan beserta kode-kode pendukung:

Listing 2: build.sbt

```

1 | name := """KIRI"""
2 |
3 | version := "1.0-SNAPSHOT"
4 |
5 | lazy val root = (project in file(".")).enablePlugins(PlayJava)
6 |
7 | scalaVersion := "2.11.6"
8 |
9 | libraryDependencies ++= Seq(
10 |   javaJdbc,
11 |   cache,
12 |   javaWs
13 | )
14 |
15 | // Play provides two styles of routers, one expects its actions to be injected, the
16 | // other, legacy style, accesses its actions statically.
17 | routesGenerator := InjectedRoutesGenerator
18 |
19 | //tambah database untuk mysql
20 | libraryDependencies += "mysql" % "mysql-connector-java" % "5.1.18"
21 | //untuk pake json
22 | libraryDependencies += "com.fasterxml.jackson.core" % "jackson-databind" % "2.6.3"
23 | //untuk pake Java Mail
24 | libraryDependencies += "com.sun.mail" % "javax.mail" % "1.5.4"
```

File “build.sbt” digunakan untuk menambahkan SBT *plugins* yang digunakan untuk aplikasi sistem usulan. Baris 20-24 kode di atas akan menambahkan *libraries* yang diperlukan untuk aplikasi ini, yaitu: Jackson Databind, JavaMail API, dan MySQL Connector/J.

Listing 3: application.conf

```

1 | # This is the main configuration file for the application.
2 | # -----
3 |
4 | # Secret key
5 | # -----
6 | # The secret key is used to secure cryptographic functions.
7 | #
8 | # This must be changed for production, but we recommend not changing it in this file.
9 | #
10 | # See http://www.playframework.com/documentation/latest/ApplicationSecret for more details.
11 | play.crypto.secret = "changeme"
12 |
13 | # The application languages
14 | # -----
15 | play.i18n.langs = [ "en" ]
16 |
17 | # Router
18 | # -----
19 | # Define the Router object to use for this application.
20 | # This router will be looked up first when the application is starting up,
21 | # so make sure this is the entry point.
22 | # Furthermore, it's assumed your route file is named properly.
23 | # So for an application router like 'my.application.Router',
24 | # you may need to define a router file 'conf/my.application.routes'.
25 | # Default to Routes in the root package (and conf/routes)
26 | play.http.router = my.application.Routes
27 |
28 | # Database configuration
29 | # -----
30 | # You can declare as many datasources as you want.
31 | # By convention, the default datasource is named 'default'
32 | #
33 | db.default.driver=com.mysql.jdbc.Driver
34 | db.default.url="jdbc:mysql://localhost/tirtayasa"
35 | db.default.username=root
36 |
37 | # Evolutions
38 | # -----
39 | # You can disable evolutions if needed
40 | play.evolutions.enabled=false
41 |
42 | # You can disable evolutions for a specific datasource if necessary
43 | play.evolutions.db.default.enabled=false
```

Salah satu fungsi file “application.conf” adalah digunakan untuk menentukan pengaturan untuk koneksi



ke *database*. Baris 33 menyatakan jenis *driver* yang digunakan, baris 34 menyatakan alamat URL *database* yang dituju, dan baris 35 menyatakan *username* yang digunakan.

Listing 4: routes

```

1  # Routes
2  # This file defines all application routes (Higher priority routes first)
3  # ----
4
5  # Home page
6  GET      /                                controllers.Application.index()
7  GET      /testingdb                      controllers.Application.testingDB()
8  POST     /bukitjarian/handle.php         controllers.Application.handle()
9  GET      /bukitjarian                    controllers.Application.index()
10 GET      /bukitjarian/                    controllers.Assets.at(path="/public/bukitjarian", file="index.html")
11 GET      /bukitjarian/*file                controllers.Assets.at(path="/public/bukitjarian", file)
12 GET      /assets/*file                    controllers.Assets.versioned(path="/public", file: Asset)
13
14 # handle for page not found
15 GET      /*other                          controllers.Application.pagenotfound(other: String)

```

Kode di atas menyatakan pemetaan yang akan dilakukan.

Listing 5: Application.java

```

1  package controllers;
2
3  //default import
4  import play.*;
5  import play.mvc.*;
6  import views.html.*;
7
8  //tambahan import
9  import play.data.*;
10 import play.db.*;
11 import java.io.*;
12 import java.sql.*;
13 import java.util.Random;
14
15 //untuk json
16 import play.libs.Json;
17 import com.fasterxml.jackson.databind.node.ObjectNode;
18
19 public class Application extends Controller {
20     DynamicForm requestData;
21
22
23     public Result index(){
24         return redirect("/bukitjarian/");
25     }
26
27     public Result pagenotfound(String other){
28         return notFound("<h1>"+other+" not found</h1>").as("text/html");
29     }
30
31     public Result testingDB() throws IOException, SQLException{
32         java.sql.Connection connection = DB.getConnection();
33         Statement statement = connection.createStatement();
34         ResultSet result = statement.executeQuery("select * from users;");
35         StringBuilder sb = new StringBuilder();
36         sb.append("LIST USERS Tirtayasa"+ "\n");
37         while (result.next()) {
38             sb.append("userid: "+ result.getString("email") + "\npassword: " + result.getString("password")
39                 + "\n");
40         }
41         return ok(sb.toString());
42     }
43
44     public Result handle() throws IOException, SQLException{
45         this.requestData = Form.form().bindFromRequest();
46         String mode = this.requestData.get("mode");
47         if(mode.equals("login")){
48             return this.login();
49         }
50         else if(mode.equals("register")){
51             return this.register();
52         }
53         else{
54             return badRequest("failed");
55         }
56     }
57
58     private Result login() throws IOException, SQLException{
59         String userid = this.requestData.get("userid");
60         String password = this.requestData.get("password");
61         if (userid.length() > 128) {
62             return badRequest(this.return_invalid_credentials("User ID length is more than allowed (" +
63                 userid.length() + ")"));
64         }
65         if (password.length() > 32) {

```

```

64         return badRequest(this.return_invalid_credentials("Password length is more than allowed (" +
65             password.length() + ")");
66     }
67     // Retrieve the user information
68     java.sql.Connection connection = DB.getConnection();
69     Statement statement = connection.createStatement();
70     ResultSet result = statement.executeQuery("SELECT * FROM users WHERE email='"+userid+"'");
71     if(!result.next()){
72         System.out.println("userid tidak ditemukan");
73         return badRequest(this.return_invalid_credentials(null));
74     }
75
76     String hasher=result.getString("password");
77     if(!hasher.equals(password)){
78         System.out.println("password salah");
79         return badRequest(this.return_invalid_credentials(null));
80     }
81     StringBuilder privileges= new StringBuilder();
82     if (result.getInt("privilegeRoute") != 0) {
83         privileges.append(",route");
84     }
85     if (result.getInt("privilegeApiUsage") != 0) {
86         privileges.append(",apiusage");
87     }
88     if (privileges.length() > 0) {
89         privileges=new StringBuilder(privileges.substring(1));
90     }
91     ObjectNode obj = Json.newObject();
92     obj.put("status", "ok");
93     obj.put("sessionid", "e27wy7s3f08fmu13");
94     obj.put("privileges", privileges.toString());
95     return ok(obj);
96 }
97
98 private Result register() throws IOException, SQLException{
99     String email = this.requestData.get("userid");
100     String fullname = this.requestData.get("fullname");
101     String company = this.requestData.get("company");
102
103     // Check if the email has already been registered.
104     java.sql.Connection connection = DB.getConnection();
105     Statement statement = connection.createStatement();
106     ResultSet result = statement.executeQuery("SELECT email FROM users WHERE email='"+email+"'");
107     while (result.next()) {
108         return badRequest("udah ada usernya");
109     }
110
111     // Generate password tanpa fitur hash and send password (belum dilakukan)
112     String password = this.generate_password();
113     statement.executeUpdate("INSERT INTO users(email, password, privilegeApiUsage, fullName, company)
114         VALUES('"+ email + "', '"+ password + "', 1, '"+ fullname + "', '"+ company + "')");
115
116     return ok(this.well_done(null));
117 }
118
119
120 private String generate_password(){
121     return this.generate_random("abcdefghijklmnopqrstuvwxyz0123456789", 8);
122 }
123
124 private String generate_random(String chars, int length){
125     int chars_size = chars.length();
126     Random random=new Random();
127     String string = "";
128     for (int i = 0; i < length; i++) {
129         string += chars.charAt(random.nextInt(chars_size));
130     }
131     return string;
132 }
133
134 private ObjectNode well_done(String message) {
135     ObjectNode obj = Json.newObject();
136     obj.put("status", "ok");
137     if (message != null) {
138         obj.put("status", message);
139     }
140     return obj;
141 }
142
143 public ObjectNode return_invalid_credentials(String logmessage) {
144     ObjectNode obj = Json.newObject();
145     obj.put("status", "credentialfail");
146     return obj;
147 }
148 }

```

6. Melakukan pengujian terhadap fitur-fitur yang sudah dibuat
  - status** : Ada sejak rencana kerja skripsi.
  - hasil** : belum ada perkembangan.

7. Menulis dokumen skripsi.

**status** : Ada sejak rencana kerja skripsi.

**hasil** : Sudah menulis dokumen hingga bab 3 sebagian.

## Pustaka

- [1] The Eclipse Foundation, “About the Eclipse Foundation.” <https://eclipse.org/org>, 2015. [Online; diakses 25-November-2015].
- [2] GitHub Inc, “About GitHub.” <https://github.com/>, 2015. [Online; diakses 25-November-2015].
- [3] Apache Ant, “Apache Ant.” <http://ant.apache.org/>, 2015. [Online; diakses 25-November-2015].
- [4] GWT, “Overview GWT.” <http://www.gwtproject.org/>, 2015. [Online; diakses 25-November-2015].
- [5] Google Developers, “Keyhole Markup Language.” <https://developers.google.com/kml/>, 2015. [Online; diakses 26-November-2015].
- [6] Jan Goyvaerts, “The Premier website about Regular Expressions.” <http://www.regular-expressions.info/>, 2015. [Online; diakses 1-Desember-2015].
- [7] Oracle, “MySQL 5.7 Reference Manual.” <https://dev.mysql.com/doc/refman/5.7/en/>, 2015. [Online; diakses 4-November-2015].
- [8] GeoJSON, “Representasi Objek dalam Geometri.” <http://geojson.io/>, 2015. [Online; diakses 4-November-2015].
- [9] Oracle, “Java Documentation.” <https://docs.oracle.com/javase/8/>, 2015. [Online; diakses 26-November-2015].
- [10] N. Leroux and S. D. Kaper, *Play for Java*. Manning Publications Co., 2014.
- [11] Play Framework, “Play 2.4.x documentation.” <https://www.playframework.com/documentation/2.4.x/Home>, 2015. [Online; diakses 4-November-2015].
- [12] Standard ECMA-262 3rd Edition, “Introducing JSON.” <http://www.json.org/>, 2015. [Online; diakses 1-Desember-2015].
- [13] W3Schools, “AJAX Tutorial.” <http://www.w3schools.com/ajax/>, 2015. [Online; diakses 1-Desember-2015].

## 3 Pencapaian Rencana Kerja

Persentase penyelesaian skripsi sampai dengan dokumen ini dibuat dapat dilihat pada tabel berikut :

1*	2*(%)	3*(%)	4*(%)	5*	6*(%)
1	10	10			10
2	10	10			10
3	15	15			10
4	15		15		0
5	15		15		5
6	15		15		0
7	20	5	15		5
Total	100	40	60		40

Keterangan (\*)

- 1 : Bagian pengerjaan Skripsi (nomor disesuaikan dengan detail pengerjaan di bagian 5)
- 2 : Persentase total
- 3 : Persentase yang akan diselesaikan di Skripsi 1
- 4 : Persentase yang akan diselesaikan di Skripsi 2
- 5 : Penjelasan singkat apa yang dilakukan di S1 (Skripsi 1) atau S2 (skripsi 2)
- 6 : Persentase yang sudah diselesaikan sampai saat ini

Bandung, 11/10/2015

Tommy Adhitya The

Menyetujui,

Nama: Pascal Alfadian, M.Com.  
Pembimbing Tunggal