

# *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 kode situs web KIRI *Dashboard Server Side*(bahasa PHP).

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

**hasil :**

Berikut adalah kode situs web KIRI *Dashboard Server Side*(bahasa PHP):

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) > $maximum_userid_length) {
38         return_invalid_credentials("UserID length is more than allowed (". strlen($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='$userid'" ) or
46     die_nice('Failed_to_verify_user_id:_'. mysqli_error($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,
110         internalInfo)VALUES_(' $trackid ',' $tracktype ',' $trackname ',' $penalty ',' $internalinfo')") or
111         die_nice('Failed_to_add_a_new_track:_'. mysqli_error($global_mysql_link), true);
112     update_trackversion();
113 } else {
114     die_nice("The_trackId_ '$trackid' _already_existed.", true);
115 }
116 deinit_mysql();
117 well_done();
118 } elseif ($mode == $proto_mode_update_track) {
119     check_privilege($privilege_route);
120     $trackid = addslashes(retrieve_from_post($proto_trackid));
121     $newtrackid = addslashes(retrieve_from_post($proto_new_trackid));
122     $tracktype = addslashes(retrieve_from_post($proto_tracktype));
123     $trackname = addslashes(retrieve_from_post($proto_trackname));
124     $internalinfo = addslashes(retrieve_from_post($proto_internalinfo, false)) or $internalinfo = '';
125     $pathloop = retrieve_from_post($proto_pathloop) == 'true' ? 1 : 0;
126     $penalty = addslashes(retrieve_from_post($proto_penalty));
127     $transferrnodes = retrieve_from_post($proto_transferrnodes, false);
128
129 // When changed, check if the id is already existed
130 if ($newtrackid != $trackid) {
131     $result = mysqli_query($global_mysql_link, "SELECT_trackId FROM_tracks_WHERE_trackId='$newtrackid'"
132         ) or
133         die_nice('Failed_to_check_trackid_existence:_'. mysqli_error($global_mysql_link), true);
134     if (mysqli_num_rows($result) != 0) {
135         die_nice("The_new_trackId_ '$newtrackid' _already_existed.", true);
136     }
137 }

```

```

135     }
136     mysqli_query($global_mysqli_link, "UPDATE_tracks SET_trackTypeId='$tracktype',_trackId='$newtrackid',_
        trackName='$trackname',_internalInfo='$internalinfo',_pathloop='$pathloop',_penalty='$penalty'
        WHERE_trackId='$trackid'" ) or
137     die_nice('Failed_to_update_the_track:_' . mysqli_error($global_mysqli_link));
138     if (!is_null($transfereodes)) {
139         $transfereodes = addslashes($transfereodes);
140         mysqli_query($global_mysqli_link, "UPDATE_tracks SET_transferNodes='$transfereodes' WHERE_trackId='
            $trackid'" ) or
141         die_nice('Failed_to_update_the_track:_' . mysqli_error($global_mysqli_link));
142     }
143     update_trackversion();
144     deinit_mysql();
145     well_done();
146 } elseif ($mode == $proto_mode_list_tracks) {
147     check_privilege($privilege_route);
148     // Retrieve track list from database
149     $result = mysqli_query($global_mysqli_link, 'SELECT_trackTypeId,_trackId,_trackName_FROM_tracks_ORDER_BY
        _trackTypeId,_trackId') or
150     die('Cannot_retrieve_the_track_names_from_database');
151     $track_list = array();
152     while ($row = mysqli_fetch_row($result)) {
153         $track_list[] = array($row[1], htmlspecialchars($row[0] . '/' . $row[2]));
154     }
155     // Retrieve track types list result from database
156     $result = mysqli_query($global_mysqli_link, 'SELECT_trackTypeId,_name_FROM_tracktypes_ORDER_BY_
        trackTypeId') or
157     die_nice('Cannot_retrieve_the_track_types_from_database');
158     $tracktype_list = array();
159     while ($row = mysqli_fetch_row($result)) {
160         $tracktype_list[] = array($row[0], htmlspecialchars($row[1]));
161     }
162
163     // Construct json.
164     $json = array(
165         $proto_status => $proto_status_ok,
166         $proto_trackslist => $track_list,
167         $proto_tracktypeslist => $tracktype_list
168     );
169
170     deinit_mysql();
171     print(json_encode($json));
172 } elseif ($mode == $proto_mode_getdetails_track) {
173     check_privilege($privilege_route);
174     $trackid = addslashes(retrieve_from_post($proto_trackid));
175
176     // Retrieve result from database and construct in XML format
177     $result = mysqli_query($global_mysqli_link, "SELECT_trackTypeId,_trackName,_internalInfo,_AsText(geodata
        ),_pathloop,_penalty,_transferNodes_FROM_tracks WHERE_trackId='$trackid'" ) or
178     die_nice("Can't_retrieve_the_track_details_from_database:_" . mysqli_error($global_mysqli_link),
        true);
179     $i = 0;
180     $row = mysqli_fetch_row($result);
181     if ($row == FALSE) {
182         die_nice("Can't_find_track_information_for_'$trackid'", true);
183     }
184     $geodata = lineStringToLatLngArray($row[3]);
185     // Construct json.
186     $json = array(
187         $proto_status => $proto_status_ok,
188         $proto_trackid => $trackid,
189         $proto_tracktype => $row[0],
190         $proto_trackname => $row[1],
191         $proto_internalinfo => $row[2],
192         $proto_geodata => $geodata,
193         $proto_pathloop => ($row[4] > 0 ? true : false),
194         $proto_penalty => doubleval($row[5]),
195         $proto_transfereodes => is_null($row[6]) ? array('0-' . (count($geodata) - 1)) : split(',', $row[6])
196     );
197
198     deinit_mysql();
199     print(json_encode($json));
200 } elseif ($mode == $proto_mode_cleargeodata) {
201     check_privilege($privilege_route);
202     $trackid = addslashes(retrieve_from_post($proto_trackid));
203
204     mysqli_query($global_mysqli_link, "UPDATE_tracks SET_geodata=NULL,_transferNodes=NULL WHERE_trackId='
        $trackid'" ) or
205     die_nice('Failed_to_clear_the_geodata:_' . mysqli_error($global_mysqli_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_$num_matches_occurences", true);
227 }
228 fclose($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_mysqli_link, "UPDATE_ apikeys_ SET_ domainFilter=' $domainfilter ' , _description='
313     $description ' _WHERE_ verifier=' $apikey '") or
314     die_nice('Failed_to_update_API_Key:_ ' . mysqli_error($global_mysqli_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_mysqli_link, "SELECT_email_FROM_ users_ WHERE_ email=' $email '") or
325         die_nice('Cannot_check_user_id_existence:_ ' . mysqli_error($global_mysqli_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_mysqli_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_mysqli_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_mysqli_link, "SELECT_fullName, _company_FROM_ users_ WHERE_ email=' $email
348         '") or
349         die_nice('Cannot_retrieve_user_details:_ ' . mysqli_error($global_mysqli_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_mysqli_link, "UPDATE_ users_ SET_ password=' $passwordHash ' _WHERE_ email=' $email '")
377             or
378             die_nice('Cannot_update_password_for_ $email:_ ' . mysqli_error($global_mysqli_link));
379     }
380     mysqli_query($global_mysqli_link, "UPDATE_ users_ SET_ fullName=' $fullname ' , _company=' $company ' _WHERE_ email
381         =' $email '") or
382         die_nice('Cannot_update_profile_for_ $email:_ ' . mysqli_error($global_mysqli_link));
383
384     deinit_mysql();
385     well_done();
386 } else {
387     die_nice("Mode_not_understood:_ \" " . $mode . "\" , true);
388 }
389
390 /**
391  * Return invalid credential error, close mysql connection, and exit.
392  * @param string $logmessage the message to record in the log file.
393  */
394 function return_invalid_credentials($logmessage) {
395     global $proto_status, $proto_status_credentialfail, $errorlog_file, $global_mysqli_link;
396     $ip_address = $SERVER['REMOTE_ADDR'];
397     log_error("Login_ failed_ (IP= $ip_address):_ $logmessage", '...' . $errorlog_file);
398     $json = array(
399         $proto_status => $proto_status_credentialfail;
400     );
401     print(json_encode($json));
402     mysqli_close($global_mysqli_link);
403     exit(0);
404 }

```

```

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 ?>

```

Berdasarkan hasil analisa dan wawancara dengan kontributor kode, kode situs web KIRI *Dashboard Server Side* (bahasa PHP) terbagi dalam 16 bagian yang masing-masing melayani sebuah permintaan tertentu.

- **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 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[1].

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 teknik *regular expression* (baris 224). 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, JSON, dan *Regular Expression*.

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

**hasil :**

- **MySQL Spatial Extensions**

Suatu *geographic feature*[2] 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:

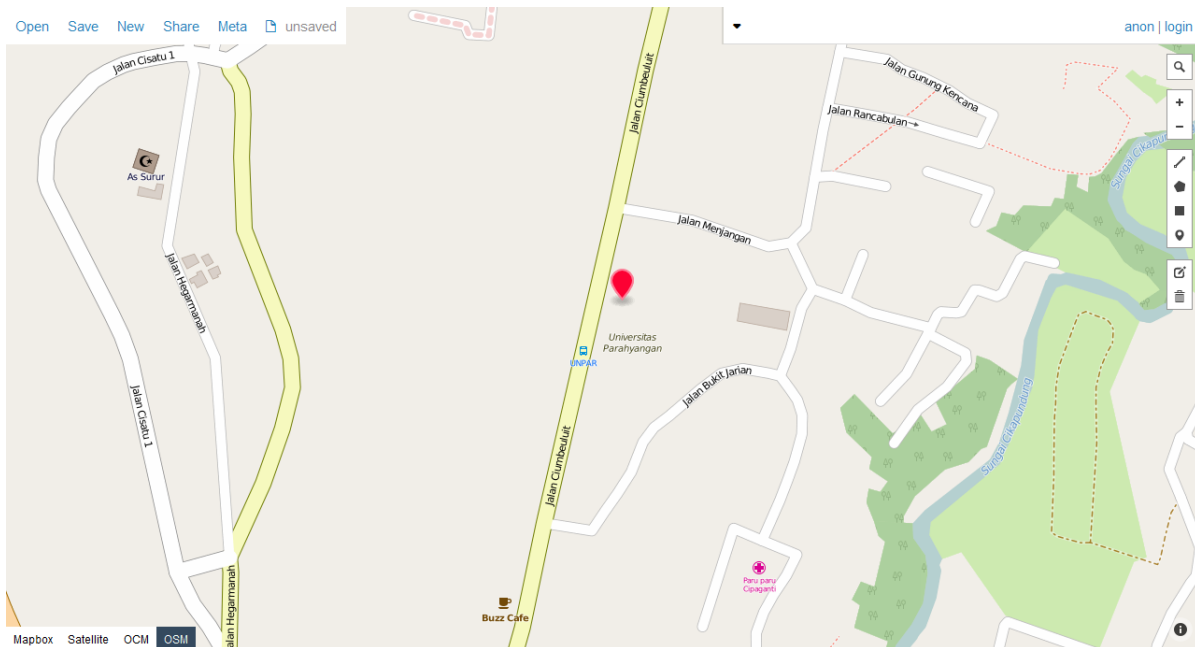
- *Geometry*
- *Point*
- *LineString*
- *Polygon*

(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[2]. 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 1).

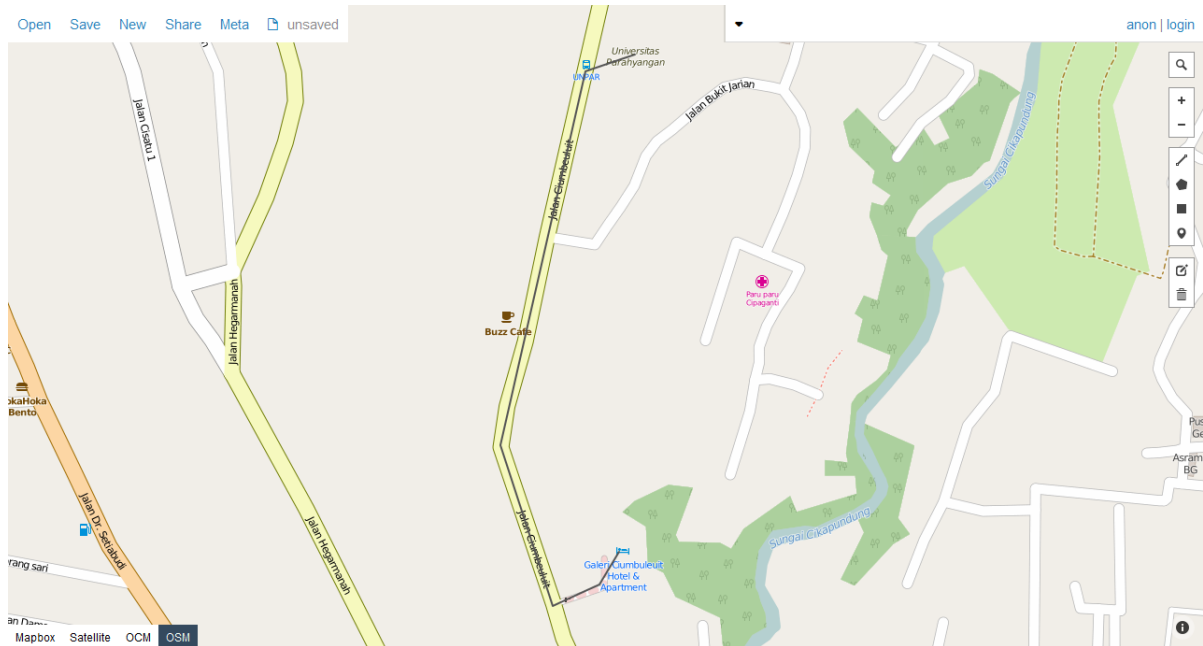


Gambar 1: Universitas Katolik Parahyangan dinyatakan dalam *Point*[3]

### *LineString*

*LineString* adalah garis yang terbentuk dari sekumpulan *Point*[2]. Dalam peta dunia, *LineString* dapat merepresentasikan sebuah sungai dan dalam peta perkotaan, *LineString* dapat merepresentasikan sebuah jalan (contoh: gambar 2). 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 2: Rute jalan dari Universitas Katolik Parahyangan menuju Galeri Ciumbuleuit dinyatakan dalam *LineString*[3]

### Format Well-Known Text (WKT)

Format Well-Known Text (WKT) adalah salah satu aturan penulisan tipe data *spatial* untuk merepresentasikan suatu *geographic feature*[2]. 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*[4]. 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[4]. 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*[4]. 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[4]. 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*[4]. 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**

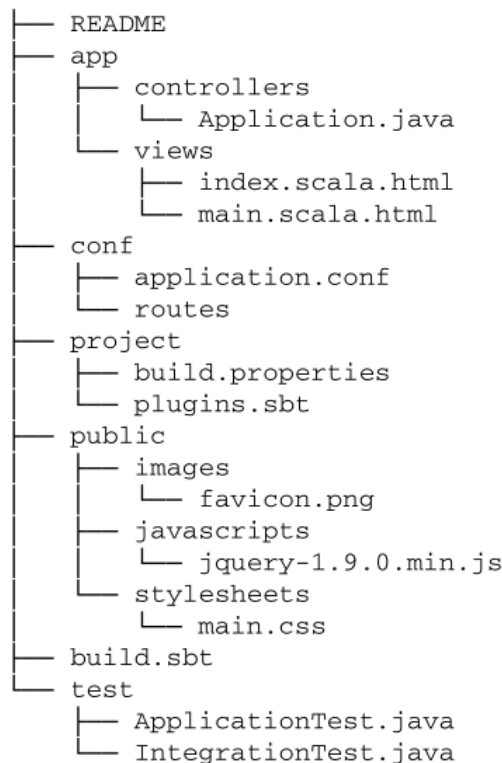
Play Framework adalah sekumpulan kerangka kode yang dapat digunakan untuk membangun suatu situs web[5]. Play Framework tidak hanya menggunakan bahasa Java dalam pembuatannya. Bahasa Scala juga digunakan Play Framework dalam beberapa bagian seperti bagian *view* dan *route*. Play Framework menggunakan 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 3). 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 3: 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 4), yaitu HTTP *method*, URL *path*, dan *action method*. Struktur *routes* seperti yang dijelaskan pada gambar 4 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”[6]. 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 4), dan terakhir adalah pemilihan *method* yang ada pada kelas “controllers” yang dipilih (contohnya: “list()”).



Gambar 4: Struktur kode file “routes”[5]

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 diletakan 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 3). 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 3) harus dengan format sebagai berikut, “namaFile.scala.html”.

Berikut adalah contoh struktur kode *views*:

```
1 | @(name:String)
2 | <!doctype html>
3 | <html>
4 |   <head>
```



```

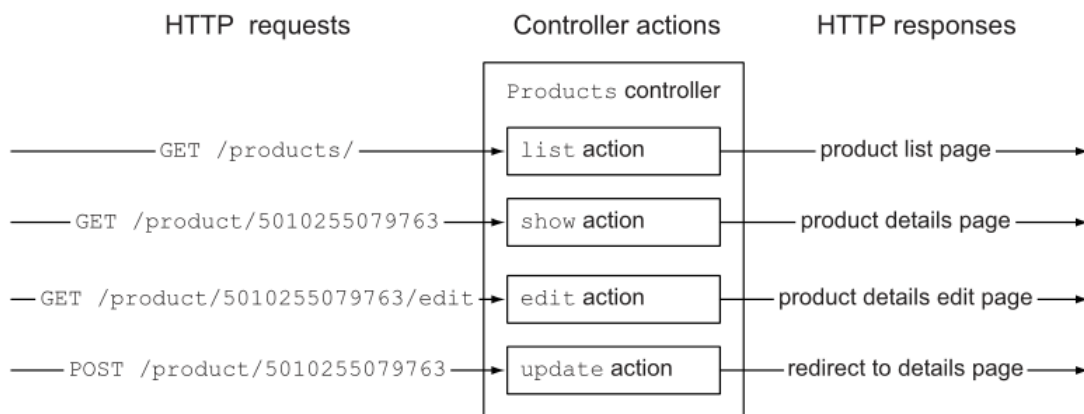
5 |         <meta charset="UTF-8">
6 |         <title>Hello </title>
7 |     </head>
8 |     <body>
9 |         <h1>Hello <em>@name</em></h1>
10 |     </body>
11 | </html>

```

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 5). Fungsi dari *controllers* dalam arsitektur MVC adalah sebagai penghubung antara *models* dan *views*.



Gambar 5: Hubungan *routes* dan *controllers* dalam memproses HTTP requests[5]

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[6]. 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 3), 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 3), 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**

- ***Regular Expression***

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

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

**hasil :** Mengerti penggunaan *controllers*, *routes*, dan *views* (sedikit). Mencoba membuat kode untuk fungsi CRUD (masih gagal).

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 :** belum ada perkembangan.

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 skripsi bab 1 dan bab 2.

## Pustaka

- [1] Google Developers, “Keyhole Markup Language.” <https://developers.google.com/kml/>, 2015. [Online; diakses 26-November-2015].
- [2] Oracle, “MySQL 5.7 Reference Manual.” <https://dev.mysql.com/doc/refman/5.7/en/>, 2015. [Online; diakses 4-November-2015].
- [3] GeoJSON, “Representasi Objek dalam Geometri.” <http://geojson.io/>, 2015. [Online; diakses 4-November-2015].
- [4] Oracle, “Java Documentation.” <https://docs.oracle.com/javase/8/>, 2015. [Online; diakses 26-November-2015].
- [5] N. Leroux and S. D. Kaper, *Play for Java*. Manning Publications Co., 2014.
- [6] Play Framework, “Play 2.4.x documentation.” <https://www.playframework.com/documentation/2.4.x/Home>, 2015. [Online; diakses 4-November-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			6
2	10	10			8
3	15	15			3
4	15		15		0
5	15		15		0
6	15		15		0
7	20	5	15	penulisan skripsi hingga bab 3 pada S1	3
Total	100	40	60		20

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