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 terkahir :

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

```
<?php
require_once '../../etc/utils.php';
require_once '../../etc/constants.php';
require_once '../../etc/PasswordHash.php';

start_working();

$mode = retrieve_from_post($proto_mode);

// Initializes MySQL and check for session
init_mysql();
if ($mode != $proto_mode_login && $mode != $proto_mode_logout && $mode != $proto_mode_register) {
    $sessionid = addslashes(retrieve_from_post($proto_sessionid));
    // Clear expired sessions
    mysqli_query($global_mysqli_link, "DELETE FROM sessions WHERE lastSeen < (NOW() - INTERVAL
        $session_expiry_interval_mysql)") or
        die_nice('Failed to clean expired sessions: ' . mysqli_error($global_mysqli_link), true);
    $result = mysqli_query($global_mysqli_link, "SELECT users.email, users.privilegeRoute, users.
        privilegeApiUsage FROM users LEFT JOIN sessions ON users.email = sessions.email WHERE sessions.
        sessionId = '$sessionid'") or
        die_nice('Failed to get user session information: ' . mysqli_error($global_mysqli_link), true);
    if (mysqli_num_rows($result) == 0) {
        deinit_mysql();
        // Construct json - session expired.
        $json = array(
            $proto_status => $proto_status_sessionexpired,
        );
        print(json_encode($json));
        exit(0);
    }
    $columns = mysqli_fetch_row($result);
    $active_userid = $columns[0];
    $privilege_route = $columns[1] != '0';
    $privilege_apiUsage = $columns[2] != '0';
}

if ($mode == $proto_mode_login) {
    $userid = addslashes(retrieve_from_post($proto_userid));
    $plain_password = addslashes(retrieve_from_post($proto_password));
    if (strlen($userid) > $maximum_userid_length) {
        return_invalid_credentials("User ID length is more than allowed (". strlen($userid) . ')');
    }
    if (strlen($plain_password) > $maximum_password_length) {
        return_invalid_credentials('Password length is more than allowed ('. strlen($password) . ')');
    }
```

```
43
44      // Retrieve the user information
45      $result = mysqli_query($global_mysqli_link, "SELECT * FROM users WHERE email='$userid'") or
46          die_nice('Failed to verify user id: ' . mysqli_error($global_mysqli_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_mysqli_link, "INSERT INTO sessions (sessionId, email) VALUES ('$sessionid', '
            $userid')") or
66          die_nice('Failed to generate session: ' . mysqli_error($global_mysqli_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_mysqli_link, "DELETE FROM sessions WHERE sessionId='$sessionid'") or
94          die_nice('Failed to logout sessionid $sessionid: ' . mysqli_error($global_mysqli_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_mysqli_link, "SELECT trackId FROM tracks WHERE trackId='$trackid'") or
107         die_nice('Failed to check trackid existence: ' . mysqli_error($global_mysqli_link), true);
108     if (mysqli_num_rows($result) == 0) {
109         mysqli_query($global_mysqli_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_mysqli_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_mysqli_link, "SELECT trackId FROM tracks WHERE trackId='$newtrackid'"
                ) or
131             die_nice('Failed to check trackid existence: ' . mysqli_error($global_mysqli_link), true);
132         if (mysqli_num_rows($result) != 0) {
133             die_nice("The new trackId '$newtrackid' already existed.", true);
134         }
```

```
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($global_mysqli_link));
138         if (!is_null($transfernodes)) {
139             $transfernodes = addslashes($transfernodes);
140             mysqli_query($global_mysqli_link, "UPDATE tracks SET transferNodes='$transfernodes'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             $proto_transfernodes => is_null($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_mysqli_link, "UPDATE tracks SET geodata=GeomFromText('$output'), transferNodes=NULL
            WHERE trackId='$trackid'") or
242         die_nice("Error updating the goedata: " . mysqli_error($global_mysqli_link), true);
243     update_trackversion();
244     deinit_mysql();
245     well_done();
246 } elseif ($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($global_mysqli_link, "DELETE FROM tracks WHERE trackId='$trackid'") or
254         die_nice('Failed to delete track $trackid: ' . mysqli_error($global_mysqli_link), true);
255     if (mysqli_affected_rows($global_mysqli_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_mysqli_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_mysqli_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_mysqli_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($global_mysqli_link));
288
289     log_statistic("$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_mysqli_link, "SELECT email FROM apikeys WHERE verifier='apikey'") or
306         die_nice('Cannot check API key owner: ' . mysqli_error($global_mysqli_link));
307     while ($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='
            $description' WHERE verifier='$apikey'") or
313         die_nice('Failed to update API Key: ' . mysqli_error($global_mysqli_link));
314
315     deinit_mysql();
316     well_done();
317 } elseif ($mode == $proto_mode_register) {
318     $email = addslashes(retrieve_from_post($proto_userid));
319     $fullname = addslashes(retrieve_from_post($proto_fullname));
320     $company = addslashes(retrieve_from_post($proto_company));
321
322     // Check if the email has already been registered.
323     $result = mysqli_query($global_mysqli_link, "SELECT email FROM users WHERE email='$email'") or
324         die_nice('Cannot check user id existence: ' . mysqli_error($global_mysqli_link));
325     if (mysqli_num_rows($result) > 0) {
326         die_nice("Ooops! Email $email has already registered. Please check your mailbox or contact
                hello@kiri.travel");
327     }
328
329     // Generate and send password
330     $password = generate_password();
331     $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
332     $passwordHash = $hasher->HashPassword($password);
333     mysqli_query($global_mysqli_link, "INSERT INTO users(email, password, privilegeApiUsage, fullName,
            company) VALUES('$email', '$passwordHash', 1, '$fullname', '$company')") or
334         die_nice('Cannot add new user $email: ' . mysqli_error($global_mysqli_link));
335     sendPassword($email, $password, $fullname);
336
337     log_statistic("$apikey_kiri", 'REGISTER', "$email/$fullname/$company");
338
339     deinit_mysql();
340     well_done();
341 } elseif ($mode == $proto_mode_getprofile) {
342         $email = $active_userid;
343
344         $result = mysqli_query($global_mysqli_link, "SELECT fullName, company FROM users WHERE email='$email
                '") or
345             die_nice('Cannot retrieve user details: ' . mysqli_error($global_mysqli_link));
346         if ($row = mysqli_fetch_row($result)) {
347             $fullname = $row[0];
348             $company = $row[1];
349         } else {
350             die_nice("User $email not found in database.");
351         }
352
353         deinit_mysql();
354         // Construct json.
355         $json = array(
356                 $proto_status => $proto_status_ok,
357                 $proto_fullname => $fullname,
358                 $proto_company => $company
359         );
360
361         print(json_encode($json));
362 } elseif ($mode == $proto_mode_update_profile) {
363     $email = $active_userid;
364     $password = addslashes(retrieve_from_post($proto_password, false));
365     $fullname = addslashes(retrieve_from_post($proto_fullname));
366     $company = addslashes(retrieve_from_post($proto_company));
367
368     // Updates password if necessary
369     if (!is_null($password) && $password != "") {
370         $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
371         $passwordHash = $hasher->HashPassword($password);
372         mysqli_query($global_mysqli_link, "UPDATE users SET password='$passwordHash' WHERE email='$email'")
                or
373             die_nice('Cannot update password for $email: ' . mysqli_error($global_mysqli_link));
374     }
375     mysqli_query($global_mysqli_link, "UPDATE users SET fullName='$fullname', company='$company' WHERE email
            ='$email'") or
376         die_nice('Cannot update profile for $email: ' . mysqli_error($global_mysqli_link));
377
378     deinit_mysql();
379     well_done();
380 } else {
381     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 ?>
```

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*, *transfernodes* 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 *regular expression* (baris 224). *Regular expression* adalah karakter atau kata spesial yang digunakan untuk menjelaskan pola pencarian[2]. 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*[3] 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*.

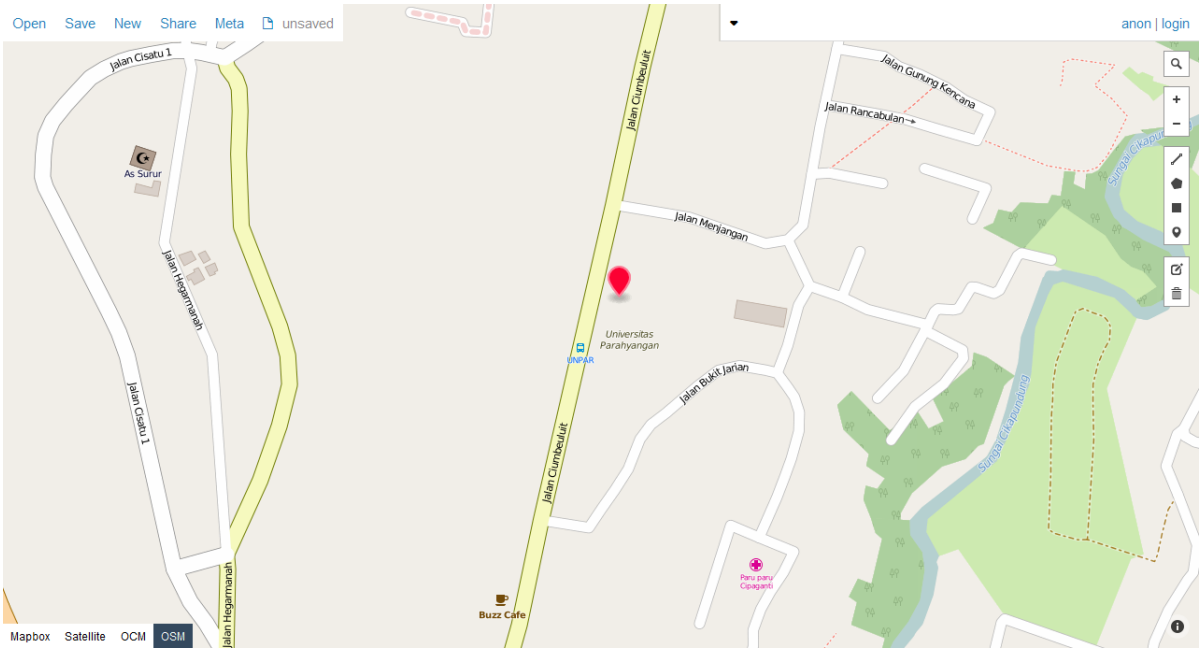
Berdasarkan kemampuan penyimpanan nilai geometri, tipe data *spatial* dapat dikelompokan ke dalam 2 jenis:

(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[3]. 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).

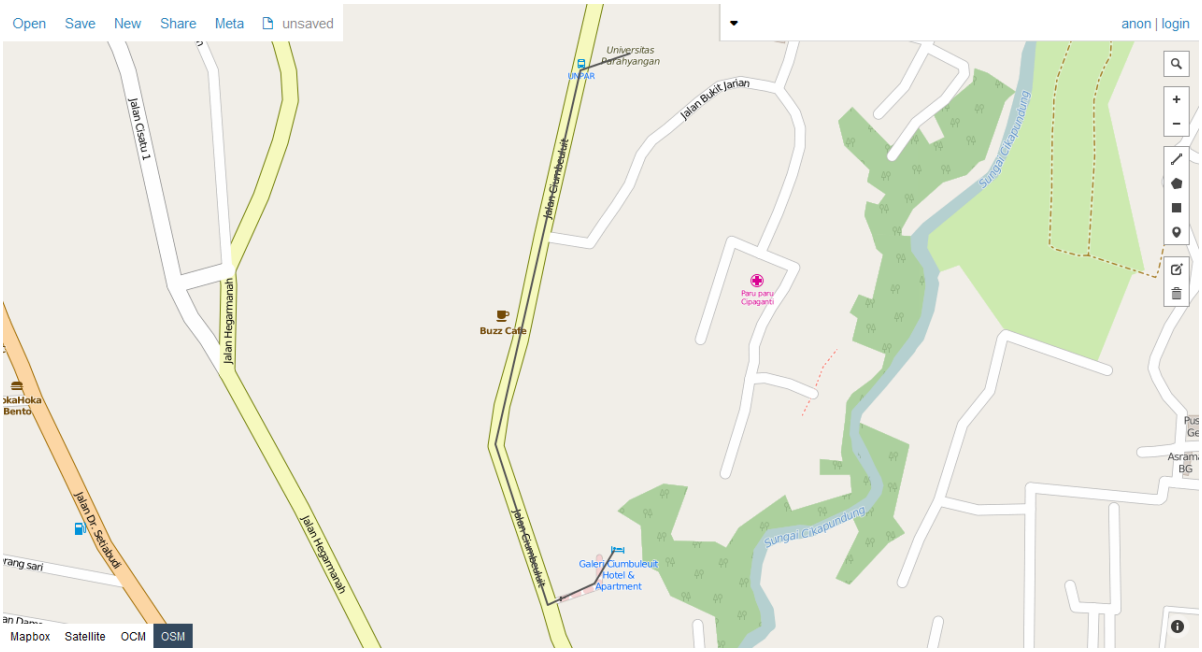


Gambar 1: Universitas Katolik Parahyangan dinyatakan dalam *Point*[4]

LineString

LineString adalah garis yang terbentuk dari sekumpulan *Point*[3]. 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 ($X_1 \ldots X_n$ dan $Y_1 \ldots Y_n$,

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*[4]

Format Well-Known Text (WKT)

Format Well-Known Text (WKT) adalah salah satu aturan penulisan tipe data *spatial* untuk merepresentasikan suatu *geographic feature*[3]. 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                        107.60407805442 81 -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                        107.60407805442 81 -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*[5]. 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:myDriver: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[5]. 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*[5]. 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[5]. 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*[5]. 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[6]. 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.

```
├── README
├── app
│   ├── controllers
│   │   └── Application.java
│   └── views
│       ├── index.scala.html
│       └── main.scala.html
├── conf
│   ├── application.conf
│   └── routes
├── project
│   ├── build.properties
│   └── plugins.sbt
├── public
│   ├── images
│   │   └── favicon.png
│   ├── javascripts
│   │   └── jquery-1.9.0.min.js
│   └── stylesheets
│       └── main.css
├── build.sbt
└── test
    ├── ApplicationTest.java
    └── IntegrationTest.java
```

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`"[7]. 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"[6]

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*:

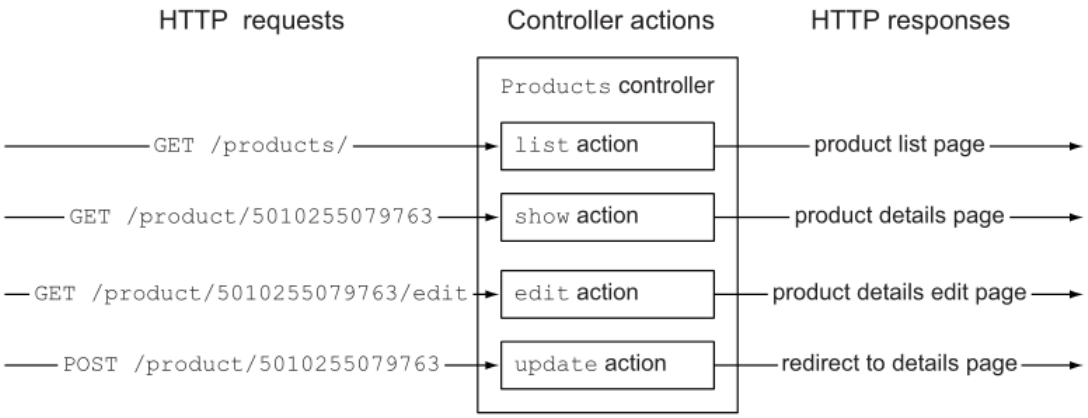
```
1  @(name:String)
2  <!doctype html>
3  <html>
4      <head>
```

```
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$_1$: tipeVariabel$_1$) (namaVariabel$_2$: tipeVariabel$_2$) ... (namaVariabel$_n$: tipeVariabel$_n$)", 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*[6]

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:

(a) *Visibility* kelas dan *method* pada kelas tersebut harus *public* (baris 5),

(b) Kelas yang dibuat harus merupakan turunan dari "`play.mvc.Controller`" (baris 5),

(c) 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[7]. 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**

JSON (*JavaScript Object Notation*) adalah sebuah format pertukaran data ringan[8]. JSON dapat dibangun dalam 2 buah struktur:

(a) 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 "`,`".

(b) 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 : 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] Jan Goyvaerts, "The Premier website about Regular Expressions." `http://www.regular-expressions.info/`, 2015. [Online; diakses 1-Desember-2015].

[3] Oracle, "MySQL 5.7 Reference Manual." `https://dev.mysql.com/doc/refman/5.7/en/`, 2015. [Online; diakses 4-November-2015].

[4] GeoJSON, "Representasi Objek dalam Geometri." `http://geojson.io/`, 2015. [Online; diakses 4-November-2015].

[5] Oracle, "Java Documentation." `https://docs.oracle.com/javase/8/`, 2015. [Online; diakses 26-November-2015].

[6] N. Leroux and S. D. Kaper, *Play for Java*. Manning Publications Co., 2014.

[7] Play Framework, "Play 2.4.x documentation." `https://www.playframework.com/documentation/2.4.x/Home`, 2015. [Online; diakses 4-November-2015].

[8] Standard ECMA-262 3rd Edition, "Introducing JSON." `http://www.json.org/`, 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			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 sidah diselesaikan sampai saat ini

Bandung, 11/10/2015

Tommy Adhitya The

Menyetujui,

Nama: Pascal Alfadian, M.Com.
Pembimbing Tunggal