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 :

   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 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 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 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 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 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 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 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 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 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, referensi subbab **??**). Baris 231-239 program akan membangun data LineString yang semula dalam format KML menjadi format WKT (subbab **??**). 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 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 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 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 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 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 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 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 dan Play Framework.
   status : Ada sejak rencana kerja skripsi.
   hasil : Sudah melakukan studi literatur mengenai MySQL Spatial Extensions. Mengerti cara menggunakan tipe data *spatial* dalam MySQL, terutama tipe data *LineString* yang digunakan dalam peneletian ini. Sudah melakukan instalasi Play Framework di laptop. Mengerti cara kerja Play Framework secara umum, bagaimana struktur *direktori* Play Framework, bagaimana *server* Play Framework menerima HTTP *requests* dari pengguna yang diterjemahkan melalui *routes* dan dilanjutkan ke *controllers* yang nantinya akan membalas HTTP *requests* dengan memberikan tampilan berupa *views*. Mengerti dan dapat membuat aplikasi "Hellow World" dengan menggunakan Play Framework.

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.

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) . ')');
    }

    // Retrieve the user information
    $result = mysqli_query($global_mysqli_link, "SELECT * FROM users WHERE email='$userid'") or
        die_nice('Failed to verify user id: ' . mysqli_error($global_mysqli_link), true);
    if (mysqli_num_rows($result) == 0) {
        deinit_mysql();
        return_invalid_credentials("User id not found: $userid");
    }
    $userdata = mysqli_fetch_assoc($result);

    // Check against the stored hash.
    $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
    if (!$hasher->CheckPassword($plain_password, $userdata['password'])) {
        log_statistic("$apikey_kiri", 'LOGIN', $userid . '/FAIL');
        deinit_mysql();
        return_invalid_credentials("Password mismatch for $userid");
    }

    log_statistic("$apikey_kiri", 'LOGIN', $userid . '/SUCCESS');

    // Create session id
    $sessionid = generate_sessionid();
    mysqli_query($global_mysqli_link, "INSERT INTO sessions (sessionId, email) VALUES ('$sessionid', '$userid')")
        or
        die_nice('Failed to generate session: ' . mysqli_error($global_mysqli_link), true);

    // Construct privilege lists
    $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     ?>
```

3 Pencapaian Rencana Kerja

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

Pustaka

[1] Google Developers, "Keyhole Markup Language." `https://developers.google.com/kml/`, 2015. [Online; diakses 26-November-2015].

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