

SKRIPSI

*PORTING PHP MENJADI JAVA/PLAY FRAMEWORK (STUDI
KASUS KIRI DASHBOARD SERVER SIDE)*



TOMMY ADHITYA THE

NPM: 2012730031

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

**PORTING PHP TO JAVA/PLAY FRAMEWORK (CASE STUDY
KIRI DASHBOARD SERVER SIDE)**



TOMMY ADHITYA THE

NPM: 2012730031

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015**

LEMBAR PENGESAHAN

PORTING PHP MENJADI JAVA/PLAY FRAMEWORK (STUDI KASUS KIRI *DASHBOARD SERVER SIDE*)

TOMMY ADHITYA THE

NPM: 2012730031

Bandung, 17 September 2015

Menyetujui,

Pembimbing Utama

Pembimbing Pendamping

Pascal Alfadian, M.Com.

«pembimbing pendamping/2»

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Aditia, PDEng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PORTING PHP MENJADI JAVA/PLAY FRAMEWORK (STUDI KASUS KIRI DASHBOARD SERVER SIDE)

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 17 September 2015

Meterai

Tommy Adhitya The
NPM: 2012730031

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris» Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini...?»

KATA PENGANTAR

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Bandung, September 2015

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xx
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	3
1.5 Metode Penelitian	3
1.6 Sistematika Penulisan	3
2 DASAR TEORI	5
2.1 MySQL Spatial Extensions	5
2.1.1 <i>Point</i>	6
2.1.2 <i>LineString</i>	6
2.1.3 Format Well-Known Text (WKT)	7
2.2 JDBC	8
2.2.1 <i>Interface</i> Connection	8
2.2.2 Kelas DriverManager	8
2.2.3 <i>Interface</i> Statement	9
2.2.4 <i>Interface</i> ResultSet	9
2.3 Play Framework	10
2.3.1 Struktur Aplikasi	10
2.3.2 <i>Routes</i>	11
2.3.3 <i>Models</i>	12
2.3.4 <i>Views</i>	12
2.3.5 <i>Controllers</i>	13
2.3.6 <i>Database</i>	14
2.4 JSON	14
2.5 <i>Regular Expression</i>	15
3 ANALISIS	17
3.1 Analisis Sistem Kini	17
3.1.1 Bagian Pemeriksaan <i>Login</i>	21
3.1.2 Bagian <i>Login</i>	21
3.1.3 Bagian <i>Logout</i>	21
3.1.4 Bagian Menambahkan Rute	22
3.1.5 Bagian Mengubah Rute	22

3.1.6	Bagian Melihat Daftar Rute	22
3.1.7	Bagian Melihat Informasi Rute secara Detail	22
3.1.8	Bagian Menghapus Data Geografis suatu Rute	23
3.1.9	Bagian Impor Data KML	23
3.1.10	Bagian Menghapus Rute	23
3.1.11	Bagian Melihat Daftar API <i>Keys</i>	24
3.1.12	Bagian Menambahkan API <i>Key</i>	24
3.1.13	Bagian Mengubah API <i>Key</i>	24
3.1.14	Bagian <i>Register</i>	25
3.1.15	Bagian Melihat Data Pribadi Pengguna	25
3.1.16	Bagian Mengubah Data Pribadi Pengguna	25
3.2	Analisis <i>Database</i> Sistem Kini	26
3.3	Analisis Sistem Usulan	26
3.3.1	<i>Routes</i>	26
3.3.2	<i>Folder</i> “public/”	27
3.3.3	<i>Controllers</i>	28
3.3.4	<i>Models</i>	28
3.4	Analisis <i>Libraries</i> Sistem Usulan	29
3.4.1	Jackson Databind	30
3.4.2	JavaMail API	31
3.4.3	MySQL Connector/J	33
3.4.4	jBCrypt	34
4	PERANCANGAN	37
4.1	Perancangan Kelas	37
4.2	<i>Controllers</i>	37
4.3	<i>Models</i>	42
4.3.1	ApiKeysManager	42
4.3.2	AuthenticationManager	43
4.3.3	Constant	45
4.3.4	TracksManager	47
4.3.5	UniqueStatusError	49
4.3.6	User	49
4.3.7	Utils	50
4.4	<i>Views</i>	51
5	IMPLEMENTASI DAN PENGUJIAN	53
5.1	Implementasi	53
5.1.1	Lingkungan Implementasi	53
5.1.2	Hasil Implementasi	53
5.2	Hasil Pengujian	53
5.2.1	Pengujian Fungsional	53
5.2.2	Pengujian Eksperimental	53
	DAFTAR REFERENSI	55
	A THE SOURCE CODE	57

DAFTAR GAMBAR

1.1	Situs web KIRI[1]	1
1.2	KIRI <i>Dashboard</i> [2]	2
2.1	Universitas Katolik Parahyangan dinyatakan dalam <i>Point</i> [3]	6
2.2	Rute jalan dari Universitas Katolik Parahyangan menuju Galeri Ciumbuleuit dinyatakan dalam <i>LineString</i> [3]	7
2.3	Struktur minimal Play Framework	11
2.4	Struktur kode file “routes”[4]	12
2.5	Hubungan <i>routes</i> dan <i>controllers</i> dalam memproses HTTP <i>requests</i> [4]	13
3.1	Struktur Kode KIRI	18
3.2	Struktur <i>folder</i> “public_html_dev”	18
3.3	Halaman dokumentasi KIRI	19
3.4	Halaman developer KIRI	20
3.5	Struktur <i>folder</i> “bukitjarian”	20
3.6	Struktur folder “sql”	26
3.7	Struktur database sistem KIRI	27
3.8	Diagram Kelas Analisis	28
3.9	Diagram <i>use case</i> KIRI <i>Dashboard</i>	29
4.1	Kelas Diagram KIRI <i>Dashboard Server Side</i>	38

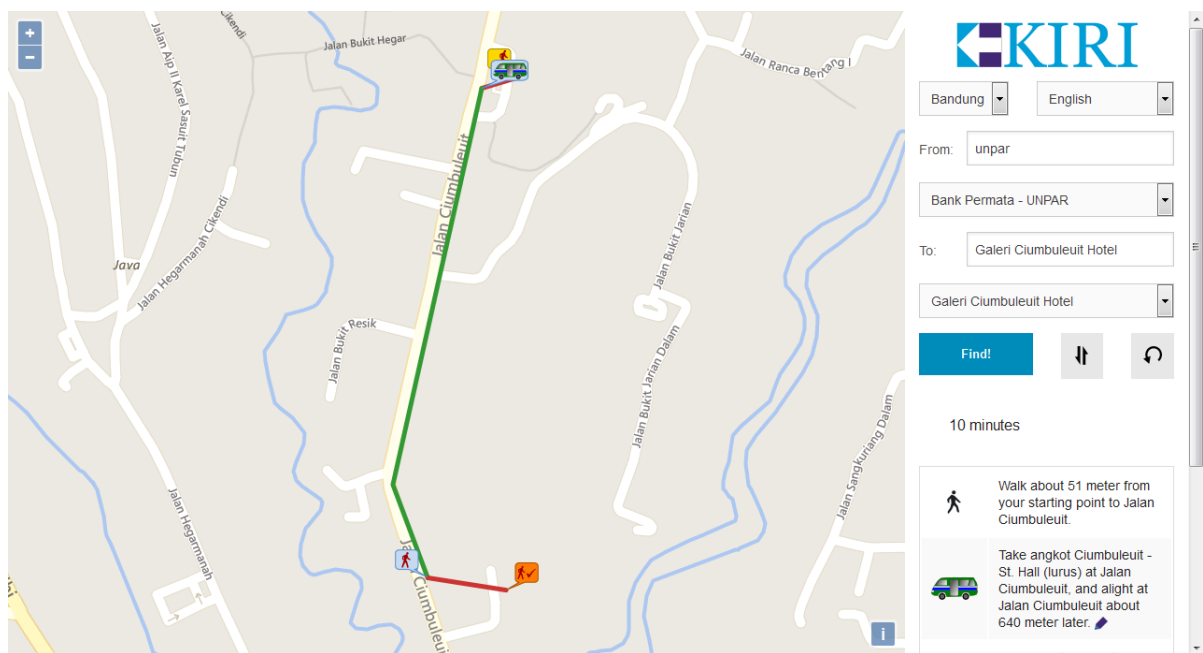
DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

KIRI[1] (Gambar 1.1) merupakan situs web untuk membantu pengguna menemukan rute transportasi umum ke tempat tujuannya. Dengan memasukkan lokasi awal serta lokasi tujuan pengguna tersebut, situs web KIRI akan memberikan langkah-langkah (contoh: berjalan sejauh berapa meter, menggunakan angkot, dan sebagainya) tercepat untuk sampai ke lokasi tujuan.

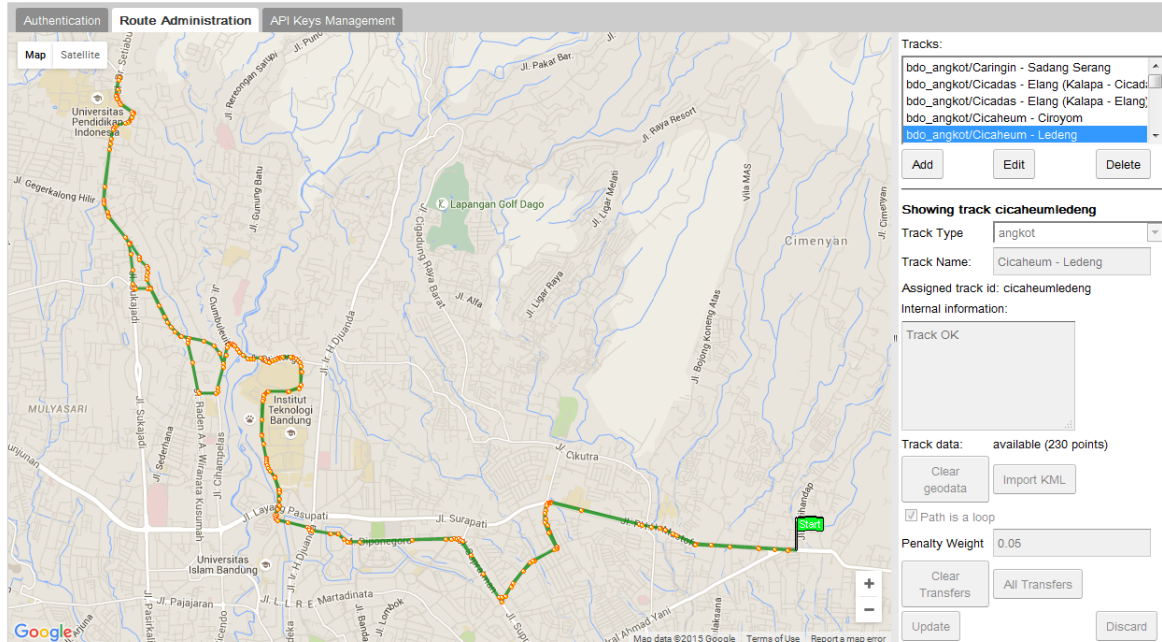


Gambar 1.1: Situs web KIRI[1]

KIRI *Dashboard*[2] (Gambar 1.2) adalah bagian dari situs web KIRI. KIRI *Dashboard* berfungsi sebagai pengatur proses CRUD (*Create*, *Read*, *Update*, dan *Delete*) daftar rute yang terdapat dalam *database* situs web KIRI. KIRI *Dashboard Server Side* menggunakan bahasa PHP dalam pembuatannya[5]. Bahasa PHP kurang cocok untuk proyek skala besar seperti *dashboard*. Salah satu penyebab bahasa PHP kurang cocok adalah karena tidak ada deklarasi dan tipe variabel dalam penggunaan bahasa PHP.

Java merupakan bahasa pemrograman yang umum digunakan oleh banyak orang. Selain umum digunakan, Java juga merupakan bahasa pemrograman yang lebih terstruktur dibandingkan dengan PHP. Adanya deklarasi dan tipe variabel pada Java membuat setiap variabel memiliki kegunaan

yang lebih jelas dan mudah dimengerti. Play Framework merupakan salah satu *framework* yang membantu implementasi Java dalam pembuatan suatu situs web. Play Framework juga cocok untuk proyek skala besar karena arsitekturnya sudah menggunakan konsep MVC (*Model View Controller*)[4].



Gambar 1.2: KIRI Dashboard[2]

Berdasarkan ditemukannya kekurangan-kekurangan pada KIRI Dashboard Server Side seperti yang telah dijelaskan, maka solusi untuk mengatasi kekurangan tersebut adalah dibuatlah penelitian ini untuk mengubah KIRI Dashboard Server Side yang semula dalam bahasa PHP menjadi bahasa Java dengan menggunakan Play Framework.

1.2 Rumusan Masalah

Berikut adalah susunan permasalahan yang akan dibahas pada penelitian ini:

1. Bagaimana isi kode KIRI Dashboard Server Side dan apa saja kekurangan yang ada di dalamnya?
2. Bagaimana cara kerja Play Framework berbasis MVC?
3. Bagaimana melakukan *porting* KIRI Dashboard Server Side yang semula dalam bahasa PHP menjadi bahasa Java dengan menggunakan Play Framework?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dibuat, maka tujuan penelitian ini dijelaskan ke dalam poin-poin sebagai berikut:

1. Mengetahui isi kode KIRI *Dashboard Server Side* dan kekurangan-kekurangan yang ada di dalamnya.
2. Mengetahui cara kerja Play Framework berbasis MVC.
3. Melakukan *porting* KIRI *Dashboard Server Side* yang semula dalam bahasa PHP menjadi bahasa Java dengan menggunakan Play Framework.

1.4 Batasan Masalah

Penelitian ini dibuat berdasarkan batasan-batasan sebagai berikut:

1. Play Framework yang digunakan selama penelitian ini adalah versi 2.4.3.
2. *Porting* Kode KIRI *Dashboard Server Side* yang dilakukan adalah berdasarkan versi terbaru dari Github dengan *username*: “pascalalfadian”[5].

1.5 Metode Penelitian

Berikut adalah metode penelitian yang digunakan dalam penelitian ini:

1. Melakukan studi literatur mengenai kode KIRI *Dashboard Server Side*, MySQL Spatial Extensions, dan Play Framework.
2. Menganalisis teori-teori untuk membangun KIRI *Dashboard Server Side* dalam bahasa Java dengan menggunakan Play Framework.
3. Merancang KIRI *Dashboard Server Side* dalam bahasa Java dengan menggunakan Play Framework.
4. Melakukan *porting* kode situs web KIRI *Dashboard Server Side* menjadi Java dengan menggunakan Play Framework.
5. Melakukan pengujian terhadap fitur-fitur yang sudah dibuat.

1.6 Sistematika Penulisan

Setiap bab dalam penelitian ini memiliki sistematika penulisan yang dijelaskan ke dalam poin-poin sebagai berikut:

1. Bab 1: Pendahuluan, yaitu membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.
2. Bab 2: Dasar Teori, yaitu membahas mengenai teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang MySQL Spatial Extensions dan Play Framework.
3. Bab 3: Analisis, yaitu membahas mengenai analisa masalah. Berisi tentang analisis kode KIRI *Dashboard Server Side* dan analisis kekurangan-kekurangan kode KIRI *Dashboard Server Side*.

4. Bab 4: Perancangan, yaitu membahas mengenai perancangan yang dilakukan sebelum melakukan tahapan implementasi. Berisi tentang perancangan fitur CRUD KIRI *Dashboard Server Side* menggunakan Play Framework, perancangan basis data, dan perancangan antarmuka KIRI *Dashboard* menggunakan Play Framework.
5. Bab 5: Implementasi dan Pengujian, yaitu membahas mengenai implementasi dan pengujian aplikasi yang telah dilakukan. Berisi tentang implementasi dan hasil pengujian aplikasi.
6. Bab 6: Kesimpulan dan Saran, yaitu membahas hasil kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat diberikan untuk penelitian berikutnya. Berisi tentang kesimpulan dan saran.

BAB 2

DASAR TEORI

2.1 MySQL Spatial Extensions

Suatu *geographic feature* [6] 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:

1. ***An entity***, contohnya adalah gunung, kolam, kota, dll.
2. ***A space***, contohnya adalah daerah, cuaca, dll.
3. ***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 *Var-char* untuk menyimpan karakter atau kata, *Int* untuk menyimpan angka, *Boolean* untuk menyimpan nilai “**true**” atau “**false**”, dan tipe data lainnya. MySQL Spatial Extensions adalah perluasan dari tipe-tipe data yang disediakan MySQL untuk menyatakan nilai geometri dari suatu *geographic feature*.

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

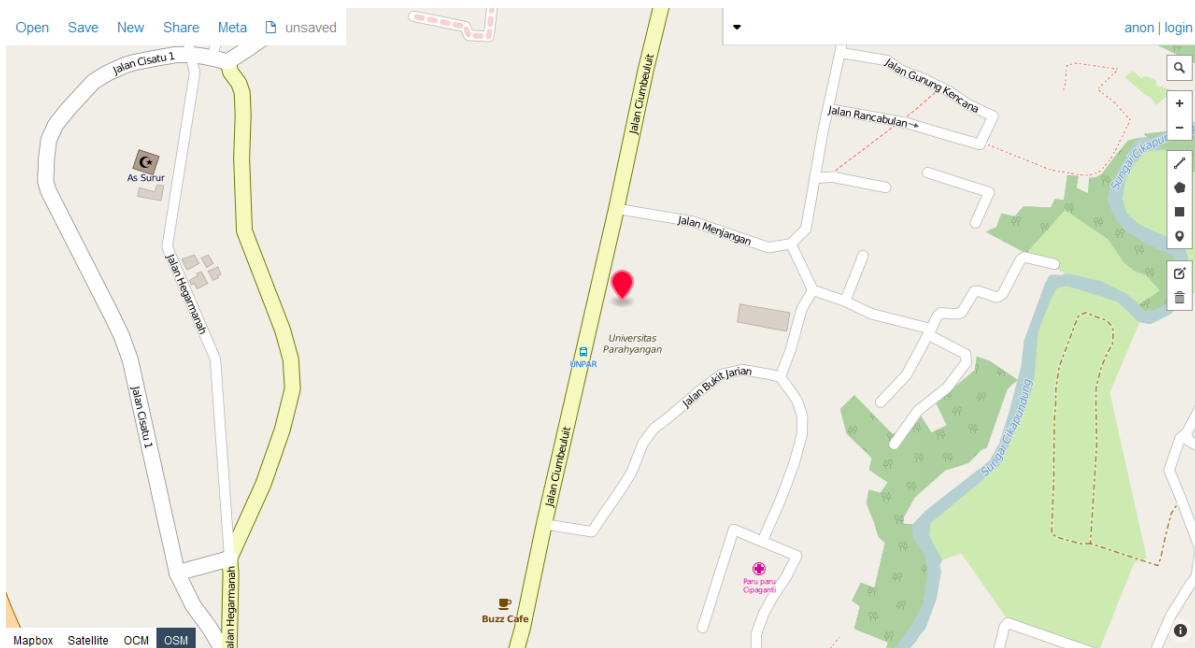
1. Tipe data yang hanya dapat menyimpan sebuah nilai geometri saja, yaitu:
 - ***Geometry***
 - ***Point***
 - ***LineString***
 - ***Polygon***

2. Tipe data yang dapat menyimpan sekumpulan nilai geometri, yaitu:

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

2.1.1 *Point*

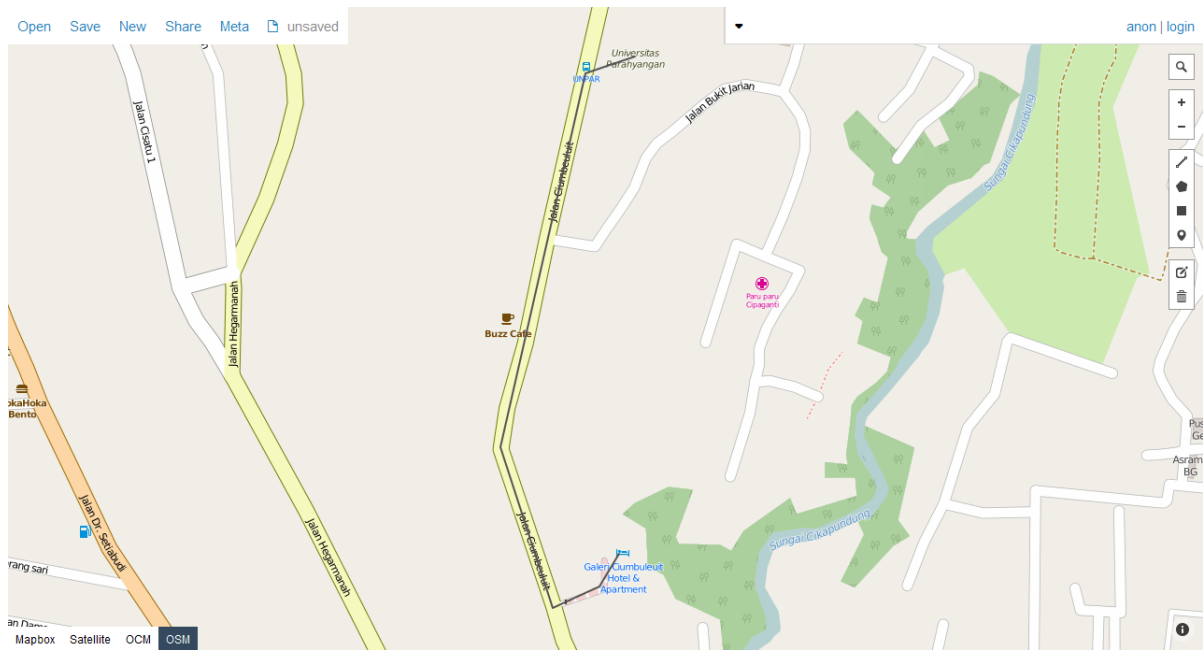
Point adalah nilai geometri yang merepresentasikan sebuah lokasi ke dalam suatu koordinat[6]. Koordinat pada *Point* terdiri dari nilai X dan Y dimana X merepresentasikan letak lokasi terhadap garis bujur dan Y merepresentasikan letak lokasi terhadap garis lintang. *Point* tidak memiliki dimensi maupun nilai batasan. Contoh representasi *Point* adalah Universitas Katolik Parahyangan direpresentasikan dalam koordinat X=107.6049079 dan Y=-6.874735 (Gambar 2.1).



Gambar 2.1: Universitas Katolik Parahyangan dinyatakan dalam *Point*[3]

2.1.2 *LineString*

LineString adalah garis yang terbentuk dari sekumpulan *Point*[6]. Dalam peta dunia, *LineString* dapat merepresentasikan sebuah sungai dan dalam peta perkotaan, *LineString* dapat merepresentasikan sebuah jalan (contoh: Gambar 2.2). Karena *LineString* merupakan sekumpulan *Point*, maka *LineString* menyimpan sekumpulan koordinat dimana setiap koordinat ($X_1 \dots X_n$ dan $Y_1 \dots Y_n$, dimana n menyatakan banyaknya *Point* dalam *LineString*) terhubung oleh garis dengan koordinat selanjutnya. Contohnya: misal terdapat sebuah *LineString* yang mengandung 3 buah *Point*, maka terdapat garis yang menghubungkan *Point* pertama dengan *Point* kedua dan *Point* kedua dengan *Point* ketiga.



Gambar 2.2: Rute jalan dari Universitas Katolik Parahyangan menuju Galeri Ciumbuleuit dinyatakan dalam *LineString*[3]

2.1.3 Format Well-Known Text (WKT)

Format Well-Known Text (WKT) adalah salah satu aturan penulisan tipe data *spatial* untuk merepresentasikan suatu *geographic feature*[6]. WKT merepresentasikan nilai geometri yang dimodelkan untuk pertukaran data geometri dalam *ASCII form*. Berikut adalah contoh format WKT:

```

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

```

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

Berikut adalah contoh penggunaan format WKT dalam MySQL:

```

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

```

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

2.2 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*[7]. JDBC dapat membantu 3 jenis aktivitas *programming* dalam menggunakan bahasa Java, yaitu:

1. Menghubungkan aplikasi Java ke suatu sumber data seperti *database*,
2. Mengirimkan *queries* dan pembaharuan *statement* ke *database*,
3. 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).

2.2.1 Interface Connection

Interface Connection adalah sebuah koneksi (*session*) dengan *database* spesifik[7]. Eksekusi SQL *statements* (subbab 2.2.3) dan penerimaan hasil kembalian (subbab 2.2.4) 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.

2.2.2 Kelas DriverManager

Kelas `DriverManager` adalah cara paling dasar untuk mengatur JDBC *drivers*[7]. 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* (subbab 2.2.1).

Parameter:

1. `url`, alamat dari *database*, formatnya adalah “*jdbc:subprotocol:subname*”,
2. `user`, *username* untuk mengakses *database*,
3. `password`, *password* dari *username*.

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

2.2.3 Interface Statement

Interface *Statement* adalah objek yang digunakan untuk melakukan eksekusi terhadap suatu *query* dan mengembalikan nilai kembalian dari eksekusi tersebut[7]. 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`.

2.2.4 Interface ResultSet

Interface *ResultSet* adalah sebuah tabel data yang merepresentasikan hasil dari sebuah eksekusi *query* pada suatu *database*[7] (subbab 2.2.3). 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.

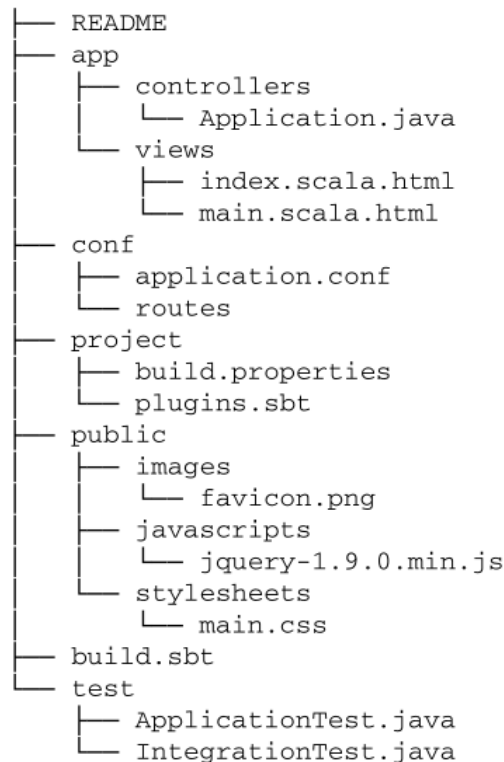
2.3 Play Framework

Play Framework adalah sekumpulan kerangka kode yang dapat digunakan untuk membangun suatu situs web[4]. 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”.

2.3.1 Struktur Aplikasi

Ketika Play Framework pertama kali ter-*install* pada komputer, Play Framework menyediakan *default* direktori dengan struktur minimal (Gambar 2.3). Berikut adalah penjelasan struktur minimal Play Framework:

1. *Folder* “app” merupakan *folder* yang berisi pola arsitektur yang dimiliki Play Framework, yaitu “models” (tidak dibuat secara *default*), “views”, dan “controllers” yang akan dijelaskan lebih lanjut pada subbab selanjutnya (subbab *Models*: 2.3.3, subbab *Views*: 2.3.4, dan subbab *Controllers*: 2.3.5).
2. *Folder* “conf” berisi *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 yang akan dijelaskan pada subbab selanjutnya (subbab 2.3.2).
3. *Folder* “project” terdapat *file* “build.properties” dan “plugins.sbt”, *file* tersebut mendeskripsikan versi Play dan SBT yang digunakan pada aplikasi.
4. *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”).
5. *File* “build.sbt” mengatur *dependencies* yang dibutuhkan dalam pembuatan aplikasi.
6. Terakhir adalah *folder* “test” yang merupakan salah satu kelebihan dari Play Framework, bagian ini berisikan *file* “Application.test” dan “Integration.test” yang dapat digunakan untuk melakukan serangkaian *testing* yang diinginkan terhadap aplikasi.



Gambar 2.3: Struktur minimal Play Framework

2.3.2 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 2.4), yaitu HTTP *method*, URL *path*, dan *action method*. Struktur *routes* seperti yang dijelaskan pada gambar 2.4 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”[8]. 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 2.4), dan terakhir adalah pemilihan *method* yang ada pada kelas “controllers” yang dipilih (contohnya: “list()”).

Gambar 2.4: Struktur kode *file* “routes”^[4]

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”.

2.3.3 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 2.3). Untuk itu perlu menambahkan *models* secara manual ke dalam struktur Play Framework. Langkah yang dilakukan untuk menambahkan *models* ke dalam Play Framework adalah:

1. Menambahkan folder “models” ke dalam folder “app”,
2. 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.

2.3.4 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 2.3) harus dengan format sebagai berikut, “namaFile.scala.html”.

Berikut adalah contoh struktur kode *views*:


```

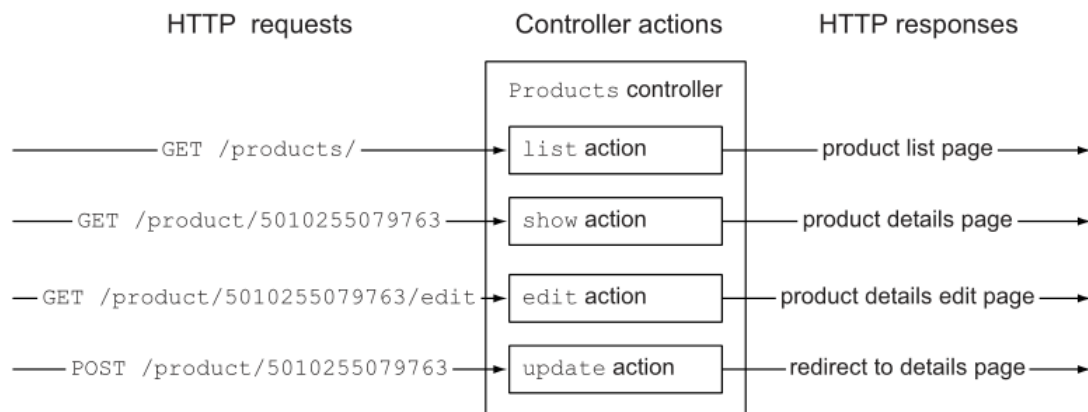
1 | @(name: String)
2 | <!doctype html>
3 | <html>
4 |   <head>
5 |     <meta charset="UTF-8">
6 |     <title>Hello</title>
7 |   </head>
8 |   <body>
9 |     <h1>Hello <em>@name</em></h1>
10 |   </body>
11 | </html>

```

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 “(namaVariabel₁: tipeVariabel₁) (namaVariabel₂: tipeVariabel₂) ... (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).

2.3.5 Controllers

Controllers merupakan bagian pada Play Framework yang terhubung langsung dengan *routes* (sub-bab 2.3.2). 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 2.5). Fungsi dari *controllers* dalam arsitektur MVC adalah sebagai penghubung antara *models* dan *views*.



Gambar 2.5: Hubungan *routes* dan *controllers* dalam memproses HTTP requests[4]

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:

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

2.3.6 Database

Play Framework menyediakan sebuah *plugin* yang dapat digunakan untuk mengatur koneksi JDBC ke berbagai jenis aplikasi *database* yang tersedia[8]. 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:

1. Menambahkan kode program ke dalam “`build.sbt`” (Gambar 2.3), 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.

2. Menambahkan kode program ke dalam “`conf/application.conf`” (Gambar 2.3), 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* (subbab 2.2). 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 (contoh kode pada subbab 2.2 baris 3-6).

2.4 JSON

JSON (*JavaScript Object Notation*) adalah sebuah format pertukaran data ringan[9]. JSON dapat dibangun dalam 2 buah struktur:

1. Sekumpulan pasangan antara nama dengan nilai. Umumnya dikenal dengan sebutan objek. Sebuah objek dalam JSON dimulai dengan karakter “{” dan diakhiri dengan karakter “}”. Di antara karakter “{” dan “}” dapat disisipkan sekumpulan pasangan “**nama:nilai**” yang dipisahkan dengan karakter “,”.
2. Sekumpulan data terstruktur. Umumnya dikenal dengan sebutan *array*. Sebuah *array* dalam JSON dimulai dengan karakter “[” dan diakhiri dengan karakter “]”. Di antara karakter “[” dan “]” dapat disisipkan sekumpulan data (dapat berupa nilai, objek atau *array*) yang dipisahkan dengan karakter “,”. 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).

2.5 *Regular Expression*

Sebuah *regular expression* (*regex*) adalah karakter atau kata istimewa yang dapat digunakan untuk melakukan pola pencarian tertentu.

BAB 3

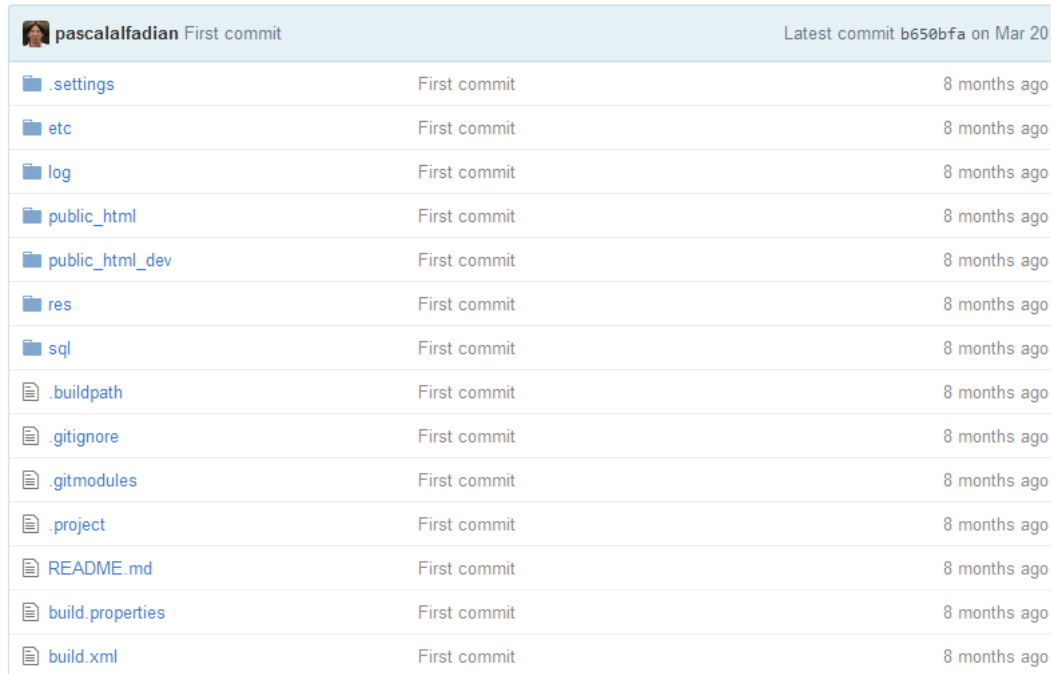
ANALISIS

3.1 Analisis Sistem Kini

Berdasarkan hasil analisa serta wawancara dengan kontributor kode KIRI, berikut adalah penjelasan secara umum mengenai isi kode KIRI (Gambar 3.1):

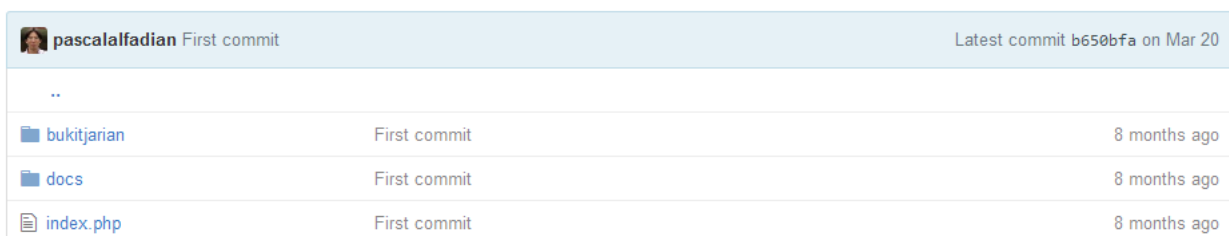
1. *Folder “settings”* merupakan *folder* yang menyimpan *file-file* pengaturan sistem KIRI untuk Eclipse. Eclipse adalah sebuah *open source* IDE (*Integrated Development Environment*) yang membantu programmer dalam membangun suatu perangkat lunak[10].
2. *Folder “etc”* merupakan *folder* yang menyimpan *file-file* untuk membangun konstanta-konstanta dan fungsi-fungsi yang digunakan oleh sistem KIRI.
3. *Folder “log”* merupakan *folder* yang berisi *file-file* untuk mencatat setiap kinerja sistem KIRI.
4. *Folder “public_html”* merupakan *folder* yang berisi *file-file* untuk membangun KIRI *Front End*.
5. *Folder “public_html_dev”* merupakan *folder* yang berisi *file-file* untuk membangun KIRI *Dashboard*.
6. *Folder “res”* merupakan *folder* yang berisi *file-file* yang bersifat publik seperti gambar, data XML, dll yang digunakan sebagai pendukung sistem KIRI.
7. *Folder “sql”* merupakan *folder* yang menyimpan *file-file* untuk membangun *database* sistem KIRI.
8. *File “.buildpath”* dan *“.project”* merupakan *file* yang menyimpan konfigurasi untuk Eclipse.
9. *File “.gitignore”* merupakan *file* yang menyimpan daftar *file* yang tidak perlu dikirimkan ke tempat penyimpanan GitHub karena *file-file* tersebut dibangkitkan oleh sistem. GitHub adalah sebuah tempat penyimpanan *online* yang dikhususkan untuk menyimpan isi kode suatu perangkat lunak[11].
10. *File “.gitmodules”* merupakan *file* yang menyimpan alamat kode eksternal yang digunakan dalam sistem KIRI.
11. *File “README.md”* merupakan *file* yang berisi keterangan singkat mengenai proyek KIRI. Isi keterangan singkat tersebut digunakan untuk mencegah sembarang orang agar tidak membuka proyek KIRI di GitHub.

12. *File* “build.properties” dan “build.xml” merupakan *file* yang digunakan Ant untuk mendistribusikan program. Ant adalah sebuah perangkat yang dapat digunakan untuk menjalankan tes dan menjalankan aplikasi dalam bahasa Java[12].



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

Gambar 3.1: Struktur Kode KIRI



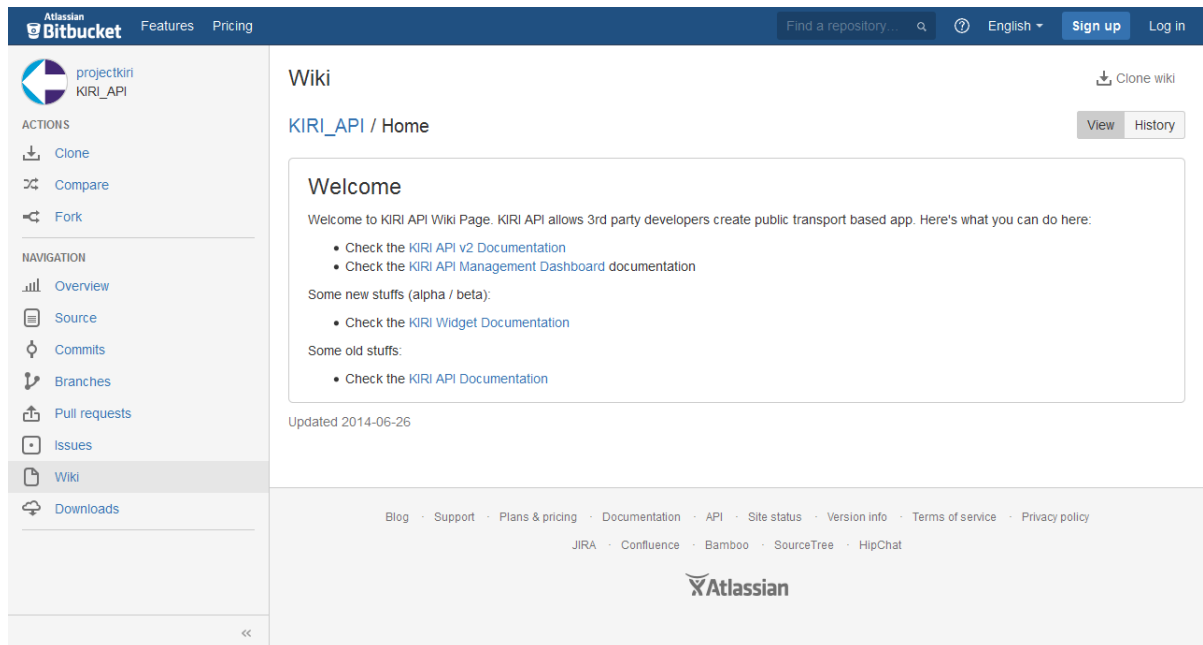
pascalalfadian First commit		Latest commit b650bfa on Mar 20
..		
📁 bukitjarian	First commit	8 months ago
📁 docs	First commit	8 months ago
📄 index.php	First commit	8 months ago

Gambar 3.2: Struktur *folder* “public_html_dev”

Penelitian ini berfokus pada pembahasan KIRI *Dashboard*, untuk itu bagian yang akan dianalisa lebih mendalam adalah pada bagian “public_html_dev” (Gambar 3.2) dan beberapa bagian-bagian pendukung untuk membangun KIRI *Dashboard*. *Folder* “public_html_dev” memiliki struktur sebagai berikut:

1. *Folder* “bukitjarian”, berisi mengenai *file-file* untuk membangun KIRI *Dashboard* (Gambar 3.5).
2. *Folder* “docs”, berisi sebuah *file* “index.php” yang bila dijalankan akan mengarahkan pengguna ke alamat https://bitbucket.org/projectkiri/kiri_api/wiki/Home (Gambar 3.3). Halaman tersebut berisi mengenai dokumentasi sistem KIRI.
3. *File* “index.php”, bila *file* ini dieksekusi maka akan mengarahkan pengguna ke alamat <http://static.kiri.travel/developer> (Gambar 3.4). Halaman tersebut berisi informasi mengenai

alamat KIRI *Dashboard*, dokumentasi KIRI, dan halaman untuk memberikan masukan kepada kontributor sistem KIRI.

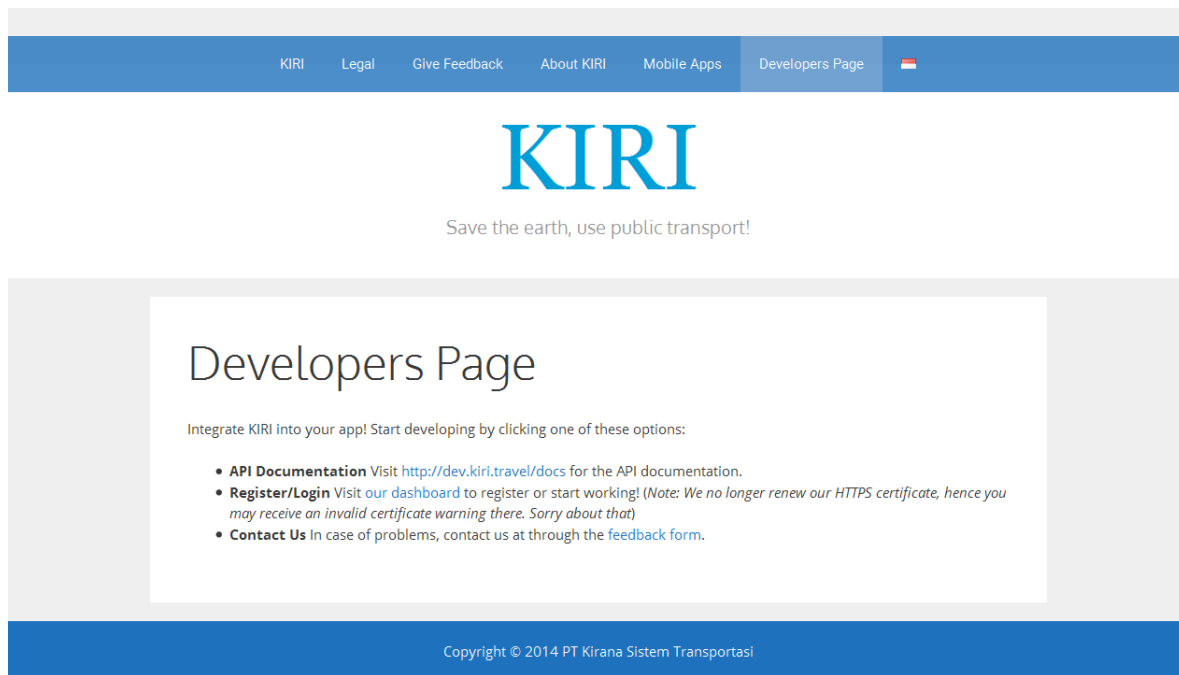


Gambar 3.3: Halaman dokumentasi KIRI

Pada direktori “public_html_dev/bukitjarian” (Gambar 3.5) terdapat bermacam-macam *file* dan *folder*. Berdasarkan wawancara dengan kontributor kode, terdapat dua bagian *file* dan *folder* yang berperan penting dalam membangun KIRI *Dashboard*:

1. “index.html”, “bukitjariangwt/”, dan “images/”. *File* dan *folder* ini dibangun dengan menggunakan perangkat lunak GWT. GWT (*Google Web Toolkit*) adalah perangkat lunak yang digunakan untuk membangun dan melakukan optimasi suatu aplikasi berbasis web[13]. Dengan GWT, pengguna tidak perlu ahli dalam menggunakan bahasa pemrograman berbasis web seperti JavaScript. GWT menggunakan bahasa Java dalam pembuatannya dan akan melakukan konversi kode Java tersebut menjadi HTML dan JavaScript. Hasil konversi menggunakan GWT akan mengalami *obfuscate* (pengacakan) sehingga sulit untuk dianalisa. Namun demikian, *file-file* ini bersifat statis dan tidak memerlukan operasi khusus di *server*, sehingga dapat disalin (*copy*) apa adanya.
2. “handle.php” merupakan kode pada sisi *server* yang bertugas untuk melayani permintaan-permintaan dari *browser* yang dieksekusi oleh “index.html”.

Seperti disebutkan sebelumnya, *file* “handle.php” berfungsi untuk melayani berbagai jenis permintaan yang dikirimkan oleh “index.html”. *File* ini dapat dibagi menjadi 16 bagian yang masing-masing melayani sebuah permintaan tertentu. Analisa lebih mendalam mengenai 16 bagian isi kode “handle.php” (kode A.1) akan dijelaskan pada 16 subbab yang akan dijelaskan pada pembahasan analisa selanjutnya.



Gambar 3.4: Halaman developer KIRI

pascalalfadian First commit		Latest commit b650bfa on Mar 20
..		
bukitjariangwt	First commit	8 months ago
images	First commit	8 months ago
BukitJarianGWT.css	First commit	8 months ago
getplaces.php	First commit	8 months ago
gettracks.php	First commit	8 months ago
handle.php	First commit	8 months ago
index.html	First commit	8 months ago

Gambar 3.5: Struktur *folder* “bukitjarian”

3.1.1 Bagian Pemeriksaan *Login*

Bagian ini terletak di baris 12-32 dari “handle.php” (kode A.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).

3.1.2 Bagian *Login*

Bagian ini terletak di baris 34-89 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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*.

3.1.3 Bagian *Logout*

Bagian ini terletak di baris 89-97 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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*.

3.1.4 Bagian Menambahkan Rute

Bagian ini terletak di baris 97-117 dari “handle.php” (kode [A.1](#)). Bagian ini akan dieksekusi 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.

3.1.5 Bagian Mengubah Rute

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

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

3.1.6 Bagian Melihat Daftar Rute

Bagian ini terletak di baris 146-172 dari “handle.php” (kode [A.1](#)). Bagian ini akan dieksekusi 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).

3.1.7 Bagian Melihat Informasi Rute secara Detail

Bagian ini terletak di baris 172-200 dari “handle.php” (kode [A.1](#)). Bagian ini akan dieksekusi 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).

3.1.8 Bagian Menghapus Data Geografis suatu Rute

Bagian ini terletak di baris 200-209 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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).

3.1.9 Bagian Impor Data KML

Bagian ini terletak di baris 209-246 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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[14].

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 2.5). Baris 231-239 program akan membangun data LineString yang semula dalam format KML menjadi format WKT (subbab 2.1.3). 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).

3.1.10 Bagian Menghapus Rute

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

3.1.11 Bagian Melihat Daftar API Keys

Bagian ini terletak di baris 261-279 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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).

3.1.12 Bagian Menambahkan API Key

Bagian ini terletak di baris 279-299 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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).

3.1.13 Bagian Mengubah API Key

Bagian ini terletak di baris 299-317 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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.

3.1.14 Bagian *Register*

Bagian ini terletak di baris 317-341 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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). *Hashing* akan dilakukan terhadap sandi acak tersebut dengan menggunakan PHP *hashing framework*[15] (baris 332). *Hashing* adalah teknik untuk memetakan data dengan ukuran yang berbeda-beda menjadi data dengan ukuran yang tetap[16]. 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).

3.1.15 Bagian Melihat Data Pribadi Pengguna

Bagian ini terletak di baris 341-362 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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).

3.1.16 Bagian Mengubah Data Pribadi Pengguna




Bagian ini terletak di baris 362-380 dari “handle.php” (kode A.1). Bagian ini akan dieksekusi 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).

3.2 Analisis *Database* Sistem Kini

Sistem KIRI menggunakan perangkat lunak MySQL sebagai sarana penyimpanan untuk mengolah data. Seperti yang dijelaskan pada subbab sebelumnya (subbab 3.1) terdapat sebuah *folder* yang menyimpan *file-file* untuk membangun *database* sistem KIRI, yaitu: *folder* “sql” (Gambar 3.1). Berdasarkan hasil analisa dan wawancara dengan kontributor kode KIRI, berikut adalah penjelasan mengenai isi *folder* “sql” (Gambar 3.6):

1. *file* “.gitignore”, merupakan *file* yang menyimpan daftar *file* yang tidak perlu dikirimkan ke tempat penyimpanan GitHub karena *file-file* tersebut dibangkitkan oleh sistem.
2. *file* “tirtayasa_structure.sql”, merupakan *file* untuk membangun tabel-tabel serta kolom-kolom (pada setiap tabel) *database* sistem KIRI.
3. *file* “tirtayasa.sql”, merupakan *file* untuk membangun isi dari kolom-kolom setiap tabel, yaitu *record-record* awal.

pascalalfadian First commit		Latest commit b650bfa on Mar 20
..		
 .gitignore	First commit	8 months ago
 tirtayasa.sql	First commit	8 months ago
 tirtayasa_structure.sql	First commit	8 months ago

Gambar 3.6: Struktur folder “sql”

3.3 Analisis Sistem Usulan

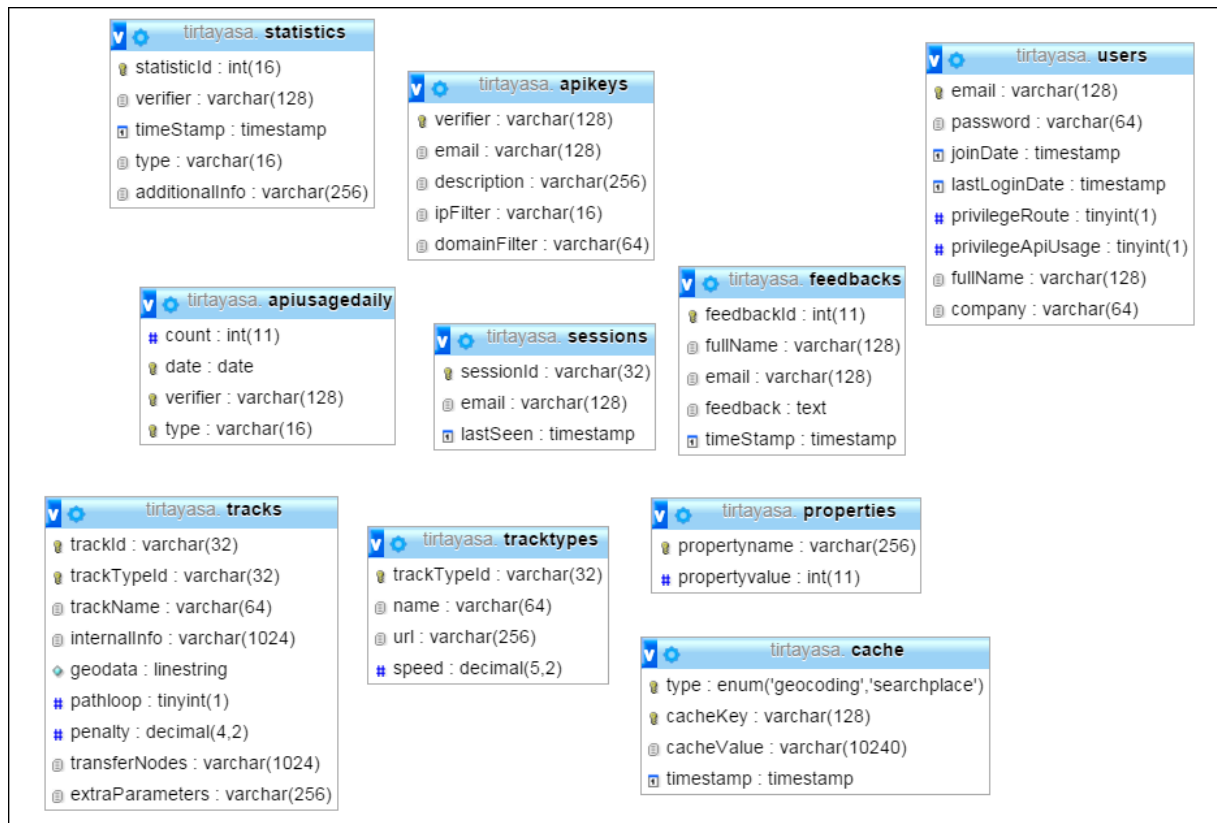
Berdasarkan hasil analisa dan wawancara dengan kontributor kode pada subbab sebelumnya (subbab 3.1 dan subbab 3.2) didapatkan poin-poin bahwa:

1. KIRI *Dashboard* terdiri dari dua bagian penting, yaitu bagian pertama adalah *file* dan *folder* yang bersifat statis, yaitu: “index.html”, “bukitjariangwt/”, dan “images/”, serta bagian kedua adalah “handle.php” yang bertugas menangani permintaan-permintaan (dari “index.html”) pada sisi *server*.
2. KIRI *Dashboard server side* (*file* “handle.php”) terbagi menjadi 16 bagian dalam menangani permintaan.

Berdasarkan kebutuhan akan poin-poin di atas, maka dalam memodelkan KIRI *Dashboard* dengan menggunakan Play Framework perlu memperhatikan beberapa hal, yaitu: *models*, *controllers*, *routes*, dan *folder* “public/”.

3.3.1 *Routes*

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



Gambar 3.7: Struktur database sistem KIRI

1. Pemetaan URL untuk menangani tampilan KIRI *Dashboard*. Hanya 1 *route* yang dibutuhkan untuk pemetaan terhadap bagian tampilan (walau fitur KIRI *Dashboard* banyak) karena kode KIRI *Dashboard* sistem kini telah menggunakan kombinasi JavaScript AJAX sedemikian rupa yang dapat mengubah tampilan KIRI *Dashboard* sesuai dengan balasan permintaan dari sisi *server*. AJAX (*asynchronous JavaScript and XML*) adalah teknik yang memungkinkan suatu situs web melakukan permintaan ke sisi *server* secara *background*[17].
2. Pemetaan URL untuk menangani permintaan-permintaan dari tampilan ke sisi *server*, yaitu KIRI *Dashboard server side (controllers)*.

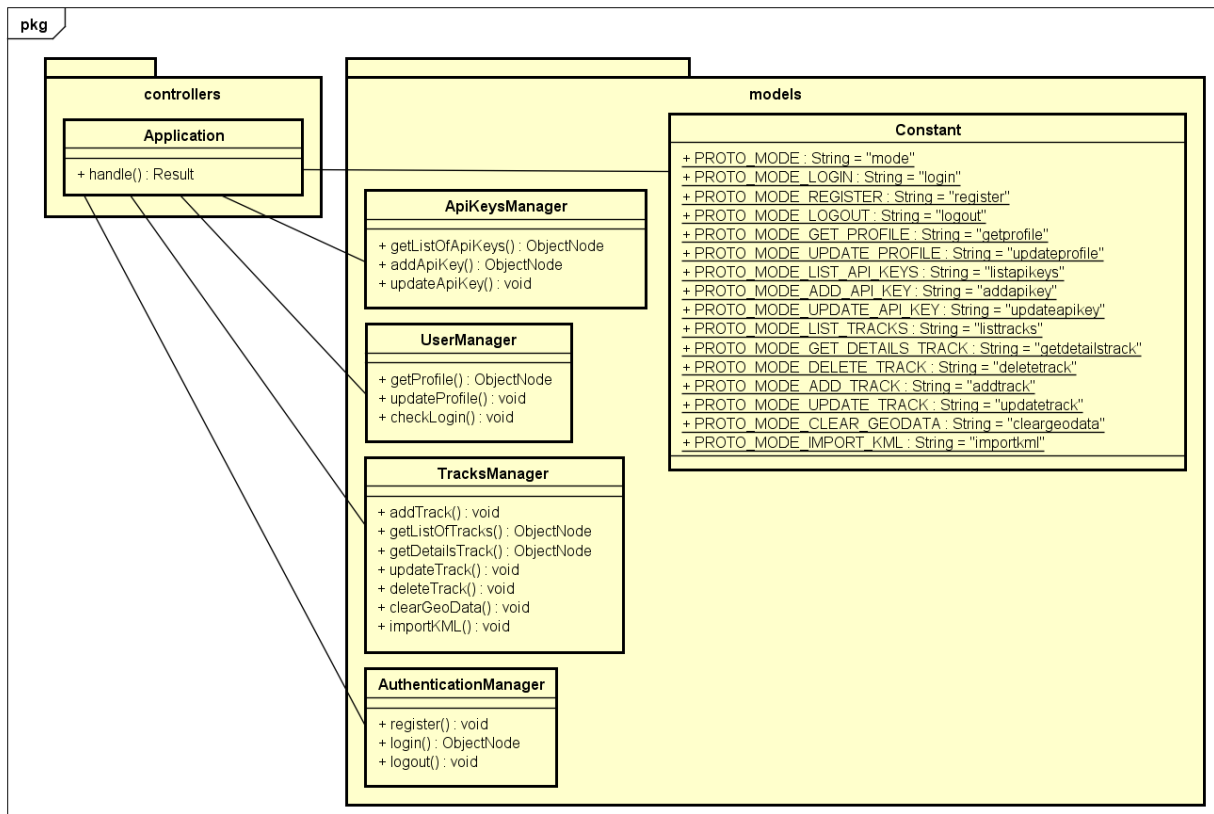
Isi lengkap dari *routes* (file “conf/routes”) akan dijelaskan pada bab perancangan (Bab 4).

3.3.2 Folder “public/”

Seperti yang telah dijelaskan pada analisa sebelumnya bahwa bagian tampilan KIRI *Dashboard* dibuat dengan menggunakan perangkat GWT dimana kode yang dihasilkan sangat sulit untuk dipelajari. Walau sulit dipelajari, bagian tampilan KIRI *Dashboard* memiliki keuntungan sendiri karena bersifat statis (dijelaskan oleh kontributor kode). Sifat statis ini memudahkan pemodelan karena dengan begitu kode dapat disalin apa adanya ke sistem usulan. Pada Play Framework, *file-file* yang bersifat statis seperti ini dapat disimpan pada bagian *folder* “public/”.

3.3.3 Controllers

Pada bagian ini akan dibuat sebuah kelas yang akan menangani permintaan-permintaan dari bagian tampilan sistem usulan (Gambar 3.8). Bagian ini berfungsi seperti “handle.php” pada sistem ini, yaitu menangani 16 jenis permintaan berbeda seperti yang telah dijelaskan pada subbab sebelumnya (Subbab 3.1).



Gambar 3.8: Diagram Kelas Analisis

3.3.4 Models

Models adalah bagian yang paling bebas dan tidak memiliki aturan pembuatan khusus pada Play Framework. Pada bagian ini peneliti memutuskan untuk membuat *models* sesuai dengan *use case* KIRI Dashboard. Untuk setiap kemampuan pengguna yang dapat dilakukan pada *use case* (Gambar 3.9) akan dibuat kelas Java di bagian *models* pada sistem usulan (Gambar 3.8). Berikut adalah penjelasan kelas-kelas yang dibuat pada *models*:

1. AuthenticationManager

Kelas ini berfungsi untuk menangani permintaan *login*, *register*, dan *logout*.

2. ApiKeysManager

Kelas ini berfungsi untuk menangani permintaan melihat daftar API *keys*, menambahkan data API *key*, dan mengubah data API *key*.

3. UserManager

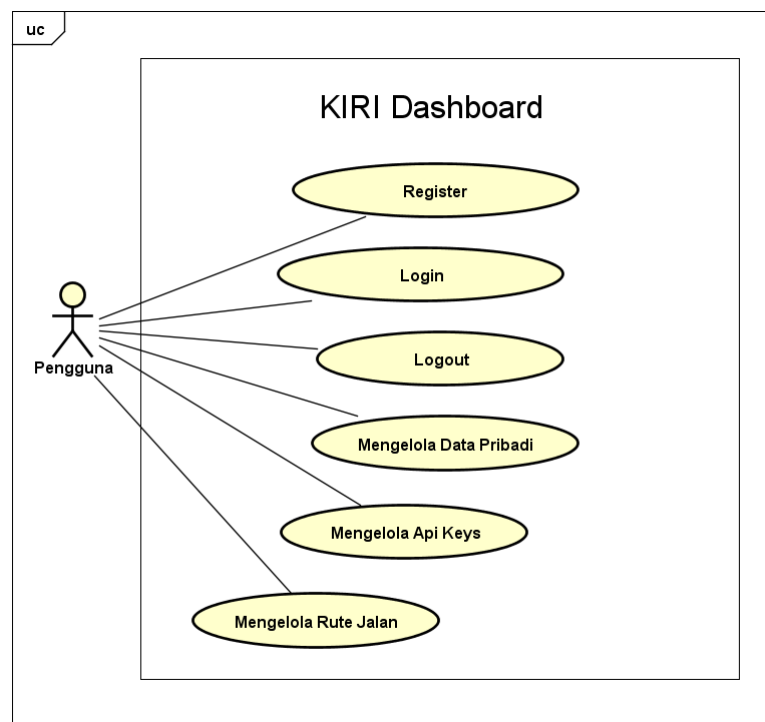
Kelas ini berfungsi untuk menangani permintaan melihat data pribadi pengguna, mengubah data pribadi pengguna, dan melakukan pemeriksaan *login*.

4. TracksManager

Kelas ini berfungsi untuk menangani permintaan melihat rute angkutan umum, menambahkan rute angkutan umum, melihat rute angkutan umum secara detail, mengubah data rute angkutan umum, menghapus rute angkutan umum, menghapus data geografis rute angkutan umum, dan melakukan impor data KML.

5. Constant

Kelas ini berfungsi untuk membangun konstanta-konstanta statis yang digunakan dalam sistem KIRI.



Gambar 3.9: Diagram *use case* KIRI Dashboard

3.4 Analisis *Libraries* Sistem Usulan

Berdasarkan hasil analisis 16 bagian sistem KIRI, dibutuhkan beberapa *library* untuk dapat melakukan *porting* beberapa bagian pada penelitian ini. Untuk mengatur penggunaan *library* pada aplikasi Play Framework dapat dilakukan pada file “build.sbt”[8] (Gambar 3.1). Daftar *library* yang dapat digunakan untuk Play Framework dapat dicari melalui Maven Repository[18]. Deklarasi untuk menambahkan sebuah *library* pada “build.sbt” terdiri dari 3 bagian, yaitu: kelompok, nama spesifik, dan versi. Berikut adalah contoh struktur kode untuk menambahkan sebuah *library* pada “build.sbt”:

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

Kode di atas menjelaskan bahwa akan ditambahkan *library* dari kelompok “mysql”, dengan nama “mysql-connector-java”, dan versi yang digunakan adalah “5.1.18”.

3.4.1 Jackson Databind

Dalam membalas permintaan dari bagian tampilan sistem ini, bagian sisi *server* akan mengirimkan pesan dalam format JSON. Jackson Databind adalah *library* Java (disarankan oleh Play Framework) yang digunakan untuk melakukan pengolahan data dalam format JSON[19]. Jackson Databind dapat mengolah sebuah data objek Java dari atau menjadi data yang berformat JSON. Berikut adalah kelas-kelas yang dapat digunakan untuk membangun data dalam format JSON:

1. ObjectNode

Kelas ini merupakan *node* yang memetakan sebuah atau sekumpulan objek pada JSON (Subbab 2.4). Berikut adalah contoh kode untuk membangun sebuah kelas ObjectNode pada Play Framework:

```
1|      ObjectNode obj = Json.newObject();
```

Berikut adalah beberapa *method* yang dimiliki oleh kelas ini:

- `public ObjectNode put(String fieldName, String v)`

Berfungsi untuk memberikan nilai berupa String ke sebuah nama.

Parameter:

- `fieldName` merepresentasikan nama.
- `v` merepresentasikan nilai berupa String.

Nilai kembalian: objek JSON dalam bentuk *node*.

- `public ObjectNode put(String fieldName, boolean v)`

Berfungsi untuk memberikan nilai berupa boolean ke sebuah nama.

Parameter:

- `fieldName` merepresentasikan nama.
- `v` merepresentasikan nilai berupa boolean.

Nilai kembalian: objek JSON dalam bentuk *node*.

- `public ObjectNode put(String fieldName, double v)`

Berfungsi untuk memberikan nilai berupa double ke sebuah nama.

Parameter:

- `fieldName` merepresentasikan nama.
- `v` merepresentasikan nilai berupa double.

Nilai kembalian: objek JSON dalam bentuk *node*.

- `public ObjectNode putNull(String fieldName)`

Berfungsi untuk memberikan nilai null ke sebuah nama.

Parameter:

- `fieldName` merepresentasikan nama.

Nilai kembalian: objek JSON dalam bentuk *node*.

- `public ArrayNode putArray(String fieldName)`

Berfungsi untuk membangun sebuah `ArrayNode` dan menambahkan `ArrayNode` tersebut ke sebuah nama.

Parameter:

- `fieldName` merepresentasikan nama.

Nilai kembalian: sebuah `ArrayNode` yang baru dibangun.

2. `ArrayNode`

Kelas ini merupakan *node* yang memetakan sebuah atau sekumpulan *array* pada JSON (Subbab 2.4). Berikut adalah contoh kode untuk membangun sebuah kelas `ArrayNode` pada Play Framework:

```
1 |      ArrayNode arr = Json.newArray();
```

Berikut adalah beberapa *method* yang dimiliki oleh kelas ini:

- `public ArrayNode add(String v)`

Berfungsi untuk menambahkan sebuah data berupa `String` ke akhir *node array*.

Parameter:

- `v` merepresentasikan nilai berupa `String`.

Nilai kembalian: *array JSON* dalam bentuk *node*.

- `public ArrayNode add(JsonNode value)`

Berfungsi untuk menambahkan sebuah data berupa *node* (dapat berupa `ObjectNode` atau `ArrayNode`) ke akhir *node array*.

Parameter:

- `v` merepresentasikan nilai berupa `ObjectNode` atau `ArrayNode`.

Nilai kembalian: *array* atau objek JSON dalam bentuk *node*.

- `public ArrayNode addAll(ArrayNode other)`

Berfungsi untuk menambahkan sebuah data berupa `ArrayNode` ke akhir *node array*.

Parameter:

- `other` merepresentasikan `ArrayNode`.

Nilai kembalian: *array JSON* dalam bentuk *node*.

3.4.2 `JavaMail API`

Pada bagian *register* sistem ini (Subbab 3.1.14), sandi yang telah dibangun oleh sistem akan dikirimkan ke *email* pengguna, untuk itu diperlukan metode pengiriman *email* oleh sistem usulan. Setelah dilakukan analisis lebih lanjut, diketahui bahwa sistem ini menggunakan `PHPMailer`[11] sebagai metode pengiriman *email*. `PHPMailer` terlalu rumit untuk diimplementasikan dan tidak tersedia di Java. `JavaMail API` merupakan *library* Java yang menyediakan layanan `SMTP`[20]. `SMTP` adalah layanan yang dapat digunakan untuk melakukan pengiriman *email* secara efisien[21]. Berikut adalah contoh struktur kode untuk mengirimkan *email* dengan menggunakan `JavaMail API`:

```

1 Properties props = new Properties();
2 props.setProperty("mail.smtp.host", "my-mail-server");
3 Session session = Session.getInstance(props);
4 try {
5     MimeMessage msg = new MimeMessage(session);
6     msg.setFrom("me@example.com");
7     msg.setRecipients(Message.RecipientType.TO,
8         "you@example.com");
9     msg.setSubject("JavaMail hello world example");
10    msg.setSentDate(new Date());
11    msg.setText("Hello, world!\n");
12    Transport.send(msg, "me@example.com", "my-password");
13 } catch (MessagingException mex) {
14     System.out.println("send failed, exception: " + mex);
15 }

```

Baris 1-3 pada contoh kode di atas menyatakan pembangunan sebuah sesi *email* dan apa saja aturan yang ingin digunakan dalam sesi tersebut. Baris 5-11 menyatakan bagaimana rancangan pesan yang ingin dikirimkan. Baris 12 menyatakan bahwa pesan akan dikirim dengan menggunakan data otentikasi pada parameter *method* tersebut (“me@example.com” sebagai *username* dan “my-password” sebagai sandi). Baris 13-15 menyatakan bagaimana penanganan jika terjadi kesalahan. Berikut adalah penjelasan seluruh kelas yang digunakan pada contoh kode di atas:

1. Properties

Kelas ini merupakan kelas yang merepresentasikan sekumpulan data yang terdiri dari pasangan kata kunci dan nilai. Setiap pasangan kata kunci dan nilai dalam kelas ini merupakan data dalam format *String*. Berikut adalah sebuah *method* yang digunakan pada penelitian ini:

- `public Object setProperty(String key, String value)`

Berfungsi untuk menambahkan sebuah pasangan kata kunci dan nilai.

Parameter:

- `key` kata kunci.
- `value` nilai.

Nilai kembalian: nilai lama berdasarkan kata kunci pada parameter (atau `null` jika belum ada kata kunci yang dibangun).

2. Session

Kelas ini merepresentasikan sebuah sesi *email*. Sebuah sesi *email* dibangun dengan memiliki pengaturan awal. Pengaturan awal dibangun berdasarkan sekumpulan kata kunci dan nilai pada kelas *Properties*. Berikut adalah sebuah *method* yang digunakan pada penelitian ini:

- `public static Session getInstance(Properties props)`

Berfungsi untuk membangun sebuah sesi *email*.

Parameter:

- `props` merupakan sekumpulan kata kunci dan nilai yang menyatakan pengaturan awal.

Nilai kembalian: sebuah sesi *email*.

3. MimeMessage

Kelas ini merepresentasikan rancangan pesan yang ingin dikirimkan. Konstruktor kelas ini memerlukan sebuah sesi *email* (kelas *Session*) untuk dapat dijalankan. Jika syarat terpenuhi,

maka konstruktor kelas ini akan membangun sebuah pesan kosong. Berikut adalah beberapa *method* yang digunakan pada penelitian ini:

- `public void setFrom(String address)`
Berfungsi untuk memberikan alamat pengirim pesan.
Parameter:
 - `address` alamat *email* pengirim pesan.
- `public void setRecipients(Message.RecipientType type, String addresses)`
Berfungsi untuk memberikan alamat penerima pesan.
Parameter:
 - `type` tipe penerima pesan.
 - `address` alamat *email* penerima pesan.
- `public void setSubject(String subject)`
Berfungsi untuk memberikan judul pesan.
Parameter:
 - `subject` judul.
- `public void setSentDate(Date d)`
Berfungsi untuk memberikan tanggal pengiriman pesan.
Parameter:
 - `d` tanggal pengiriman.
- `public void setText(String text)`
Berfungsi untuk memberikan isi pesan.
Parameter:
 - `text` isi.

4. Transport

Kelas ini merepresentasikan pengiriman pesan. Berikut adalah sebuah *method* yang digunakan pada penelitian ini:

- `public static void send(Message msg, String user, String password)`
Berfungsi untuk mengirimkan pesan.
Parameter:
 - `msg` rancangan pesan.
 - `user` alamat *email*.
 - `password` sandi yang cocok dengan alamat *email*.

3.4.3 MySQL Connector/J

Sistem kini menggunakan MySQL sebagai sarana penyimpanan datanya (Subbab 3.2). MySQL Connector/J adalah JDBC API yang dapat digunakan untuk mengakses semua jenis data yang terstruktur, terutama data yang tersimpan dalam suatu *Relational Database* (Subbab 2.2).

Struktur kode untuk melakukan akses ke *database* telah dijelaskan pada subbab sebelumnya (Subbab 2.2). Namun demikian, struktur kode tersebut masih rentan terhadap serangan *SQL injection*. *SQL injection* adalah sebuah serangan pada *database* dengan mengubah *SQL statement* sedemikian rupa sehingga dapat menghancurkan suatu *database* sistem[17]. Berikut adalah contoh struktur kode untuk mencegah terjadinya serangan *SQL injection*:

```

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

```

Perbedaan antara kode di atas dengan kode pada subbab sebelumnya (Subbab 2.2) adalah pada baris ke 6, yaitu penggunaan kelas *PreparedStatement* (kode di atas) dan kelas *Statement* (subbab sebelumnya). Kelas *PreparedStatement* dapat menangani serangan *SQL injection* karena dapat menangani penulisan karakter jenis apapun (contoh: “;”). Berikut adalah beberapa *method* yang dimiliki kelas *PreparedStatement*:

- **public void setString(int parameterIndex, String x)**

Berfungsi untuk memberikan nilai *String* ke indeks parameter tujuan.

Parameter:

- **parameterIndex** merepresentasikan indeks dengan urutan 1,2,3,dst.
- **x** nilai yang akan dimasukkan.

- **public ResultSet executeQuery()**

Berfungsi untuk melakukan eksekusi terhadap *query* yang telah dibangun.

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

- **public int executeUpdate()**

Berfungsi untuk melakukan eksekusi terhadap *query* yang telah dibangun.

Nilai kembalian: jumlah baris yang berhasil dieksekusi dalam *database* atau 0 jika tidak ada baris yang dieksekusi.

3.4.4 jBCrypt

Berdasarkan analisa pada bagian *register* sistem kini (Subbab 3.1.14) didapatkan informasi bahwa dilakukan *hashing* pada sandi yang sudah diacak. *Hashing* yang dilakukan adalah menggunakan sebuah *PHP hashing framework*[15]. Berdasarkan hasil wawancara dengan kontributor kode KIRI, algoritma yang digunakan pada *framework* tersebut adalah *bcrypt*. *jBCrypt* adalah *library* Java untuk melakukan *hashing* data dengan menggunakan algoritma *bcrypt*[22]. Berikut adalah contoh kode untuk melakukan *hashing* suatu data:

```

1 String hashed = BCrypt.hashpw(password, BCrypt.gensalt());

```

Berikut adalah contoh kode untuk melakukan pengecekan antara sandi sebelum dilakukan *hashing* dan sandi yang sudah dilakukan *hashing*:

```
1 | if (BCrypt.checkpw(candidate, hashed))  
2 |     System.out.println("It matches");  
3 | else  
4 |     System.out.println("It does not match");
```

Baris 1 kode di atas menjelaskan bahwa untuk melakukan pengecekan digunakan *method* “**checkpw**”. Parameter “**candidate**” adalah sandi sebelum dilakukan *hashing* dan parameter “**hashed**” adalah sandi yang sudah dilakukan *hashing*.

Berikut adalah penjelasan secara detail seluruh *method* yang dimiliki kelas BCrypt:

- **public static String hashpw(String password, String salt)**

Berfungsi untuk melakukan *hashing* pada sebuah sandi dengan menggunakan algoritma bcr-ypt.

Parameter:

- **password** merepresentasikan indeks dengan urutan 1,2,3,dst.
- **salt** kunci yang digunakan.

Nilai kembalian: kata sandi yang telah dilakukan *hash*.

- **public static boolean checkpw(String plaintext, String hashed)**

Berfungsi untuk melakukan pengecekan apakah sebuah sandi cocok dengan sebuah data *hash*.

Parameter:

- **plaintext** sandi.
- **hashed** data *hash*.

Nilai kembalian: **true** jika cocok dan **false** jika tidak cocok.

- **public static String gensalt()**

Berfungsi untuk membangun kunci dalam format *String* yang dapat digunakan untuk melakukan *hashing* suatu sandi.

Parameter: Nilai kembalian: sebuah kunci.

BAB 4

PERANCANGAN

4.1 Perancangan Kelas

Seperti yang telah dijelaskan pada bab analisis, bahwa untuk memodelkan KIRI *Dashboard Server Side* dalam Play Framework membutuhkan *models*, *views*, dan *controllers*. Berdasarkan hasil analisis yang dilakukan pada bab analisis, telah dirancang diagram kelas untuk memenuhi kebutuhan dalam membangun aplikasi sistem usulan (Gambar 4.1). Deskripsi kelas beserta fungsi dari diagram kelas akan dijelaskan pada subbab selanjutnya.

4.2 *Controllers*

Controllers terdiri dari sebuah kelas, yaitu kelas Application. Kelas Application digunakan untuk menerima permintaan dari bagian tampilan sistem usulan dan digunakan juga sebagai pengirim pesan untuk membalas permintaan dari bagian tampilan. Deskripsi seluruh *method* dari kelas Application adalah sebagai berikut:

- `public Result index()`
Berfungsi untuk mengarahkan pengguna ke halaman “/bukitjarian/”.
Nilai kembalian: halaman *login KIRI Dashboard*.
- `public Result pagenotfound(String other)`
Berfungsi untuk memberikan informasi kepada pengguna bahwa halaman yang dituju tidak ada dalam sistem.
Parameter:
 - `other` halaman yang dituju.
Nilai kembalian: halaman *page not found*.
- `public Result handle()`
Berfungsi untuk menangani pembagian 16 permintaan pengguna (16 bagian yang telah dijelaskan pada bab analisis).
Nilai kembalian: pesan berhasil/kesalahan dalam format JSON yang sesuai dengan permintaan pengguna.

Gambar 4.1: Kelas Diagram KIRI *Dashboard Server Side*

- `private ObjectNode login(String userid, String password)`

Berfungsi untuk menangani permintaan *login*.

Parameter:

- `userid` *username* pengguna.
- `password` sandi pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi sesi id, hak akses terhadap rute, dan hak akses terhadap API *keys*.

- `private ObjectNode register(String email, String fullname, String company)`

Berfungsi untuk menangani permintaan *register*.

Parameter:

- `email` *email* pengguna.
- `fullname` nama lengkap pengguna.
- `company` nama perusahaan pengguna.

Nilai kembalian: pesan dalam format JSON yang menandakan *register* berhasil.

- `private ObjectNode logout(String sessionid)`

Berfungsi untuk menangani permintaan *logout*.

Parameter:

- `sessionid` sesi id yang didapat ketika pengguna berhasil *login*.

Nilai kembalian: pesan dalam format JSON yang menandakan *logout* berhasil.

- `private ObjectNode getProfile(User user)`

Berfungsi untuk menangani permintaan melihat data pribadi pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi nama lengkap dan nama perusahaan pengguna.

- `private ObjectNode updateProfile(User user, String newPassword, String newFullName, String newCompany)`

Berfungsi untuk mengubah data pribadi pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `newPassword` sandi baru pengguna.
- `newFullName` nama lengkap baru pengguna.
- `newCompany` nama perusahaan baru pengguna.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa mengubah data pribadi pengguna berhasil.

- `private ObjectNode getListOfApiKeys(User user)`

Berfungsi untuk melihat daftar API *keys* yang dimiliki pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi daftar API *keys* pengguna.

- `private ObjectNode addApiKey(User user, String domainFilter, String description)`

Berfungsi untuk menambahkan sebuah API *key* ke dalam daftar yang dimiliki pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `domainFilter` nama *domain* pengguna.
- `description` penjelasan tambahan untuk pengguna.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa penambahan sebuah API *key* berhasil dilakukan.

- `private ObjectNode updateApiKey(User user, String apiKey, String domainFilter, String description)`

Berfungsi untuk mengubah data nama domain dan deskripsi sebuah API *key* yang dimiliki pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `apiKey` data API *key* yang ingin diubah pengguna.
- `domainFilter` nama *domain* baru pengguna.
- `description` penjelasan tambahan baru untuk pengguna.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa mengubah data sebuah API *key* berhasil dilakukan.

- `private ObjectNode getListOfTracks(User user)`

Berfungsi untuk melihat daftar rute angkutan umum yang dimiliki oleh sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi daftar rute angkutan umum sistem KIRI.

- `private ObjectNode getDetailsTrack(User user, String trackID)`

Berfungsi untuk melihat data sebuah rute angkutan umum secara detail.

Parameter:

- **user** data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- **trackID** ID rute angkutan umum yang ingin dilihat secara detail.

Nilai kembalian: pesan dalam format JSON yang berisi sebuah data rute angkutan umum secara detail.

- **private ObjectNode deleteTrack(User user, String trackID)**

Berfungsi untuk menghapus sebuah rute angkutan umum milik sistem KIRI.

Parameter:

- **user** data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- **trackID** ID rute angkutan umum yang ingin dihapus.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa menghapus sebuah rute angkutan umum berhasil dilakukan.

- **private ObjectNode addTrack(User user, String trackID, String trackName, String trackType, String penalty, String internalInfo)**

Berfungsi untuk menambahkan sebuah rute angkutan umum ke sistem KIRI.

Parameter:

- **user** data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- **trackID** ID rute angkutan umum yang ingin ditambahkan.
- **trackName** nama rute angkutan umum yang ingin ditambahkan.
- **trackType** tipe angkutan umum yang ingin ditambahkan.
- **penalty** pengali bobot rute angkutan umum.
- **internalInfo** informasi seputar rute angkutan umum yang ingin ditambahkan.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa menambahkan sebuah rute angkutan umum berhasil dilakukan.

- **private ObjectNode updateTrack(User user, String trackID, String newTrackID, String trackType, String trackName, String internalInfo, String loop, String penalty, String transferNodes)**

Berfungsi untuk mengubah data sebuah rute angkutan umum milik sistem KIRI.

Parameter:

- **user** data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- **trackID** ID rute angkutan umum yang ingin diubah.
- **newTrackID** ID rute angkutan umum baru sebagai pengganti ID lama.
- **trackType** tipe angkutan umum baru.
- **trackName** nama rute angkutan umum baru.
- **internalInfo** informasi seputar rute angkutan umum baru.
- **loop** informasi apakah titik awal rute = titik akhir rute (1 atau 0).

- `penalty` pengali bobot rute angkutan umum baru.
- `transferNodes` informasi apakah node dapat dilakukan pemindahan atau tidak.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa mengubah sebuah rute angkutan umum berhasil dilakukan.

- `private ObjectNode clearGeoData(User user, String trackID)`

Berfungsi untuk menghapus data geografis sebuah rute angkutan umum milik sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `trackID` ID rute angkutan umum yang ingin diubah.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa menghapus data geografis sebuah rute angkutan umum berhasil dilakukan.

- `private ObjectNode importKML(User user, String trackID, File dataKML)`

Berfungsi untuk melakukan impor data KML (data geografis) ke sebuah rute angkutan umum milik sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `trackID` ID rute angkutan umum yang ingin ditambahkan data geografis.
- `dataKML` data KML.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa impor data KML ke sebuah rute angkutan umum berhasil dilakukan.

4.3 Models

Models terdiri dari 7 buah kelas. *Models* merupakan bagian pada Play Framework yang melakukan pemrosesan data secara detail. Deskripsi kelas beserta fungsi dari kelas tersebut akan dijelaskan ke dalam subbab selanjutnya.

4.3.1 ApiKeysManager

Kelas ini merupakan kelas untuk mengelola data-data API *keys* yang dimiliki oleh pengguna KIRI *Dashboard*. Kelas ini untuk menangani permintaan: melihat daftar API *keys*, menambahkan API *keys* dan mengubah API *keys*. Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public ObjectNode getListOfApiKeys(User user)`

Berfungsi untuk mendapatkan daftar API *keys* pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi daftar API *keys* pengguna.

- `public ObjectNode addApiKey(User user, String domainFilter, String description)`

Berfungsi untuk menambahkan sebuah API *key* ke dalam daftar yang dimiliki pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `domainFilter` nama *domain* pengguna.
- `description` penjelasan tambahan untuk pengguna.

Nilai kembalian: pesan dalam format JSON yang menandakan bahwa penambahan API *key* berhasil dilakukan.

- `public void updateApiKey(User user, String apiKey, String domainFilter, String description)`

Berfungsi untuk mengubah nama *domain* dan deskripsi dari sebuah API *key* yang dimiliki pengguna.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `apiKey` data API *key* yang ingin diubah pengguna.
- `domainFilter` nama *domain* baru pengguna.
- `description` penjelasan tambahan baru untuk pengguna.

- `private void checkPrivilege(boolean privilegeApiUsage)`

Berfungsi untuk melakukan pengecekan apakah pengguna memiliki hak akses terhadap API *keys* atau tidak.

Parameter:

- `privilegeApiUsage` hak akses pengguna terhadap API *keys*.

- `private String generateApiKey()`

Berfungsi untuk membangun sebuah API *key* baru.

Nilai kembalian: sebuah API *key* dalam format *String* yang dibangun secara acak.

4.3.2 AuthenticationManager

Kelas ini merupakan kelas untuk mengelola proses otentikasi pengguna terhadap sistem KIRI *Dashboard*. Kelas ini untuk menangani permintaan: *login*, *register*, dan *logout*. Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public void register(String email, String fullname, String company)`

Berfungsi untuk mendaftarkan data diri pengguna untuk mendapatkan hak akses terhadap KIRI *Dashboard*.

Parameter:

- `email` alamat *email* pengguna (berperan sebagai *username*).
- `fullname` nama lengkap pengguna.
- `company` nama perusahaan pengguna.

- `public ObjectNode login(String userid, String password)`

Berfungsi untuk melakukan otentikasi sebagai pengguna terhadap KIRI *Dashboard*.

Parameter:

- `userid` alamat *email* yang sebelumnya telah didaftarkan pengguna.
- `password` sandi milik pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi tentang data sesi dan hak akses (terhadap rute dan API *keys*) dan pesan yang menandakan bahwa *login* berhasil dilakukan.

- `public void logout(String sessionid)`

Berfungsi untuk melakukan *logout* sebagai pengguna terhadap KIRI *Dashboard*.

Parameter:

- `sessionid` data sesi yang dibangun pada saat melakukan *login*.

- `private void sendPassword(String email, String password, String fullname)`

Berfungsi untuk mengirimkan sandi yang telah dibangun oleh sistem KIRI *Dashboard* ke alamat *email* pengguna.

Parameter:

- `email` alamat *email* pengguna.
- `password` sandi yang telah dibangun sistem KIRI *Dashboard*.
- `fullname` nama lengkap pengguna.

- `private String generateSessionID()`

Berfungsi untuk membangun data sesi baru.

Nilai kembalian: sebuah data sesi dalam format *String* yang dibangun secara acak.

- `private String generatePassword()`

Berfungsi untuk membangun sebuah sandi baru.

Nilai kembalian: sebuah data sandi dalam format *String* yang dibangun secara acak.

- `private void returnInvalidCredentials(String logMessage)`

Berfungsi untuk melemparkan dan mencatat informasi kesalahan bila terjadi kesalahan pada saat proses otentikasi.

Parameter:

- `logMessage` informasi kesalahan yang dilakukan.

- `private void logError(String message)`

Berfungsi untuk mencatat informasi kesalahan bila terjadi kesalahan pada saat proses otentikasi.

Parameter:

- `message` informasi kesalahan yang dilakukan.

4.3.3 Constant

Kelas ini merupakan kelas yang berisi mengenai konstanta-konstanta statis yang digunakan dalam sistem KIRI. Karena sifat konstanta yang statis, maka deklarasi di Java dibuat *final*. Berikut adalah seluruh konstanta yang digunakan pada kelas ini:

- `String APIKEY_KIRI`: API *key* milik sistem KIRI.
- `String PROTO_MODE`: mode permintaan.
- `String PROTO_MODE_LOGIN`: mode permintaan *login*.
- `String PROTO_MODE_REGISTER`: mode permintaan *register*.
- `String PROTO_MODE_LOGOUT`: mode permintaan *logout*.
- `String PROTO_MODE_GET_PROFILE`: mode permintaan melihat data diri pengguna.
- `String PROTO_MODE_UPDATE_PROFILE`: mode permintaan mengubah data diri pengguna.
- `String PROTO_MODE_LIST_API_KEYS`: mode permintaan melihat daftar API *keys* milik pengguna.
- `String PROTO_MODE_ADD_API_KEY`: mode permintaan menambahkan sebuah API *key* ke daftar milik pengguna.
- `String PROTO_MODE_UPDATE_API_KEY`: mode permintaan mengubah sebuah API *key* milik pengguna.
- `String PROTO_MODE_LIST_TRACKS`: mode permintaan melihat daftar rute angkutan umum.
- `String PROTO_MODE_GET_DETAILS_TRACK`: mode permintaan melihat detail rute angkutan umum.
- `String PROTO_MODE_DELETE_TRACK`: mode permintaan menghapus sebuah rute angkutan umum.
- `String PROTO_MODE_ADD_TRACK`: mode permintaan menambahkan sebuah rute angkutan umum.
- `String PROTO_MODE_UPDATE_TRACK`: mode permintaan mengubah data sebuah rute angkutan umum.
- `String PROTO_MODE_CLEAR_GEODATA`: mode permintaan menghapus data geografis sebuah rute angkutan umum.
- `String PROTO_MODE_IMPORT_KML`: mode permintaan melakukan impor data KML untuk sebuah rute angkutan umum.
- `String PROTO_STATUS`: status sistem.
- `String PROTO_STATUS_OK`: status sistem berhasil.
- `String PROTO_FULL_NAME`: nama lengkap.

- String `PROTO_COMPANY`: nama perusahaan.
- String `PROTO_MESSAGE`: pesan.
- String `PROTO_API_KEYS_LIST`: daftar API *keys*.
- String `PROTO_SESSION_ID`: data sesi.
- String `PROTO_PRIVILEGES`: hak akses.
- String `PROTO_VERIFIER`: pemeriksa.
- String `PROTO_TRACK_ID`: ID rute angkutan umum.
- String `PROTO_TRACKS_LIST`: daftar rute angkutan umum.
- String `PROTO_TRACK_TYPE`: tipe angkutan umum.
- String `PROTO_TRACK_NAME`: nama rute angkutan umum.
- String `PROTO_TRACK_TYPES_LIST`: daftar tipe angkutan umum.
- String `PROTO_INTERNAL_INFO`: keterangan tambahan.
- String `PROTO_GEO_DATA`: data geografis.
- String `PROTO_PATH_LOOP`: rute angkutan umum dimana titik awal = titik akhir.
- String `PROTO_PENALTY`: pengali bobot rute angkutan umum.
- String `PROTO_TRANSFER_NODES`: daftar *node* yang dapat dipindahkan.
- String `PROTO_USER_ID`: ID pengguna.
- String `PROTO_PASSWORD`: sandi.
- String `PROTO_DOMAIN_FILTER`: nama *domain*.
- String `PROTO_DESCRIPTION`: deskripsi.
- String `PROTO_NEW_TRACK_ID`: ID rute angkutan umum baru.
- String `PROTO_UPLOADED_FILE`: mengirimkan data.
- String `ERROR`: pesan kesalahan.
- String `ERROR_CREDENTIAL_FAIL`: pesan kesalahan pada bagian otentikasi.
- String `ERROR_SESSION_EXPIRED`: pesan kesalahan data sesi telah habis waktu.
- String `ERROR_MODE_NOT_FOUND`: pesan kesalahan mode tidak ditemukan.
- int `MAX_FILE_SIZE`: data maksimum yang dapat diterima oleh sistem KIRI.

4.3.4 TracksManager

Kelas ini merupakan kelas untuk mengelola seluruh rute angkutan umum sistem KIRI. Kelas ini untuk menangani permintaan: menambahkan rute angkutan umum, melihat daftar rute angkutan umum, melihat rute angkutan umum secara detail, mengubah data rute angkutan umum, menghapus rute angkutan umum, menghapus data geografis sebuah angkutan umum, dan melakukan impor data KML. Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public void addTrack(User user, String trackID, String trackName, String trackType, String penalty, String internalInfo)`

Berfungsi untuk menambahkan sebuah rute angkutan umum ke sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `trackID` ID rute angkutan umum yang ingin ditambahkan.
- `trackName` nama rute angkutan umum yang ingin ditambahkan.
- `trackType` tipe angkutan umum yang ingin ditambahkan.
- `penalty` pengali bobot rute angkutan umum.
- `internalInfo` informasi seputar rute angkutan umum yang ingin ditambahkan.

- `public ObjectNode getListOfTracks(User user)`

Berfungsi untuk melihat daftar rute angkutan umum yang dimiliki oleh sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.

Nilai kembalian: pesan dalam format JSON yang berisi daftar rute angkutan umum sistem KIRI.

- `public ObjectNode getDetailsTrack(User user, String trackID)`

Berfungsi untuk melihat data sebuah rute angkutan umum secara detail.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `trackID` ID rute angkutan umum yang ingin dilihat secara detail.

Nilai kembalian: pesan dalam format JSON yang berisi sebuah data rute angkutan umum secara detail.

- `public void updateTrack(User user, String trackID, String newTrackID, String trackType, String trackName, String internalInfo, String loop, String penalty, String transferNodes)`

Berfungsi untuk mengubah data sebuah rute angkutan umum milik sistem KIRI.

Parameter:

- `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
- `trackID` ID rute angkutan umum yang ingin diubah.

- `newTrackID` ID rute angkutan umum baru sebagai pengganti ID lama.
 - `trackType` tipe angkutan umum baru.
 - `trackName` nama rute angkutan umum baru.
 - `internalInfo` informasi seputar rute angkutan umum baru.
 - `loop` informasi apakah titik awal rute = titik akhir rute (1 atau 0).
 - `penalty` pengali bobot rute angkutan umum baru.
 - `transferNodes` informasi apakah node dapat dilakukan pemindahan atau tidak.
- `public void deleteTrack(User user, String trackID)`
Berfungsi untuk menghapus sebuah rute angkutan umum milik sistem KIRI.
Parameter:
 - `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
 - `trackID` ID rute angkutan umum yang ingin dihapus.
 - `public void clearGeoData(User user, String trackID)`
Berfungsi untuk menghapus data geografis sebuah rute angkutan umum milik sistem KIRI.
Parameter:
 - `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
 - `trackID` ID rute angkutan umum yang ingin diubah.
 - `public void importKML(User user, String trackID, File dataKML)`
Berfungsi untuk melakukan impor data KML (data geografis) ke sebuah rute angkutan umum milik sistem KIRI.
Parameter:
 - `user` data sesi dan hak akses (rute dan API *key*) yang dimiliki pengguna.
 - `trackID` ID rute angkutan umum yang ingin ditambahkan data geografis.
 - `dataKML` data KML.
 - `private void updateTrackVersion()`
Berfungsi untuk memperbaharui versi rute angkutan umum dalam sistem KIRI.
 - `private void checkPrivilege(boolean privilegeRoute)`
Berfungsi untuk melakukan pengecekan apakah pengguna memiliki hak akses terhadap rute angkutan umum sistem KIRI atau tidak.
 - `private ArrayNode lineStringToLatLngArray(String lineString)`
Berfungsi untuk melakukan format data dalam format LINESTRING menjadi data *String* yang dapat dibaca oleh bagian tampilan sistem KIRI.

4.3.5 UniqueStatusError

Kelas ini merupakan kelas untuk melemparkan 2 jenis pesan kesalahan pada sistem KIRI, yaitu: “credentialfail” dan “sessionexpired”. Berikut adalah sebuah konstanta yang digunakan pada kelas ini:

- `private String status`: pesan kesalahan.

Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public String getStatus()`
Berfungsi untuk mendapatkan pesan kesalahan.
Nilai kembalian: pesan kesalahan.
- `public void setStatus(String status)`
Berfungsi untuk mengubah pesan kesalahan.
Parameter:
 - `status` pesan kesalahan baru.

4.3.6 User

Kelas ini merupakan kelas untuk mengelola data-data pribadi pengguna sistem KIRI *Dashboard*. Kelas ini untuk menangani permintaan: pemeriksaan *login*, melihat data pribadi pengguna, dan mengubah data pribadi pengguna. Konstruktor pada kelas ini memiliki fungsi yang sama dengan bagian pemeriksaan *login* sistem ini. Berikut adalah seluruh konstanta yang digunakan pada kelas ini:

- `private String sessionId`: data sesi yang dibangun saat melakukan *login*.
- `private String activeUserID`: alamat *email* pengguna.
- `private boolean privilegeRoute`: hak akses terhadap rute angkutan umum sistem KIRI.
- `private boolean privilegeApiUsage`: hak akses terhadap API *keys* milik pengguna.

Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public ObjectNode getProfile()`
Berfungsi untuk menangani permintaan melihat data pribadi pengguna.
Nilai kembalian: pesan dalam format JSON yang berisi nama lengkap dan nama perusahaan pengguna.
- `public void updateProfile(String newPassword, String newFullName, String newCompany)`
Berfungsi untuk mengubah data pribadi pengguna.
Parameter:
 - `newPassword` sandi baru pengguna.
 - `newFullName` nama lengkap baru pengguna.
 - `newCompany` nama perusahaan baru pengguna.

- `public String getActiveUserID()`
 Berfungsi untuk mendapatkan alamat *email* pengguna.
 Nilai kembalian: alamat *email* pengguna.
- `public boolean isPrivilegeRoute()`
 Berfungsi untuk mengecek apakah pengguna memiliki hak akses terhadap rute angkutan umum.
 Nilai kembalian: `true` atau `false`.
- `public boolean isPrivilegeApiUsage()`
 Berfungsi untuk mengecek apakah pengguna memiliki hak akses terhadap API *keys*.
 Nilai kembalian: `true` atau `false`.

4.3.7 Utils

Kelas ini merupakan kelas penyedia beberapa *method* yang umum digunakan oleh sistem KIRI. Beberapa bagian pada sistem usulan menggunakan beberapa *method* yang cara kerjanya sama, untuk itu dibuatlah kelas ini. Berikut adalah seluruh *method* yang digunakan pada kelas ini:

- `public static String generateRandom(String chars, int length)`
 Berfungsi untuk membangun sebuah data *String* secara acak.
 Parameter:
 - `chars` daftar karakter yang hanya digunakan dalam membangun data acak.
 - `length` panjang data yang ingin dibangun.
 Nilai kembalian: data dalam format *String* yang dibangun secara acak.
- `public static ObjectNode wellDone()`
 Berfungsi untuk membangun sebuah objek JSON sebagai penanda keberhasilan suatu bagian pada sistem KIRI.
 Nilai kembalian: sebuah objek dalam format JSON sebagai penanda keberhasilan suatu bagian pada sistem KIRI.
- `public static void dieNice(String message)`
 Berfungsi untuk menghentikan sistem KIRI karena terjadi kesalahan sistem ataupun pengguna.
 Parameter:
 - `message` deskripsi kesalahan yang dilakukan oleh sistem atau pengguna.
 Nilai kembalian: data dalam format *String* yang dibangun secara acak.
- `public static void logStatistic(String verifier, String type, String additional_info)`
 Berfungsi untuk melakukan pencatatan bila terjadi kesalahan pada sistem KIRI.
 - `verifier` API *keys* aplikasi yang menggunakan sistem KIRI.

- `type` bagian tempat terjadi kesalahan.
- `additional_info` informasi tambahan.

4.4 *Views*

Disalin apa adanya.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Menjelaskan mengenai spesifikasi laptop.

5.1.2 Hasil Implementasi

Penjelasan mengenai hasil 16 metode.

5.2 Hasil Pengujian

5.2.1 Pengujian Fungsional

Pengujian dilakukan sendiri.

5.2.2 Pengujian Eksperimental

Pengujian dilakukan oleh kontributor kode KIRI.

DAFTAR REFERENSI

- [1] Pascal Alfadian, “KIRI.” <http://kiri.travel/>, 2014. [Online; diakses 1-Oktober-2015].
- [2] Pascal Alfadian, “KIRI.” <https://dev.kiri.travel/bukitjarian/>, 2014. [Online; diakses 1-Oktober-2015].
- [3] GeoJSON, “Representasi Objek dalam Geometri.” <http://geojson.io/>, 2015. [Online; diakses 4-November-2015].
- [4] N. Leroux and S. D. Kaper, *Play for Java*. Manning Publications Co., 2014.
- [5] Pascal Alfadian, “TirtayasaGH.” <https://github.com/pascalalfadian/TirtayasaGH>, 2014. [Online; diakses 1-Oktober-2015].
- [6] Oracle, “MySQL 5.7 Reference Manual.” <https://dev.mysql.com/doc/refman/5.7/en/>, 2015. [Online; diakses 4-November-2015].
- [7] Oracle, “Java Documentation.” <https://docs.oracle.com/javase/8/>, 2015. [Online; diakses 26-November-2015].
- [8] Play Framework, “Play 2.4.x documentation.” <https://www.playframework.com/documentation/2.4.x/Home>, 2015. [Online; diakses 4-November-2015].
- [9] Standard ECMA-262 3rd Edition, “Introducing JSON.” <http://www.json.org/>, 2015. [Online; diakses 1-Desember-2015].
- [10] The Eclipse Foundation, “About the Eclipse Foundation.” <https://eclipse.org/org>, 2015. [Online; diakses 25-November-2015].
- [11] GitHub Inc, “About GitHub.” <https://github.com/>, 2015. [Online; diakses 25-November-2015].
- [12] Apache Ant, “Apache Ant.” <http://ant.apache.org/>, 2015. [Online; diakses 25-November-2015].
- [13] GWT, “Overview GWT.” <http://www.gwtproject.org/>, 2015. [Online; diakses 25-November-2015].
- [14] Google Developers, “Keyhole Markup Language.” <https://developers.google.com/kml/>, 2015. [Online; diakses 26-November-2015].
- [15] Solar Designer, “Portable PHP Password Hashing Framework.” <http://www.openwall.com/phpass/>, 2004. [Online; diakses 14-April-2016].

- [16] A. Konheim, *7. HASHING FOR STORAGE: DATA MANAGEMENT*. Wiley-Interscience, 2010.
- [17] W3Schools, “THE WORLD’S LARGEST WEB DEVELOPER SITE.” <http://www.w3schools.com/>, 2015. [Online; diakses 1-Desember-2015].
- [18] Frodriguez, “Maven Repository.” <http://mvnrepository.com/>, 2006. [Online; diakses 14-April-2016].
- [19] FasterXML, LLC, “Jackson JSON Processor Wiki.” <http://wiki.fasterxml.com/JacksonHome>, 2016. [Online; diakses 13-April-2016].
- [20] Oracle, Project Kenai dan Cognisync, “SMTP Transport.” <https://java.net/projects/javamail/pages/SMTPTransport>, 2014. [Online; diakses 14-April-2016].
- [21] J. Klensin, “Simple Mail Transfer Protocol.” <https://tools.ietf.org/html/rfc5321>, 2008. [Online; diakses 14-April-2016].
- [22] Niels Provos dan Niels Provos, “jBCrypt.” <http://www.mindrot.org/projects/jBCrypt/>, 2015. [Online; diakses 14-April-2016].

LAMPIRAN A

THE SOURCE CODE

Listing A.1: handle.php

```

1 <?php
2 require_once '../etc/Utils.php';
3 require_once '../etc/constants.php';
4 require_once '../etc/PasswordHash.php';
5
6 start_working();
7
8 $mode = retrieve_from_post($proto_mode);
9
10 // Initializes MySQL and check for session
11 init_mysql();
12 if ($mode != $proto_mode_login && $mode != $proto_mode_logout && $mode != $proto_mode_register) {
13     $sessionid = addslashes(retrieve_from_post($proto_sessionid));
14     // Clear expired sessions
15     mysql_query($global_mysql_link, "DELETE FROM sessions WHERE lastSeen < (NOW() - INTERVAL
        $session_expiry_interval mysql)"); or
16     die_nice('Failed to clean expired sessions: ' . mysql_error($global_mysql_link), true);
17     $result = mysql_query($global_mysql_link, "SELECT users.email, users.privilegeRoute, users.
        privilegeApiUsage FROM users LEFT JOIN sessions ON users.email=sessions.email WHERE sessions.
        sessionid=$sessionid"); or
18     die_nice('Failed to get user session information: ' . mysql_error($global_mysql_link), true);
19     if (mysql_num_rows($result) == 0) {
20         deinit_mysql();
21         // Construct json - session expired.
22         $json = array(
23             $proto_status => $proto_status_sessionexpired,
24         );
25         print(json_encode($json));
26         exit(0);
27     }
28     $columns = mysql_fetch_row($result);
29     $active_userid = $columns[0];
30     $privilege_route = $columns[1] != '0';
31     $privilege_apiUsage = $columns[2] != '0';
32 }
33
34 if ($mode == $proto_mode_login) {
35     $userid = addslashes(retrieve_from_post($proto_userid));
36     $plain_password = addslashes(retrieve_from_post($proto_password));
37     if (strlen($userid) > $maximum_userid_length) {
38         return_invalid_credentials("User ID length is more than allowed (". strlen($userid) . ')');
39     }
40     if (strlen($plain_password) > $maximum_password_length) {
41         return_invalid_credentials("Password length is more than allowed (". strlen($password) . ')');
42     }
43
44     // Retrieve the user information
45     $result = mysql_query($global_mysql_link, "SELECT * FROM users WHERE email='$userid'"); or
46     die_nice('Failed to verify user id: ' . mysql_error($global_mysql_link), true);
47     if (mysql_num_rows($result) == 0) {
48         deinit_mysql();
49         return_invalid_credentials("User id not found: $userid");
50     }
51     $userdata = mysql_fetch_assoc($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     $sessionid = generate_sessionid();
65     mysql_query($global_mysql_link, "INSERT INTO sessions (sessionid, email) VALUES ('$sessionid', '$
        $userid')"); or
66     die_nice('Failed to generate session: ' . mysql_error($global_mysql_link), true);
67
68     // Construct privilege lists

```

```

69     $privileges = '';
70     if ($userdata['privilegeRoute'] != 0) {
71         $privileges .= ",$proto_privilege_route";
72     }
73     if ($userdata['privilegeApiUsage'] != 0) {
74         $privileges .= ",$proto_privilege_apiUsage";
75     }
76     if (strlen($privileges) > 0) {
77         $privileges = substr($privileges, 1);
78     }
79
80     // Construct json.
81     $json = array(
82         $proto_status => $proto_status_ok,
83         $proto_sessionid => $sessionid,
84         $proto_privileges => $privileges
85     );
86
87     deinit_mysql();
88     print(json_encode($json));
89 } elseif ($mode == $proto_mode_logout) {
90     $sessionid = addslashes(retrieve_from_post($proto_sessionid));
91
92     // Remove the session information
93     $result = mysqli_query($global_mysql_link, "DELETE FROM sessions WHERE sessionId='$sessionid'") or
94         die_nice('Failed to logout sessionid $sessionid: ' . mysqli_error($global_mysql_link), true);
95     deinit_mysql();
96     well_done();
97 } elseif ($mode == $proto_mode_add_track) {
98     check_privilege($privilege_route);
99     $trackid = addslashes(retrieve_from_post($proto_trackid));
100    $trackname = addslashes(retrieve_from_post($proto_trackname));
101    $tracktype = addslashes(retrieve_from_post($proto_tracktype));
102    $penalty = addslashes(retrieve_from_post($proto_penalty));
103    $internalinfo = addslashes(retrieve_from_post($proto_internalinfo, false)) or $internalinfo = '';
104
105    // Check if the id is already existed
106    $result = mysqli_query($global_mysql_link, "SELECT trackId FROM tracks WHERE trackId='$trackid'") or
107        die_nice('Failed to check trackid existence: ' . mysqli_error($global_mysql_link), true);
108    if (mysqli_num_rows($result) == 0) {
109        mysqli_query($global_mysql_link, "INSERT INTO tracks (trackId, trackTypeId, trackName, penalty,
110            internalInfo) VALUES ('$trackid', '$tracktype', '$trackname', '$penalty', '$internalinfo')") or
111            die_nice('Failed to add a new track: ' . mysqli_error($global_mysql_link), true);
112        update_trackversion();
113    } else {
114        die_nice("The trackId '$trackid' already existed.", true);
115    }
116    deinit_mysql();
117    well_done();
118 } elseif ($mode == $proto_mode_update_track) {
119     check_privilege($privilege_route);
120     $trackid = addslashes(retrieve_from_post($proto_trackid));
121     $newtrackid = addslashes(retrieve_from_post($proto_new_trackid));
122     $tracktype = addslashes(retrieve_from_post($proto_tracktype));
123     $trackname = addslashes(retrieve_from_post($proto_trackname));
124     $internalinfo = addslashes(retrieve_from_post($proto_internalinfo, false)) or $internalinfo = '';
125     $pathloop = retrieve_from_post($proto_pathloop) == 'true' ? 1 : 0;
126     $penalty = addslashes(retrieve_from_post($proto_penalty));
127     $transfernodes = retrieve_from_post($proto_transfernodes, false);
128
129     // When changed, check if the id is already existed
130     if ($newtrackid != $trackid) {
131         $result = mysqli_query($global_mysql_link, "SELECT trackId FROM tracks WHERE trackId='$newtrackid'") or
132             die_nice('Failed to check trackid existence: ' . mysqli_error($global_mysql_link), true);
133         if (mysqli_num_rows($result) != 0) {
134             die_nice("The new trackId '$newtrackid' already existed.", true);
135         }
136     }
137     mysqli_query($global_mysql_link, "UPDATE tracks SET trackTypeId='$tracktype', trackId='$newtrackid',
138         trackName='$trackname', internalInfo='$internalinfo', pathloop='$pathloop', penalty='$penalty'
139         WHERE trackId='$trackid'") or
140         die_nice('Failed to update the track: ' . mysqli_error($global_mysql_link));
141     if (!is_null($transfernodes)) {
142         $transfernodes = addslashes($transfernodes);
143         mysqli_query($global_mysql_link, "UPDATE tracks SET transferNodes='$transfernodes' WHERE trackId='$
144             trackid'") or
145             die_nice('Failed to update the track: ' . mysqli_error($global_mysql_link));
146     }
147     update_trackversion();
148     deinit_mysql();
149     well_done();
150 } elseif ($mode == $proto_mode_list_tracks) {
151     check_privilege($privilege_route);
152     // Retrieve track list from database
153     $result = mysqli_query($global_mysql_link, 'SELECT trackTypeId, trackId, trackName FROM tracks ORDER
154         BY trackTypeId, trackId') or
155         die('Cannot retrieve the track names from database');
156     $track_list = array();
157     while ($row = mysqli_fetch_row($result)) {
158         $track_list[] = array($row[1], htmlspecialchars($row[0] . '/' . $row[2]));
159     }
160     // Retrieve track types list result from database
161     $result = mysqli_query($global_mysql_link, 'SELECT trackTypeId, name FROM tracktypes ORDER BY
162         trackTypeId') or
163         die('Cannot retrieve the track types from database');
164     $tracktype_list = array();
165     while ($row = mysqli_fetch_row($result)) {
166         $tracktype_list[] = array($row[0], htmlspecialchars($row[1]));
167     }

```



```

251
252 // Check if the id is already existed
253 mysql_query($global_mysql_link, "DELETE FROM tracks WHERE trackId='$trackid'") or
254     die_nice('Failed to delete track $trackid: ' . mysql_error($global_mysql_link), true);
255 if (mysql_affected_rows($global_mysql_link) == 0) {
256     die_nice("The track $trackid was not found in the database", true);
257 }
258 update_trackversion();
259 deinit_mysql();
260 well_done();
261 } elseif ($mode == $proto_mode_list_apikeys) {
262     check_privilege($privilege_apiUsage);
263     // Retrieve api key list from database
264     $result = mysql_query($global_mysql_link, "SELECT verifier, domainFilter, description FROM apikeys
265         WHERE email='$active_userid' ORDER BY verifier") or
266         die_nice('Cannot retrieve the API keys list from database: ' . mysql_error($global_mysql_link));
267     $apikey_list = array();
268     while ($row = mysql_fetch_row($result)) {
269         $apikey_list[] = array($row[0], $row[1], $row[2]);
270     }
271     // Construct json.
272     $json = array(
273         $proto_status => $proto_status_ok,
274         $proto_apikeys_list => $apikey_list,
275     );
276     deinit_mysql();
277     print(json_encode($json));
278 } elseif ($mode == $proto_mode_add_apikey) {
279     check_privilege($privilege_apiUsage);
280     $domainfilter = addslashes(retrieve_from_post($proto_domainfilter));
281     $description = addslashes(retrieve_from_post($proto_description));
282     $apikey = generate_apikey();
283     // Retrieve api key list from database
284     $result = mysql_query($global_mysql_link, "INSERT INTO apikeys(verifier, email, domainFilter,
285         description) VALUES('$apikey', '$active_userid', '$domainfilter', '$description')") or
286         die_nice('Cannot insert a new api key: ' . mysql_error($global_mysql_link));
287     log_statistic("$apikey_kiri", 'ADDAPIKEY', $userid . $apikey);
288     // Construct json.
289     $json = array(
290         $proto_status => $proto_status_ok,
291         $proto_verifier => $apikey,
292     );
293     deinit_mysql();
294     print(json_encode($json));
295 } elseif ($mode == $proto_mode_update_apikey) {
296     check_privilege($privilege_apiUsage);
297     $apikey = addslashes(retrieve_from_post($proto_verifier));
298     $domainfilter = addslashes(retrieve_from_post($proto_domainfilter));
299     $description = addslashes(retrieve_from_post($proto_description));
300     // Ensure that this user has access to the apikey
301     $result = mysql_query($global_mysql_link, "SELECT email FROM apikeys WHERE verifier='$apikey'") or
302         die_nice('Cannot check API key owner: ' . mysql_error($global_mysql_link));
303     while ($row = mysql_fetch_row($result)) {
304         if ($row[0] != $active_userid) {
305             die_nice("User $active_userid does not have privilege to update API Key $apikey");
306         }
307     }
308     mysql_query($global_mysql_link, "UPDATE apikeys SET domainFilter='$domainfilter', description=
309         '$description' WHERE verifier='$apikey'") or
310         die_nice('Failed to update API Key: ' . mysql_error($global_mysql_link));
311     deinit_mysql();
312     well_done();
313 } elseif ($mode == $proto_mode_register) {
314     $email = addslashes(retrieve_from_post($proto_userid));
315     $fullname = addslashes(retrieve_from_post($proto_fullname));
316     $company = addslashes(retrieve_from_post($proto_company));
317     // Check if the email has already been registered.
318     $result = mysql_query($global_mysql_link, "SELECT email FROM users WHERE email='$email'") or
319         die_nice('Cannot check user id existence: ' . mysql_error($global_mysql_link));
320     if (mysql_num_rows($result) > 0) {
321         die_nice("Oops! Email $email has already registered. Please check your mailbox or contact
322             hello@kiri.travel");
323     }
324     // Generate and send password
325     $password = generate_password();
326     $hasher = new PasswordHash($passwordhash_cost_log2, $passwordhash_portable);
327     $passwordHash = $hasher->HashPassword($password);
328     mysql_query($global_mysql_link, "INSERT INTO users(email, password, privilegeApiUsage, fullname,
329         company) VALUES('$email', '$passwordHash', 1, '$fullname', '$company')") or
330         die_nice('Cannot add new user $email: ' . mysql_error($global_mysql_link));
331     sendPassword($email, $password, $fullname);
332     log_statistic("$apikey_kiri", 'REGISTER', "$email/$fullname/$company");
333     deinit_mysql();
334     well_done();
335 } elseif ($mode == $proto_mode_getprofile) {
336     $email = $active_userid;
337 }

```



```

344     $result = mysqli_query($global_mysqli_link, "SELECT_fullName,_company_FROM_users_WHERE_email='
345         $email'" ) or
346         die_nice('Cannot_retrieve_user_details:_'. mysqli_error($global_mysqli_link));
347     if ($row = mysqli_fetch_row($result)) {
348         $fullname = $row[0];
349         $company = $row[1];
350     } else {
351         die_nice("User_$email_not_found_in_database.");
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='$email'"
373             ) or
374             die_nice('Cannot_update_password_for_$email:_'. mysqli_error($global_mysqli_link));
375     }
376     mysqli_query($global_mysqli_link, "UPDATE_users_SET_fullName='$fullname',_company='$company'_WHERE_
377         email='$email'" ) or
378         die_nice('Cannot_update_profile_for_$email:_'. mysqli_error($global_mysqli_link));
379     deinit_mysql();
380     well_done();
381 } else {
382     die_nice("Mode_not_understood:_\" . $mode . "\" , true);
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 ?>

```