

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 consectetur.

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	xviii
DAFTAR TABEL	xix
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
3 ANALISIS	15
3.1 Analisis Sistem Kini	15
3.2 Analisis <i>Database</i> Sistem Kini	15
3.3 Analisis teori-teori untuk membangun KIRI <i>Dashboard Server Side</i> dalam bahasa Java dengan menggunakan Play Framework	15
DAFTAR REFERENSI	17
A THE SOURCE CODE	19

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 <i>file</i> “routes”[4]	12
2.5	Hubungan <i>routes</i> dan <i>controllers</i> dalam memproses HTTP <i>requests</i> [4]	13

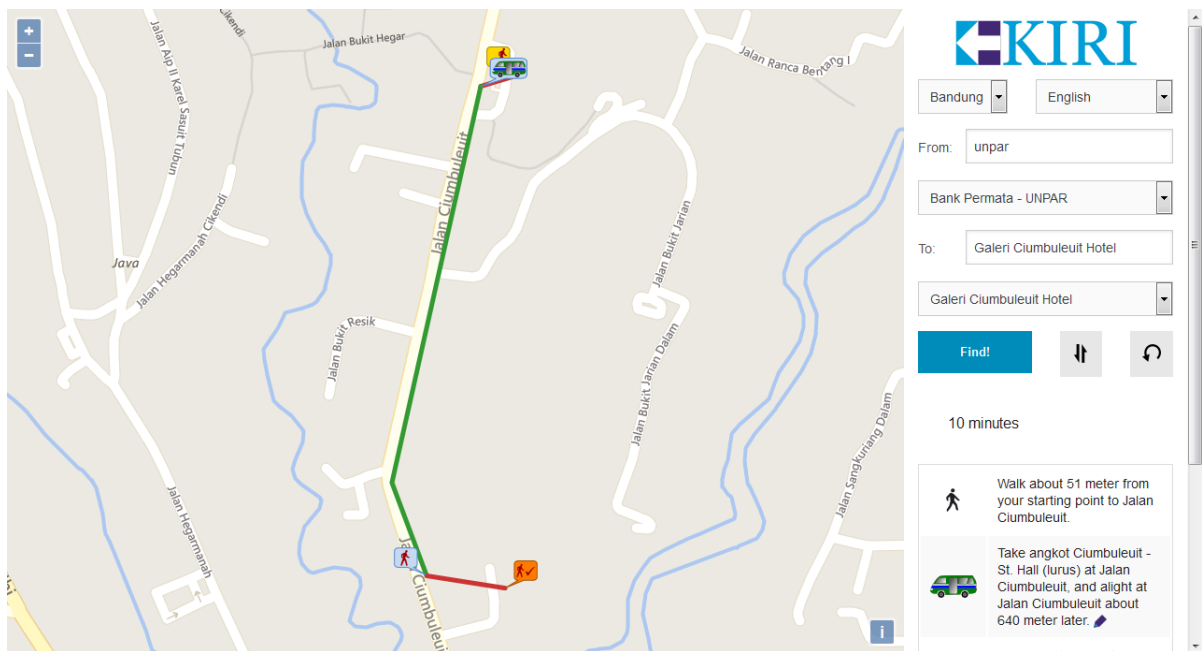
DAFTAR TABEL

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.

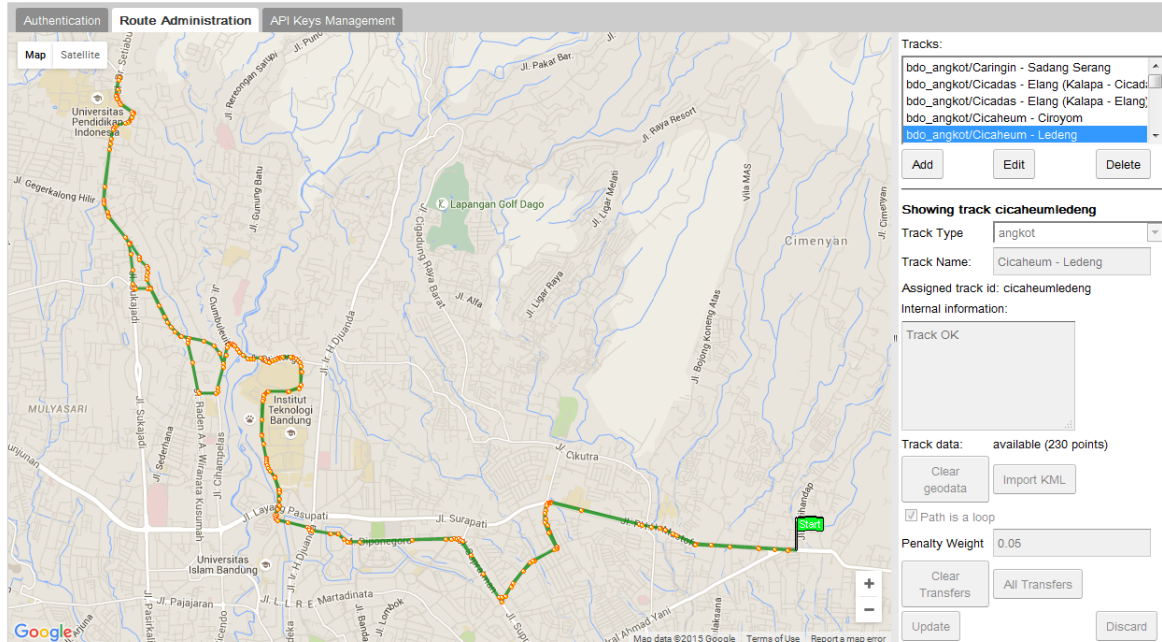


Gambar 1.1: Situs web KIRI[1]

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 penulisan 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:

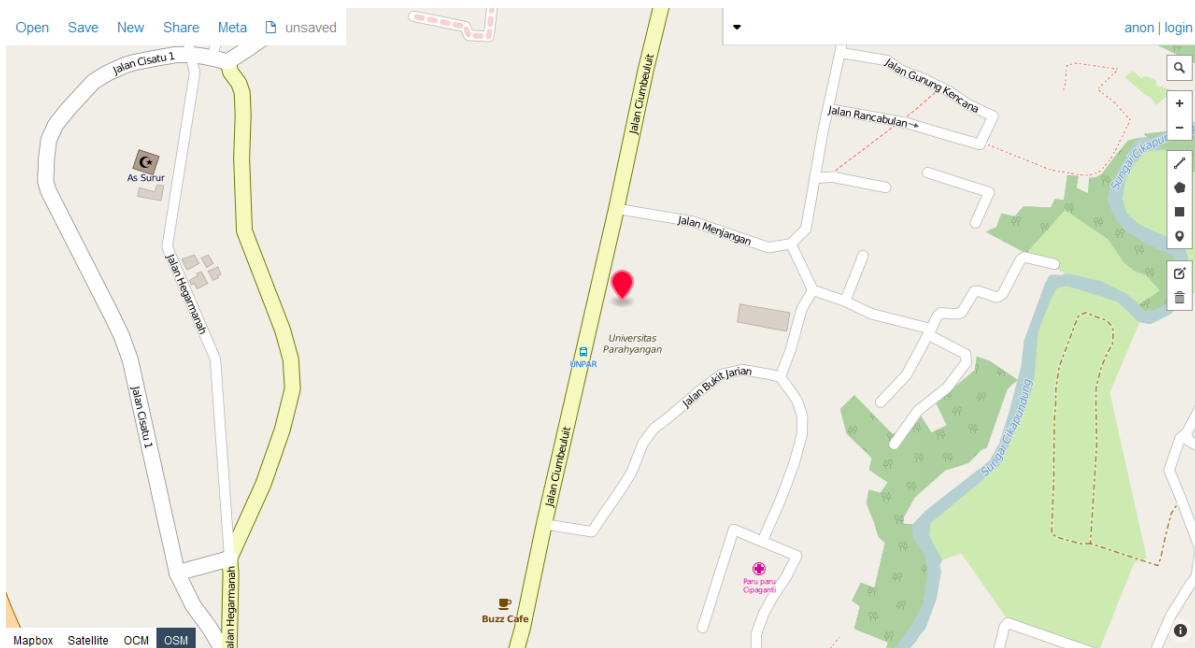
1. Tipe data yang hanya dapat menyimpan sebuah nilai geometri saja, yaitu:
 - ***Geometry***
 - ***Point***
 - ***LineString***
 - ***Polygon***

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

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

2.1.1 *Point*

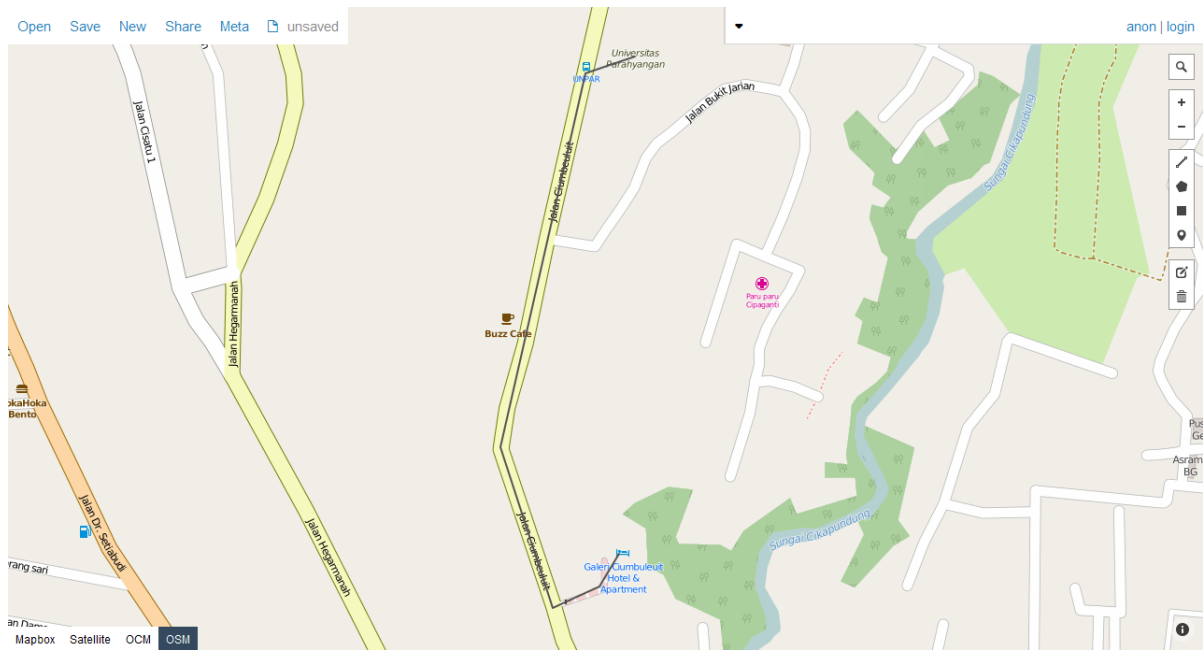
Point adalah nilai geometri yang merepresentasikan sebuah lokasi ke dalam suatu koordinat[6]. Koordinat pada *Point* terdiri dari nilai X dan Y dimana X merepresentasikan letak lokasi terhadap garis bujur dan Y merepresentasikan letak lokasi terhadap garis lintang. *Point* tidak memiliki dimensi maupun 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[8]. 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*[8]. 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[8]. 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*[8] (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.

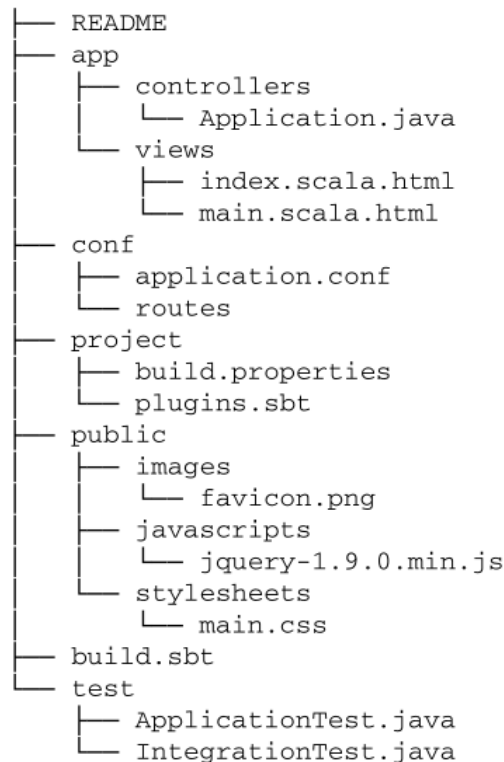
2.3 Play Framework

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 mengenai 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 mengenai *file* “application.conf” yang menyimpan pengaturan-pengaturan seperti kumpulan *log*, koneksi ke *database*, jenis *port* tempat *server* bekerja, dll. *Folder* “conf” juga berisi *file* “routes” yang mengatur bagaimana HTTP *requests* nantinya akan diproses lebih lanjut 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”[9]. 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*:


```

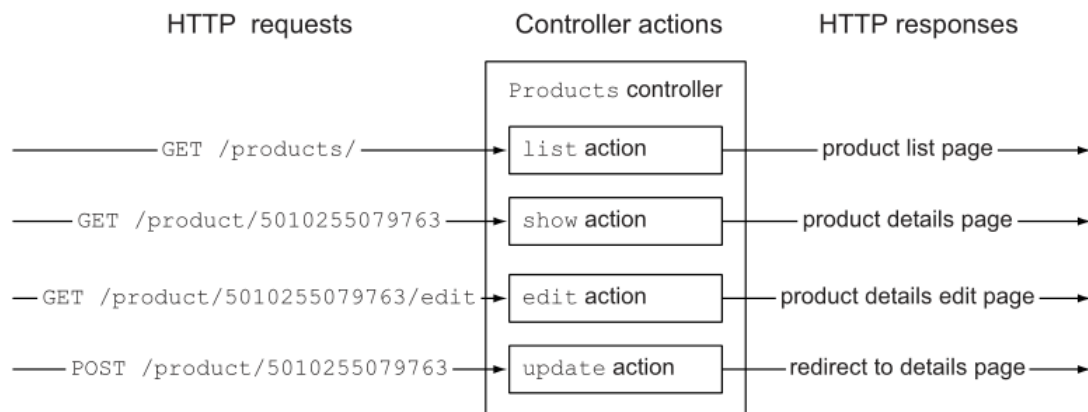
1 | @(name: String)
2 | <!doctype html>
3 | <html>
4 |   <head>
5 |     <meta charset="UTF-8">
6 |     <title>Hello</title>
7 |   </head>
8 |   <body>
9 |     <h1>Hello <em>@name</em></h1>
10 |   </body>
11 | </html>

```

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[9]. 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).

BAB 3

ANALISIS

3.1 Analisis Sistem Kini

3.2 Analisis *Database* Sistem Kini

3.3 Analisis teori-teori untuk membangun KIRI *Dashboard Server Side* dalam bahasa Java dengan menggunakan Play Framework

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, “Trail: JDBC(TM) Database Access.” <https://docs.oracle.com/javase/tutorial/jdbc/>, 2015. [Online; diakses 5-November-2015].
- [8] Oracle, “Package java.sql.” <http://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>, 2015. [Online; diakses 6-November-2015].
- [9] Play Framework, “Play 2.4.x documentation.” <https://www.playframework.com/documentation/2.4.x/Home>, 2015. [Online; diakses 4-November-2015].

LAMPIRAN A

THE SOURCE CODE

Listing A.1: MyFurSet.java

```
1
2 import java.util.ArrayList;
3 import java.util.Collections;
4 import java.util.HashSet;
5
6 /**
7  *
8  * @author Lionov
9  */
10
11 //class for set of vertices close to furthest edge
12 public class MyFurSet {
13     protected int id; //id of the set
14     protected MyEdge FurthestEdge; //the furthest edge
15     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
16     protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each
17         trajectory
18     protected ArrayList<Integer> closeID; //store the ID of all vertices
19     protected ArrayList<Double> closeDist; //store the distance of all vertices
20     protected int totaltrj; //total trajectories in the set
21
22     /**
23      * Constructor
24      * @param id : id of the set
25      * @param totaltrj : total number of trajectories in the set
26      * @param FurthestEdge : the furthest edge
27      */
28     public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
29         this.id = id;
30         this.totaltrj = totaltrj;
31         this.FurthestEdge = FurthestEdge;
32         set = new HashSet<MyVertex>();
33         ordered = new ArrayList<ArrayList<Integer>>();
34         for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
35         closeID = new ArrayList<Integer>(totaltrj);
36         closeDist = new ArrayList<Double>(totaltrj);
37         for (int i = 0; i < totaltrj; i++) {
38             closeID.add(-1);
39             closeDist.add(Double.MAX_VALUE);
40         }
41     }
42
43     /**
44      * set a vertex into the set
45      * @param v : vertex to be added to the set
46      */
47     public void add(MyVertex v) {
48         set.add(v);
49     }
50
51     /**
52      * check whether vertex v is a member of the set
53      * @param v : vertex to be checked
54      * @return true if v is a member of the set, false otherwise
55      */
56     public boolean contains(MyVertex v) {
57         return this.set.contains(v);
58     }
59 }
```