SKRIPSI

ANALISI WAKTU TEMPUH KOTA BANDUNG

FRASETIAWAN HIDAYAT

NPM: 2010730121

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

UNDERGRADUATE THESIS

ANALYSIS OF TRAVEL TIME BANDUNG CITY

FRASETIAWAN HIDAYAT

NPM: 2010730121

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

ABSTRAK

Dalam melakukan suatu perjalanan , manusia melalui suatu jalur yang relatif konstan dimana jalur tersebut akan menjadi rutinitas yang akan dilalui. Dari jalur tersebut sering kali terjadi kemacetan dan biasanya kemacetan itu terjadi pada jam-jam tertentu.

Pada kota-kota besar sering terjadi kemacetan. Efeknya adalah keterlambatan yang akan mempengaruhi seluruh rangkaian kegiatan yang telah direncanakan. Bandung adalah salah satunya dari kota besar yang sering mengalami kemacetan ini dan terkadang kemacetan sendiri tidak dapat diprediksi.Kemacetan ini sendiri bisa dianalisis dengan menentukan pada pukul berapa sajakah terjadi kemacetan pada jalur yang ditempuh.

Dengan memanfaatkan Google Direction yang dimana Google Direction itu sendiri adalah suatu layanan web untuk menghitung arah antar lokasi. Dengan layanan web ini bisa mendapatkan data waktu tempuh dari lokasi awal sampai lokasi tujuan dengan input berupa URL beserta dengan parameter wajib dan beberapa parameter opsional yang akan menghasilkan output JSON atau XML.

Aplikasi sederhana yang akan dibangun berbasis Java memanfaatkan jsoup bisa melakukan *request* ke layanan Google Direction.Pengujian dari aplikasi sederhana ini dilakukan dengan menggunakan *test case*. Berdasarkan hasil pengujian, aplikasi dapat berjalan dengan baik dan memberikan keluaran file .csv yang akan dianalisis untuk memberikan waktu terbaik dalam melakukan perjalanan dengan bantuan aplikasi Microsoft Excel. Hasil pengujian aplikasi sederhana ini membuktikan bahwa Google Direction API dapat dimanfaatkan untuk menganalisis waktu tempuh antar titik agar mendapatkan waktu tempuh yang optimal.

Kata-kata kunci: Kemacetan, Kota Bandung, Google Direction, JSON, Java, jsoup, Microsoft Excel.

ABSTRACT

In doing a travel, man through a relatively constant path where the path will be a routine to be traversed. From this path there is often a traffic jam and usually the jam occurs at certain hours.

In big cities there are frequent congestion. The effect is the delay that will affect the entire set of planned activities. Bandung is one of the big cities that often experience this bottleneck and sometimes the congestion itself can not be predicted. The congestion itself can be analyzed by determining at what time there are congestion on the path taken.

By using Google Direction which Google Direction itself is a web service to calculate the direction between locations. With this web service can get data travel time from the initial location to the destination location with input in the form of URL along with mandatory parameters and some optional parameters that will produce JSON or XML output.

Simple applications that will be built based on Java jsoup can make request to service Google Direction.Testing of this simple application is done by using a test case. Based on the test results, the application can run well and provide output .csv files to be analyzed to provide the best time to travel with the help of Microsoft Excel applications. The results of testing this simple application proves that Google Direction API can be utilized to analyze the travel time between points in order to get the travel time that optimalukan a journey, humans through a relatively constant path where the path will be a routine to be traversed. From this path there is often a traffic jam and usually the jam occurs at certain hours.

In big cities there are frequent congestion. The effect is the delay that will affect the entire set of planned activities. Bandung is one of the big cities that often experience this bottleneck and sometimes the congestion itself can not be predicted. The congestion itself can be analyzed by determining at what time there are congestion on the path taken.

By using Google Direction which Google Direction itself is a web service to calculate the direction between locations. With this web service can get data travel time from the initial location to the destination location with input in the form of URL along with mandatory parameters and some optional parameters that will produce JSON or XML output.

Simple applications that will be built based on Java jsoup can make request to service Google Direction.Testing of this simple application is done by using a test case. Based on the test results, the application can run well and provide output .csv files to be analyzed to provide the best time to travel with the help of Microsoft Excel applications. The results of testing this simple application proves that Google Direction API can be

used to analyze the travel time between points in order to get optimal travel time

Keywords: Congestion, Bandung City, Google Direction, JSON, Java, jsoup, Microsoft Excel.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Dalam melakukan kegiatan dan rutinitas, manusia akan melakukan perpindahan tempat dari suatu tempat ke tempat lain. Salah satu contohnya adalah melakukan kegiatan perkuliahan. Dalam melakukan kegiatan tersebut, mahasiswa harus berpindah dari rumah ke tempat perkuliahan diselengarakan. Dalam melakukan suatu perpindahan itu, kita melalui suatu jalur yang relatif konstan dimana jalur tersebut akan menjadi rutinitas yang akan dilalui. Dari jalur tersebut sering kali terjadi kemacetan dan biasanya kemacetan itu terjadi pada jam-jam tertentu.

Pada kota-kota besar sering terjadi kemacetan. Efeknya adalah keterlambatan yang akan mempengaruhi seluruh rangkaian kegiatan yang telah direncanakan. Bandung adalah salah satunya dari kota besar yang sering mengalami kemacetan ini dan terkadang kemacetan sendiri tidak dapat diprediksi.

Dengan demikian, untuk merencanakan segalanya agar berjalan sesuai dengan rencana, perlu untuk mengetahui waktu tempuh yang paling cepat dari jalur yang relatif konstan agar tidak terjebak dalam kemacetan. Kemacetan ini sendiri bisa dianalisis dengan menentukan pada pukul berapa sajakah terjadi kemacetan pada jalur yang ditempuh.

Salah satu teknologi yang telah ada, *Google Direction* adalah suatu layanan web untuk menghitung arah antar lokasi. Layanan web ini didesain menghitung arah alamat statis untuk penempatan konten aplikasi pada peta (*Google Maps*). Dengan layanan web ini juga kita bisa mendapatkan data waktu tempuh dari lokasi awal sampai lokasi tujuan dengan input berupa URL beserta dengan parameter wajib dan beberapa parameter opsional yang bisa disesuaikan dengan kebutuhan seperti waktu keberangkatan dan model lalu lintas apakah optimis atau pesimis yang akan mempengaruhi waktu tempuh. Pesimis adalah model lalu lintas dengan memperhitungkan kemacetan dan optimis adalah model lalu lintas yang tidak memperhitungkan kemacetan.*Google Direction* ini sendiri memiliki output berupa JSON atau XML.

Layanan web sendiri adalah setiap layanan yang tersedia melalui internet. Layanan web ini sendiri menggunakan suatu format sistem pesan yang terstandarisasi yang bisa diakses oleh aplikasi lain. Layanan web ini juga tidak terikat pada satu sistem operasi atau bahasa pemrograman agar bisa diakses oleh aplikasi lain. contoh format dari layanan web adalah JSON dan XML.

Google Direction sendiri menggunakan protokol HTTP untuk bisa saling berkomunikasi dengan aplikasi. Protokol HTTP merupakan protokol yang berjalan diatas protokol TCP

pada port 80 yang digunakan untuk mengirim dokumen atau halaman. Pesan protokol http diformat untuk dapat ditampilkan pada aplikasi.

Dalam penelitian ini, akan dibuat sebuah perangkat lunak yang dapat menampilkan hasil analisis dari data yang didapatkan dari Google Direction API. tujuan aplikasi ini adalah untuk membantu mengambil keputusan pada jam berapakah harus melakukan perjalanan dengan waktu tempuh yang tercepat dengan data-data yang telah ada dalam kurun waktu 7 hari. Aplikasi ini memanfaatkan layanan dari *Google* yaitu *Google Direction* untuk mendapatkan data-data waktu tempuh dari suatu jalur. Pada penelitian ini menggunakan 2 sampel yaitu : menghitung waktu tempuh dari Universitas Katolik Parahyangan dengan alamat Jln. Ciumbuleuit No.94 dan Komplek Amaya Residence, menghitung waktu tempuh dari Universitas Katolik Parahyangan dengan alamat Jln. Ciumbuleuit No.94 dan Komplek Taman Puspa Indah.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah dijelaskan, rumusan masalah pada penelitian ini adalah:

- Bagaimana cara menggunakan Google Direction API dalam bahasa Java?
- Bagaimana memanfaatkan layanan Google Direction API untuk memberikan kesimpulan waktu perjalanan terbaik?
- Kapan waktu terbaik untuk berangkat/pulang untuk dua sampel tempat yang dimaksud?

1.3 Tujuan

Berdasarkan rumusan masalah di atas, maka tujuan dari penelitian ini adalah:

- memahami cara menggunakan Google Direction API.
- memahami layanan Google Direction API untuk memberikan kesimpulan waktu perjalanan terbaik.
- memutuskan kapan waktu terbaik untuk berangkat/pulang untuk dua sampel yang dimaksud.

1.4 Batasan Masalah

Batasan masalah yang akan digunakan untuk peneliatian ini adalah:

1. Output dari permintaan komunikasi menggunakan format JSON.
2. Cakupan wilayah yang akan dihitung waktu tempuhnya adalah kota Bandung.
3. Waktu tempuh dihitung setiap jam dalam satu hari.
4. Waktu tempuh dihitung setiap hari dalam seminggu.

5. Menghitung Waktu tempuh dengan sampel yang beralamat Jln. Ciumbuleuit No.94, Komplek Amaya Residence dan Komplek Taman Puspa Indah.

6. Program dijalankan selalu dari hari Senin.

1.5 Metodologi

Dalam penyusunan skripsi ini mengikuti langkah-langkah metodologi penelitian sebagai berikut :

1. Melakukan studi pustaka untuk dijadikan referensi dalam melakukan pembangunan aplikasi Analisis waktu tempuh kota Bandung,

2. Melakukan analisis *Google Direction* untuk mendapatkan hasil waktu tempuh dari tujuan asal ke tujuan akhir,

3. Melakukan perancangan perangkat lunak,

4. Melakukan uji coba sesuai dengan sampel,

5. Melakukan penarikan kesimpulan dan saran pada hasil analisis tersebut.

1.6 Sistematika Pembahasan

Sistematika penulisan laporan pada skripsi ini adalah sebagai berikut :

1. Bab Pendahuluan
   Bab 1 berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan dalam pelaksanaan penelitian ini.

2. Bab Dasar Teori
   Bab 2 berisi tentang definisi-definisi dasar teori tentang *Google direction* beserta teori pendukung lainnya.

3. Bab Analisis
   Bab 3 berisi analisis *Google Direction*, analisis teori pendukung lainnya dan analisis perangkat lunak.

4. Bab Perancangan
   Bab 4 berisi tentang pembahasan menegenai perancangan perangkat lunak.

5. Bab Impelemntasi dan Pengujian
   Bab 5 berisi tentang pengimplementasian perangkat lunak.

6. Bab Kesimpulan dan Saran
   Bab 6 berisi penarikan kesimpulan selama menyelesaikan skripsi dan saran yang diusulkan untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

Pada bab ini akan diuraikan teori-teori yang akan digunakan untuk pembangunan aplikasi ke analisis kota Bandung. Teori-teori tersebut adalah tentang protokol HTTP, *library* Jsoup meliputi kelas jsoup dan Connection. Selain itu akan dibahas juga mengenai *Google Direction API*, *JavaScript Object Notation (JSON)* meliputi kelas pada *library* JSON : JSONObject.

2.1 Protokol HTTP

HTTP adalah protokol di balik World Wide Web. Dengan setiap transaksi web, HTTP dipanggil. HTTP adalah di balik setiap permintaan dokumen web atau grafis, setiap klik link hypertext, dan setiap penyerahan formulir. Web adalah tentang penyebaran informasi melalui Internet, dan HTTP adalah protokol yang digunakan untuk melakukannya.

2.1.1 Transaksi HTTP

Berikut akan diilustrasikan transaksi web umum, menunjukkan HTTP yang dipertukarkan antara program *client* dan *program* server. [1]:

- berikut diberikan sebuah url : http://hypothetical.ora.com:80/.
- Browser akan mengintepretasikan URL tersebut sebagai berikut :
  - *http* : // : menggunakan protokol HTTP.
  - *hypothetical.ora.com* : menghubungi komputer melalui jaringan dengan hostname hypothetical.ora.com.
  - : 80 : Terhubung ke komputer di port 80. Nomor port IP nomor dari 1 sampai 65535. Jika titik dua dan nomor port dihilangkan, nomor port diasumsikan nomor port *default* HTTP, yang merupakan 80.
  - : Apapun setelah nama host dan nomor port opsional dianggap sebagai jalan dokumen. Dalam ilustrasi ini, jalan dokumen adalah .
- Pada ilustrasi ini browser menghubungkan ke hypothetical.ora.com pada port 80 menggunakan protokol HTTP. Pesan bahwa browser mengirimkan ke server adalah sebagai berikut:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/
    jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
    5.01; Windows NT)
Host: hypothetical.ora.com
Connection: Keep-Alive
```

Gambar 2.1: HTTP Request[1]

- Pada baris pertama pada request (Gambar 2.1) disebut dengan request line dan diawali dengan *request method*(metode permintaan), dalam gambar tersebut adalah GET. *Request method* diikuti dengan *resource* yang diinginkan, dalam gambar tersebut adalah /. *Request line* diakhiri dengan versi protokol yang digunakan dalam gambar diatas adalah HTTP/1.1.

- baris kedua dan baris-baris berikutnya sampai ditemukan baris kosong, berisi request headers dalam format *nama-header:nilai-header*. pada gambar 2.1 terdapat header host yang menandakan bahwa browser ingin mengakses situs dari nilai yang ada di header host.

- Dibawah header-header pada gambar 2.1 terdapat baris kosong di akhir *request*. pada *request*, baris kosong memisahkan antara *request headers* dengan *request body*(tubuh permintaan).

Setelah *client* memberikan *request* server memberikan *response*. Dari kasus diatas berikut adalah sebagai berikut :

```
HTTP/1.1 200 OK
Date: Mon, 06 Dec 1999 20:54:26 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 04 Oct 1996 14:06:11 GMT
ETag: "2f5cd-964-381e1bd6"
Accept-Ranges: bytes
Content-length: 327
Connection: close
Content-type: text/html

<title>Sample Homepage</title>
<img src="/images/oreilly_mast.gif">
<h1>Welcome</h1>
Hi there, this is a simple web page.  Granted,
it may not be as elegant as some other web
pages you've seen on the net, but there are
some common qualities:

<ul>
  <li> An image,
  <li> Text,
  <li> and a <a href="/example2.html"> hyperlink. </a>
</ul>
```

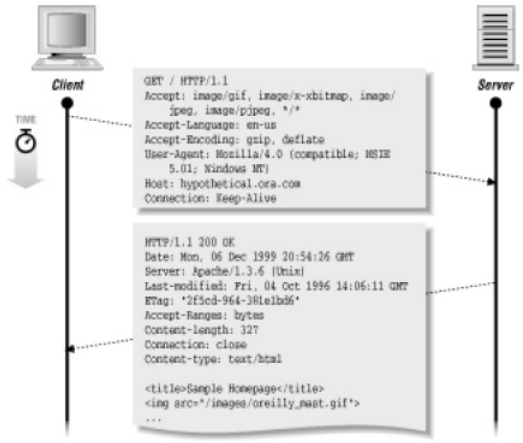
Gambar 2.2: HTTP Respond[1]

- Pada baris pertama pada respon (Gambar 2.2) disebut *status line*, dan diawali dengan versi protokol yang digunakan, dalam kasus ini HTTP/1.1. *Status line* diikuti dengan

3 dijit kode status, dalam kasus ini 200. *Status line* diakhiri dengan representasi tekstual dari status tersebut dalam kasus ini *OK*.

- Baris kedua dan baris-baris berikutnya sampai ditemukan baris kosong, berisi request headers dalam format *nama-header:nilai-header*. pada gambar 2.2 terdapat header server yang menandakan bahwa server yang digunakan untuk melayani request.
- Setelah baris kosong adaah *body* dari *response*, gambar 2.2 berupa teks HTML.
- Pada gambar 2.2 ada kebutuhan akan *file* oreilly_mast.gif di HTML ini. *File* tersebut akan diunduh secara terpisah, tetapi juga dengan protokol HTTP.

Setelah semua terjadi dan dibaca dengan baik, maka baris kosong dan teks dokumen muncul. dengan demikian transaksi yang terjadi adalah sebagai berikut :



Gambar 2.3: Transaksi Sederhana[1]

2.1.2 Kode Status

Kode status adalah bilangan bulat tiga dijit yang menyatakan status dari pemrosesan permintaan yang dikirimkan. Berikut adalah beberapa kode status yang umum ditemui :

Kode Status	Status	Deskripsi
200	OK	Request berhasil diproses dengan baik.
301	Moved Permanently	Resource yang diminta sudah berpindah ke URI yang lain secara permanen.
302	Found	Resource yang diminta untuk sementara bepindah pada URL yang lain. Untuk alasan historis, client diperkenankan untuk mengubah metode permintaan dan POST menjadi GET.
307	Temporary Redirect	Resource yang diminta untuk sementara berpindah pada URL yang lain. Mirip dengan status 302 namun client tidak diperkenankan mengubah metode permintaan dari POST menjadi GET.
400	Bad Request	Server tidak dapat memproses permintaan karena ada kesalahan adri client
401	Unauthorized	Server tidak dapat memproses permintaan karena kredensial diperlukan dan client tidak menyediakannya.
404	Not Found	Resource yang diminta tidak tersedia pada server.
500	Interna Server Error	Server mengalami masalah internal, sehingga tidak dapat memprises permintaan yang dikirimkan.
501	Not Implemented	Server belum atau tidak mendukung fungsionalitas yang diminta oleh client.
503	Service Unavailable	Server tidak dapat menjawab permintaan client, karena terlalu sibuk atau perawatan. Status ini mengindikasikan client dapat mencoba lagi setelah jangka waktu tertentu.

Tabel 2.1: Tabel Kode Status

kode status yang tersedia dikelompokan menjadi lima, diindikasikan oleh dijit pertama dari kode tersebut:

- 1xx(informational): Request diterima, dan proses dilanjutkan.
- 2xx(Successfull): Request diterima, dan dimengertian dengan baik.
- 3xx(Redirection): Aksi tambahan diperlukan untuk menyelesaikan permintaan.

- 4xx(Client Error): Terjadi kesalahan dan client harus memperbaikinya
- 5xx(Server Error): Terjadi kesalahan pada sisi server.

2.1.3 *Request method*

Request method menentukan karakteristik dari permintaan yang dikirimkan. Ada 2 *method* yang sudah dikenal umum yaitu GET dan POST. Selain kedua method tersebut, ada beberapa *method-method* lain yang dapat juga digunakan pada protokol HTTP seperti dijelaskan pada tabel berikut:

Metode	**Deskripsi**
GET	Metode yang paling umum digunakan, dan digunakan untuk mendapatkan konten dari resource yang ditentukan pada request.
POST	Metode ini digunakan untuk meminta server memproses data yang dikirimkan. Pada umumnya, metode POST diikuti dengan request body, yang berisi parameter-parameter yang dikirimkan
HEAD	Metode HEAD mirip dengan metode GET, tetapi bedanya di sini server tidak mengembalikan konten body, melaikan hanya sampai response headers saja.
PUT	Metode ini digunakan untuk membuat atau menggantikan resource yang ditentukan pada request.
DELETE	Metode ini digunakan untuk menghapus resource dari server.

Tabel 2.2: Tabel Request Method

2.1.4 *Response Headers*

Response Headers digunakan untuk meberikan informasi-informasi tambahan pada sebuah jawaban. Sama seperti *request header*, setiap header terdiri dari nama dan nilai, dan terpisah oleh titik dua dan spasi(:). Tabel berikut menjelaskan beberapa header yang umum dipakai:

Header	Deskripsi
Content-Type	Header ini menunjukan tipe media dari konten yang akan diberikan. Pada bentuk sederhana, nilai dari header ini berisi dari kode tipe MIME(Multipurpose Internet Mail Extension). Beberapa kode tipe MIME yang umum antara lain: text/plain untuk teks, text/html untuk halaman HTML; image/gif, image/jpg, image/png untuk gambar berformat GIF, JPEG, PNG; dan application/json untuk data JSON.
Cache-control	Header ini mengatur bagaimana konten yang dikirimkan dapat dikirimkan sementara di client. Pada konten-konten statis seperti gambar, secara default konten akan disimpan pada client dalam jangka waktu tertentu, sehingga jika dibutuhkan dalam waktu dekat di masa depan, tidak perlu mengirimkan permintaan lagi ke server. jika secara eksplisit diinginkan konten diminta lagi setiap kali diperlukan, dapat mengisi header ini dengan nilai no-cache.
Location	Header ini digunakan untuk beberapa jenis jawaban untuk menunjukan lokasi sumberdaya dalam bentuk URI. Pada jawaban dengan kode 3xx, nilai dari header ini menunjukan lokasi baru yang harus dituju.

Tabel 2.3: Tabel Response Headers

2.2 *Library* jsoup

Jsoup adalah sebuah *library* java untuk bekerja dengan HTML dunia nyata. Jsoup menyediakan API yang sangat nyaman untuk mengekstrak dan memanipulasi data, menggunakan DOM(Document Object Model) terbaik, CSS, dan *method* yang mirip dengan jquery. Jsoup mengimplementasikan spesifikasi standar *WHATWG HTML5* dan mengurai HTML menjadi DOM(Document Object Model) yang sama dengan peramban modern lakukan. Jsoup sendiri dirancang untuk menangani semua jenis HTML yang biasa ditemukan dengan membuat *parsing tree* yang dapat dimengerti.

Dalam subbab berikut akan dijelaskan fungsi dan beberapa kelas dari jsoup[2].

2.2.1 Fungsi jsoup

berikut adalah fungsi dari jsoup :

- menghimpun dan mengurai HTML dari URL, file, atau *string*.
- mencari dan mengambil data, menggunakan *DOM traversal* atau *CSS selectors*.
- memanipulasi elemen HTML, atribut, dan teks.
- membersihkan konten yang dikirimkan pengguna terhadap daftar putih yang aman, untuk mencegah serangan XSS.
- memberi *output* HTML yang rapi.

2.2.2 Kelas- kelas jsoup

Jsoup

Kelas ini merupakan inti untuk mengakses fungsi jsoup. Seluruh method dalam kelas ini merupakan *static method* sehingga kelas ini tidak perlu dikonstruksi. Salah satu method yang dimiliki kelas ini adalah sebagai berikut :

- **public static Connection connect(String url)**

  Berfungsi untuk membuat koneksi baru dengan suatu situs web.

  Parameter:

  - **url**: URL situs web dengan protokol HTTP.

  Kembalian: koneksi dengan situs web.

Connection

Kelas ini merupakan interface yang menyediakan pengambilan data dari situs web. Beberapa method yang dimiliki kelas ini adalah sebagai berikut:

- **Connection data(String key, String value)**

  Berfungsi untuk menambahkan parameter data yang bisa dikirim melalui metode HTTP GET atau POST.

  Parameter:

  - **key**: kunci data.
  - **value**: nilai data.

  Kembalian: koneksi yang sama tetapi sudah diubah.

- **Connection ignoreContentType(boolean ignoreContentType)**

  Berfungsi untuk Mengabaikan tipe konten dokumen saat *parsing* respon.

  Parameter:

  - **ignoreContentType**: set true jika ingin jenis konten diabaikan pada *parsing* respon dalam dokumen.

  Kembalian: koneksi pada situs web.

- **Connection.Response execute() throws IOException**

  Berfungsi untuk mengeksekusi **request** dari **Connection**.

  Kembalian: objek respon.

- **String body()**

  Berfungsi untuk mendapatkan *body* respon sebagai string biasa.

  Kembalian: *string* dari *body*.

2.3 Google Direction

Google Maps Directions adalah layanan yang menghitung arah antar lokasi menggunakan permintaan HTTP. Anda bisa mencari arah untuk beberapa moda transportasi, termasuk angkutan umum, mengemudi, berjalan atau bersepeda. Arah bisa menetapkan tempat asal, tujuan dan titik jalan baik sebagai string teks atau sebagai koordinat garis lintang/garis bujur. Layanan ini didesain untuk menghitung arah alamat statis (sudah diketahui sebelumnya) untuk penempatan konten aplikasi pada peta.

2.3.1 Permintaan Arah

Permintaan Google Maps Directions mengambil bentuk berikut:

```
https://maps.googleapis.com/maps/api/directions/json?parameters
```

Listing 2.1: *Request* Google Directions

HTTP disarankan untuk aplikasi yang berisi data pengguna sensitif, seperti lokasi pengguna, dalam permintaan. URL Google Maps Directions API dibatasi sekitar 2000 karakter, setelah Pengkodean URL. Karena sebagian URL Google Maps Directions API bisa melibatkan banyak lokasi sepanjang lintasan. Pada subbab berikutnya akan dijelaskan parameter apa saja yang digunakan pada permintaan ke layanan ini.

2.3.2 Parameter Permintaan

Beberapa parameter tertentu diperlukan sementara yang lainnya bersifat opsional. Sebagaimana standar dalam URL, semua parameter dipisah menggunakan karakter ampersand (&). Daftar parameter dan kemungkinan nilainya disebutkan di bawah ini[3].

Parameter yang diperlukan

- **origin** adalah alamat, nilai garis lintang/garis bujur tekstual, atau ID tempat asal yang ingin Anda hitung arahnya. ketentuan dari alamat dari origin adalah sebagai berikut :

  - Jika Anda meneruskan sebuah alamat sebagai string, layanan Directions akan melakukan geocode atas string itu dan mengubahnya menjadi koordinat garis lintang/garis bujur untuk menghitung arah. Koordinat ini mungkin berbeda dengan yang dikembalikan oleh Google Maps Geocoding API, misalnya pintu masuk bangunan dan bukan pusatnya.

- Jika Anda meneruskan koordinat, itu akan digunakan tanpa diubah untuk menghitung arah. Pastikan tidak ada spasi di antara nilai garis lintang dan garis bujur.
- ID Tempat harus diawali dengan **place_id:**. ID tempat hanya bisa ditetapkan jika permintaan menyertakan kunci API atau ID klien Google Maps API for Work. Anda bisa mendapatkan ID tempat dari Google Maps Geocoding API dan Google Places API (termasuk Place Autocomplete).

- **destination** adalah alamat, nilai garis lintang/garis bujur tekstual, atau ID tempat tujuan yang ingin Anda hitung arahnya. Opsi untuk parameter destination sama dengan opsi untuk parameter origin yang dijelaskan di atas.

- **key** adalah kunci API aplikasi Anda. Kunci ini mengidentifikasi aplikasi Anda untuk keperluan manajemen kuota.

Parameter yang opsional

- **mode** (default-nya adalah driving) adalah menetapkan moda transportasi yang akan digunakan saat menghitung arah.

- **waypoint** adalah menetapkan larik titik jalan. Titik jalan mengubah rute dengan mengarahkannya melalui lokasi yang ditetapkan. Titik jalan ditetapkan berupa koordinat garis lintang/garis bujur, ID tempat, atau alamat yang akan di-geocode. ID Tempat harus diawali dengan **place_id:**. ID tempat hanya bisa ditetapkan jika permintaan menyertakan kunci API atau ID klien Google Maps API for Work. Titik jalan hanya didukung untuk arah mengemudi, berjalan dan bersepeda.

- **alternative** adalah jika diatur ke true, menetapkan bahwa layanan Directions mungkin menyediakan lebih dari satu rute alternatif dalam respons. Perhatikan, memberikan alternatif rute bisa meningkatkan waktu respons dari server.

- **avoid** adalah menunjukkan rute yang dihitung harus menghindari fitur yang ditandai. Parameter ini mendukung argumen berikut:
  - **tolls** menunjukkan rute yang dihitung harus menghindari jalan/jembatan tol.
  - **highways** menunjukkan rute yang dihitung harus menghindari jalan raya.
  - **ferries** menunjukkan rute yang dihitung harus menghindari penyeberangan feri.
  - **indoor** menunjukkan rute yang dihitung harus menghindari tangga dalam ruangan untuk arah berjalan dan arah angkutan umum. Hanya permintaan yang menyertakan kunci API atau ID klien Google Maps API for Work yang akan menerima tangga dalam ruangan secara default.

- **language** adalah menetapkan bahasa yang digunakan untuk mengembalikan hasil.

- **unit** adalah menetapkan sistem satuan yang akan digunakan saat menampilkan hasil.

- **region** adalah menetapkan kode wilayah, ditetapkan sebagai nilai yang berisi dua karakter ccTLD ("top-level domain").

- **arrival_time** adalah menetapkan waktu kedatangan yang diinginkan untuk arah angkutan umum, dalam detik sejak tengah malam, 1 Januari 1970 UTC. Anda bisa menetapkan **departure_time** atau **arrival_time**, namun tidak boleh duanya.

- **departure_time** adalah menetapkan waktu keberangkatan yang diinginkan. Anda bisa menetapkan waktu berupa integer dalam detik sejak tengah malam 1 Januari 1970 UTC. Atau, Anda bisa menetapkan nilai now, yang mengatur waktu keberangkatan ke waktu saat ini (dikoreksi ke detik terdekat).

- **traffic_model** (default-nya adalah **best_guess**) adalah menetapkan asumsi yang akan digunakan saat menghitung waktu dalam lalu lintas. Pengaturan ini memengaruhi nilai yang dikembalikan di bidang **duration_in_traffic** dalam respons, yang berisi prediksi waktu dalam lalu lintas berdasarkan rata-rata historis. Parameter **traffic_model** hanya bisa ditetapkan untuk arah mengemudi yang permintaannya menyertakan departure_time, dan hanya jika permintaan menyertakan kunci API atau ID klien Google Maps API for Work.Nilai yang tersedia untuk parameter ini adalah:

  - **best_guess** (default) menunjukkan **duration_in_traffic** yang dikembalikan harus berupa perkiraan waktu tempuh terbaik berdasarkan informasi riwayat kondisi lalu lintas dan lalu lintas saat ini. Lalu lintas saat ini menjadi kian penting bila **departure_time** semakin dekat ke waktu sekarang.
  - **pessimistic** menunjukkan **duration_in_traffic** yang dikembalikan lebih lama dari waktu tempuh sesungguhnya di hari-hari biasa, meskipun hari-hari tertentu dengan kondisi lalu lintas yang buruk mungkin melebihi nilai ini.
  - **optimistic** menunjukkan **duration_in_traffic** yang dikembalikan harus lebih singkat dari waktu tempuh sesungguhnya di hari biasa, meskipun hari-hari tertentu dengan kondisi lalu lintas yang baik bisa lebih cepat dari nilai ini.

- **transit_mode** adalah menetapkan satu atau beberapa mode angkutan umum yang disukai. Parameter ini hanya bisa ditetapkan untuk arah angkutan umum, dan hanya jika permintaan menyertakan kunci API atau ID klien Google Maps API for Work. Parameter ini mendukung argumen berikut:

  - **bus** menunjukkan rute yang sudah dihitung akan mengutamakan perjalanan dengan bus.
  - **subway** menunjukkan rute yang sudah dihitung akan mengutamakan perjalanan dengan kereta bawah tanah.
  - **train** menunjukkan rute yang sudah dihitung akan mengutamakan perjalanan dengan kereta api.
  - **tram** menunjukkan rute yang sudah dihitung akan mengutamakan perjalanan dengan trem dan kereta ringan.
  - **rail** menunjukkan rute yang sudah dihitung akan mengutamakan perjalanan dengan kereta api, trem, kereta ringan, dan kereta bawah tanah. Ini sama dengan **transit_mode=train|tram|subway**.

- **transit_routing_preference** adalah menetapkan preferensi untuk rute angkutan umum. Dengan parameter ini, Anda bisa mencondongkan opsi yang dikembalikan, bukannya menerima rute default terbaik yang dipilih oleh API. Parameter ini hanya bisa ditetapkan untuk arah angkutan umum, dan hanya jika permintaan menyertakan kunci API atau ID klien Google Maps API for Work. Parameter ini mendukung argumen berikut:
  - **less_walking** menunjukkan rute yang sudah dihitung akan mengutamakan jumlah berjalan kaki yang terbatas.
  - **fewer_transfers** menunjukkan rute yang sudah dihitung akan mengutamakan jumlah ganti angkutan yang terbatas.

2.3.3 *Response* Arah

Response Arah dikembalikan dalam format yang ditunjukkan oleh flag output dalam jalur permintaan URL. Hasil *response* yang dikeluarkan adalah jalur yang dilalui menggunakan format JSON yang terdapat elemen-elemen yang menjelaskan jalur yang dilewati. Pada subbab berikutnya akan dijelaskan elemen-elemen yang ada pada *output* yang dihasilkan dari permintaan arah.

2.3.4 Elemen *Response* Arah

Berikut adalah penjelasan dari setiap elemen *output* yang dihasilkan dari permintaan arah :

- **Status** adalah status *response* dari permintaan yang dikirimkan, isinya dapat berupa salah satu dari berikut ini :
  - **OK** jika permintaan berhasil, dan permintaan akan mengandung informasi tambahan terkait hasil pencarian.
  - **NOT_FOUND** jika salah satu dari **origin** atau **destination** bukan berupa *latitude, longitude* dan tidak dapat ditemukan.
  - **ZERO_RESULTS** jika Google tidak berhasil menemukan rute yang diminta.
  - **INVALID_REQUEST** jika ada parameter wajib yang tidak diberikan, atau ada parameter yang tidak valid.
  - **OVER_QUERY_LIMIT** yang berarti jumlah permintaan sudah melebihi kuota.
  - **REQUEST_DENIED** jika permintaan ditolak.
- **geocoded_waypoints** adalah hasil *geocoding* dari **origin, destination,** maupun *waypoints* pada permintaan. *Geocoding* pada API ini adalah proses konversi dari lokasi maupun nama tempat menjadi *place_id*.
- **routes** adalah *array* dari objek yang berisi informasi detail setiap alternatif rute yang ditemukan. elemenn dari **routes** akan dijelaskan pada subsubbab berikutnya.

Elemen dari *routes*

Setiap elemen dari **routes** adalah objek yang memiliki anggota sebagai berikut :

- **summary** adalah ringkasan dari alternatif rute ini, untuk membedakan dengan rute alternatif lainnya.
- **legs** adalah *array* yang berisi objek yang mempresentasikan *leg*. *Leg* adalah subrute untuk setiap *waypoints* yang diberikan (jika parameter opsional *waypoints* diberikan). Jika *waypoints* tidak diberikan, *array* ini akan berisi satu elemen saja. Penjelasan setiap elemen *legs* akan dijelaskan pada subsubbab berikutnya.
- **waypoint_order** adalah *array* yang berisi urutan *waypoint* yang baru, jika parameter *waypoints* diawali dengan *optimized:true*.
- **overview_polyline** adalah berisi daftar titik-titik yang dilalui oleh rute yang didapatkan. Titik-titik rute ini sudah disederhanakan (tidak detail), dan diringkas dengan format *encoded polyline*.
- **bounds** adalah menyatakan kotak yang menyelubungi rute yang diberikan. Kotak ini direpresentasikan dalam sebuah objek yang mengandung dua anggota yaitu : *northeast*(kanan-atas) dan *southwest*(kiri-bawah). Setiap anggota berupa objek lain yang mengandung dua anggota yaitu : *lat* yang merepresentasikan *latitude* dan *lng* yang merepresentasikan *longitude*.
- **copyrights** adalah berisi teks *copyright* yang harus ditampilkan kepada pengguna.
- **warnings** adalah *array string* yang berisi peringatan yang harus ditampilkan kepada pengguna, jika ada.
- **fare** adalah informasi biaya transportasi publik yang harus dikeluarkan, jika parameter *mode* berisi *transit* dan Google memiliki informasi tarif untuk setiap moda yang digunakan. Informasi ini belum tersedia di Indonesia.

Elemen dari *legs*

Setiap elemen dari **legs** adalah sebagai berikut :

- **steps** adalah *array* yang berisi objek yang menyatakan setiap langkah yang harus diambil. Penjelaasan setiap elemen *steps* dijelaskan pada subsubbab berikutnya.
- **distance** adalah menyatakan jarak yang harus ditempuh pada *leg* ini, berupa objek yang berisi dua anggota yaitu *value* yang merepresentasikan angka yang menyatakan jarak dalam meter dan *text* yang merepresentasikan jarak dalam fotmat teks yang dapat dibaca manusia.
- *duration* adalah menyatakan waktu yang dibutuhkan untuk menempuh *leg* ini, berupa objek yang berisi dua anggota yaitu : *value* yang merepresentasikan angka yang menyatakan waktu dalam detik dan *text* yang merepresentasikan waktu yang dibutuhkan dalam format teks yang dapat dibaca manusia.

- **duration_in_traffic** adalah menyatakan waktu mirip dengan duration. perbedaannya pada elemen ini memperhitungkan faktor kepadatan lalu lintas.
- **arrival_time** dan **departure_time** adalah waktu sampai di *destination* dan waktu keberangkatan ke *destination*, jika parameter *mode* berisi *transit*. berupa objek yang mengandung tiga anggota yaitu : *value* yang merepresentasikan waktu sampai sesuai dengan objek *date* pada *javascript*, *text* yang merepresentasi waktu sampai dalam format teks yang dapat dibaca manusia, dan *time_zone* yang merepresentasikan zona waktu pada lokasi akhir *leg*.
- **start_location** dan **end_location** adalah berisi lokasi awal dan akhir dari *leg* ini, berupa objek yang memiliki doa anggota yaitu : *lat* yang merepresentasikan *latitude* dan *lng* yang merepresentasikan *longitude*.
- **start_address** dan **end_address** adalah berisi lokasi awal dan akhir dari *leg* ini, dalam format teks yang dapat dibaca manusia.

Elemen dari *steps*

Setiap elemen dari **steps** adalah sebagai berikut :

- **html_instructions** adalah berisi instruksi *step* ini, dalam format HTML.
- **distance** adalah jarak dari *step* ini, dengan format yang sama seperti anggota *duration* pada elemen *legs* di atas.
- **start_location** dan **end_location** adalah lokasi awal dan akhir dari *step* ini, dengan format yang sama seperti anggota **start_location** dan **end_location** pada elemen *legs* di atas.
- **polyline** adalah berisi daftar titik-titik yang dilalui pada *step* ini. titik- titik rute ini diringkas dengan format *encoded polyline*.
- **steps** adalah *array* yang berisi *sub-step* dari *step* ini, jika parameter *mode* berisi *transit*. Formatnya sama dengan elemen step ini.
- **transit_details** adalah berisi detail transit, jika parameter *mode* berisi *transit*. Penjelasan objek **transit_details** akan dijelaskan pada subsubbab berikutnya.

Elemen dari *transit_details*

Setiap elemen dari **transit_details** adalah sebagai berikut :

- **name** adalah berisi nama jalur ini.
- **short_name** adalah berisi nama jalur yang lebih singkat, biasanya kode jalur.
- **color** adalah berisi warna yang umum digunakan untuk merepresentasikan jalur ini, dalam format string heksadesimal.

- **agencies** adalah *array* yang tiap elemennya berupa objek yang merepresentasikan penyedia layanan, dan mengandung tiga anggota yaitu : *name* yang merepresentasikan nama penyedia layanan, *url* yang merepresentasikan alamat situs web, dan *phone* yang mere[resentasikan nomor telepon. Informasi ini wajib ditampilkan ke pengguna.
- **url** adalah alamat situs web dari jalur ini.
- **icon** adalah URL untuk mendapatkan gambar yang merepresentasikan jalur ini.
- **text_color** adalah berisi warna yang umum digunakan untuk teksyang merepresentasikan jalur ini dalam format string heksadesimal.
- **vehicle** adalah berisi informasi kendaraan yang digunakan pada jalur inim dalam bentuk objek yang mengandung empat anggota yaitu : *name* yang merepresentasikan nama kendaraan, *type* yang merepresentasikan tipe kendaraan, *icon* yang merepresentasikan URL gambar kendaraan, *local_icon* yang merepresenasikan gambar kendaraan secara lokal.

2.4 *JavaScript Object Notation (JSON)*

JSON (JavaScript Object Notation) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (generate) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll[4].

2.4.1 Struktur JSON

JSON terbuat dari dua struktur :

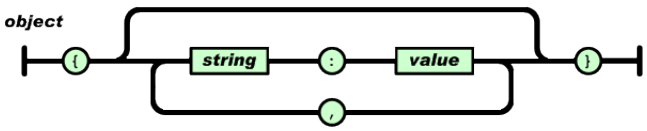
- Kumpulan pasangan nama/nilai.
- Daftar nilai terurutkan (an ordered list of values).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini.

2.4.2 Bentuk-Bentuk JSON

- Objek

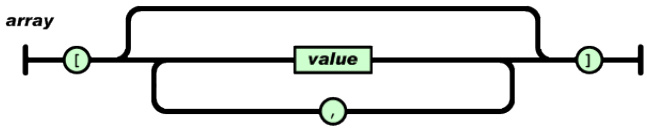
  Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan (kurung kurawal buka) dan diakhiri dengan (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama atau nilai dipisahkan oleh , (koma).



Gambar 2.4: JSON Object

- *Array*

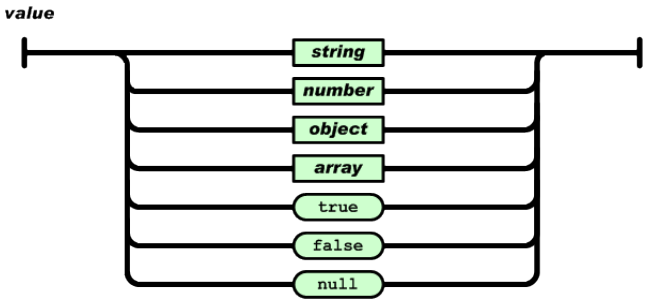
  Array adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



Gambar 2.5: JSON Array

2.4.3 Value JSON

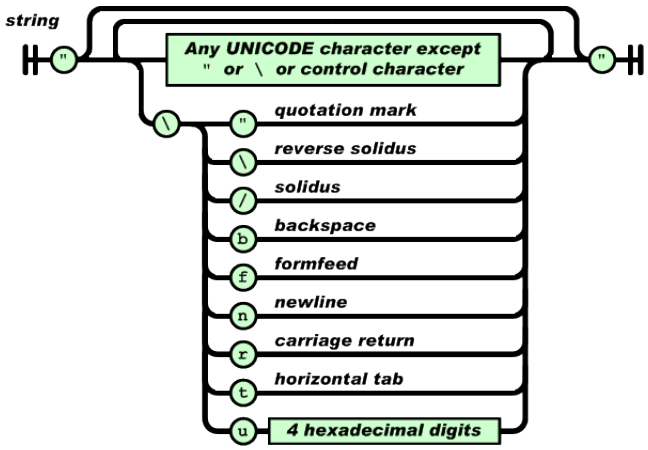
Nilai(*value*)dapat berupa sebuah string dalam tanda kutip ganda, atau angka, atau true atau false atau null, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



Gambar 2.6: Value

- *String*

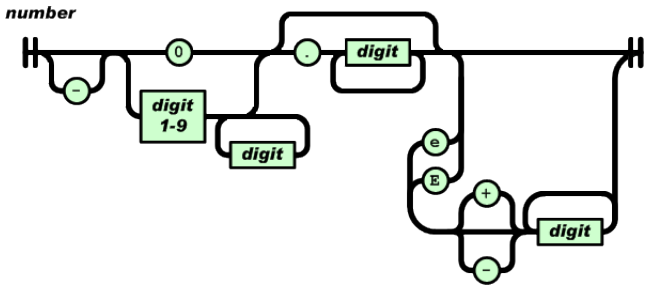
  String adalah kumpulan dari nol atau lebih karakter Unicode, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan backslash escapes "üntuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Gambar 2.7: String

- Angka

  Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Gambar 2.8: Angka

2.4.4 kelas-kelas pada *Library* JSON

Subbab-subbab berikut menjelaskan beberapa kelas dari *library* JSON[1].

JSONObject

Kelas ini merepresentasikan sebuah objek JSON yang merupakan koleksi yang tak terurut dari pasangan nama dan nilai. Bentuk eksternal objek JSON adalah sebuah string dibungkus dalam kurung kurawal dengan titik dua antara nama dan nilai-nilai, dan koma antara nilai-nilai dan nama. Nilai-nilai dapat salah satu dari jenis: Boolean, JSONArray, JSONObject, Nomor, String, atau benda JSONObject.NULL. beberapa *method* dan *constructor* yang dimiliki kelas ini adalah sebagai berikut:

- **public JSONObject(String source) throws JSONException**

  Berfungsi untuk membangun JSONObject dari sumber JSON string teks.

  Parameter:

  - **source**: Sebuah string dimulai dengan {(kurung kurawal kiri) dan berakhir dengan} (kurung kurawal kanan).

[1] https://stleary.github.io/JSON-java/

- **public String getString(String key)throws JSONException**

  Berfungsi untuk mendapatkan objek nilai yang terkait dengan kunci.

  Parameter:

  - **key**: kunci data.

  Kembalian: Sebuah string yang merupakan nilai.

- **public String optString(String key)**

  Berfungsi untuk mendapatkan string opsional terkait dengan kunci. Ia mengembalikan string kosong jika tidak ada kunci yang ditemukan. Jika nilai tidak string dan tidak null, maka dikonversi ke string.

  Parameter:

  - **key**: kunci data.

  Kembalian: Sebuah string yang merupakan nilai.

- **public JSONArray getJSONArray(String key) throws JSONException**

  Berfungsi untuk mendapatkan nilai JSONArray terkait dengan kunci.

  Parameter:

  - **key**: kunci data.

  Kembalian: Sebuah JSONArray yang merupakan nilai.

- **public JSONObject getJSONObject(String key) throws JSONException**

  Berfungsi untuk mendapatkan nilai JSONObject terkait dengan kunci.

  Parameter:

  - **key**: kunci data.

  Kembalian: Sebuah JSONObject yang merupakan nilai.

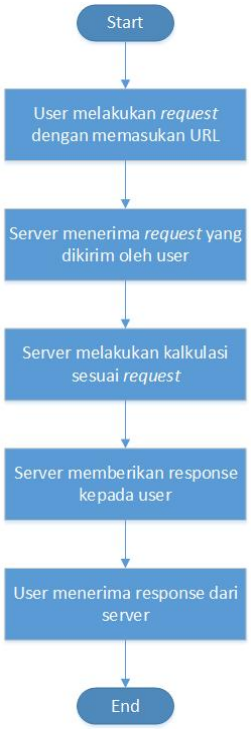
BAB 3

ANALISIS

Berdasarkan hasil studi pustaka yang telah dilakukan, pada bab ini akan dijelaskan hasil analisis berupa uraian dari perangkat lunak yang akan dibangun, analisis google direction API, diagram use-case beserta dengan skenario dan analisis diagram kelas.

3.1 Flow Chart Alur Layanan Google Direction

Dalam mengakses layanan Google Direction sesuai dengan subbab 2.3 yang berjalan pada protokol HTTP, terjadi transaksi data yang bergerak antara *user* dan *server* Google. Dengan menggunakan diagram *flow chart* akan memudahkan dalam pembangunan perangkat lunak dan mengetahui alur transaksi dari layanan Google Direction. Diagram *flow chart* yang menunjukan alur transaksi layanan Google Direction dapat dilihat pada Gambar 3.1



Gambar 3.1: Flow Chart Alur Layanan Google Direction

3.2 Analisis permintaan ke layanan Google Direction

Sesuai dengan subbab 2.3.1 permintaan dari google direction ini menggunakan protokol HTTP. Permintaan tersebut menghubungi hostname www.google.com dengan port default untuk port HTTP yaitu 80. Permintaan tersebut disertai dengan parameter-parameter opsional lainnya untuk mendapatkan data yang diinginkan.

3.2.1 Parameter yang digunakan

Untuk mendapatkan data waktu tempuh yang beragam untuk menganalisis waktu tempuh dari 2 titik sesuai dengan 2.3.2, parameter opsional yang digunakan adalah : **departure_time** dan **traffic_model**. Dari memanipulasi kedua parameter tersebut akan menghasilkan data waktu tempuh yang beragam. Selain itu memanipulasi nilai parameter pada **destination** dan **origin** juga akan mempengaruhi data waktu tempuh yang dihasilkan karena pada perhitungan dari masing-masing **destination** ke **origin** akan menghasilkan waktu tempuh yang berbeda. Dari masing-masing **destination** ke **origin**juga memiliki jam kepadatan tertentu dimana nilai waktu tempuh akan berbeda dengan jam-jam lainnya sesuai dengan **departure_time**. Parameter **traffic_model** ini juga mempengaruhi nilai yang watu tempuh dikeluarkan tergantung model apakah yang digunakan yang telah dibahas pada subbab 2.3.2.

```
https://maps.googleapis.com/maps/api/directions/json?...traffic_model=best_guess
```

Listing 3.1: Traffic_model : best_guess

```
https://maps.googleapis.com/maps/api/directions/json?...traffic_model=optimistic
```

Listing 3.2: Traffic_model : optimistic

```
https://maps.googleapis.com/maps/api/directions/json?...traffic_model=pessimistic
```

Listing 3.3: Traffic_model : pessimistic

3.3 Analisis response dari layanan Google Directions

Pada saat melakukan permintaan, Server akan memberikan *response* dengan format JSON. Response yang diterima adalah hasil perhitungan dari *origin* ke *destination*. Dari response ini terdapat banyak data didalamnya.

Data waktu tempuh pada hasil response permintaan ada pada *duration_in_traffic* dimana *duration_in_traffic* ini adalah salah satu elemen dari *legs* (subsubbab 2.3.4) yang merupakan sebuah *json array* dan *legs* ini sendiri adalah salah satu elemen dari *routes* yang merupakan elemen dari *response* yang diterima.

```
{
   "geocoded_waypoints" :[
        ...
     "routes" : [
     ...
         "legs" : [
         ...
            "duration_in_traffic" : {
          "text" : "57 menit",
          "value" : 3439
        },
          ...
     "status" : "OK"
```

```
}
```

Listing 3.4: Hasil *response* Google Directions

3.4 Gambaran Umum Perangkat Lunak

Perangkat lunak yang akan dibangun adalah perangkat lunak untuk menghitung waktu tempuh dari 2 titik yang ditentukan. Perangkat lunak yang akan dibangun ini bertujuan untuk membantu menganalisis pada jam berapakah waktu tempuh paling cepat dalam waktu 1 minggu terhitung dari hari senin. Selain itu, perangkat lunak ini bertujuan untuk membantu pengambilan keputusan pengguna untuk menentukan pada jam berapakah pengguna melakukan perjalanan agar tidak terjebak dalam kemacetan. Perangkat lunak ini berjalan pada protokol HTTP. Perangkat lunak ini dibangun pada perangkat komputer(desktop) yang berfungsi sebagai penghitung waktu tempuh dengan memanfaatkan Google Direction API. Perangkat lunak mengeluarkan *output* berupa file yang berekstensi .csv untuk mencatat seluruh data yang diterima oleh perangkat lunak dari layanan Google Direction dan menggunakan aplikasi *Microsoft Excel* untuk membantu menganalisis data dengan cara me-*generate* bagan secara manual oleh pengguna. Perangkat lunak ini akan diuji coba sesuai dengan sampel sebagai berikut : menghitung waktu tempuh antar lokasi yang beralamat Jln. Ciumbuleuit No.94 dan Komplek Amaya Residence; menghitung waktu tempuh antar lokasi yang beralamat Jln. Ciumbuleuit No.94 dan Komplek Taman Puspa Indah. Penetapan sampel untuk memudahkan mendapatkan waktu tempuh dengan alamat yang konstant dan memudahkan untuk output yang dikeluarkan.

3.5 Analisis Perangkat Lunak

Perangkat lunak yang akan dibangun adalah perangkat lunak yang dapat melakukan penghitungan waktu tempuh tercepat berdasarkan *request-request* yang dikirimkan oleh user dalam jangka waktu 1 minggu terhitung dari senin. perangkat lunak dibangun dengan menggunakan bahasa pemrograman Java dan membutuhkan *library* jsoup yang akan digunakan untuk membantu perancangan dan pengimplementasian perangkat lunak yang akan dibangun oleh penulis. Berikut adalah fitur-fitur yang akan dibangun pada perangkat lunak:

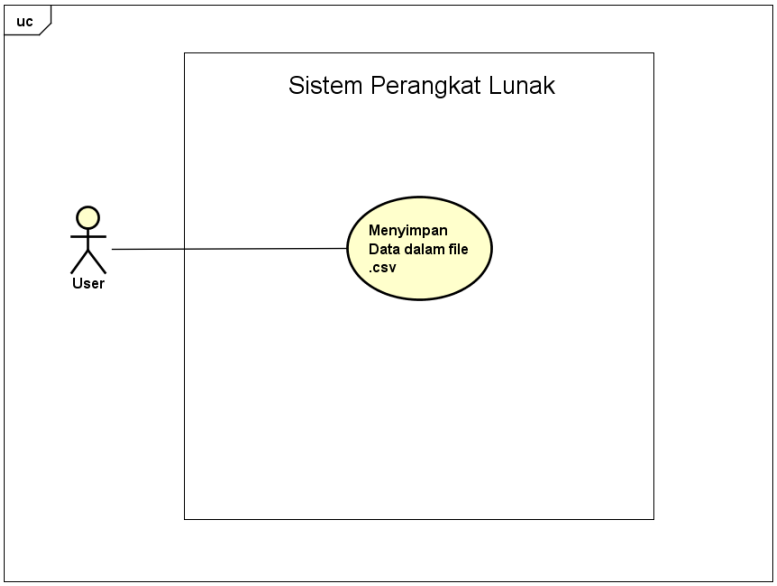
1. Mengekstaksi data waktu tempuh dari keluaran *response* Google Direction dan menampilkan pukul berapa yang memiliki waktu tempuh terbaik dalam kurun waktu 1 minggu.

2. Menyimpan data-data waktu tempuh dari keluaran *response* Google Direction pada file berekstensi .csv.

3.6 Analisis *Use Case*

3.6.1 Diagram *Use Case*

Diagram use case pada perangkat lunak yang akan dibangun hanya mengandung satu aktor, yaitu User. Diagram use case dapat dilihat pada Gambar 3.2.



Gambar 3.2: Diagram *Use Case* Perangkat Lunak

Berdasarkan subbab 3.5. dari dua fitur yang akan dibuat, dibentuk satu *use case* antara lain:

- **Menyimpan Data dalam file .csv**, User dapat menyimpan data dari penghitungan antar 2 titik : sampel dan Unpar.

3.6.2 Skenario *Use Case*

1. Menghitung Waktu Tempuh

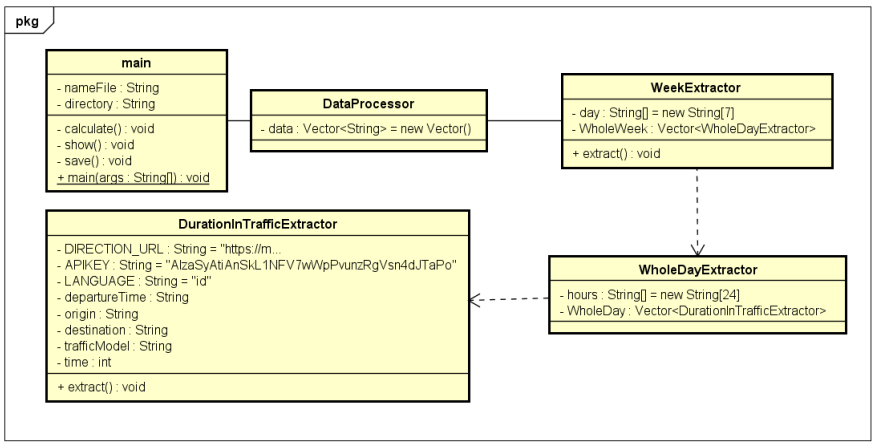
   - Nama : Menyimpan Data dalam file .csv.
   - Aktor : User.
   - Deskripsi : Menyimpan data dari hasil penghitungan dari tempat asal tempat asal ke tempat tujuan.
   - Kondisi awal : User memulai program.
   - Kondisi akhir : User berhasil menyimpan data dan membuka file tersebut dengan bantuan excel.
   - Skenario Utama :

No	Aksi Aktor	Reaksi Sistem
1	User memulai program	Sistem menampilkan GUI dari perangkat lunak
2	User melakukan input	
3	User menekan tombol save	Sistem mengolah data seusai dengan input dari user
4		Sistem menampilkan layar untuk penyimpanan file
5	User melakukan input untuk penyimpanan file	Sistem melakukan penyimpanan file seusai input dari user
6		Sistem mebuka file tersebut dengan menggunakan aplikasi Microsoft Excel

- Eksepsi :
  - tidak ada model traffic yang dipilih.
  - user belum memilih sampel mana yang akan dihitung.
  - user belum memilih tanggal.
  - user memilih tanggal pada masa lampau dan hari ini.

3.7 Analisis Kelas

Diagram kelas analisis untuk perangkat lunak ditunjukkan pada Gambar 3.3



Gambar 3.3: Diagram Kelas untuk Perangkat Lunak

Penjelasan dari kelas-kelas lainnya sebagai berikut:

1. **DurationInTrafficExtractor** adalah kelas yang bertugas untuk mengirimkan permintaan ke layanan Google Direction dan menekstraksi data waktu tempuh.
2. **WholeDayExtractor** adalah kelas yang bertugas menekstraksi waktu tempuh dalam 1 hari.

3. **WholeWeekExtractor** adalah kelas yang bertugas untuk menekstraksi waktu tempuh dalam 1 minggu.

4. **DataProsesor** adalah kelas yang bertugas sebagai tempat penyimpanan data dan bertugas untuk proses penyimpanan data ke dalam file.

5. **main** adalah kelas yang bertugas sebagai tampilan utama pada perangkat lunak ini.

BAB 4

PERANCANGAN

Berdasarkan analisis yang telah dilakukan, terdapat beberapa hal yang perlu dirancang untuk pembangunan perangkat lunak sederhana analisis waktu tempuh kota Bandung ini. Pada bab ini akan dijelaskan mengenai perancangan aplikasi yang akan dibangun meliputi perancangan file keluaran, perancangan antarmuka, kelas diagram kelas rinci beserta deskripsi dan fungsinya.

4.1 Kebutuhan Masukan dan Keluaran

Perangkat lunak yang dibangun merupakan perangkat lunak untuk melakukan ekstraksi data dari layanan Google Direction. Pada perancangan perangkat lunak ini menggunakan *library* jsoup untuk melalukan *request* ke layanan Google Direction. Selain itu perangkat lunak ini akan menggunakan *library* JSON untuk mengekstraksi *output* JSON yang merupakan *response* dari *request* yang diminta. Masukan dan keluaran perangkat lunak adalah sebagai berikut :

4.1.1 Masukan

Masukan dari perangkat lunak sederhana ini adalah parameter-parameter yang digunakan untuk melakukan request. Parameter-parameter tersebut adalah : *origin, departure_time* dalam bentuk unix, dan *traffic_model*. Nilai parameter *origin* adalah nilai *longitude* dan *latitude* yang disatukan menjadi sebuah string. *Longitude* dan *latitude* itu sendiri adalah *longitude* dan *latitude* dari masing-masing sample. Nilai parameter *departure_time* adalah nilai unix yang di-*parsing* menjadi bentuk string dari sebuah tanggal. Nilai parameter *traffic_model* adalah sebuah string salah satu dari *best_guess, optimistic, pessimistic*.

4.1.2 Keluaran

Keluaran dari perangkat lunak sederhana ini adalah sebuah file dengan ekstensi *Comma Separated Value* (.csv) dimana dalam file ini berisi data hasil ekstraksi dari *request* ke layanan Google Direction.

4.2 Parameter *request* ke layanan Google Direction

Sesuai dengan subbab 3.2, *request* berbentuk sebuah url dengan memasukan parameter-parameter yang dibutuhkan keladam url untuk melakukan suatu *request*. Nilai dari parameter *origin* yang akan digunakan pada perangkat lunak ini dibagi menjadi 2 yaitu : sampel 1

dengan nilai "'-6.9536001,107.6193958"' dan sampel 2 dengan nilai "'-6.937021,107.6643817"'. Nilai dari parameter *destination* yang akan digunakan pada perangkat lunak ini adalah "'-6.8746025,107.6024968"' yang merupakan nilai *longitude* dan *latitude* dari Universitas Katolik Parahyangan.

4.3 Rancangan file keluaran

Sesuai dengan subsubbab 4.1.2, data pada file ini memiliki format tersusun dari nilai-nilai yang dipisahkan oleh koma(,). Menggunakan aplikasi Microsoft Excel, setiap nilai-nilai yang dipisahkan oleh koma(,) ini di direpresentasikan dengan masing-masing kolom. Sesuai masukan dari pengguna, file keluaran ini sendiri terbagi menjadi tiga tergantung dari *traffic_model* yang dipilih. Berikut ini merupakan tiga macam keluaran yang akan dihasilkan oleh perangkat lunak sederhana ini :

- file ekstensi .csv dengan tiga nilai : file ini akan dihasilkan jika pengguna memilih satu *traffic_model* dimana nilai pertama adalah nilai yang merepresentasikan hari, nilai kedua adalah nilai merepresentasikan jam dan nilai yang terakhir adalah nilai waktu tempuh sesuai dengan yang dipilih oleh pengguna.
- file ekstensi .csv dengan empat nilai : file ini akan dihasilkan jika pengguna memilih dua *traffic_model* dimana nilai pertama adalah nilai yang merepresentasikan hari, nilai kedua adalah nilai merepresentasikan jam, nilai yang ketiga dan keempat adalah nilai waktu tempuh sesuai dengan yang dipilih oleh pengguna. Kombinasi pilihan pengguna yang memungkinkan adalah *best_guess* dan *optimistic*, *best_guess* dan *pessimistic*, *optimistic* dan *pessimistic*.
- file ekstensi .csv dengan lima nilai : file ini akan dihasilkan jika pengguna memilih satu *traffic_model* dimana nilai pertama adalah nilai yang merepresentasikan hari, nilai kedua adalah nilai merepresentasikan jam, nilai ketiga, keempat dan kelima adalah nilai waktu tempuh sesuai dengan yang dipilih oleh pengguna dengan kombinasi pilihan pengguna adalah *best_guess*, *optimisic* dan *pessimistic*.

4.4 Diagram Kelas Rinci

Diagram kelas rinci diperoleh dari hasil pengembangan diagram kelas analisis pada subbab 3.7. Diagram kelas rinci dapat dilihat pada Gambar 4.1. Deskripsi kelas beserta fungsi dari diagram kelas rinci tersebut adalah sebagai berikut:

1. DurationIntrafficExtractor

   Kelas ini merupakan kelas yang bertugas untuk melakukan *request* ke Google Direction API dan mengekstraksi nilai waktu dalam satuan detik dari *response* balasan dari *request* yang diminta. Kelas ini melakukan satu *request* dalam satu waktu. Kelas ini juga melakukan penghitungan konversi dari nilai waktu yang bersatuan detik menjadi menit. Atribut yang dimiliki oleh kelas ini antara lain :

   - **private String DIRECTION_URL :** merupakan url dasar untuk melakukan *request* ke Google Direction API.

- **private String APIKEY :** merupakan sebuah kunci API yang akan dimasukan kedalam url dasar sebagai salah satu parameter untuk dapat melacak penggunaannya.
- **private String LANGUAGE :** merupakan salah satu parameter untuk url dasar yang berfungsi untuk memberikan *response* dari Google Direction dalam suatu bahasa.
- **private String departureTime :** merupakan salah satu parameter untuk url dasar untuk menentukan waktu keberangkatan dari suatu titik ke tempat tujuan.
- **private String origin :** merupakan parameter wajib untuk url dasar yang merepresentasikan suatu titik berupa *longitude* dan *latitude* dari suatu tempat yang dijadikan acuan dasar tempat keberangkatan ke tempat tujuan.
- **private String destination :** merupakan parameter wajib untuk url dasar yang merepresentasikan suatu titik berupa *longitude* dan *latitude* dari suatu tempat yang dijadikan acuan dasar tempat tujuan.
- **private int time :** merupakan sebuah attribut yang berisikan waktu dalam hitungan menit yang didapatkan dari hasil ekstraksi *response* **duration_in_traffic** dari suatu *request*.

Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

- **public DurationInTrafficExtractor(String unix, String origin, String destination, String trafficModel)**

  merupakan konstruktor dari kelas ini. Fungsinya untuk menginistansiasi dari masing-masing atribut yang dimiliki kelas ini.

  Parameter:

  - **unix**: nilai waktu berbentuk string yang telah dikonversi kedalam bentuk unix yang merepresentasikan waktu keberangkatan dari tempat asal ke tempat tujuan.
  - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
  - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
  - **trafficModel**: nilai string yang merepresentasikan model traffic yang akan digunakan.

- **public void setTime(int time)**

  Berfungsi untuk menetapkan nilai dari atribut *time*.

  Parameter:

  - **time**: nilai waktu yang akan ditetapkan.

- **public int getTime()**

  Berfungsi untuk mendapatkan nilai yang dari atribut *time*.

  Kembalian: Sebuah integer yang merupakan nilai dari atribut *time*.

- **public void extract()**

  Berfungsi untuk menetapkan seluruh parameter pada url dasar dan melakukan *request* ke layanan Google dan mendapatkan *response*-nya. Setelah mendapatkan *response*-nya method ini melakukan ekstraksi untuk mendapatkan waktu tempuh pada suatu waktu.

- **public String getDepartureTimeHours()**

  Berfungsi untuk mendapatkan nilai jam dalam bentuk string dari atribut *departureTime*.

  Kembalian: Sebuah string yang merupakan nilai jam dari atribut *departureTime*.

2. WholeDayExtractor

Kelas ini merupakan kelas yang bertugas untuk mendapatkan nilai waktu tempuh selama satu hari penuh. Kelas ini terdiri dari *DurationIntrafficExtractor* dimana setiap *DurationInTrafficExtractor* merepresentasikan *request* ke Google Direction API dalam setiap jam. Atribut yang dimiliki oleh kelas ini antara lain :

- **private String[] hours :** merupakan sebuah atribut array yang memiliki ukuran duapuluh empat dimana setiap string dalam atribut ini merepresentasikan setiap jam.
- **private DurationInTrafficExtractor[] wholeDay :** merupakan sebuah atribut array yang memiliki ukuran duapuluh empat dimana setiap *DurationInTraffic* dalam atribut ini merepresentasikan waktu tempuh dalam setiap jamnya yang memiliki nilai yang berbeda.

Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

- **public void initialize(String unix, String origin, String destination, String trafficModel) throws ParseException**

  Berfungsi untuk menetapkan seluruh parameter pada url dasar pada setiap *DurationInTrafficExtractor* dalam array *wholeDay* dengan menentukan parameter-parameter ke setiap *DurationInTrafficExtractor*.

  Parameter:

  - **unix**: nilai waktu berbentuk string yang telah dikonversi kedalam bentuk unix yang merepresentasikan waktu keberangkatan dari tempat asal ke tempat tujuan.
  - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
  - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
  - **trafficModel**: nilai string yang merepresentasikan model traffic yang akan digunakan.

- **public void extract() throws IOException**

  Berfungsi untuk melakukan *request* ke layanan Google dan mendapatkan *response*-nya pada setiap *DurationInTrafficExtractor*. Setelah mendapatkan *response*-nya method ini melakukan ekstraksi untuk mendapatkan waktu tempuh pada setiap *DurationInTrafficExtractor*.

- **public DurationInTrafficExtractor[] getWholeDay()**

  Berfungsi untuk mendapatkan setiap *DurationInTrafficExtractor* yang ada didalam array *wholeDay*.

  Kembalian: Sebuah array *DurationIntrafficExtractor*.

- **public String[] getHours()**

  Berfungsi untuk mendapatkan nilai string setiap jam.

  Kembalian: Sebuah array string.

3. WeekExtractor

   Kelas ini merupakan kelas yang bertugas untuk mendapatkan nilai waktu tempuh selama tujuh hari. Kelas ini terdiri dari *WholeDayExtractor* dimana setiap *WholeDayExtractor* merepresentasikan *request* ke Google Direction API dalam setiap hari. Atribut yang dimiliki oleh kelas ini antara lain :

   - **private String[] day :** merupakan sebuah atribut array yang memiliki ukuran tujuh dimana setiap string dalam atribut ini merepresentasikan setiap harinya.
   - **private WholeDayExtractor[] oneWeek :** merupakan sebuah atribut array yang memiliki ukuran tujuh dimana setiap *DurationInTraffic* dalam atribut ini merepresentasikan waktu tempuh dalam setiap harinya yang memiliki nilai yang berbeda.

   Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

   - **public void initialize(String date, String origin, String destination, String trafficModel) throws ParseException**

     Berfungsi untuk menetapkan seluruh parameter pada url dasar pada setiap *WholeDayExtractor* dalam array *oneWeek* dengan menentukan parameter-parameter ke setiap *WholeDayExtractor*.

     Parameter:

     - **unix**: nilai waktu berbentuk string yang telah dikonversi kedalam bentuk unix yang merepresentasikan waktu keberangkatan dari tempat asal ke tempat tujuan.
     - **origin**: nilai *longitude* dan *latitude* yang disatukan mejadi string yang merepresentasikan tempat asal keberangkatan.
     - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
     - **trafficModel**: nilai string yang merepresentasikan model traffic yang akan digunakan.

- **public void extract() throws IOException**

  Berfungsi untuk melakukan *request* ke layanan Google dan mendapatkan *response*-nya pada setiap *WholeDayExtractor*. Setelah mendapatkan *response*-nya method ini melakukan ekstraksi untuk mendapatkan waktu tempuh pada setiap *WholeDayExtractor*.

- **public WholeDayExtractor[] getOneWeek()**

  Berfungsi untuk mendapatkan setiap *WholeDayExtractor* yang ada didalam array *oneWeek*.

  Kembalian: Sebuah array *WholeDayExtractor*.

- **public String[] getDay()**

  Berfungsi untuk mendapatkan nilai string setiap harinya.

  Kembalian: Sebuah array string.

4. DataProcessor

   Kelas ini merupakan kelas yang bertugas untuk memproses semua data yang didapatkan. Selain itu Kelas ini juga bertugas untuk melakukan penulisan data-data kedalam file dengan ekstensi *.csv*. Atribut yang dimiliki oleh kelas ini antara lain :

   - **private String csvSplitBy :** merupakan atribut untuk memisahkan antar data yang didapatkan untuk dituliskan kedalam file.
   - **private Vector<String> data :** merupakan atribut *Vector* dimana data yang didapatkan dari hasil ekstraksi disimpan.
   - **private Vector<String> trafficModel :** merupakan atribut *Vector* yang merepresentasikan model traffic yang digunakan oleh data yang disimpan dalam *Vector data*.

   Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

   - **public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox trafficModel) throws ParseException, IOExcception**

     Berfungsi untuk menetapkan parameter-parameter untuk melakukan *request* ke layanan Google dimana model traffic yang digunakan adalah **satu** model traffic dan mendapatkan hasil ekstraksinya. Hasil-hasil ekstraksi tersebut dimasukan kedalam *Vector data*.

     Parameter:

     - **date**: merupakan sebuah *JFormattedTextField* yang memiliki nilai tanggal yang merepresentasikan tanggal berapa yang akan dihitung.
     - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
     - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.

- **trafficModel1**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.
- **trafficModel2**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.

- **public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox trafficModel1, JCheckBox trafficModel2) throws ParseException, IOException**

  Berfungsi untuk menetapkan parameter-parameter untuk melakukan *request* ke layanan Google dimana model traffic yang digunakan adalah **dua** model traffic dan mendapatkan hasil ekstraksinya. Hasil-hasil ekstraksi tersebut dimasukan kedalam *Vector data*.

  Parameter:

  - **date**: merupakan sebuah *JFormattedTextField* yang memiliki nilai tanggal yang merepresentasikan tanggal berapa yang akan dihitung.
  - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
  - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
  - **trafficModel1**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.
  - **trafficModel2**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.

- **public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox trafficModel1, JCheckBox trafficModel2, JCheckBox trafficModel3) throws ParseException, IOException**

  Berfungsi untuk menetapkan parameter-parameter untuk melakukan *request* ke layanan Google dimana model traffic yang digunakan adalah **tiga** model traffic dan mendapatkan hasil ekstraksinya. Hasil-hasil ekstraksi tersebut dimasukan kedalam *Vector data*.

  Parameter:

  - **date**: merupakan sebuah *JFormattedTextField* yang memiliki nilai tanggal yang merepresentasikan tanggal berapa yang akan dihitung.
  - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
  - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
  - **trafficModel1**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.
  - **trafficModel2**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.
  - **trafficModel3**: merupakan sebuah *JCheckBox* yang dipilih untuk dihitung waktu tempuhnya.

- **public void saveFile(String directory, String fileName) throws IOException**

  Berfungsi untuk menuliskan data yang ada dalam *Vector data* kedalam sebuah file berekstensi *.csv*.

  Parameter:

  - **directory**: merupakan sebuah string yang merepresentasikan *directory* penyimpanan file.
  - **fileName**: merupakan sebuah string yang merepresentasikan nama file yang akan disimpan.

5. DurationTimeController

Kelas ini merupakan kelas yang bertugas sebagai jembatan penghubung antara *graphical user interface* dengan kelas *DataProcessor*. Atribut yang dimiliki oleh kelas ini antara lain :

- **private DataProcessor processor :** merupakan atribut *DataProsesor* dimana kelas ini bertugas memproses data yang diinput melalui *graphical user interface*.

Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

- **public void doCalculate(JFormattedTextField date, String origin, String destination, JCheckBox trafficModel1, JCheckBox trafficModel2, JCheckBox trafficModel3) throws ParseException, IOException**

  Berfungsi untuk meneruskan parameter-parameter yang diinput melalui *graphical user interface* dan memerintah *processor* untuk melakukan pemrosesan data.

  Parameter:

  - **date**: merupakan sebuah *JFormattedTextField* yang dinput melalui *graphical user interface*. *JFormattedTextField* tersebut memiliki nilai tanggal yang merepresentasikan tanggal berapa yang akan dihitung.
  - **origin**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat asal keberangkatan.
  - **destination**: nilai *longitude* dan *latitude* yang disatukan menjadi string yang merepresentasikan tempat tujuan dari suatu keberangkatan.
  - **trafficModel1**: merupakan sebuah *JCheckBox* yang diinput melalui *graphical user interface*. *JCheckBox* tersebut merupakan input yang dipilih untuk dihitung waktu tempuhnya.
  - **trafficModel2**: merupakan sebuah *JCheckBox* yang diinput melalui *graphical user interface*. *JCheckBox* tersebut merupakan input yang dipilih untuk dihitung waktu tempuhnya.
  - **trafficModel3**: merupakan sebuah *JCheckBox* yang diinput melalui *graphical user interface*. *JCheckBox* tersebut merupakan input yang dipilih untuk dihitung waktu tempuhnya.

- **public void saveData(String dir, String filename) throws IOException**

  Berfungsi untuk meneruskan parameter-parameter yang diinput melalui *graphical user interface* dan memerintah *processor* untuk melakukan pemrosesan penyimpanan data.

  Parameter:

  - **dir**: merupakan sebuah string yang dinput melalui *graphical user interface*. String tersebut memiliki nilai yang merepresentasikan *directory* penyimpanan file.
  - **filename**: merupakan sebuah string yang dinput melalui *graphical user interface*. String tersebut memiliki nilai yang merepresentasikan nama file.

6. FileTypeFilter

   Kelas ini merupakan kelas yang bertugas sebagai *filter* file yang memiliki suatu ekstensi dengan kelas. Dalam kelas ini peneliti mem-*filter* file dengan berekstensi *.csv*. Atribut yang dimiliki oleh kelas ini antara lain :

   - **private String extension :** merupakan atribut yang menentukan ekstensi yang di-*filter*.
   - **private String description :** merupakan atribut deskripsi dari ekstensi yang di-*filter*.

   Method-method yang dimiliki kelas ini merupakan action method dengan rincian sebagai berikut:

   - **public boolean accept(File f)**

     Berfungsi untuk memeriksa input dari *graphical user interface*.

     Parameter:

     - **f**: merupakan sebuah file.

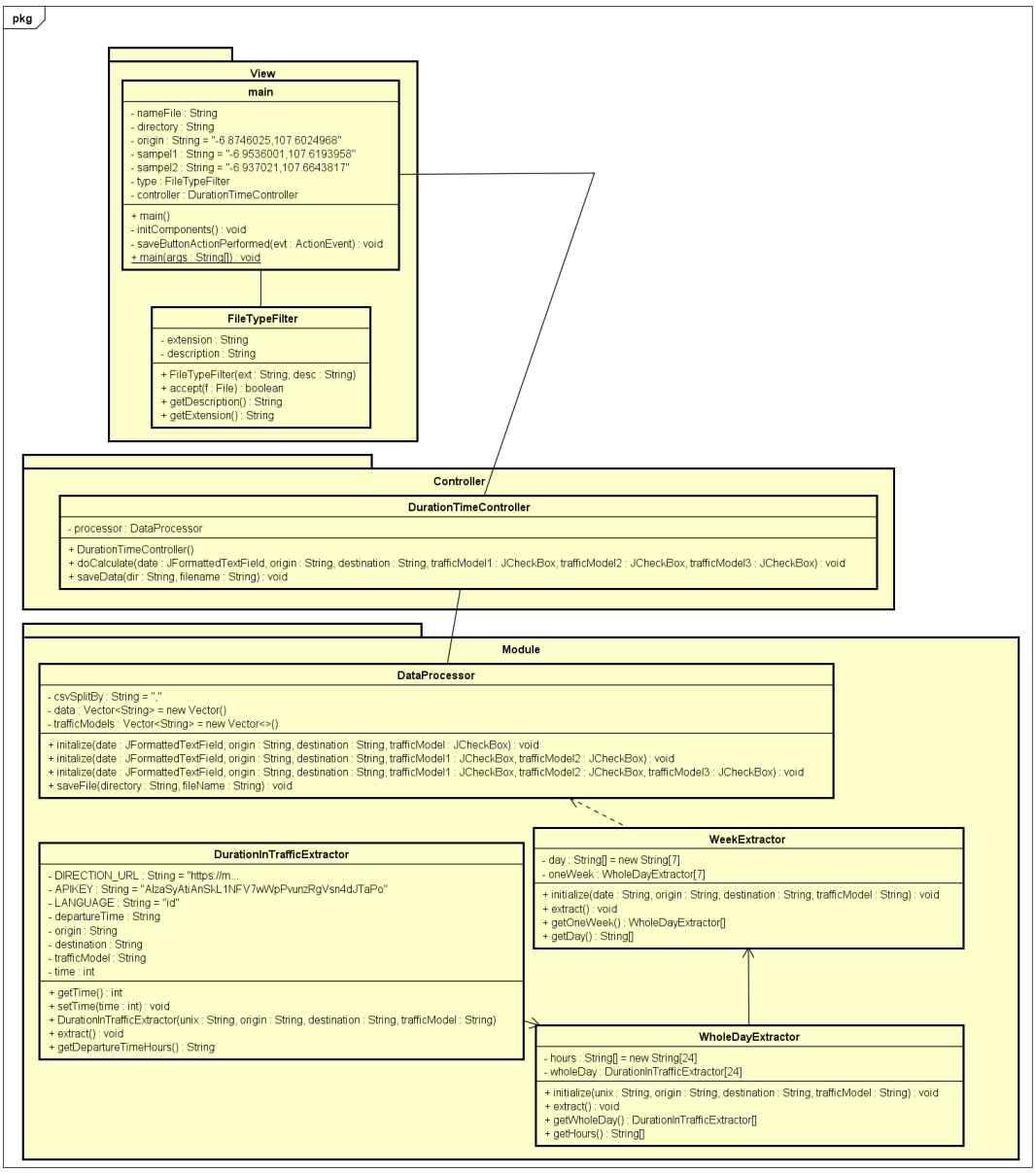
     Kembalian: Sebuah boolean.
   - **public String getDescription()**

     Berfungsi untuk mendapatkan nilai yang dari atribut *descripstion*.

     Kembalian: Sebuah string yang merupakan nilai dari atribut *descripstion*.
   - **public String getExtension()**

     Berfungsi untuk mendapatkan nilai yang dari atribut *extension*.

     Kembalian: Sebuah string yang merupakan nilai dari atribut *extension*.



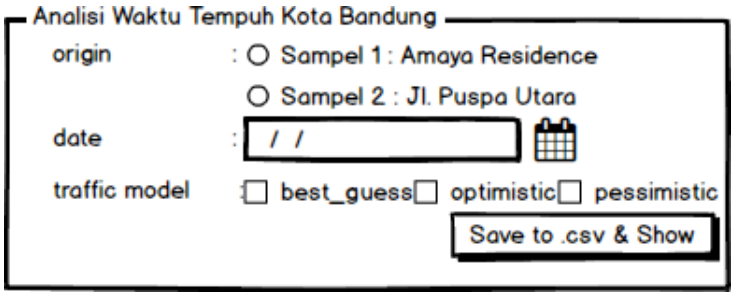
Gambar 4.1: Kelas Diagram Rinci

4.5 Perancangan Antarmuka

Untuk memenuhi kebutuhan interaksi antara pengguna dengan sistem, maka dirancanglah sebuah antarmuka dari perangkat lunak Analisis Waktu Tempuh Kota Bandun. Rancangan antarmuka dibagi menjadi dua antarmuka antara lain:

1. Antarmuka utama.

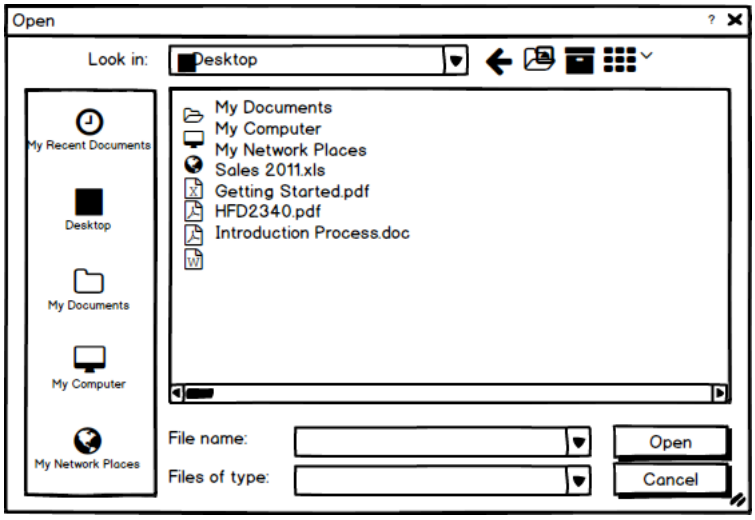
   Antarmuka ini adalah antarmuka utama dari perangkat lunak. Komponen antarmuka ini terdiri dari dua buah *radio button*, *date picker*, tiga buah *check box*, dan sebuah tombol *save* seperti yang ditunjukkan pada Gambar 4.2. Untuk dapat menyimpan data yang sudah diolah pengguna perlu melakukan input dan memilih sesuai dengan pilihan yang ada di antarmuka yang sesuai kemudian menekan tombol *save*. Jika berhasil pengguna akan diarahkan ke aplikasi Microsoft Excel yang berisikan data-data yang telah didapat.



Gambar 4.2: Antarmuka Utama

2. Antarmuka *file chooser*.

Antarmuka ini adalah antarmuka untuk menentukan *directory* penyimpanan file dan menentukan nama file yang akan disimpan. Komponen antarmuka ini seperti yang ditunjukkan pada Gambar 4.3..



Gambar 4.3: Antarmuka *File Chooser*

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas dua bagian, yaitu Implementasi Perangkat Lunak dan Pengujian Perangkat Lunak. Bagian implementasi berisi penjelasan lingkungan pengembangan perangkat lunak dan hasil implementasi. Sedangkan bagian pengujian berisi hasil pengujian perangkat lunak yang telah dibangun.

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Implementasi perangkat lunak ini dilakukan di sebuah komputer peneliti untuk keperluan pengujian dan penarikan kesimpulan. Komputer tersebut memiliki spesifikasi sebagai berikut :

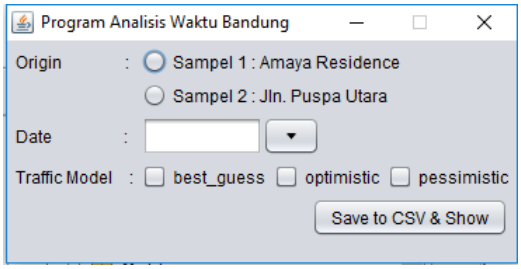
1. Processor : 1.3GHz
2. RAM: 4.00 GB DDR3
3. Sistem Operasi: Windows 10 Home 64-bit
4. Versi Java : 1.8.0_92
5. Koneksi Internet : *bandwidth up to* 1,2MBps
6. Versi Microsoft Excel : 2016

5.1.2 Implementasi Kode Program

Kode program pada perangkat lunak ditulis dalam bahasa pemrograman Java. Penulisan kode program dibagi menjadi tiga *package* yaitu :*Module*, *Controller* dan *View*. Tujuan penulisan program dibagi menjadi tiga *package* adalah untuk memudahkan proses debuging. Didalam *package Module* merupakan kode-kode program yang menjalankan semua fungsi mulai dari *request* sampai penyimpanan file. Untuk kode program yang ada pada *package Controller* merupakan kode program yang berfungsi untuk menjembatani antara tampilan dengan fungsi-fungsi untuk menjalankan perangkat lunak. Tampilan perangkat lunak ditulis didalam *package View* agar dapat mendukung interaksi antarmuka agar interaksi aplikasi lebih interaktif. Penulisan kode program menggunakan *library* : jsoup dengan versi 1.10.1, JSON dengan versi 20160810 dan swingx-all dengan versi 1.6.4. Untuk kode-kode program tersebut dapat dilihat pada lampiran A, lampiran B dan lampiran C.

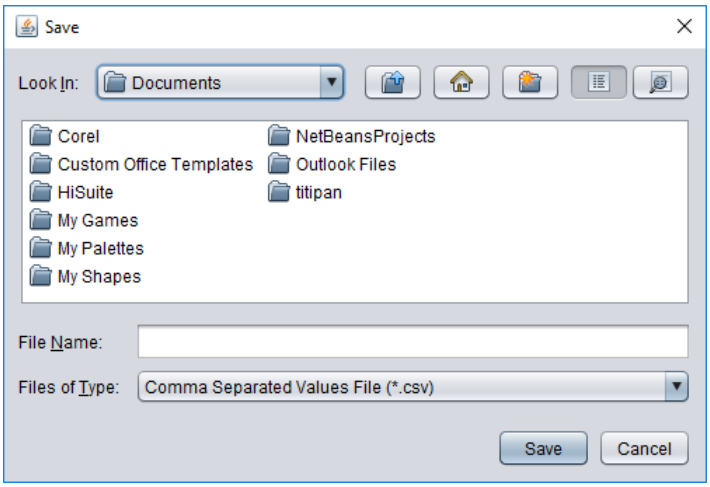
5.1.3 Tampilan antarmuka

Berikut ini merupakan hasil implementasi antarmuka untuk perangkat lunak analisis waktu tempuh kota Bandung. Pada Gambar 5.1 merupakan tampilan utama dari perangkat lunak yang memiliki tiga buah input yaitu : *radio button*, *datepicker*, *check box*. Terdapat 1 buah tombol *save* yang digunakan untuk melakukan ekstraksi data dan penyimpanan data.



Gambar 5.1: Implementasi Antarmuka Utama

Pada Gambar 5.2 merupakan tampilan *file chooser* dari perangkat lunak yang memiliki satu buah *window* untuk memilih *directory* penyimpanan file. Terdapat satu buah input untuk memberi nama file. Selain itu tampilan juga terdapat filter file untuk menyimpan data dengan suatu ekstensi tertentu. Terdapat dua buah tombol untuk fitur penyimpanan yaitu : *save* dimana tombol ini berfungsi untuk mengeksekusi penyimpanan file kemudian membuka file tersebut dengan aplikasi Microsoft Excel dan *cancel* untuk membatalkan penyimpanan file.



Gambar 5.2: Implementasi *file chooser*

5.2 Pengujian

5.2.1 Pengujian Fungsional

Pengujian fungsional perangkat lunak sederhana analisis waktu tempuh kota Bandung dengan memanfaatkan Google Direction API dilakukan untuk mengetahui kesesuaian reaksi perangkat lunak dengan reaksi yang diharapkan berdasarkan aksi pengguna terhadap perangkat lunak. Pengujian ini dilakukan pada lingkungan implementasi sesuai pada subbab 5.1.1. Terdapat 4 tes kasus yang diujikan, detail serta hasilnya dapat dilihat pada Tabel 5.1.

Beberapa data hasil ekstraksi pada pengujian ini bisa dilihat pada Lampiran D. Spesifikasi pengujian fungsional adalah sebagai berikut :

1. Pengujian dilakukan 14 kali untuk masing-masing sampel sesuai dengan tes kasus.
2. Masukan tanggal yang digunakan adalah 8 Mei 2017 dan 15 Mei 2017.
3. Tes kasus nomor 2 dilakukan sebanyak 2 kali untuk masing-masing sampel : 1 kali dengan masukan 8 Mei 2017 dan 1 kali dengan masukan 15 Mei 2017
4. Tes kasus nomor 3 dilakukan sebanyak 6 kali untuk masing-masing sampel : 1 kali dengan masukan 8 Mei 2017 dengan kombinasi model *traffic best_guess* dan *optimistic*, 1 kali dengan masukan 15 Mei 2017 dengan kombinasi model *traffic best_guess* dan *optimistic*, 1 kali dengan masukan 8 Mei 2017 dengan kombinasi model *traffic best_guess* dan *pessimistic*, 1 kali dengan masukan 15 Mei 2017 dengan kombinasi model *traffic best_guess* dan *pessimistic*, 1 kali dengan masukan 8 Mei 2017 dengan kombinasi model *traffic optimistic* dan *pessimistic*, dan 1 kali dengan masukan 15 Mei 2017 dengan kombinasi model *traffic optimistic* dan *pessimistic*
5. Tes kasus nomor 4 dilakukan sebanyak 6 kali untuk masing-masing sampel : 3 kali dengan masukan 8 Mei 2017 dengan masing-masing model *traffic* dan 3 kali dengan masukan 15 Mei 2017 dengan masing-masing model *traffic*.

No	**Aksi Pengguna**	**Reaksi yang diharapkan**	**Reaksi perangkat lunak**
1	Pengguna menjalankan program	Antarmuka utama ditampilkan	sesuai
2	Pengguna memilih salah satu sampel, memilih tanggal, memilih ketiga model *traffic* dan menekan tombol save	File berhasil disimpan dan ditampilkan dengan aplikasi microsoft excel	sesuai
3	Pengguna memilih salah satu sampel, memilih tanggal, memilih dua diantara tiga model *traffic* dan menekan tombol save	File berhasil disimpan dan ditampilkan dengan aplikasi microsoft excel	sesuai
4	Pengguna memilih salah satu sampel, memilih tanggal, memilih salah satu model *traffic* dan menekan tombol save	File berhasil disimpan dan ditampilkan dengan aplikasi microsoft excel	sesuai

		Jika pengguna belum memilih salah satu dari sampel menampilkan pesan "Silahkan pilih sampel yang akan dihitung antara sampel 1 atau sampel 2"	sesuai
		Jika pengguna belum memilih salah satu dari traffic_model menampilkan pesan "Anda harus memilih minimal salah satu dari 3 traffic model yang telah disediakan"	sesuai
		Jika pengguna belum memilih tanggal menampilkan pesan "Anda belum memilih tanggal, silahkan pilih tanggal"	sesuai
		Jika pengguna memilih tanggal yang sudah lampau atau hari ini menampilkan pesan "Tanggal yang anda masukan adalah masa lampau atau hari ini, silahkan pilihlah tanggal yang akan datang"	sesuai
		Jika pengguna memilih tanggal yang bukan hari senin menampilkan pesan "Tanggal yang anda pilih bukan hari senin, silahkan pilih tanggal yang merupakan hari senin"	sesuai

Tabel 5.1: Tabel Hasil Pengujian Fungsional

1

5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan dengan melakukan eksperimen dari hasil ekstraksi data yang ada pada Lampiran D dengan cara membuat analisis dari bagan yang datanya berasal dari hasil ekstraksi data tersebut. Bagan itu sendiri dapat dibuat dengan memanfaatkan aplikasi Microsoft Excel secara manual. Hasil pengujian eksperimental dapat dilihat pada Lampiran E yang menunjukkan perbedaan waktu tempuh pada setiap jamnya dari masing-masing sampel. Hasil pengujian eksperimental dirangkum sebagai berikut.

Pada grafik waktu masing-masing model dalam seminggu yang dapat dilihat pada Lampiran E, bahwa waktu tempuh untuk setiap hari relatif memiliki waktu tempuh yang sama setiap jamnya terkecuali pada hari jumat. Pada hari jumat, waktu tempuh cenderung menurun pada pukul 12, dan lalu menaik kembali setelah itu. Hal tersebut diperkirakan oleh karena mayoritas warga indonesia beragama muslim dan melaksanakan ibadah shalat jumat pada pukul 12. Selain itu, waktu tempuh mulai menaik sejak dari pukul 6 hingga mencapai pukul 18, dan setelah itu akan selalu menurun. Waktu tempuh maksimum pada setiap harinya berada pada sekitar pukul 18. Hal tersebut diperkirakan terjadi karena pada jam tersebut merupakan saat sebagian besar orang selesai beraktifitas dan pulang ke rumah masing-masing. Sedangkan waktu tempuh yang paling minimun ada pada setiap harinya berada pada sekitar pukul 3. Hal tersebut diperkirakan terjadi karena pada jam tersebut sebagian orang masih beristirahat dirumah masing-masing.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Dari hasil pembangunan perangkat lunak analisis waktu tempuh kota Bandung dengan memanfaatkan layanan Google Direction dan hasil data yang didapatkan, didapatkanlah kesimpulan-kesimpulan sebagai berikut :

1. Telah berhasil mengimplementasi Google Direction API dalam bahasa pemrograman Java menggunakan *library* jsoup. Dengan *library* jsoup, dapat melakukan *request* ke layanan Google Direction menggunakan url yang telah mengandung parameter-parameter yang diperlukan.

2. Telah berhasil mengekstraksi data dengan menggunakan *library* JSON. Dengan *library* JSON, *response* dari *request* ke layanan Google Direction dapat diekstraksi dengan cara mengekstrak nilai dari *duration_in_traffic*.

3. Sesuai dengan apa yang telah dipaparkan pada subbab 5.2.2, Waktu terbaik untuk melakukan perjalanan pada sampel 1 adalah **hari rabu pukul 4**. Pada sampel 2 waktu terbaik untuk melakukan perjalanan adalah hari **hari rabu pukul 4**.

6.2 Saran

Dari hasil penelitian termasuk kesimpulan yang didapat, berikut adalah beberapa saran untuk pengembangan:

1. Penelitian ini memanfaatkan Google Direction API dengan mengembalikan *response* dengan format JSON. Oleh karena itu, sebaiknya perangkat lunak yang akan dibangun bisa menangani segala format keluaran/*response* dari layanan ini.

2. Pada parameter *APIKEY* yang terintergrasi dengan akun *Google* dimana parameter tersebut digunakan untuk melakukan *request* ke layanan Google Direction memiliki batas untuk melakukan *request*. Oleh karena itu, sebaiknya perhatikan kuota sebelum pengembang melakukan *request* atau memperbesar kuota *request* dengan membayar sejumlah uang ke pihak Google untuk berjaga-jaga terjadi melebihi batas kuota pada saat melakukan *request*.

DAFTAR REFERENSI

[1] Wong, C. (2000) *Http pocket reference: Hypertext transfer protocol*. O'Reilly Media.

[2] Hedley, J. (2016) *jsoup: Java HTML Parser*, 1.10.1 edition.

[3] Kotwal, A. (2017) *Google Direction API*. Google.

[4] International, E. (2013) *Standard ECMA- 404 - The JSON Data Interchange Format*, 1st edition.

LAMPIRAN A

KODE PROGRAM PADA *PACKAGE MODULE*

Listing A.1: DurationInTrafficExtractor.java

```
package Module;

import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import org.json.JSONArray;
import org.json.JSONObject;
import org.jsoup.Connection;
import org.jsoup.Jsoup;

public class DurationInTrafficExtractor {

    private final String DIRECTION_URL = "https://maps.googleapis.com/maps/api/directions/json";
    private final String APIKEY = "AIzaSyAtiAnSkL1NFV7wWpPvunzRgVsn4dJTaPo";
    private final String LANGUAGE = "id";
    private final String departureTime;
    private final String origin;
    private final String destination;
    private final String trafficModel;
    private int time;

    public int getTime() {
        return time;
    }

    public void setTime(int time) {
        this.time = time;
    }

    public DurationInTrafficExtractor(String unix, String origin, String destination, String
        trafficModel) {
        this.departureTime = unix;
        this.origin = origin;
        this.destination = destination;
        this.trafficModel = trafficModel;
    }

    public void extract() throws IOException {
        String content;
        JSONObject jsonResponse;
        //Connection connection = HttpConnection.connect(this.DIRECTION_URL);
        Connection connection = Jsoup.connect(this.DIRECTION_URL);
                connection.timeout(180000);
        connection.data("key", this.APIKEY);
        connection.data("origin", this.origin);
        connection.data("destination", this.destination);
        connection.data("language", this.LANGUAGE);
        connection.data("departure_time", this.departureTime);
        connection.data("traffic_model", this.trafficModel);
        connection.ignoreContentType(true);
        content = connection.execute().body();
        jsonResponse = new JSONObject(content);

        if (jsonResponse.getString("status").equals("OK")) {
            JSONArray routes = jsonResponse.getJSONArray("routes");
            JSONObject route = routes.getJSONObject(0);
            JSONArray legs = route.getJSONArray("legs");
            JSONObject leg = legs.getJSONObject(0);
            JSONObject duration = leg.getJSONObject("duration_in_traffic");
            int durationValue = Integer.parseInt(duration.optString("value"));
            if (durationValue%60 == 0) {
                this.setTime(durationValue/60);
            }else{
```

```
                this.setTime((durationValue/60)+1);
            }
        } else {
            System.err.println(jsonResponse.getString("status"));
        }
    }

    public String getDepartureTimeHours() {
        Calendar tempCal = new GregorianCalendar();
        Long tempLong = new Long(this.departureTime);
        tempLong =  tempLong*1000;
        Date tempDate = new Date(tempLong);
        tempCal.setTime(tempDate);
        return tempCal.get(Calendar.HOUR_OF_DAY) + "";
    }
}
```

Listing A.2: WholeDayExtractor.java

```
package Module;

import java.io.IOException;
import java.text.ParseException;
public class WholeDayExtractor {
    private final DurationInTrafficExtractor[] wholeDay = new DurationInTrafficExtractor[24];
    private final String[] hours = new String[24];

    public void initialize(String unix, String origin, String destination, String trafficModel)
         throws ParseException{
        Long unixDate = new Long(unix);//temp.getTimeInMillis()/1000;
        long counterUnixperHour = 3600;
        for (int i = 0; i < wholeDay.length; i++) {
            wholeDay[i] =  new DurationInTrafficExtractor(unixDate.toString(), origin, destination
                 , trafficModel);
            unixDate += counterUnixperHour;
        }
    }

    public void extract() throws IOException{
        for (int i = 0; i < wholeDay.length; i++) {
            wholeDay[i].extract();
            hours[i] =  wholeDay[i].getDepartureTimeHours();
        }
    }

    public DurationInTrafficExtractor[] getWholeDay() {
        return wholeDay;
    }

    public String[] getHours() {
        return hours;
    }
}
```

Listing A.3: WeekExtractor.java

```
package Module;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class WeekExtractor {

    private final WholeDayExtractor[] oneWeek =  new WholeDayExtractor[7];
    private final String[] day = new String[7];

    public void initialize(String date, String origin, String destination, String trafficModel)
         throws ParseException{
        SimpleDateFormat format = new SimpleDateFormat("dd MM yyyy");
        Calendar temp = new GregorianCalendar();
        Date tempDate = format.parse(date);
        temp.setTime(tempDate);
        temp.set(Calendar.HOUR_OF_DAY, 0);
        temp.set(Calendar.MINUTE, 0);
        Long unixDate = temp.getTimeInMillis()/1000;
        long counterUnixperDay = 86400;
        for (int i = 0; i < oneWeek.length; i++) {
            oneWeek[i] = new WholeDayExtractor();
            oneWeek[i].initialize(unixDate.toString(), origin, destination, trafficModel);
            unixDate = unixDate + counterUnixperDay;
```

```
        }
    }

    public void extract() throws IOException{
        for (int i = 0; i < oneWeek.length; i++) {
            oneWeek[i].extract();
            day[i] = i + 1 + "";
        }
    }

    public WholeDayExtractor[] getOneWeek() {
        return oneWeek;
    }

    public String[] getDay() {
        return day;
    }
}
```

Listing A.4: DataProcessor.java

```
package Module;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.ParseException;
import java.util.Vector;
import javax.swing.JCheckBox;
import javax.swing.JFormattedTextField;

public class DataProcessor {

    private final String csvSplitBy = ",";
    private final Vector<String> data = new Vector();
    private final Vector<String> trafficModels = new Vector<>();

    public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox
        trafficModel) throws ParseException, IOException {
        data.clear();
        trafficModels.clear();
        WeekExtractor temp = new WeekExtractor();
        temp.initialize(date.getText(), origin, destination, trafficModel.getText());
        temp.extract();
        trafficModels.add(trafficModel.getText());

        DurationInTrafficExtractor[] tempDuration;
        String[] tempHours;
        WholeDayExtractor[] tempWeek = temp.getOneWeek();
        String[] tempDays = temp.getDay();
        for (int i = 0; i < tempWeek.length && i < tempDays.length; i++) {
            tempDuration = tempWeek[i].getWholeDay();
            tempHours = tempWeek[i].getHours();
            for (int j = 0; j < tempDuration.length && j < tempHours.length; j++) {
                data.add(tempDays[i] + csvSplitBy + tempHours[j] + csvSplitBy + tempDuration[j].
                    getTime());
            }
        }
    }

    public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox
        trafficModel1, JCheckBox trafficModel2) throws ParseException, IOException {
        data.clear();
        trafficModels.clear();
        WeekExtractor temp1 = new WeekExtractor();
        WeekExtractor temp2 = new WeekExtractor();

        temp1.initialize(date.getText(), origin, destination, trafficModel1.getText());
        temp1.extract();
        temp2.initialize(date.getText(), origin, destination, trafficModel2.getText());
        temp2.extract();
        trafficModels.add(trafficModel1.getText());
        trafficModels.add(trafficModel2.getText());

        DurationInTrafficExtractor[] tempDuration1;
        DurationInTrafficExtractor[] tempDuration2;
        String[] tempHours1;
        String[] tempHours2;
        WholeDayExtractor[] tempWeek1 = temp1.getOneWeek();
        WholeDayExtractor[] tempWeek2 = temp2.getOneWeek();
        String[] tempDays1 = temp1.getDay();

        for (int i = 0; i < tempWeek1.length && i < tempDays1.length; i++) {
            tempDuration1 = tempWeek1[i].getWholeDay();
```

```
            tempDuration2 = tempWeek2[i].getWholeDay();
            tempHours1 = tempWeek1[i].getHours();
            tempHours2 = tempWeek2[i].getHours();
            for (int j = 0; j < tempDuration1.length && j < tempHours1.length; j++) {
                data.add(tempDays1[i] + csvSplitBy + tempHours1[j] + csvSplitBy + tempDuration1[j
                    ].getTime() + csvSplitBy + tempDuration2[j].getTime());
            }
        }
    }

    public void initalize(JFormattedTextField date, String origin, String destination, JCheckBox
         trafficModel1, JCheckBox trafficModel2, JCheckBox trafficModel3) throws ParseException,
         IOException {
        data.clear();
        trafficModels.clear();
        WeekExtractor temp1 = new WeekExtractor();
        WeekExtractor temp2 = new WeekExtractor();
        WeekExtractor temp3 = new WeekExtractor();

        temp1.initialize(date.getText(), origin, destination, trafficModel1.getText());
        temp1.extract();
        temp2.initialize(date.getText(), origin, destination, trafficModel2.getText());
        temp2.extract();
        temp3.initialize(date.getText(), origin, destination, trafficModel3.getText());
        temp3.extract();
        trafficModels.add(trafficModel1.getText());
        trafficModels.add(trafficModel2.getText());
        trafficModels.add(trafficModel3.getText());

        DurationInTrafficExtractor[] tempDuration1;
        DurationInTrafficExtractor[] tempDuration2;
        DurationInTrafficExtractor[] tempDuration3;
        String[] tempHours1;
        String[] tempHours2;
        String[] tempHouts3;
        WholeDayExtractor[] tempWeek1 = temp1.getOneWeek();
        WholeDayExtractor[] tempWeek2 = temp2.getOneWeek();
        WholeDayExtractor[] tempWeek3 = temp3.getOneWeek();
        String[] tempDays1 = temp1.getDay();

        for (int i = 0; i < tempWeek1.length && i < tempDays1.length; i++) {
            tempDuration1 = tempWeek1[i].getWholeDay();
            tempDuration2 = tempWeek2[i].getWholeDay();
            tempDuration3 = tempWeek3[i].getWholeDay();
            tempHours1 = tempWeek1[i].getHours();
            tempHours2 = tempWeek2[i].getHours();
            tempHouts3 = tempWeek3[i].getHours();
            for (int j = 0; j < tempDuration1.length && j < tempHours1.length; j++) {
                data.add(tempDays1[i] + csvSplitBy + tempHours1[j] + csvSplitBy + tempDuration1[j
                    ].getTime() + csvSplitBy + tempDuration2[j].getTime() + csvSplitBy +
                    tempDuration3[j].getTime());
            }
        }
    }

    public void saveFile(String directory, String fileName) throws IOException {
        File file = new File(directory + "\\" + fileName);
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
            for (String data1 : data) {
                writer.write(data1);
                writer.newLine();
            }
            writer.flush();
        }
    }
}
```

LAMPIRAN B

KODE PROGRAM PADA *PACKAGE CONTROLLER*

Listing B.1: DurationTimeController.java

```
package Controller;

import Module.DataProcessor;
import java.io.IOException;
import java.text.ParseException;
import javax.swing.JCheckBox;
import javax.swing.JFormattedTextField;

public class DurationTimeController {

    private final DataProcessor processor;

    public DurationTimeController (){
        this.processor = new DataProcessor();
    }

    public void doCalculate(JFormattedTextField date, String origin, String destination, JCheckBox
          trafficModel1, JCheckBox trafficModel2, JCheckBox trafficModel3) throws ParseException,
         IOException{
        if (trafficModel3==null) {
            if(trafficModel2==null){
                processor.initalize(date, origin, destination, trafficModel1);
            }else{
                processor.initalize(date, origin, destination, trafficModel1, trafficModel2);
            }
        }else{
            processor.initalize(date, origin, destination, trafficModel1, trafficModel2,
                 trafficModel3);
        }
    }

    public void saveData(String dir, String filename) throws IOException{
        processor.saveFile(dir, filename);
    }
}
```

LAMPIRAN C

KODE PROGRAM PADA *PACKAGE VIEW*

Listing C.1: main.java

```
package View;

import Controller.DurationTimeController;
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.text.ParseException;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class main extends javax.swing.JFrame {

    private String nameFile;
    private String directory;
    private final DurationTimeController controller;
    private final FileTypeFilter type = new FileTypeFilter(".csv", "Comma Separated Values File");
    private final JFileChooser chooser = new JFileChooser();
    private final String origin = "-6.8746025,107.6024968";
    private final String sampel1 = "-6.9536001,107.6193958";
    private final String sampel2 = "-6.937021,107.6643817";

    public main() {
        this.controller = new DurationTimeController();
        chooser.setFileFilter(type);
        initComponents();
    }

    private void initComponents() {

        buttonGroup1 = new javax.swing.ButtonGroup();
        trafficModelLabel = new javax.swing.JLabel();
        dateLabel = new javax.swing.JLabel();
        originLabel = new javax.swing.JLabel();
        bestGuessCheckBox = new javax.swing.JCheckBox();
        optimistCheckBox = new javax.swing.JCheckBox();
        pessimistCheckBox = new javax.swing.JCheckBox();
        saveButton = new javax.swing.JButton();
        sampel1RadioButton = new javax.swing.JRadioButton();
        sampel2RadioButton = new javax.swing.JRadioButton();
        datePicker = new org.jdesktop.swingx.JXDatePicker();
        datePicker.getEditor().setEditable(false);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Program Analisis Waktu Bandung");
        setResizable(false);

        trafficModelLabel.setText("Traffic Model     :");

        dateLabel.setText("Date                    :");

        originLabel.setText("Origin                  : ");

        bestGuessCheckBox.setText("best_guess");

        optimistCheckBox.setText("optimistic");

        pessimistCheckBox.setText("pessimistic");

        saveButton.setText("Save to CSV & Show");
```

```
        saveButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                saveButtonActionPerformed(evt);
            }
        });

        buttonGroup1.add(sampel1RadioButton);
        sampel1RadioButton.setText("Sampel 1 : Amaya Residence");

        buttonGroup1.add(sampel2RadioButton);
        sampel2RadioButton.setText("Sampel 2 : Jln. Puspa Utara");

        datePicker.setFormats("dd MM yyyy");
        datePicker.setLinkDay(new Date());

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.
                        createSequentialGroup()
                        .addComponent(originLabel)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
                            TRAILING)
                            .addComponent(sampel2RadioButton, javax.swing.GroupLayout.Alignment.
                                LEADING)
                            .addComponent(sampel1RadioButton, javax.swing.GroupLayout.Alignment.
                                LEADING)))
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.
                        createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(saveButton)
                        .addGroup(layout.createSequentialGroup()
                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
                                .LEADING, false)
                                .addComponent(trafficModelLabel, javax.swing.GroupLayout.
                                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                                    MAX_VALUE)
                                .addComponent(dateLabel, javax.swing.GroupLayout.DEFAULT_SIZE,
                                    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
                                .LEADING)
                                .addComponent(datePicker, javax.swing.GroupLayout.PREFERRED_SIZE,
                                    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                                    PREFERRED_SIZE)
                                .addGroup(layout.createSequentialGroup()
                                    .addComponent(bestGuessCheckBox)
                                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                                        RELATED)
                                    .addComponent(optimistCheckBox)
                                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                                        RELATED)
                                    .addComponent(pessimistCheckBox))))))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(originLabel)
                    .addComponent(sampel1RadioButton))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(sampel2RadioButton)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(dateLabel)
                    .addComponent(datePicker, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.
                        GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(6, 6, 6)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(trafficModelLabel)
                    .addComponent(optimistCheckBox)
                    .addComponent(pessimistCheckBox)
                    .addComponent(bestGuessCheckBox))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(saveButton)
                .addContainerGap(18, Short.MAX_VALUE))
        );

        pack();
```

```
    }// </editor-fold>

    private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            if (!datePicker.getEditor().getText().equals("")) {
                Calendar calendar = new GregorianCalendar();
                calendar.setTime(datePicker.getDate());
                long temp = calendar.getTime().getTime() - datePicker.getLinkDay().getTime();
                if (calendar.get(Calendar.DAY_OF_WEEK) != 2) {
                    System.out.println(calendar.get(Calendar.DAY_OF_WEEK));
                    JOptionPane.showMessageDialog(this, "Tanggal yang anda pilih bukan hari senin
                        ", "ERROR", JOptionPane.ERROR_MESSAGE);
                } else {
                    if (temp <= 0) {
                        JOptionPane.showMessageDialog(this, "Tanggal yang anda masukan adalah masa
                            lampau atau hari ini, Pilihlah tanggal yang akan datang", "ERROR",
                            JOptionPane.ERROR_MESSAGE);
                    } else {
                        if (sampel1RadioButton.isSelected()) {
                            if (bestGuessCheckBox.isSelected()) {
                                if (optimistCheckBox.isSelected()) {
                                    if (pessimistCheckBox.isSelected()) {
                                        controller.doCalculate(datePicker.getEditor(), origin,
                                            sampel1, bestGuessCheckBox, optimistCheckBox,
                                            pessimistCheckBox);
                                        int result = chooser.showSaveDialog(this);
                                        if (result == JFileChooser.APPROVE_OPTION) {
                                            Desktop dt = Desktop.getDesktop();
                                            nameFile = chooser.getSelectedFile().getName() + type.
                                                getExtension();
                                            directory = chooser.getCurrentDirectory().toString();
                                            controller.saveData(directory, nameFile);
                                            File file = new File(directory + "\\" + nameFile);
                                            dt.open(file);
                                        }
                                    } else {
                                        controller.doCalculate(datePicker.getEditor(), origin,
                                            sampel1, bestGuessCheckBox, optimistCheckBox, null);
                                        int result = chooser.showSaveDialog(this);
                                        if (result == JFileChooser.APPROVE_OPTION) {
                                            Desktop dt = Desktop.getDesktop();
                                            nameFile = chooser.getSelectedFile().getName() + type.
                                                getExtension();
                                            directory = chooser.getCurrentDirectory().toString();
                                            controller.saveData(directory, nameFile);
                                            File file = new File(directory + "\\" + nameFile);
                                            dt.open(file);
                                        }
                                    }
                                } else if (pessimistCheckBox.isSelected()) {
                                    controller.doCalculate(datePicker.getEditor(), origin, sampel1
                                        , bestGuessCheckBox, pessimistCheckBox, null);
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
                                        directory = chooser.getCurrentDirectory().toString();
                                        controller.saveData(directory, nameFile);
                                        File file = new File(directory + "\\" + nameFile);
                                        dt.open(file);
                                    }
                                } else {
                                    controller.doCalculate(datePicker.getEditor(), origin, sampel1
                                        , bestGuessCheckBox, null, null);
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
                                        directory = chooser.getCurrentDirectory().toString();
                                        controller.saveData(directory, nameFile);
                                        File file = new File(directory + "\\" + nameFile);
                                        dt.open(file);
                                    }
                                }
                            } else if (optimistCheckBox.isSelected()) {
                                if (pessimistCheckBox.isSelected()) {
                                    controller.doCalculate(datePicker.getEditor(), origin, sampel1
                                        , optimistCheckBox, pessimistCheckBox, null);
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
```

```
                                directory = chooser.getCurrentDirectory().toString();
                                controller.saveData(directory, nameFile);
                                File file = new File(directory + "\\" + nameFile);
                                dt.open(file);
                            }
                        } else {
                            controller.doCalculate(datePicker.getEditor(), origin, sampel1
                                , optimistCheckBox, null, null);
                            int result = chooser.showSaveDialog(this);
                            if (result == JFileChooser.APPROVE_OPTION) {
                                Desktop dt = Desktop.getDesktop();
                                nameFile = chooser.getSelectedFile().getName() + type.
                                    getExtension();
                                directory = chooser.getCurrentDirectory().toString();
                                controller.saveData(directory, nameFile);
                                File file = new File(directory + "\\" + nameFile);
                                dt.open(file);
                            }
                        }
                    } else if (pessimistCheckBox.isSelected()) {
                        controller.doCalculate(datePicker.getEditor(), origin, sampel1,
                            pessimistCheckBox, null, null);
                        int result = chooser.showSaveDialog(this);
                        if (result == JFileChooser.APPROVE_OPTION) {
                            Desktop dt = Desktop.getDesktop();
                            nameFile = chooser.getSelectedFile().getName() + type.
                                getExtension();
                            directory = chooser.getCurrentDirectory().toString();
                            controller.saveData(directory, nameFile);
                            File file = new File(directory + "\\" + nameFile);
                            dt.open(file);
                        }
                    } else {
                        JOptionPane.showMessageDialog(this, "Anda harus memilih minimal
                            salah satu dari 3 traffic model yang telah disediakan", "ERROR
                            ", JOptionPane.ERROR_MESSAGE);
                    }
                } else if (sampel2RadioButton.isSelected()) {
                    if (bestGuessCheckBox.isSelected()) {
                        if (optimistCheckBox.isSelected()) {
                            if (pessimistCheckBox.isSelected()) {
                                controller.doCalculate(datePicker.getEditor(), origin,
                                    sampel2, bestGuessCheckBox, optimistCheckBox,
                                    pessimistCheckBox);
                                int result = chooser.showSaveDialog(this);
                                if (result == JFileChooser.APPROVE_OPTION) {
                                    Desktop dt = Desktop.getDesktop();
                                    nameFile = chooser.getSelectedFile().getName() + type.
                                        getExtension();
                                    directory = chooser.getCurrentDirectory().toString();
                                    controller.saveData(directory, nameFile);
                                    File file = new File(directory + "\\" + nameFile);
                                    dt.open(file);
                                }
                            } else {
                                controller.doCalculate(datePicker.getEditor(), origin,
                                    sampel2, bestGuessCheckBox, optimistCheckBox, null);
                                int result = chooser.showSaveDialog(this);
                                if (result == JFileChooser.APPROVE_OPTION) {
                                    Desktop dt = Desktop.getDesktop();
                                    nameFile = chooser.getSelectedFile().getName() + type.
                                        getExtension();
                                    directory = chooser.getCurrentDirectory().toString();
                                    controller.saveData(directory, nameFile);
                                    File file = new File(directory + "\\" + nameFile);
                                    dt.open(file);
                                }
                            }
                        } else if (pessimistCheckBox.isSelected()) {
                            controller.doCalculate(datePicker.getEditor(), origin, sampel2
                                , bestGuessCheckBox, pessimistCheckBox, null);
                            int result = chooser.showSaveDialog(this);
                            if (result == JFileChooser.APPROVE_OPTION) {
                                Desktop dt = Desktop.getDesktop();
                                nameFile = chooser.getSelectedFile().getName() + type.
                                    getExtension();
                                directory = chooser.getCurrentDirectory().toString();
                                controller.saveData(directory, nameFile);
                                File file = new File(directory + "\\" + nameFile);
                                dt.open(file);
                            }
                        } else {
                            controller.doCalculate(datePicker.getEditor(), origin, sampel2
                                , bestGuessCheckBox, null, null);

```

```
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
                                        directory = chooser.getCurrentDirectory().toString();
                                        controller.saveData(directory, nameFile);
                                        File file = new File(directory + "\\" + nameFile);
                                        dt.open(file);
                                    }
                                }
                            } else if (optimistCheckBox.isSelected()) {
                                if (pessimistCheckBox.isSelected()) {
                                    controller.doCalculate(datePicker.getEditor(), origin, sampel2
                                        , optimistCheckBox, pessimistCheckBox, null);
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
                                        directory = chooser.getCurrentDirectory().toString();
                                        controller.saveData(directory, nameFile);
                                        File file = new File(directory + "\\" + nameFile);
                                        dt.open(file);
                                    }
                                } else {
                                    controller.doCalculate(datePicker.getEditor(), origin, sampel2
                                        , optimistCheckBox, null, null);
                                    int result = chooser.showSaveDialog(this);
                                    if (result == JFileChooser.APPROVE_OPTION) {
                                        Desktop dt = Desktop.getDesktop();
                                        nameFile = chooser.getSelectedFile().getName() + type.
                                            getExtension();
                                        directory = chooser.getCurrentDirectory().toString();
                                        controller.saveData(directory, nameFile);
                                        File file = new File(directory + "\\" + nameFile);
                                        dt.open(file);
                                    }
                                }
                            } else if (pessimistCheckBox.isSelected()) {
                                controller.doCalculate(datePicker.getEditor(), origin, sampel2,
                                    pessimistCheckBox, null, null);
                                int result = chooser.showSaveDialog(this);
                                if (result == JFileChooser.APPROVE_OPTION) {
                                    Desktop dt = Desktop.getDesktop();
                                    nameFile = chooser.getSelectedFile().getName() + type.
                                        getExtension();
                                    directory = chooser.getCurrentDirectory().toString();
                                    controller.saveData(directory, nameFile);
                                    File file = new File(directory + "\\" + nameFile);
                                    dt.open(file);
                                }
                            } else {
                                JOptionPane.showMessageDialog(this, "Harus pilih minimal salah
                                    satu dari 3", "ERROR", JOptionPane.ERROR_MESSAGE);
                            }
                        } else {
                            JOptionPane.showMessageDialog(this, "Harus pilih antara sampel 1 atau
                                sampel 2", "ERROR", JOptionPane.ERROR_MESSAGE);
                        }
                    }
                }
            } else {
                JOptionPane.showMessageDialog(this, "Pilihlah tanggal", "ERROR", JOptionPane.
                    ERROR_MESSAGE);
            }
        } catch (ParseException | IOException ex) {
            Logger.getLogger(main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.
                getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException | javax.
            swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(main.class.getName()).log(java.util.logging.Level.
                SEVERE, null, ex);
        }
```

```
1
2           java.awt.EventQueue.invokeLater(new Runnable() {
3               @Override
4               public void run() {
5                   new main().setVisible(true);
6               }
7           });
8       }
9
10      private javax.swing.JCheckBox bestGuessCheckBox;
11      private javax.swing.ButtonGroup buttonGroup1;
12      private javax.swing.JLabel dateLabel;
13      private org.jdesktop.swingx.JXDatePicker datePicker;
14      private javax.swing.JCheckBox optimistCheckBox;
15      private javax.swing.JLabel originLabel;
16      private javax.swing.JCheckBox pessimistCheckBox;
17      private javax.swing.JRadioButton sampel1RadioButton;
18      private javax.swing.JRadioButton sampel2RadioButton;
19      private javax.swing.JButton saveButton;
20      private javax.swing.JLabel trafficModelLabel;
21  }
```

Listing C.2: FileTypeFilter.java

```
22  package View;
23
24  import java.io.File;
25  import javax.swing.filechooser.*;
26
27  public class FileTypeFilter extends FileFilter{
28
29      private final String extension;
30      private final String description;
31
32      public FileTypeFilter(String ext, String desc){
33          extension = ext;
34          description = desc;
35      }
36
37      @Override
38      public boolean accept(File f) {
39          if (f.isDirectory()) {
40              return true;
41          }else{
42              return f.getName().endsWith(extension);
43          }
44      }
45
46      @Override
47      public String getDescription() {
48          return description + String.format(" (*%s)", extension);
49      }
50
51      public String getExtension() {
52          return extension;
53      }
54  }
```

LAMPIRAN D

DATA HASIL PENGUJIAN

Pengujian perangkat lunak analisis waktu tempuh kota Bandung mengguji ekstraksi data dari tempat asal ke tujuan. Sampel yang digunakan adalah dua sampel yaitu : tujuan asal Amaya Residence ke Universitas Katolik Parahyangan dan tujuan asal Jalan Puspa Utara ke Universitas Katolik Parahyangan. Pengujian ini dilakukan dengan menggunakan input tanggal 8 Mei 2017 dan 15 Mei 2017. Berikut adalah data hasil ekstraksi waktu tempuh dari kedua sampel tersebut :

1	0	33	28	42
1	1	32	29	40
1	2	32	29	37
1	3	31	28	36
1	4	31	27	36
1	5	32	28	38
1	6	36	30	47
1	7	39	31	55
1	8	39	32	54
1	9	40	33	57
1	10	43	35	63
1	11	44	36	66
1	12	44	35	64
1	13	45	36	69
1	14	48	37	72
1	15	49	38	73
1	16	49	39	77
1	17	53	42	78
1	18	46	37	67
1	19	43	35	61
1	20	41	34	57
1	21	40	33	54
1	22	38	31	49
1	23	35	30	45
2	0	34	29	42
2	1	32	28	40
2	2	31	29	37

2	3	31	28	36
2	4	31	28	35
2	5	32	28	37
2	6	36	30	47
2	7	39	32	55
2	8	39	32	55
2	9	40	33	56
2	10	43	34	62
2	11	44	35	63
2	12	45	36	65
2	13	46	37	71
2	14	48	38	71
2	15	48	38	71
2	16	50	39	76
2	17	54	42	79
2	18	46	38	67
2	19	44	36	63
2	20	42	35	59
2	21	41	34	55
2	22	38	32	50
2	23	35	30	45
3	0	34	29	42
3	1	33	29	40
3	2	32	29	37
3	3	31	28	36
3	4	30	28	34
3	5	33	29	38
3	6	36	30	49
3	7	39	32	56
3	8	40	32	56
3	9	41	33	58
3	10	44	35	63
3	11	44	35	65
3	12	44	36	64
3	13	47	37	70
3	14	48	38	73
3	15	50	39	73
3	16	50	40	79
3	17	54	43	82
3	18	48	39	69
3	19	44	36	63
3	20	43	35	59

3	21	41	34	59
3	22	38	32	51
3	23	35	30	46
4	0	34	29	42
4	1	33	29	41
4	2	32	28	37
4	3	32	29	37
4	4	31	28	36
4	5	33	29	38
4	6	37	30	48
4	7	39	32	56
4	8	40	33	55
4	9	41	33	58
4	10	44	35	63
4	11	45	36	66
4	12	45	36	66
4	13	47	37	71
4	14	49	39	74
4	15	49	39	72
4	16	50	40	77
4	17	53	42	78
4	18	46	37	68
4	19	45	37	64
4	20	43	35	61
4	21	42	34	57
4	22	38	32	51
4	23	35	30	46
5	0	33	29	42
5	1	33	28	42
5	2	32	29	39
5	3	32	28	37
5	4	31	28	36
5	5	33	29	39
5	6	36	30	48
5	7	39	32	56
5	8	40	33	57
5	9	41	33	59
5	10	44	35	64
5	11	46	37	68
5	12	41	33	57
5	13	46	36	71
5	14	50	39	75

5	15	49	39	74
5	16	52	40	82
5	17	57	44	85
5	18	51	40	74
5	19	48	38	69
5	20	45	36	66
5	21	44	36	61
5	22	41	34	57
5	23	37	31	49
6	0	34	30	44
6	1	34	29	42
6	2	32	28	40
6	3	32	28	38
6	4	32	28	40
6	5	33	28	41
6	6	34	29	45
6	7	36	30	49
6	8	38	31	53
6	9	40	33	57
6	10	43	35	63
6	11	44	35	65
6	12	46	36	70
6	13	48	38	73
6	14	52	41	79
6	15	50	40	76
6	16	50	40	76
6	17	51	41	76
6	18	51	41	76
6	19	52	42	76
6	20	53	42	77
6	21	51	41	75
6	22	48	39	66
6	23	41	35	57
7	0	36	31	47
7	1	34	29	43
7	2	33	29	40
7	3	32	29	39
7	4	31	28	38
7	5	33	29	39
7	6	34	28	44
7	7	35	29	46
7	8	38	31	52

7	9	40	33	56
7	10	45	36	65
7	11	47	37	70
7	12	49	38	73
7	13	51	40	76
7	14	51	41	78
7	15	51	41	77
7	16	50	40	74
7	17	50	41	73
7	18	49	40	71
7	19	48	39	68
7	20	46	37	66
7	21	43	36	60
7	22	39	32	52
7	23	35	30	46

Tabel D.1: Data sampel 1 pada tanggal 8 Mei 2017 - 14 Mei 2017

1	0	33	28	42
1	1	32	29	40
1	2	32	29	37
1	3	31	28	36
1	4	31	27	36
1	5	32	28	38
1	6	36	30	47
1	7	39	31	55
1	8	39	32	54
1	9	40	33	57
1	10	43	35	63
1	11	44	36	66
1	12	44	35	64
1	13	45	36	69
1	14	48	37	72
1	15	49	38	73
1	16	49	39	77
1	17	53	42	78
1	18	46	37	67
1	19	43	35	61
1	20	41	34	57
1	21	40	33	54
1	22	38	31	49
1	23	35	30	45

2	0	34	29	42
2	1	32	28	40
2	2	31	29	37
2	3	31	28	36
2	4	31	28	35
2	5	32	28	37
2	6	36	30	47
2	7	39	32	55
2	8	39	32	55
2	9	40	33	56
2	10	43	34	62
2	11	44	35	63
2	12	45	36	65
2	13	46	37	71
2	14	48	38	71
2	15	48	38	71
2	16	50	39	76
2	17	54	42	79
2	18	46	38	67
2	19	44	36	63
2	20	42	35	59
2	21	41	34	55
2	22	38	32	50
2	23	35	30	45
3	0	34	29	42
3	1	33	29	40
3	2	32	29	37
3	3	31	28	36
3	4	30	28	34
3	5	33	29	38
3	6	36	30	49
3	7	39	32	56
3	8	40	32	56
3	9	41	33	58
3	10	44	35	63
3	11	44	35	65
3	12	44	36	64
3	13	47	37	70
3	14	48	38	73
3	15	50	39	73
3	16	50	40	79
3	17	54	43	82

3	18	48	39	69
3	19	44	36	63
3	20	43	35	59
3	21	41	34	59
3	22	38	32	51
3	23	35	30	46
4	0	34	29	42
4	1	33	29	41
4	2	32	28	37
4	3	32	29	37
4	4	31	28	36
4	5	33	29	38
4	6	37	30	48
4	7	39	32	56
4	8	40	33	55
4	9	41	33	58
4	10	44	35	63
4	11	45	36	66
4	12	45	36	66
4	13	47	37	71
4	14	49	39	74
4	15	49	39	72
4	16	50	40	77
4	17	53	42	78
4	18	46	37	68
4	19	45	37	64
4	20	43	35	61
4	21	42	34	57
4	22	38	32	51
4	23	35	30	46
5	0	33	29	42
5	1	33	28	42
5	2	32	29	39
5	3	32	28	37
5	4	31	28	36
5	5	33	29	39
5	6	36	30	48
5	7	39	32	56
5	8	40	33	57
5	9	41	33	59
5	10	44	35	64
5	11	46	37	68

5	12	41	33	57
5	13	46	36	71
5	14	50	39	75
5	15	49	39	74
5	16	52	40	82
5	17	57	44	85
5	18	51	40	74
5	19	48	38	69
5	20	45	36	66
5	21	44	36	61
5	22	41	34	57
5	23	37	31	49
6	0	34	30	44
6	1	34	29	42
6	2	32	28	40
6	3	32	28	38
6	4	32	28	40
6	5	33	28	41
6	6	34	29	45
6	7	36	30	49
6	8	38	31	53
6	9	40	33	57
6	10	43	35	63
6	11	44	35	65
6	12	46	36	70
6	13	48	38	73
6	14	52	41	79
6	15	50	40	76
6	16	50	40	76
6	17	51	41	76
6	18	51	41	76
6	19	52	42	76
6	20	53	42	77
6	21	51	41	75
6	22	48	39	66
6	23	41	35	57
7	0	36	31	47
7	1	34	29	43
7	2	33	29	40
7	3	32	29	39
7	4	31	28	38
7	5	33	29	39

7	6	34	28	44
7	7	35	29	46
7	8	38	31	52
7	9	40	33	56
7	10	45	36	65
7	11	47	37	70
7	12	49	38	73
7	13	51	40	76
7	14	51	41	78
7	15	51	41	77
7	16	50	40	74
7	17	50	41	73
7	18	49	40	71
7	19	48	39	68
7	20	46	37	66
7	21	43	36	60
7	22	39	32	52
7	23	35	30	46

Tabel D.2: Data sampel 1 pada tanggal 15 Mei 2017 - 21 Mei 2017

1	0	31	27	40
1	1	31	27	37
1	2	30	28	35
1	3	30	28	34
1	4	29	26	34
1	5	31	27	37
1	6	37	30	50
1	7	38	32	51
1	8	37	31	50
1	9	38	31	52
1	10	41	33	59
1	11	43	35	61
1	12	42	34	60
1	13	42	34	60
1	14	45	36	67
1	15	47	37	68
1	16	50	38	78
1	17	54	42	82
1	18	49	39	72
1	19	42	35	61
1	20	39	32	54

1	21	37	31	49
1	22	35	29	47
1	23	33	28	43
2	0	31	27	39
2	1	30	27	37
2	2	30	27	35
2	3	30	27	34
2	4	30	27	35
2	5	31	27	36
2	6	37	30	50
2	7	37	31	51
2	8	37	31	52
2	9	38	31	52
2	10	40	33	56
2	11	41	34	58
2	12	42	34	58
2	13	43	34	61
2	14	46	37	68
2	15	47	37	69
2	16	50	39	78
2	17	54	42	84
2	18	49	40	72
2	19	45	36	64
2	20	40	33	56
2	21	38	32	51
2	22	35	30	47
2	23	33	28	43
3	0	31	27	39
3	1	31	28	37
3	2	31	28	35
3	3	30	27	34
3	4	30	27	35
3	5	31	27	37
3	6	37	30	51
3	7	38	31	51
3	8	38	32	54
3	9	39	32	54
3	10	41	34	60
3	11	43	34	61
3	12	41	34	60
3	13	44	36	64
3	14	48	39	75

3	15	51	40	76
3	16	53	42	83
3	17	56	44	88
3	18	51	41	75
3	19	45	37	64
3	20	40	34	55
3	21	39	33	54
3	22	36	30	47
3	23	33	28	42
4	0	31	27	40
4	1	31	27	40
4	2	30	27	35
4	3	30	28	35
4	4	30	27	35
4	5	31	27	36
4	6	37	31	50
4	7	38	32	53
4	8	38	32	52
4	9	40	33	55
4	10	42	34	60
4	11	43	35	63
4	12	44	35	64
4	13	45	36	66
4	14	49	38	75
4	15	49	39	76
4	16	51	40	81
4	17	56	43	85
4	18	49	39	72
4	19	45	37	63
4	20	41	34	57
4	21	39	33	53
4	22	36	31	49
4	23	33	28	43
5	0	31	27	39
5	1	31	27	38
5	2	30	28	36
5	3	30	27	35
5	4	30	27	35
5	5	31	27	36
5	6	36	29	50
5	7	37	31	51
5	8	38	32	53

5	9	38	32	54
5	10	42	34	60
5	11	45	35	64
5	12	38	31	53
5	13	44	36	63
5	14	51	40	79
5	15	52	41	82
5	16	54	42	86
5	17	58	45	91
5	18	54	42	80
5	19	50	40	73
5	20	45	37	66
5	21	44	36	63
5	22	41	33	56
5	23	35	29	46
6	0	32	28	41
6	1	32	27	40
6	2	31	27	39
6	3	30	27	37
6	4	31	27	38
6	5	31	27	40
6	6	33	28	42
6	7	34	29	46
6	8	37	31	50
6	9	39	32	53
6	10	41	34	59
6	11	44	37	64
6	12	46	37	71
6	13	51	39	78
6	14	53	41	82
6	15	51	41	76
6	16	49	39	72
6	17	49	39	73
6	18	48	39	70
6	19	50	40	72
6	20	53	43	79
6	21	53	42	77
6	22	48	39	68
6	23	40	33	55
7	0	33	28	44
7	1	32	27	40
7	2	31	27	38

7	3	31	28	38
7	4	30	27	37
7	5	32	28	37
7	6	32	28	42
7	7	36	30	47
7	8	39	32	55
7	9	42	34	59
7	10	46	37	69
7	11	50	39	76
7	12	53	41	82
7	13	56	43	85
7	14	54	42	79
7	15	47	39	68
7	16	46	38	67
7	17	48	39	68
7	18	46	38	67
7	19	46	38	65
7	20	44	37	62
7	21	42	35	58
7	22	37	31	49
7	23	33	28	42

Tabel D.3: Data sampel 2 pada tanggal 15 Mei 2017 - 21 Mei 2017

1	0	31	27	40
1	1	31	27	37
1	2	30	28	35
1	3	30	28	34
1	4	29	26	34
1	5	31	27	37
1	6	37	30	50
1	7	38	32	51
1	8	37	31	50
1	9	38	31	52
1	10	41	33	59
1	11	43	35	61
1	12	42	34	60
1	13	42	34	60
1	14	45	36	67
1	15	47	37	68
1	16	50	38	78
1	17	54	42	82

1	18	49	39	72
1	19	42	35	61
1	20	39	32	54
1	21	37	31	49
1	22	35	29	47
1	23	33	28	43
2	0	31	27	39
2	1	30	27	37
2	2	30	27	35
2	3	30	27	34
2	4	30	27	35
2	5	31	27	36
2	6	37	30	50
2	7	37	31	51
2	8	37	31	52
2	9	38	31	52
2	10	40	33	56
2	11	41	34	58
2	12	42	34	58
2	13	43	34	61
2	14	46	37	68
2	15	47	37	69
2	16	50	39	78
2	17	54	42	84
2	18	49	40	72
2	19	45	36	64
2	20	40	33	56
2	21	38	32	51
2	22	35	30	47
2	23	33	28	43
3	0	31	27	39
3	1	31	28	37
3	2	31	28	35
3	3	30	27	34
3	4	30	27	35
3	5	31	27	37
3	6	37	30	51
3	7	38	31	51
3	8	38	32	54
3	9	39	32	54
3	10	41	34	60
3	11	43	34	61

3	12	41	34	60
3	13	44	36	64
3	14	48	39	75
3	15	51	40	76
3	16	53	42	83
3	17	56	44	88
3	18	51	41	75
3	19	45	37	64
3	20	40	34	55
3	21	39	33	54
3	22	36	30	47
3	23	33	28	42
4	0	31	27	40
4	1	31	27	40
4	2	30	27	35
4	3	30	28	35
4	4	30	27	35
4	5	31	27	36
4	6	37	31	50
4	7	38	32	53
4	8	38	32	52
4	9	40	33	55
4	10	42	34	60
4	11	43	35	63
4	12	44	35	64
4	13	45	36	66
4	14	49	38	75
4	15	49	39	76
4	16	51	40	81
4	17	56	43	85
4	18	49	39	72
4	19	45	37	63
4	20	41	34	57
4	21	39	33	53
4	22	36	31	49
4	23	33	28	43
5	0	31	27	39
5	1	31	27	38
5	2	30	28	36
5	3	30	27	35
5	4	30	27	35
5	5	31	27	36

5	6	36	29	50
5	7	37	31	51
5	8	38	32	53
5	9	38	32	54
5	10	42	34	60
5	11	45	35	64
5	12	38	31	53
5	13	44	36	63
5	14	51	40	79
5	15	52	41	82
5	16	54	42	86
5	17	58	45	91
5	18	54	42	80
5	19	50	40	73
5	20	45	37	66
5	21	44	36	63
5	22	41	33	56
5	23	35	29	46
6	0	32	28	41
6	1	32	27	40
6	2	31	27	39
6	3	30	27	37
6	4	31	27	38
6	5	31	27	40
6	6	33	28	42
6	7	34	29	46
6	8	37	31	50
6	9	39	32	53
6	10	41	34	59
6	11	44	37	64
6	12	46	37	71
6	13	51	39	78
6	14	53	41	82
6	15	51	41	76
6	16	49	39	72
6	17	49	39	73
6	18	48	39	70
6	19	50	40	72
6	20	53	43	79
6	21	53	42	77
6	22	48	39	68
6	23	40	33	55

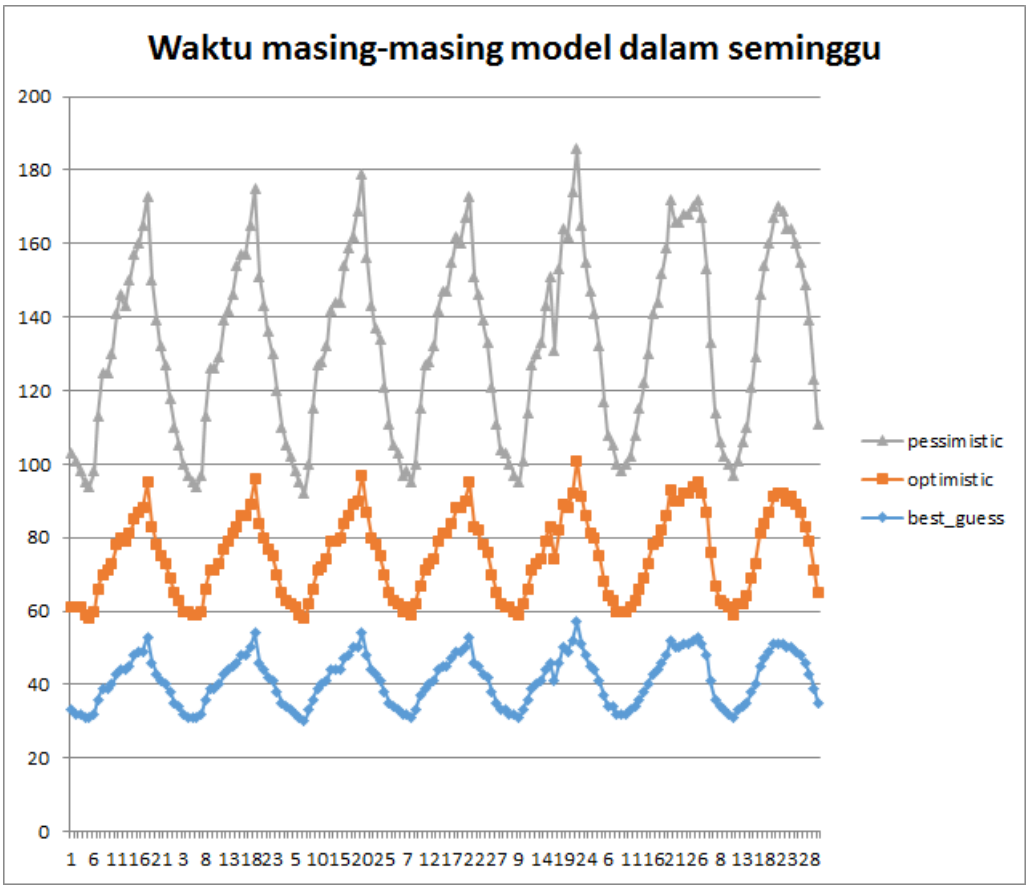
7	0	33	28	44
7	1	32	27	40
7	2	31	27	38
7	3	31	28	38
7	4	30	27	37
7	5	32	28	37
7	6	32	28	42
7	7	36	30	47
7	8	39	32	55
7	9	42	34	59
7	10	46	37	69
7	11	50	39	76
7	12	53	41	82
7	13	56	43	85
7	14	54	42	79
7	15	47	39	68
7	16	46	38	67
7	17	48	39	68
7	18	46	38	67
7	19	46	38	65
7	20	44	37	62
7	21	42	35	58
7	22	37	31	49
7	23	33	28	42

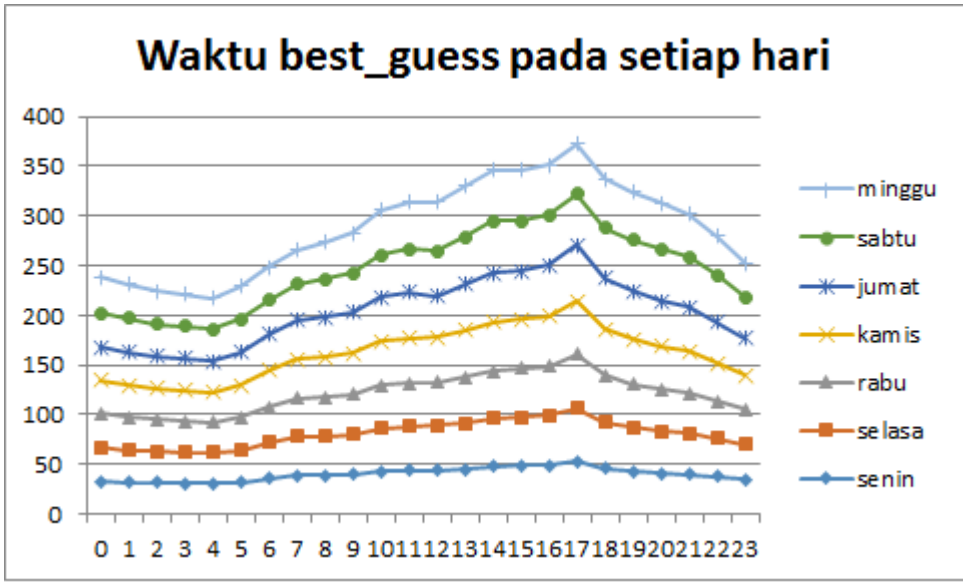
Tabel D.4: Data sampel 2 pada tanggal 8 Mei 2017 - 14 Mei 2017

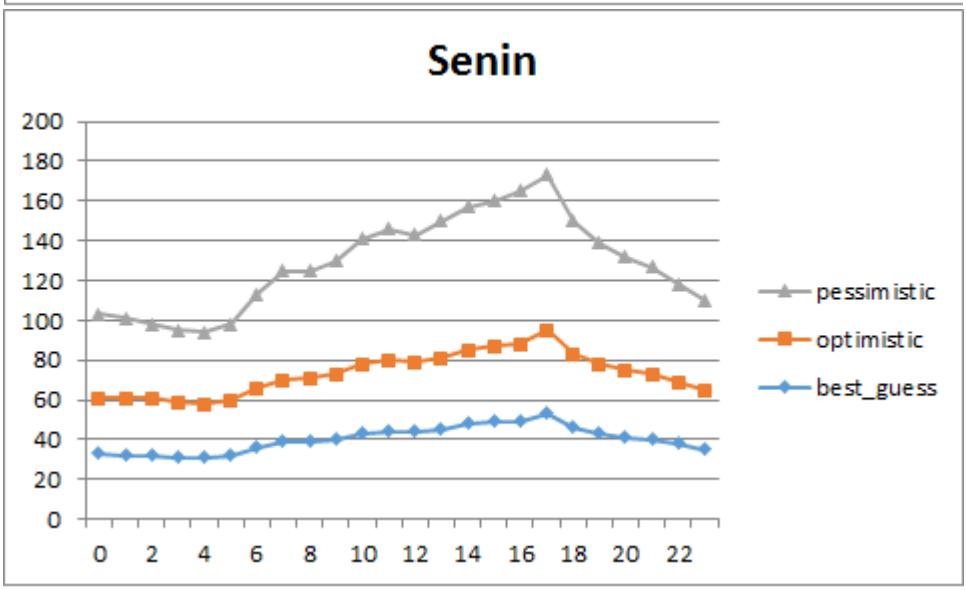
LAMPIRAN E

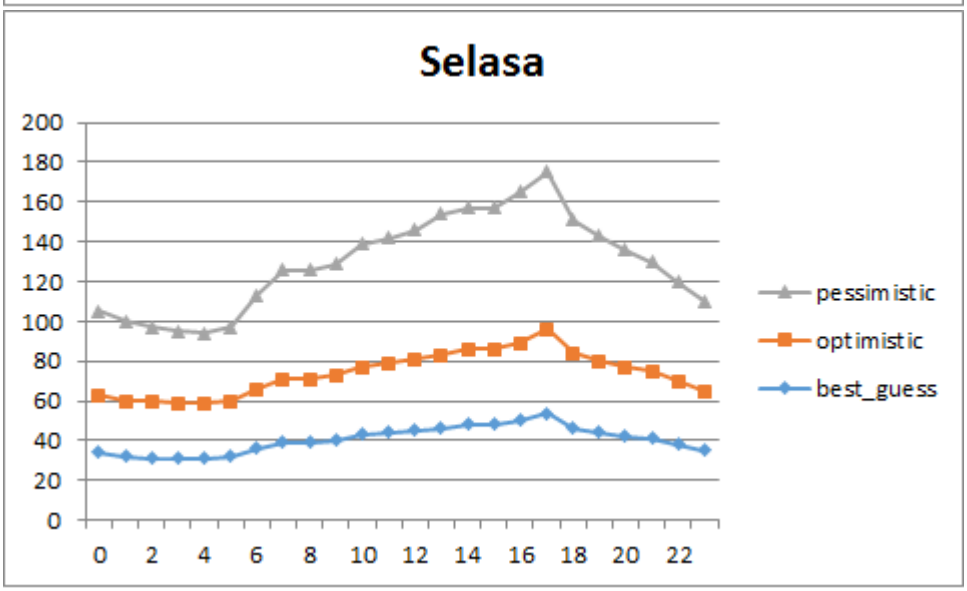
HASIL PENGUJIAN EKSPERIMENTAL

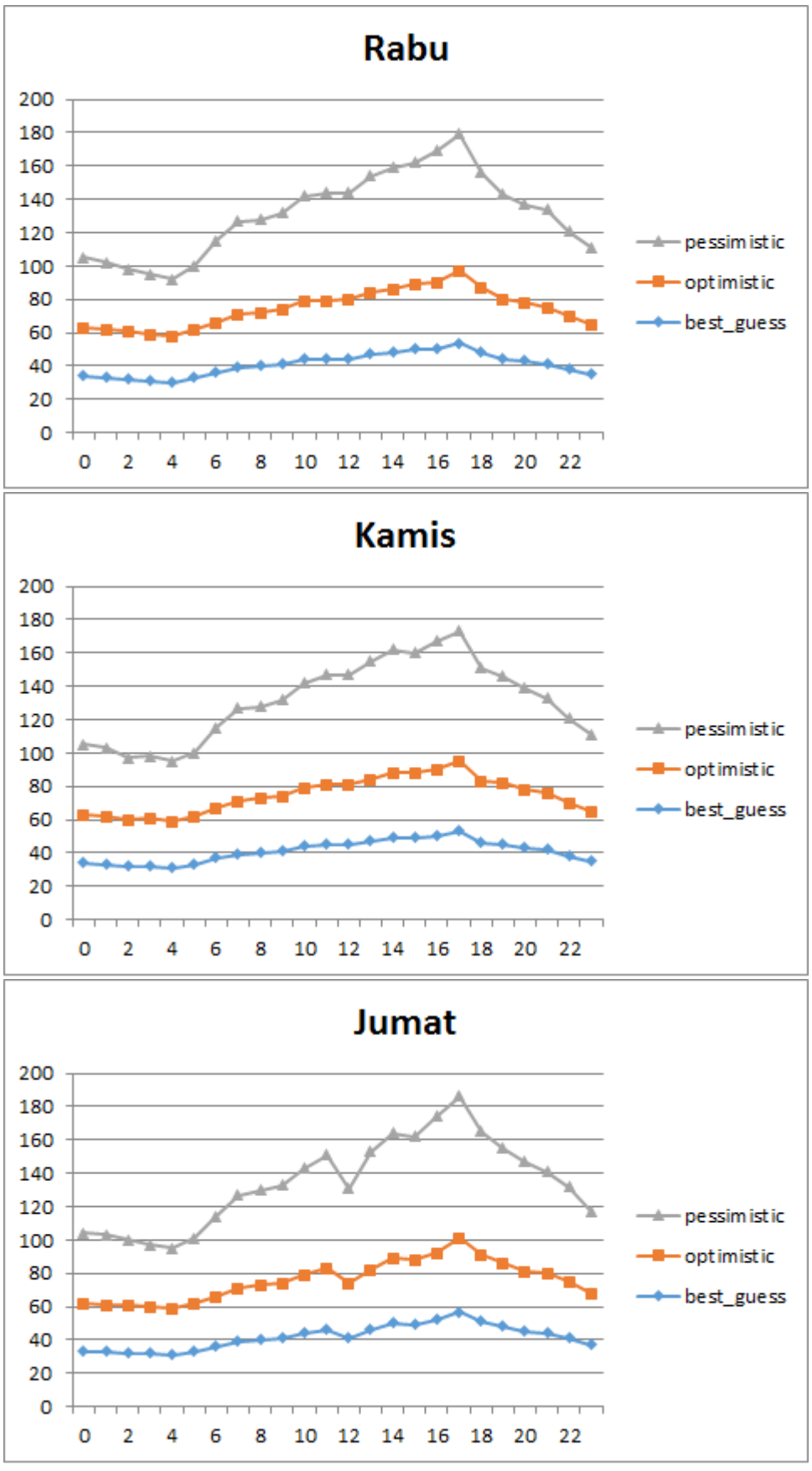
Hasil pengujian eksperimental ini berupa bagan yang dibuat dari data pada Lampiran D. Berikut adalah bagan hasil dari pengujian eksperimental :

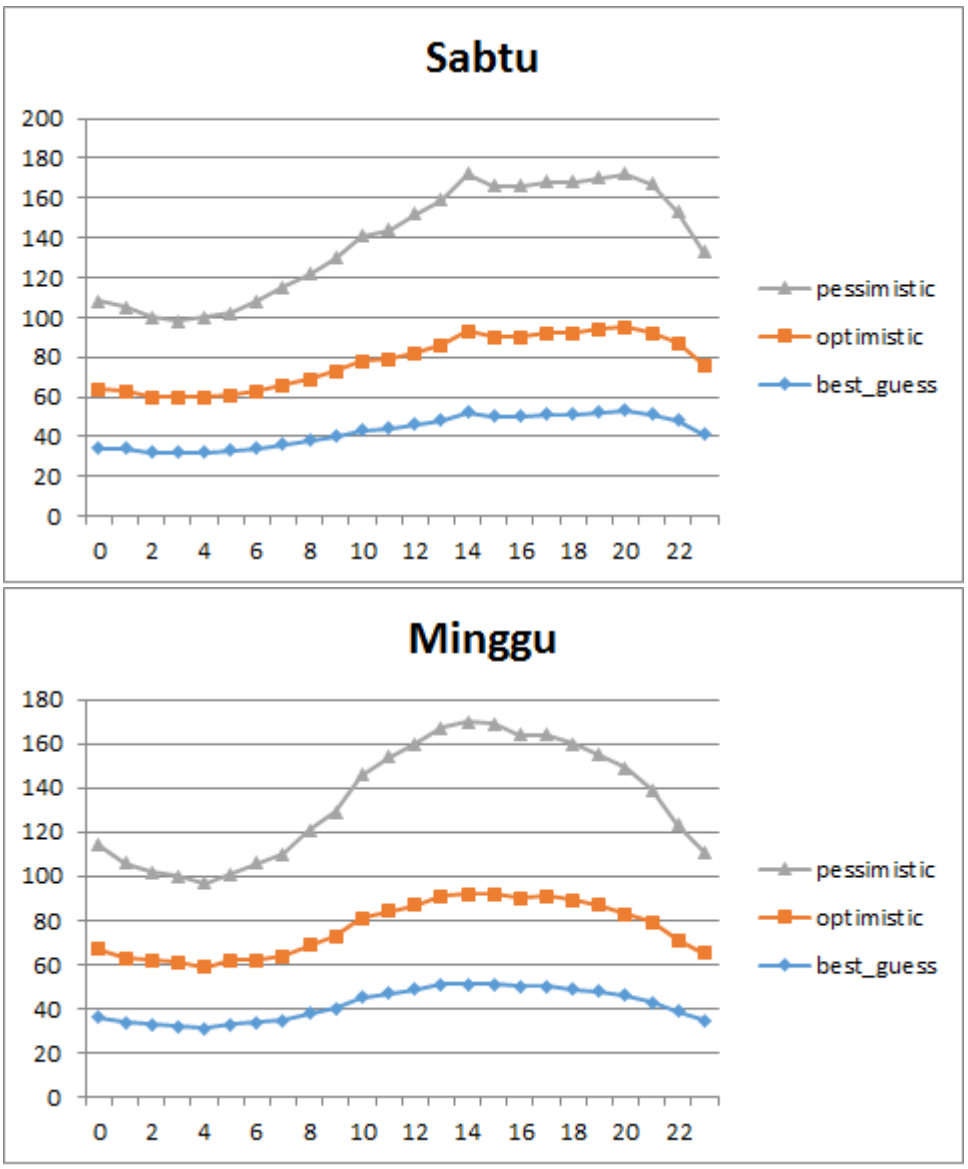



Waktu best_guess pada setiap hari
400
350
300
250
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
minggu
sabtu
jumat
kamis
rabu
selasa
senin

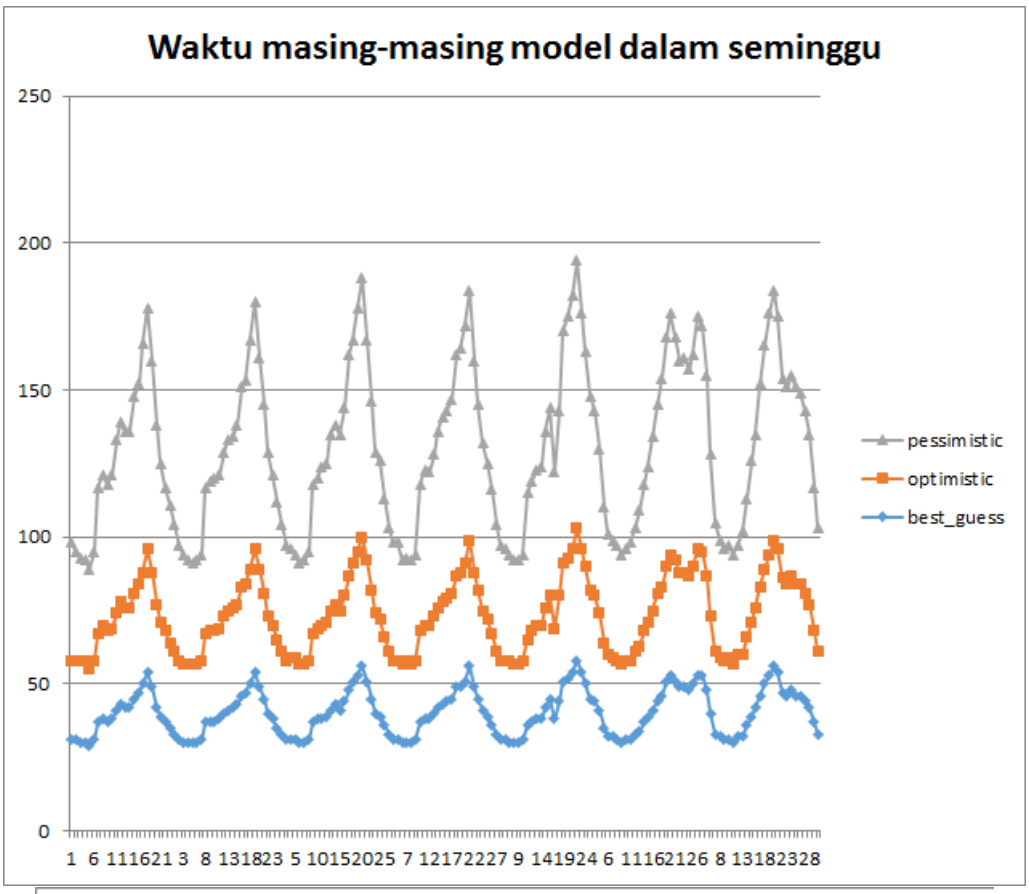

Senin
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

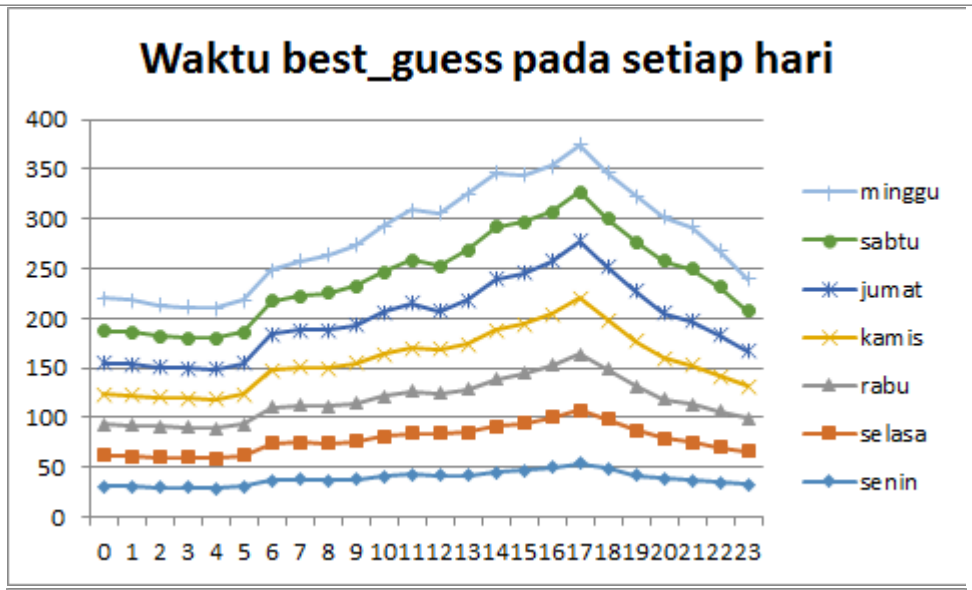

Selasa
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

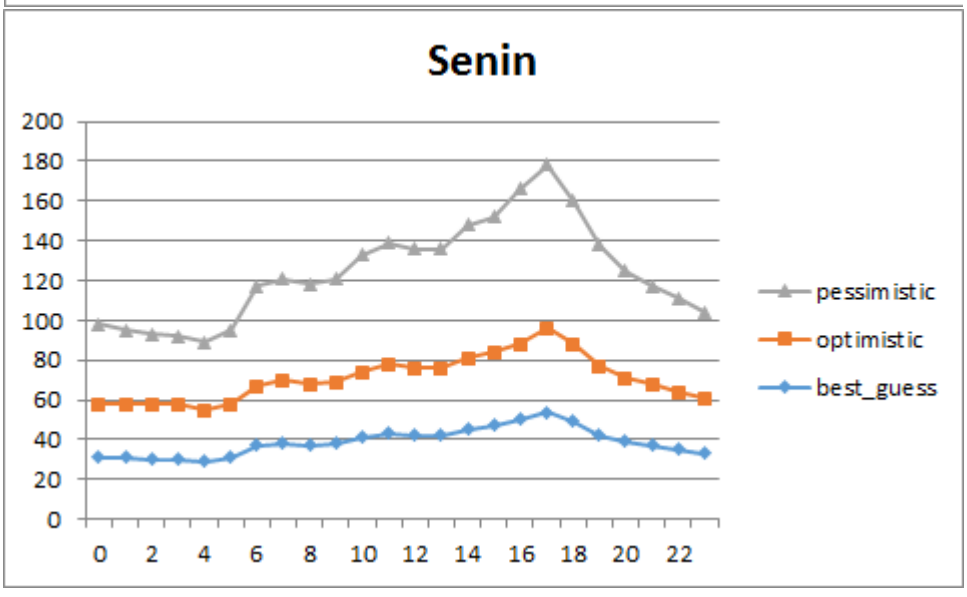

Rabu
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess
Kamis
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess
Jumat
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

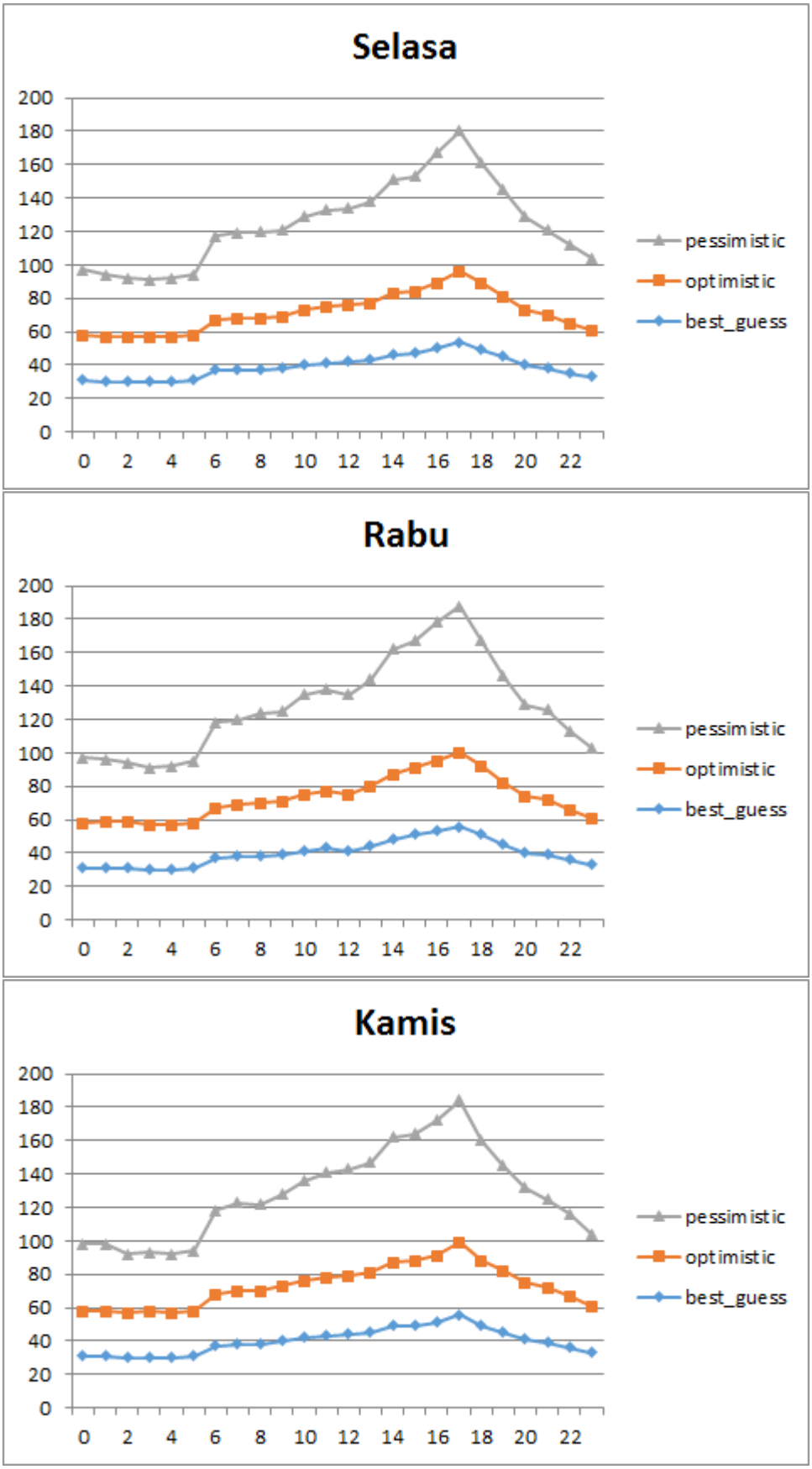


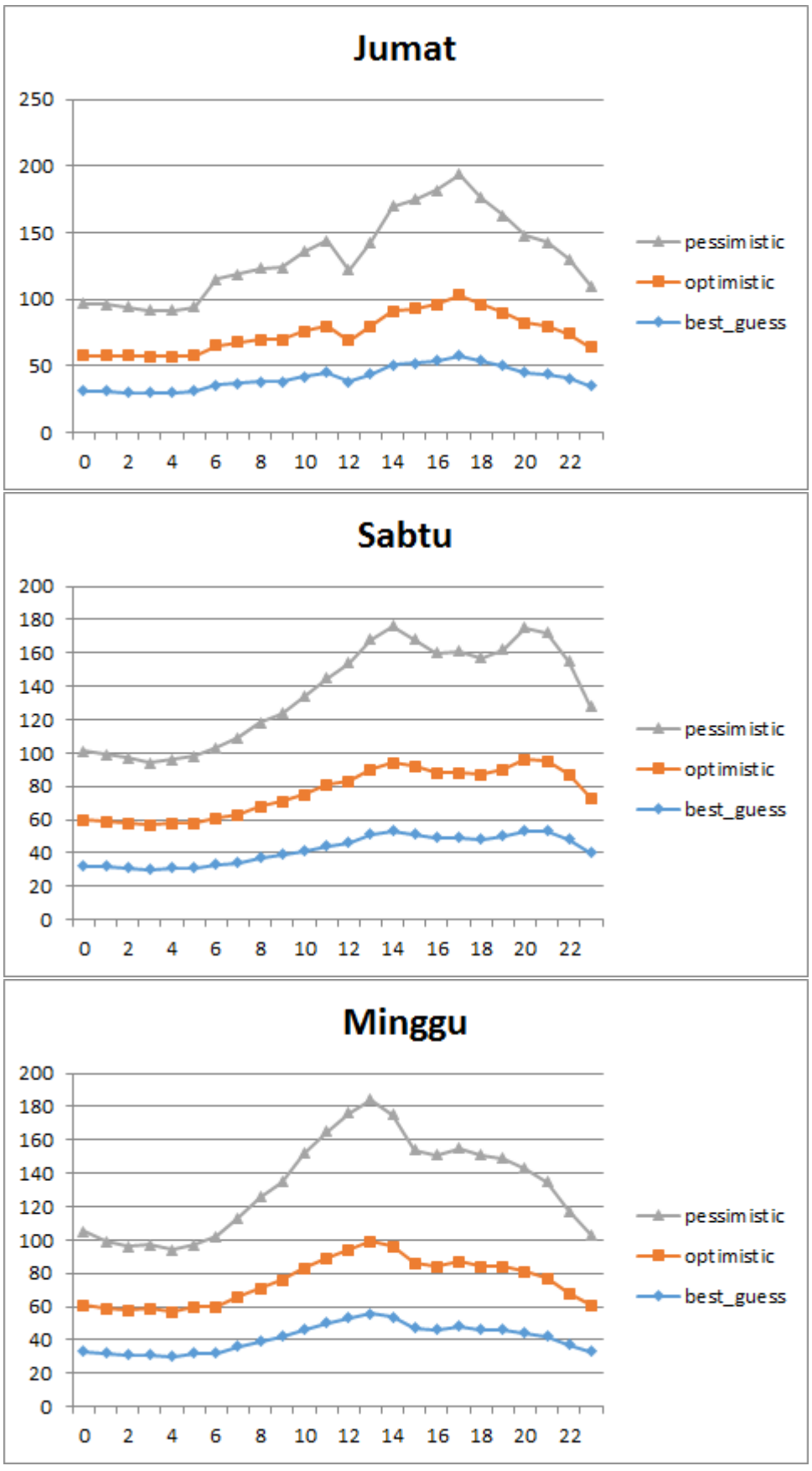
Gambar E.1: Hasil Pengujian Eksperimental sampel 1 08 Mei 2017


Waktu masing-masing model dalam seminggu
250
200
150
100
50
0
1 6 111621 3 8 131823 5 10152025 7 12172227 9 141924 6 11162126 8 13182328
pessimistic
optimistic
best_guess

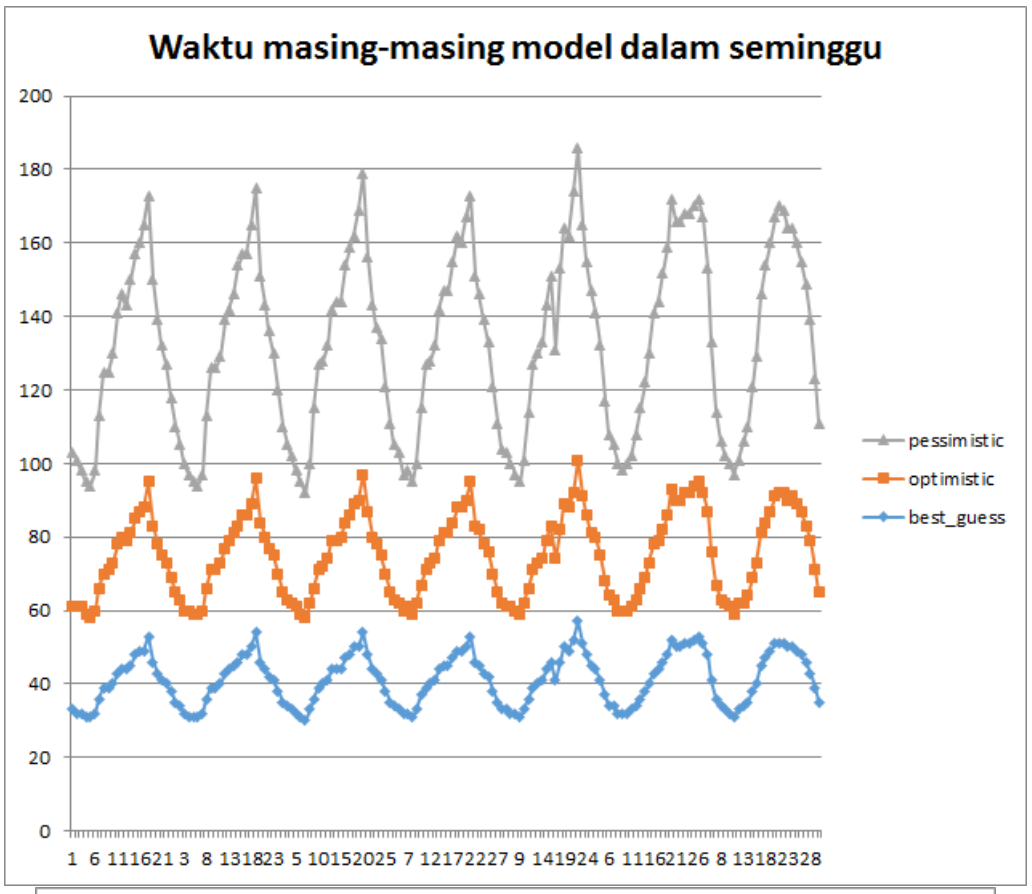

Waktu best_guess pada setiap hari
400
350
300
250
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10111213141516171819202122 23
minggu
sabtu
jumat
kamis
rabu
selasa
senin

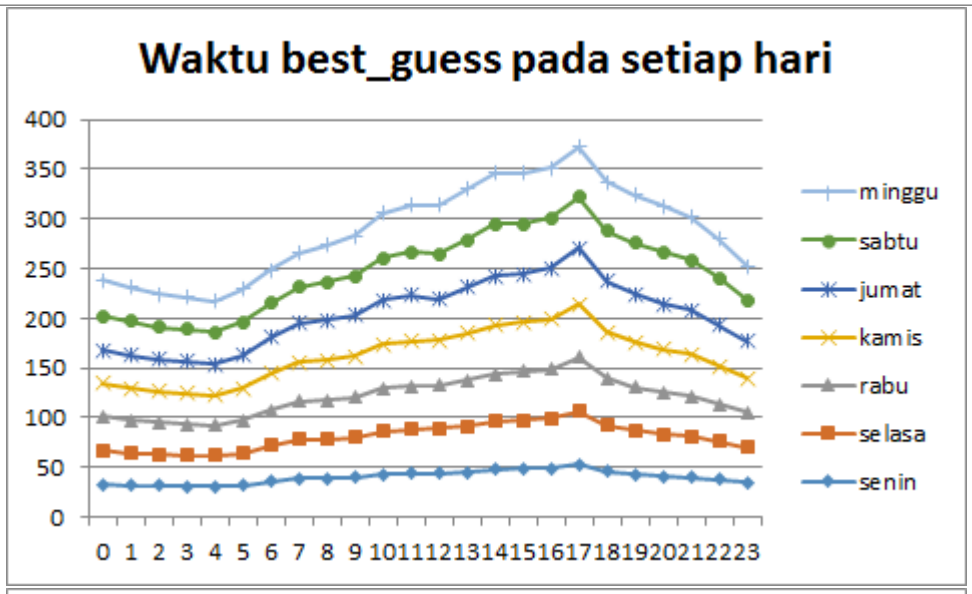

Senin
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

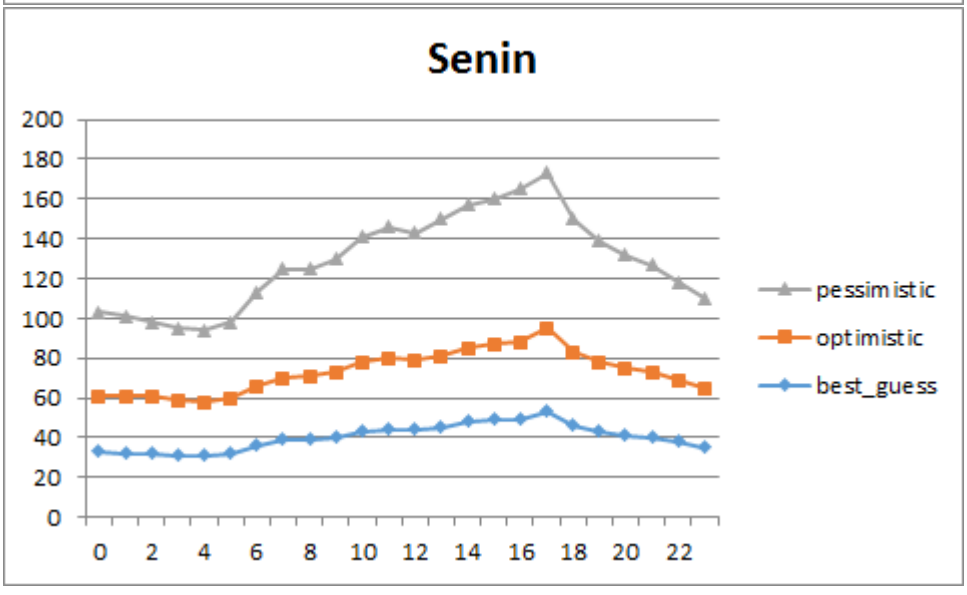

Selasa
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess
Rabu
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess
Kamis
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

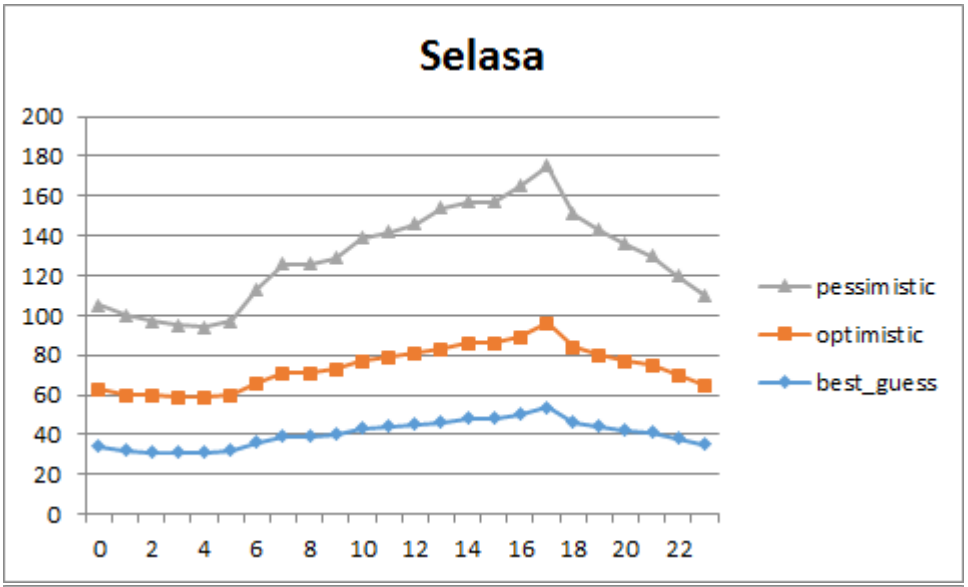


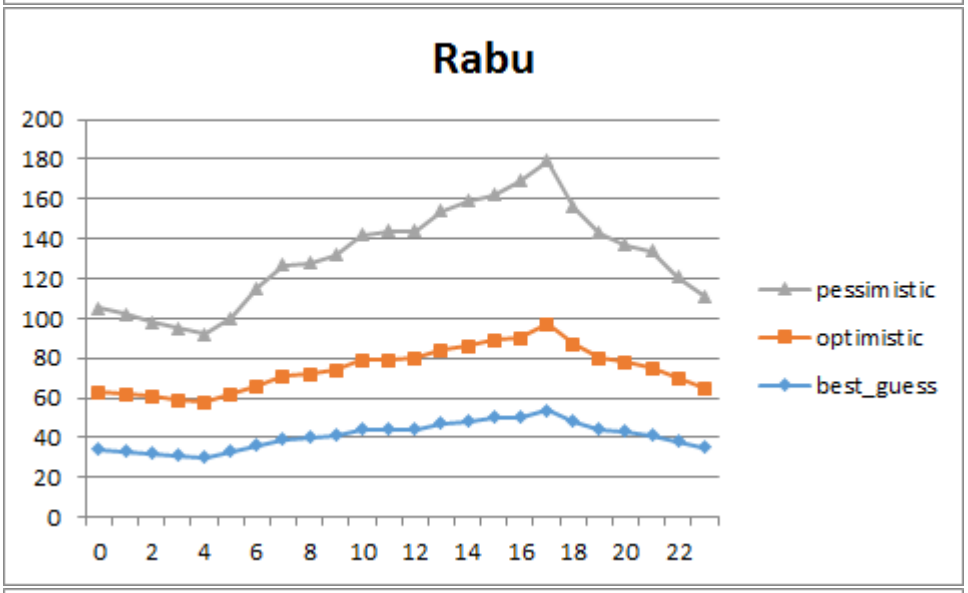
Gambar E.2: Hasil Pengujian Eksperimental sampel 2 15 Mei 2017

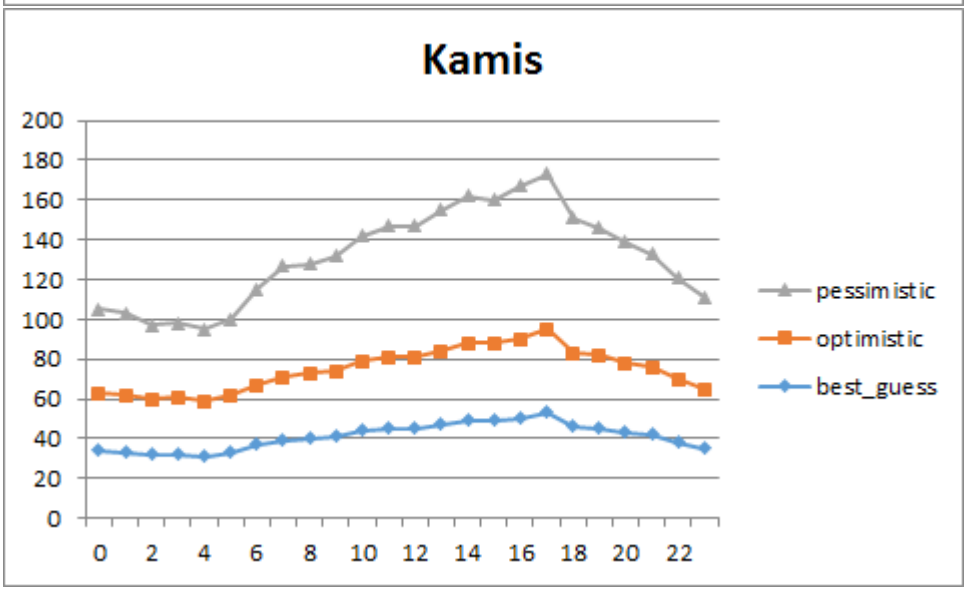

Waktu masing-masing model dalam seminggu
200
180
160
140
120
100
80
60
40
20
0
1 6 111621 3 8 131823 5 10152025 7 12172227 9 141924 6 11162126 8 13182328
pessimistic
optimistic
best_guess

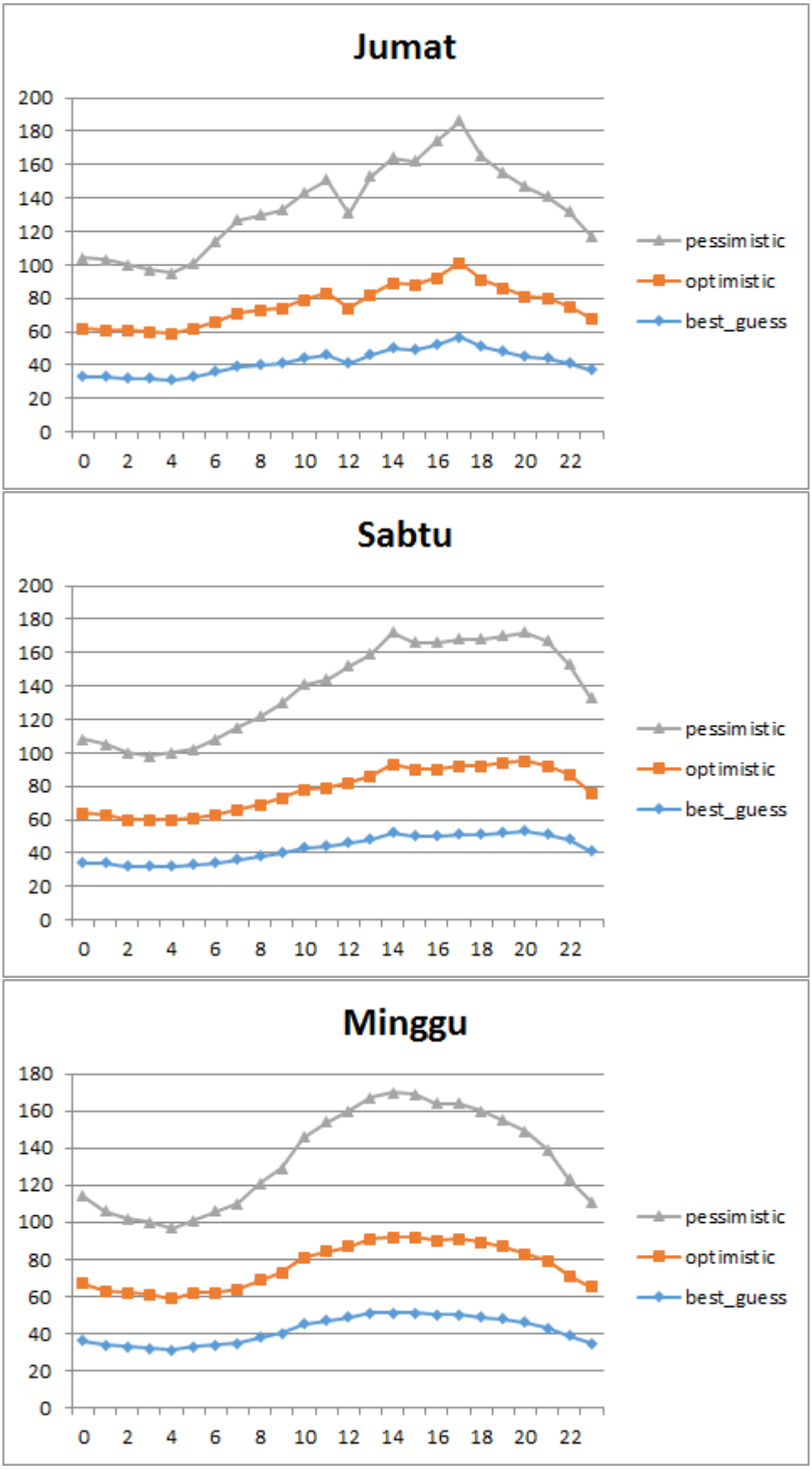

Waktu best_guess pada setiap hari
400
350
300
250
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10111213141516171819202122 23
minggu
sabtu
jumat
kamis
rabu
selasa
senin


Senin
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

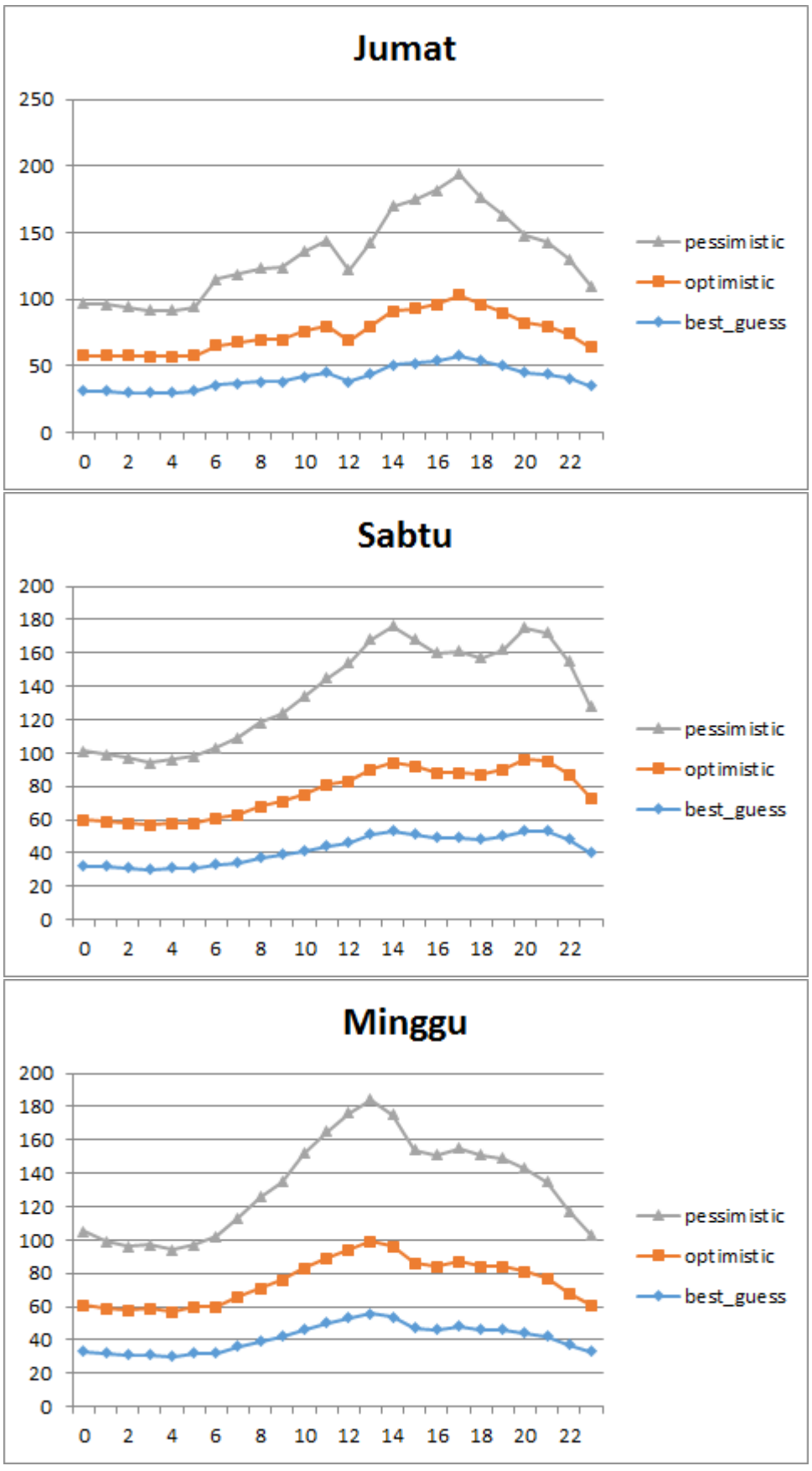

Selasa
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess


Rabu
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

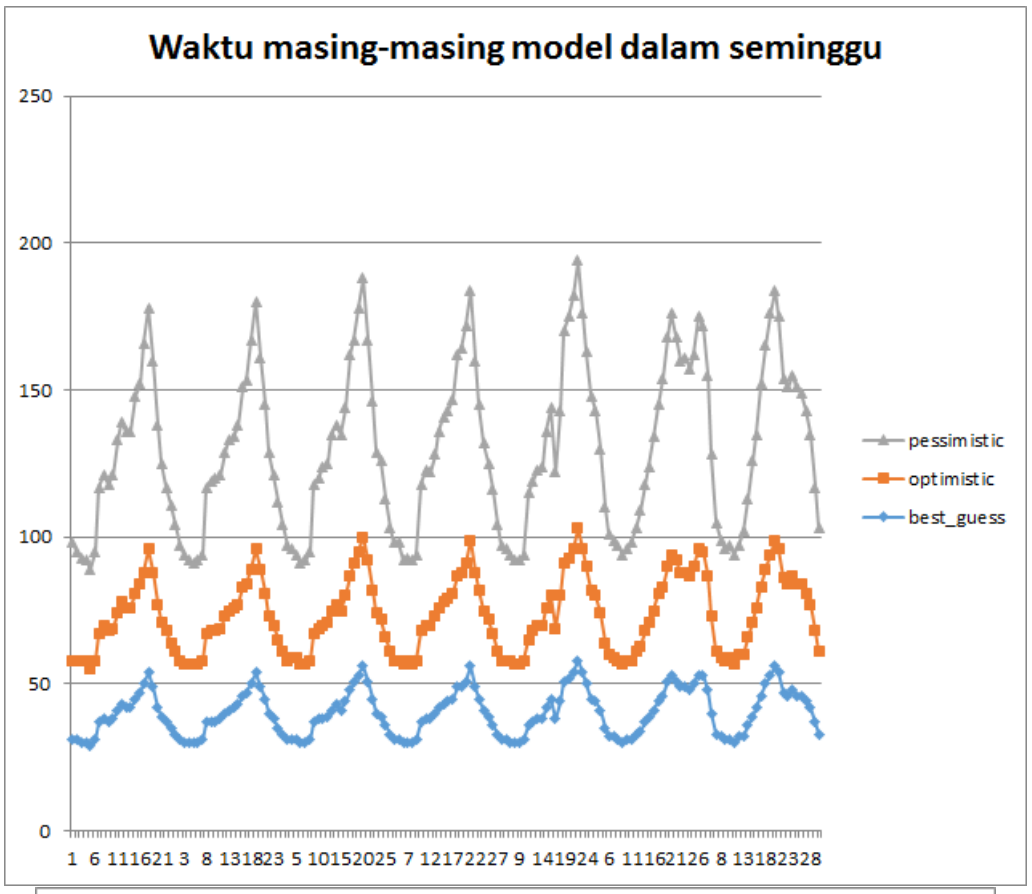

Kamis
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess

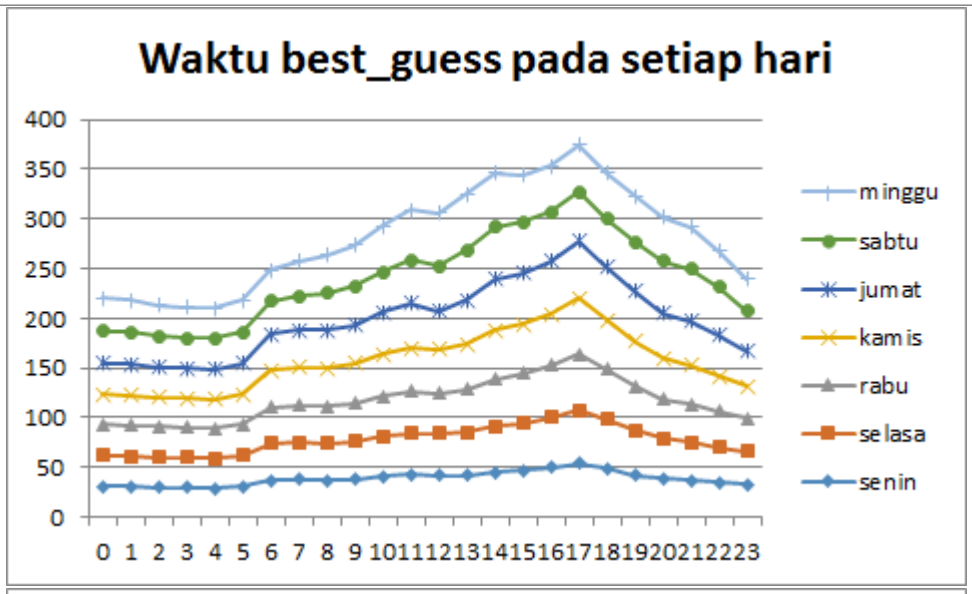


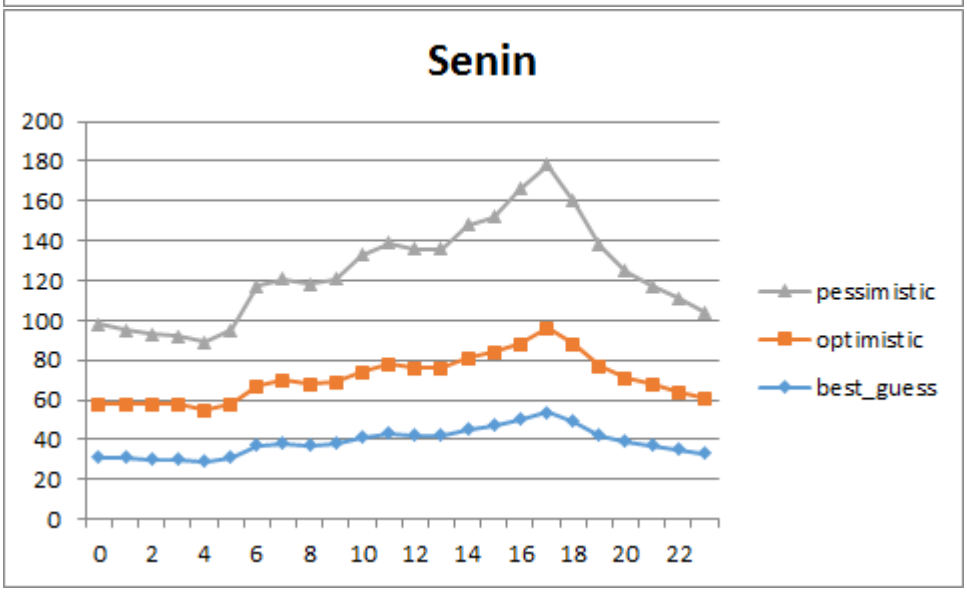
Gambar E.3: Hasil Pengujian Eksperimental sampel 1 08 Mei 2017

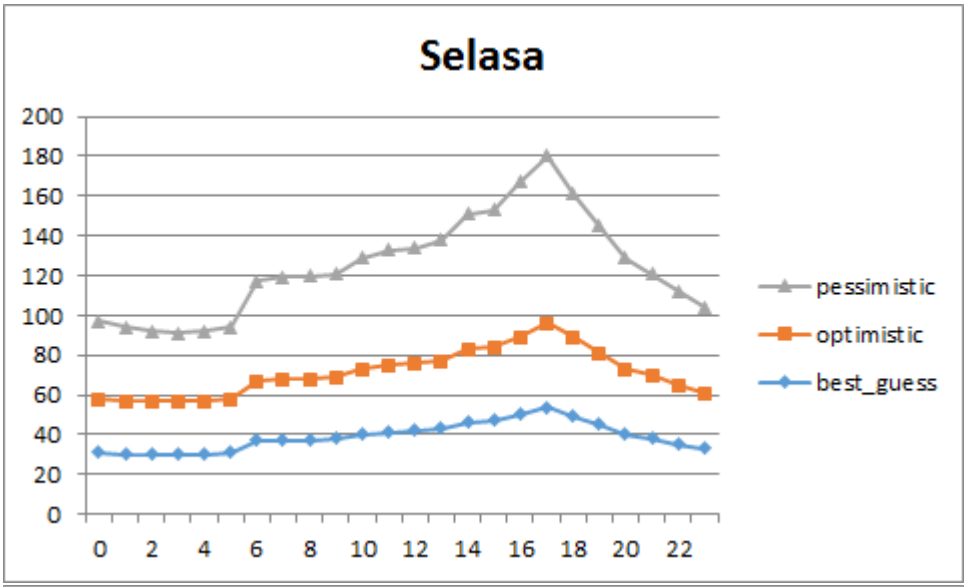


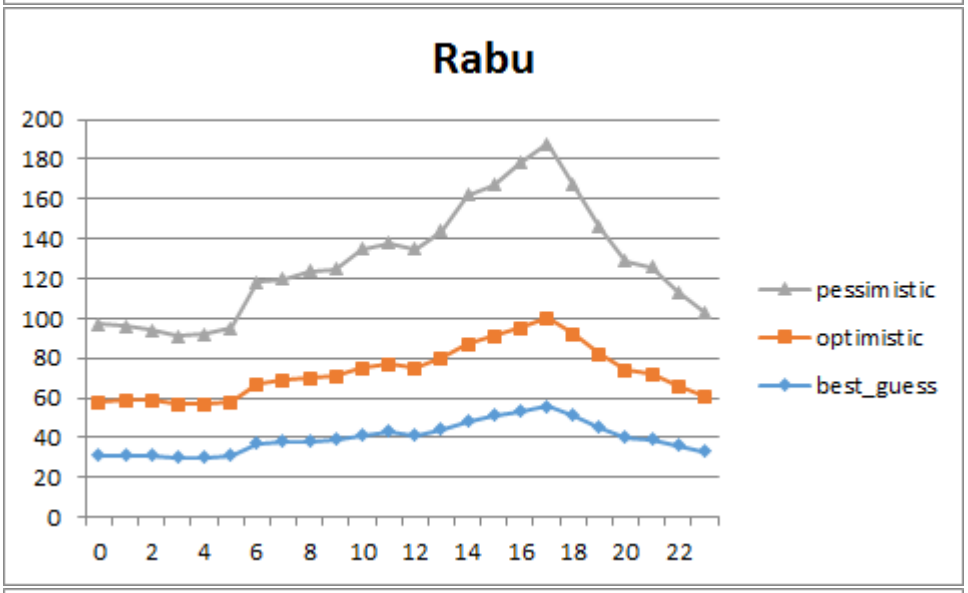
Gambar E.4: Hasil Pengujian Eksperimental sampel 2 08 Mei 2017

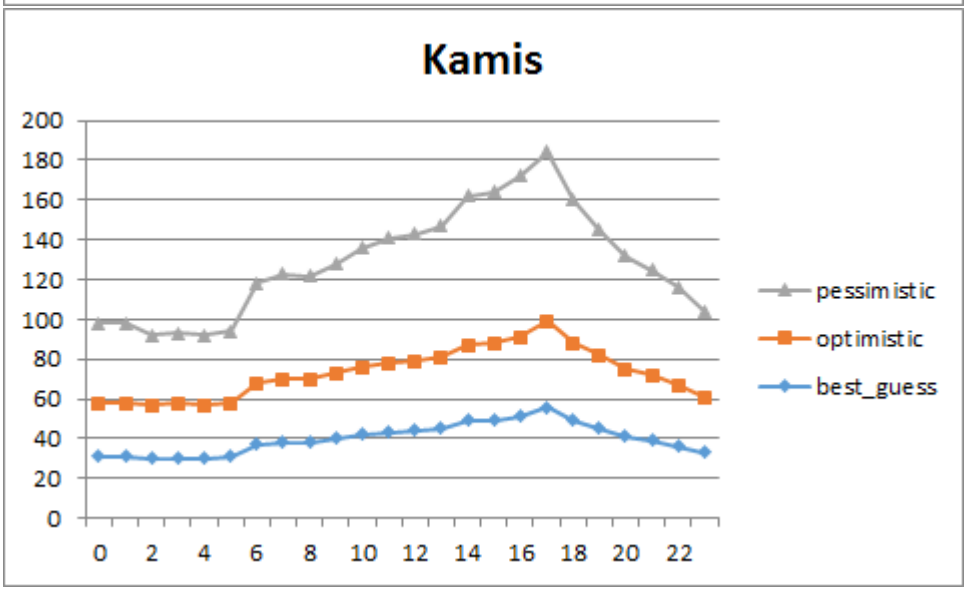

Waktu masing-masing model dalam seminggu
250
200
150
100
50
0
1 6 111621 3 8 131823 5 10152025 7 12172227 9 141924 6 11162126 8 13182328
pessimistic
optimistic
best_guess

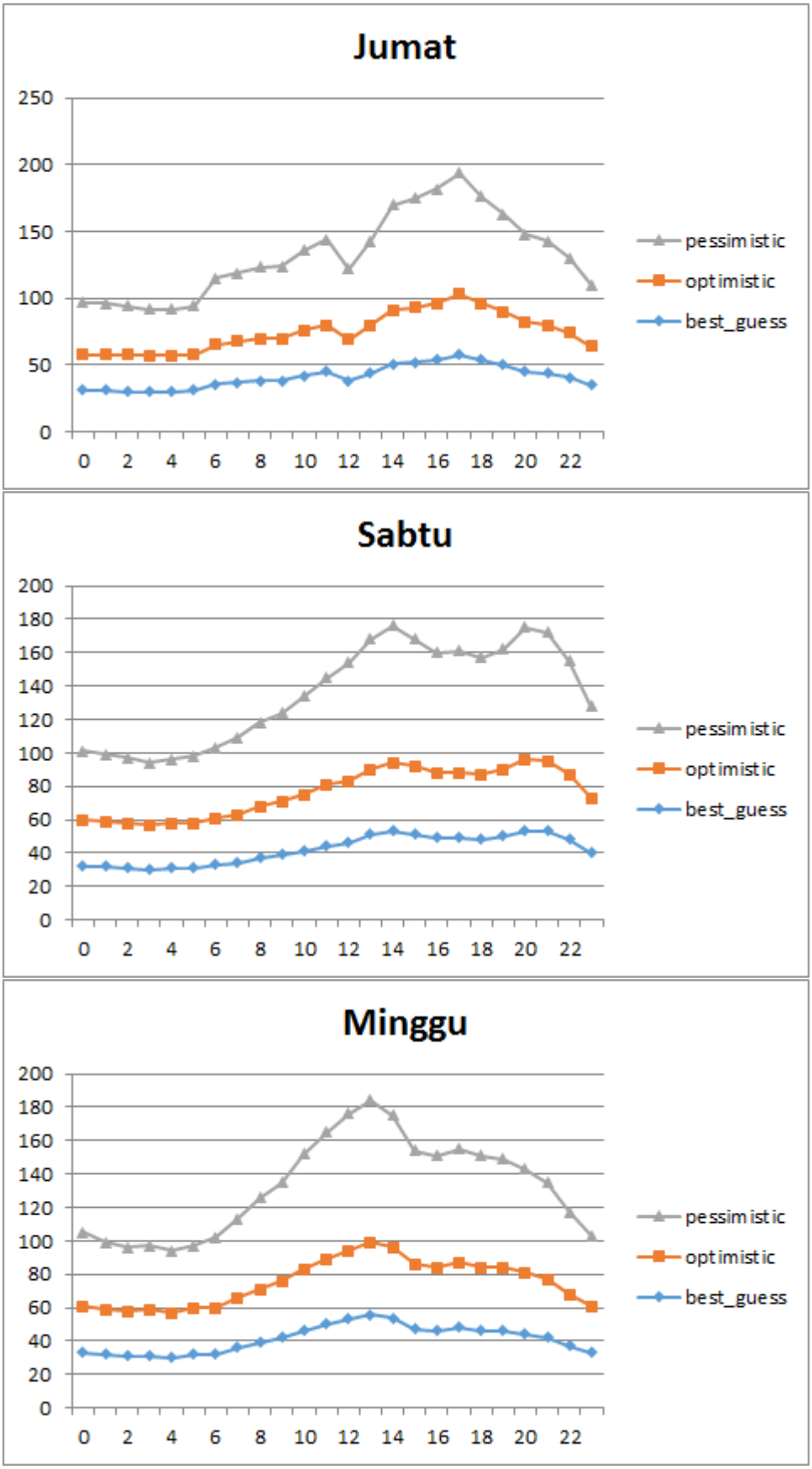

Waktu best_guess pada setiap hari
400
350
300
250
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10111213141516171819202122 23
minggu
sabtu
jumat
kamis
rabu
selasa
senin


Senin
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess


Selasa
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess


Rabu
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess


Kamis
200
180
160
140
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14 16 18 20 22
pessimistic
optimistic
best_guess



Gambar E.5: Hasil Pengujian Eksperimental sampel 2 15 Mei 2017