



## SKRIPSI

VISUALISASI DATA HISTORI KIRI PADA GOOGLE MAPS



Jonathan Laksamana Purnomo

NPM:

PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS  
UNIVERSITAS KATOLIK PARAHYANGAN  
2020



## **UNDERGRADUATE THESIS**

### **VISUALIZATION OF KIRI HISTORICAL DATA ON GOOGLE MAPS**



**Jonathan Laksamana Purnomo**

**NPM:**

**DEPARTMENT OF INFORMATICS  
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES  
PARAHYANGAN CATHOLIC UNIVERSITY  
2020**



## **LEMBAR PENGESAHAN**

### **VISUALISASI DATA HISTORI KIRI PADA GOOGLE MAPS**

**Jonathan Laksamana Purnomo**

**NPM:**

**Bandung, 3 Desember 2020**

**Menyetujui,**

**Pembimbing**

**Pascal Alfadian, M.Comp.**

**Ketua Tim Penguji**

**Anggota Tim Penguji**

**«penguji 1»**

**«penguji 2»**

**Mengetahui,**

**Ketua Program Studi**

**Mariskha Tri Adithia, P.D.Eng**



## **PERNYATAAN**

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

### **VISUALISASI DATA HISTORI KIRI PADA GOOGLE MAPS**

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

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

Dinyatakan di Bandung,  
Tanggal 3 Desember 2020

Meterai  
Rp. 6000

Jonathan Laksamana Purnomo  
NPM:



## **ABSTRAK**

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

**Kata-kata kunci:** KIRI, *Navigation System, Data History, Marker Clustering, Heat Map, Google Javascript API*



## **ABSTRACT**

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

**Keywords:** KIRI, *Navigation System, Data History, Marker Clustering, Heat Map, Google Javascript API*



*Dipersembahkan untuk Tuhan YME, keluarga, para dosen, teman-teman yang telah memberi dukungan dalam pembuatan skripsi ini, serta diri sendiri.*



## **KATA PENGANTAR**

«Tuliskan kata pengantar dari anda di sini . . . »

Bandung, Desember 2020

Penulis



# DAFTAR ISI

<b>KATA PENGANTAR</b>	<b>xv</b>
<b>DAFTAR ISI</b>	<b>xvii</b>
<b>DAFTAR GAMBAR</b>	<b>xix</b>
<b>DAFTAR TABEL</b>	<b>xxi</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Tujuan . . . . .	2
1.4 Batasan Masalah . . . . .	3
1.5 Metodologi . . . . .	3
1.6 Sistematika Pembahasan . . . . .	3
<b>2 LANDASAN TEORI</b>	<b>5</b>
2.1 KIRI Website . . . . .	5
2.2 CSV . . . . .	6
2.2.1 CSV Format . . . . .	6
2.3 Node.js . . . . .	6
2.3.1 Struktur File Node.js Project . . . . .	7
2.3.2 Node Package Manager . . . . .	7
2.3.3 NPM CLI . . . . .	7
2.4 Expressjs . . . . .	8
2.4.1 Instalasi . . . . .	8
2.4.2 Struktur File Express.js . . . . .	8
2.4.3 Routing . . . . .	9
2.4.4 Menampilkan File Statis . . . . .	9
2.5 Google Maps Javascript API . . . . .	10
2.5.1 Map . . . . .	10
2.5.2 Sistem Kordinat Google Maps . . . . .	12
2.6 Marker . . . . .	13
2.7 Marker Clusterer . . . . .	15
2.8 HeatMap . . . . .	15
<b>3 ANALISIS</b>	<b>17</b>
3.1 Analisis Data Histori KIRI . . . . .	17
3.2 Deskripsi Perangkat Lunak . . . . .	18
3.3 Analisis Perangkat Lunak . . . . .	18
3.3.1 Analisis Kebutuhan Perangkat Lunak . . . . .	19
3.3.2 Use Case Diagram . . . . .	19
3.3.3 Use Case Skenario . . . . .	20

3.3.4	Data Flow Diagram . . . . .	20
3.3.5	Flow Chart Diagram . . . . .	22
3.3.6	Input dan Output . . . . .	22
<b>4</b>	<b>PERANCANGAN</b>	<b>25</b>
4.1	Perancangan Antarmuka . . . . .	25
4.2	Perancangan Diagram Fungsi . . . . .	26
4.3	Perancangan Diagram Kapabilitas . . . . .	26
4.3.1	Diagram Kapabilitas Web Client . . . . .	26
4.3.2	Diagram Kapabilitas Backend . . . . .	27
4.3.3	Diagram Kapabilitas Google API . . . . .	28
4.4	Perancangan Diagram Interaksi . . . . .	28
4.5	Perancangan Pseudocode . . . . .	29
4.5.1	Perancangan Pseudocode Pada Web Client . . . . .	30
<b>5</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>33</b>
5.1	Implementasi . . . . .	33
5.1.1	Lingkungan Implementasi . . . . .	33
5.1.2	Implementasi Antarmuka Perangkat Lunak . . . . .	33
5.1.3	Implementasi Perangkat Lunak . . . . .	35
5.1.4	Router . . . . .	39
5.1.5	Controller antara <i>Web client</i> dan <i>Backend Logic</i> . . . . .	40
5.1.6	Berkomunikasi dengan <i>google maps javascript api</i> . . . . .	45
<b>DAFTAR REFERENSI</b>		<b>47</b>
<b>A</b>	<b>KODE PROGRAM</b>	<b>49</b>
<b>B</b>	<b>HASIL EKSPERIMEN</b>	<b>51</b>

## DAFTAR GAMBAR

1.1	Tampilan Utama Website KIRI . . . . .	1
1.2	Tampilan Heat Map . . . . .	2
1.3	Tampilan Marker Clustering . . . . .	2
2.1	Fitur-Fitur Pada Aplikasi KIRI . . . . .	5
2.2	Add Marker . . . . .	13
2.3	Contoh Marker Clustering . . . . .	15
2.4	Contoh HeatMap . . . . .	15
3.1	Alur Komunikasi . . . . .	18
3.2	Use Case Diagram . . . . .	19
3.3	Data Flow Diagram Level 0 . . . . .	21
3.4	Data Flow Diagram Level 1 . . . . .	21
3.5	KIRI Flow Chart . . . . .	22
4.1	Rancangan Antarmuka . . . . .	25
4.2	Rancangan Diagram Kelas . . . . .	26
4.3	Rancangan Diagram Kapabilitas . . . . .	27
4.4	Rancangan Diagram Kapabilitas . . . . .	27
4.5	Rancangan Diagram Kapabilitas . . . . .	28
4.6	Rancangan Diagram Interaksi . . . . .	29
5.1	Tampilan Awal Antarmuka . . . . .	34
5.2	Tampilan setelah memilih marker clustering . . . . .	34
5.3	Tampilan setelah memilih Heat Map . . . . .	35
B.1	Hasil 1 . . . . .	51
B.2	Hasil 2 . . . . .	51
B.3	Hasil 3 . . . . .	51
B.4	Hasil 4 . . . . .	51



## DAFTAR TABEL

2.1	Tabel Properti Pada Objek Marker . . . . .	14
2.2	Tabel Fungsi Pada Kelas Marker . . . . .	14
3.1	Tabel Skenario Memfilter Data . . . . .	20
3.2	Tabel Skenario Melihat Visualisasi Data Pada Map . . . . .	20
4.1	Handling User Input . . . . .	30
4.2	Menampilkan data histori kiri dengan menggunakan <i>google maps</i> . . . . .	31
4.3	Load data histori KIRI . . . . .	31
4.4	Handling Filter Data . . . . .	32

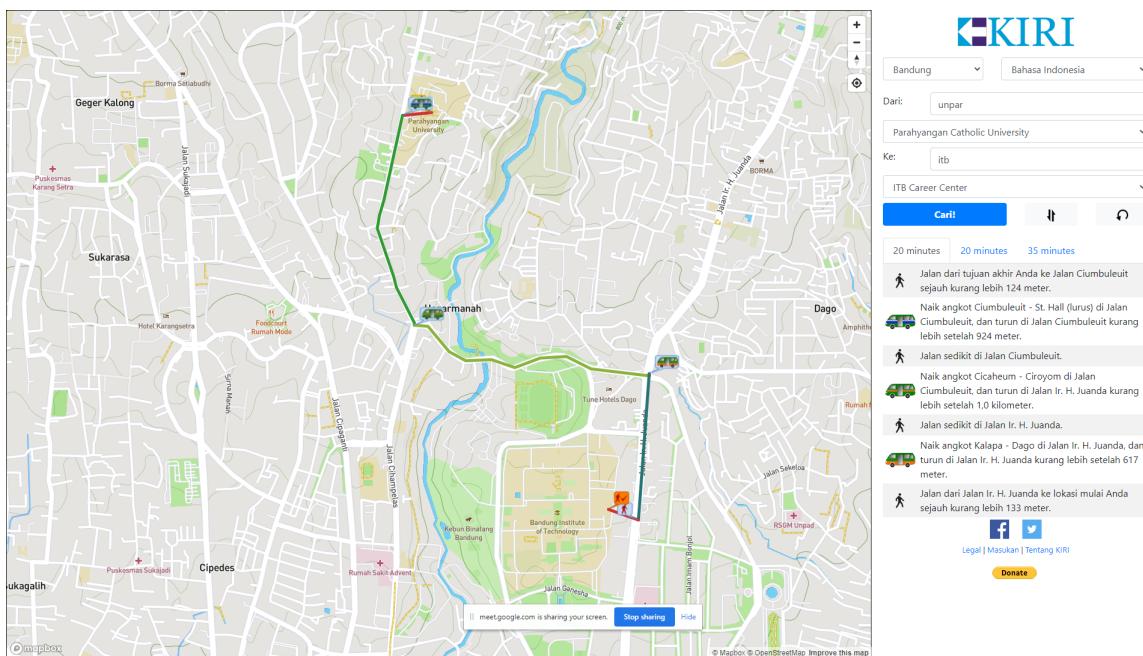


# BAB 1

## PENDAHULUAN

### 1.1 Latar Belakang

Kemajuan teknologi memudahkan manusia untuk mencari berbagai macam informasi. Salah satu informasi yang dapat diperoleh adalah informasi tentang navigasi transportasi publik. KIRI adalah perangkat lunak yang berguna sebagai navigasi antar kota menggunakan transportasi publik dengan menggunakan perangkat peta digital[1]. Pada awal pembuatannya KIRI dibuat untuk tujuan komersial. Namun karena dinilai kurang sukses, projek KIRI sekarang menjadi open source projek yang dapat di akses. Bentuk tampilan aplikasi KIRI dapat dilihat pada gambar 1.1.



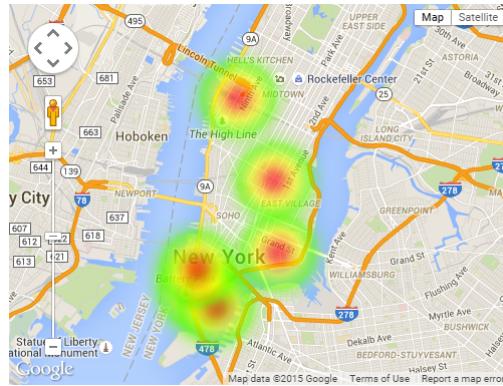
Gambar 1.1: Tampilan Utama Website KIRI

Pada perangkat lunak KIRI seluruh aktivitas yang dilakukan oleh user sudah tercatat. Data yang tercatat disebut juga dengan data histori. Data histori kiri memiliki jumlah record yang cukup banyak sehingga memungkinkan untuk mendapatkan informasi dari data tersebut. Tetapi data histori tersebut belum diolah secara maksimal.

Visualisasi Data adalah teknik untuk mengkomunikasikan data atau informasi dengan menggunakan objek visual seperti *graphic*, *chart*, *diagram*, *dll*. Salah satu objek visual yang dapat digunakan untuk merepresentasikan data adalah *Google Maps*.

Metode yang akan digunakan dalam memvisualisasikan data adalah *Heat Map* dan *Marker Clustering*. *Heat Map* adalah teknik visualisasi data yang menunjukkan besarnya suatu fenomena sebagai warna dalam dua dimensi. Sedangkan *Marker Clustering* adalah teknik visualisasi data yang mengelompokkan *marker* atau *pointer* yang jarak *latitude* dan *longitude* nya saling berdekatan

antara suatu *marker* dengan marker yang lainnya. Contoh bentuk *heat map* dan *marker clustering* dapat dilihat pada gambar 1.2 dan 1.3



Gambar 1.2: Tampilan Heat Map



Gambar 1.3: Tampilan Marker Clustering

Pada skripsi ini akan dibangun perangkat lunak yang dapat memvisualisasikan data histori KIRI. Perangkat lunak ini akan menggunakan metode visualisasi *heat map* dan *marker clustering* dari hasil visualisasi tersebut akan diambil suatu pola kesimpulan dari data histori KIRI.

## 1.2 Rumusan Masalah

Rumusan masalah dari topik ini adalah sebagai berikut:

- Bagaimana memvisualisasikan data histori KIRI?
- Bagaimana menemukan pola dari data histori KIRI?
- Bagaimana penerapan metode *heat map* pada visualisasi data histori KIRI?
- Bagaimana penerapan metode *marker clustering* pada visualisasi data histori KIRI?

## 1.3 Tujuan

Tujuan dari topik ini adalah sebagai berikut:

- Mempelajari *Google Maps Javascript API*.
- Melakukan observasi data.
- Mengimplementasikan metode *Heat map* pada visualisasi data histori KIRI.
- Mengimplementasikan metode *Marker clustering* pada visualisasi data histori KIRI.

## 1.4 Batasan Masalah

Belum ditentukan.

## 1.5 Metodologi

Metodologi yang digunakan dalam penelitian ini adalah:

1. Mempelajari *Google Maps Javascript API* khususnya *Heat Map* dan *Marker Clustering*.
2. Analisis masalah perangkat lunak yang akan dibangun.
3. Merancang perangkat lunak yang akan dibangun.
4. Membangun perangkat lunak yang mengimplementasikan *Heat Map* atau *Marker Clustering* dengan memanfaatkan *Google Maps Javascript API*.
5. Menentukan pola dari hasil visualisasi data.
6. Analisis hasil pengujian dan mengambil kesimpulan.

## 1.6 Sistematika Pembahasan

Laporan penelitian tersusun ke dalam enam bab secara sistematis sebagai berikut.

- Bab 1 Pendahuluan  
Berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- Bab 2 Dasar Teori  
Berisi metode penentuan pola, *library Google Maps* dan bahasa pemrograman *Javascript*
- Bab 3 Analisis  
Berisi analisis masalah terkait implementasi *Goole Map*, studi kasus teknik penentuan pola yang diimplementasikan, dan gambaran umum perangkat lunak yang meliputi diagram aktivitas dan diagram kelas.
- Bab 4 Perancangan Perangkat Lunak  
Berisi perancangan perangkat lunak yang akan dibangun, meliputi perancangan antarmuka, diagram kelas lengkap dan masukan perangkat lunak.
- Bab 5 Implementasi dan Pengujian  
Berisi implementasi antarmuka perangkat lunak, pengujian fungsional, pengujian eksperimental serta kesimpulan dari pengujian.
- Bab 6 Kesimpulan dan Saran  
Berisi kesimpulan dari awal hingga akhir penelitian dan saran untuk penelitian berikutnya.



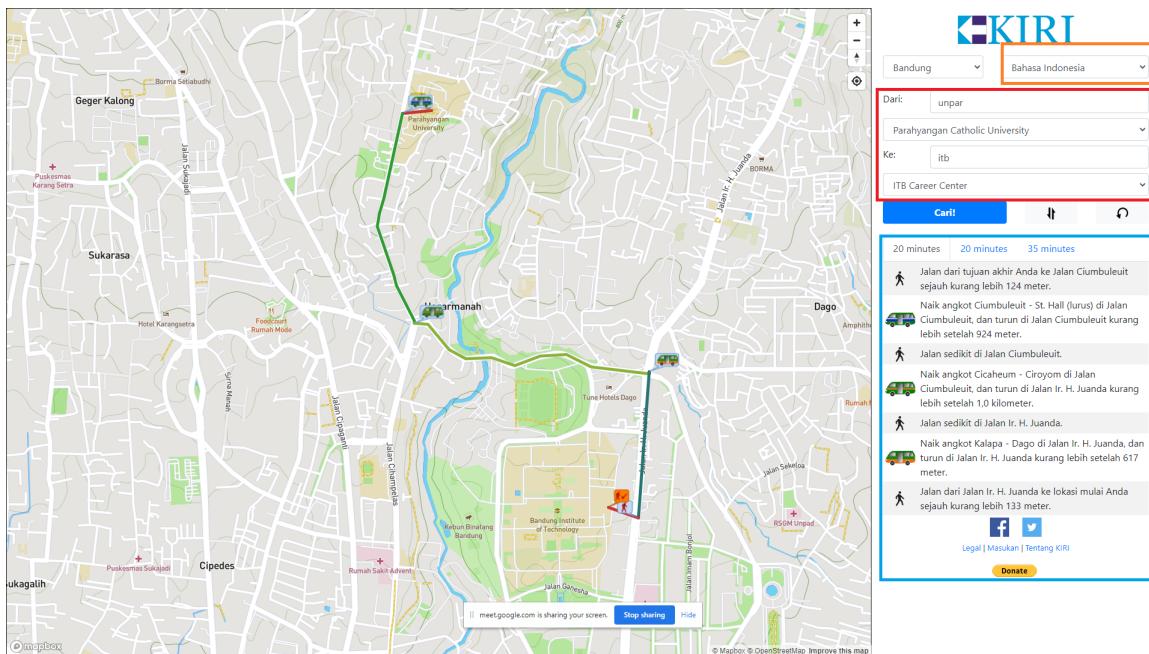
## BAB 2

### LANDASAN TEORI

Pada bab ini dijelaskan dasar-dasar teori mengenai KIRI website, JSON, CSV, dan *Google Maps Javascript API*

#### 2.1 KIRI Website

KIRI adalah aplikasi navigasi angkutan umum berbasis web yang melayani Bandung dan kota-kota lain di Indonesia.[1]. Pada awal pembuatannya KIRI dibuat untuk tujuan komersial. Namun karena dinilai kurang sukses projek KIRI sekarang menjadi open source projek yang dapat diakses. Aplikasi KIRI memiliki beberapa fitur sebagai berikut:



Gambar 2.1: Fitur-Fitur Pada Aplikasi KIRI

1. Pemilihan rute tercepat menggunakan angkutan kota. Dapat dilihat pada gambar 2.1 pada kotak berwarna merah.
2. Memiliki fitur multi bahasa. Dapat dilihat pada gambar 2.1 pada kotak berwarna oranye.
3. Dapat menampilkan instruksi lengkap mencapai tujuan. Dapat dilihat pada gambar 2.1 pada kotak berwarna biru.

## 2.2 CSV

CSV (*Comma Separated Values*) adalah format penyajian data yang sudah umum digunakan pada program *spreadsheet*, CSV memiliki format memisahkan setiap value dengan simbol titik koma (,) dan menggunakan baris baru sebagai penanda pemisah antar element data[2].

### 2.2.1 CSV Format

Tidak ada spesifikasi *formal* dalam penulisan csv. Format csv yang akan dituliskan pada dokumen ini adalah format yang paling banyak diimplementasikan:

- Setiap field diletakan pada baris terpisah dan dipisahkan oleh *line break (CRLF)*, contoh:

```
1     aaa , bbb , ccc CRLF
2     zzz , yyy , xxx CRLF
```

- *Field* terakhir pada format csv tidak harus menggunakan *line break (CRLF)*, contoh:

```
1     aaa , bbb , ccc CRLF
2     zzz , yyy , xxx
```

- Memungkinkan adanya eksternal *header* yang memiliki aturan penulisan yang sama dengan element pada csv. Nilai pada eksternal *header* akan mewakili nama yang tercantum pada *field* jumlah eksternal *header* harus sama dengan jumlah kolom pada normal record, contoh:

```
1     field_name , field_name , field_name CRLF
2     aaa , bbb , ccc CRLF
3     zzz , yyy , xxx CRLF
```

- Memungkinkan ada satu atau lebih *field* yang dipisahkan oleh koma (,). Setiap baris harus memiliki jumlah *field* yang sama pada satu *file*. Spasi dianggap sebuah element pada *field* yang tidak dapat diabaikan. Pada field terakhir sebuah baris tidak boleh diberikan simbol koma (,), contoh:

```
1     aaa , bbb , ccc
```

- Setiap *Field* bisa menggunakan simbol *double quotes*. Jika *field* tidak menggunakan simbol *double quotes* maka *double quotes* tidak akan ditampilkan pada *fields*

```
1     "aaa" , "bbb" , "ccc" CRLF
2     zzz , yyy , xxx
```

- Jika simbol *double quotes* merupakan salah satu elemen pada *field*, Maka simbol *double quotes* tersebut harus diesepe dengan memberikan simbol *double quotes* lain didalam *field* tersebut, contoh:

```
1     "aaa" , "b" "bb" , "ccc"
```

## 2.3 Node.js

*Node.js* merupakan *asynchronous event-driven JavaScript runtime*. Dengan menggunakan *Node.js* memungkinkan untuk menjalankan perintah *javascript* tanpa menggunakan *web browser*, *Node.js* memungkinkan untuk menjalankan dan melakukan *server-side scripting*[3]. *Node.js* didesain untuk membuat *network applications* yang *scaleable*. Berikut ini contoh syntax *Node.js*:

```

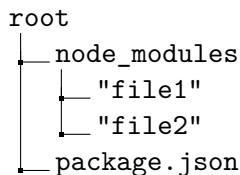
1 const http = require('http');
2
3 const hostname = '127.0.0.1';
4 const port = 3000;
5
6 const server = http.createServer((req, res) => {
7   res.statusCode = 200;
8   res.setHeader('Content-Type', 'text/plain');
9   res.end('Hello World');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${
14   hostname}:${port}/`);
15 });

```

### 2.3.1 Struktur File Node.js Project

Pada saat membuat projek baru, Node.js membuat *file-file* dasar secara otomatis. *File-file* tersebut memiliki fungsi tersendiri. Folder *node\_modules* berfungsi sebagai penyimpanan semua *library* yang diinstall melalui npm. Semua *library* yang diinstall melalui npm akan dicatat pada file *package.json*. Hal ini bertujuan untuk mempermudah proses *maintenance library* pada projek *node.js*.

Struktur dari *file-file* akan berbentuk seperti:



### 2.3.2 Node Package Manager

*Node Package Manager*(NPM) adalah *software registry* yang dimiliki oleh *Node.js*, NPM memungkinkan pengguna untuk mempublikasi atau menggunakan *software library*[4]. NPM terdiri dari tiga komponen penting yaitu:

- NPM website.
- NPM CLI (*Command Line Interface*).
- NPM *Registry*.

NPM memiliki beberapa kegunaan antara lain:

- Mendownload *Software Library*.
- Menjalankan *packages* tanpa harus mendownload *npx*.
- Mempublikasikan *Tools* atau *Software Library*.

### 2.3.3 NPM CLI

*NPM* akan terinstall secara otomatis pada perangkat keras ketika menambahkan *node.js*. Untuk mempermudah proses pembuatan perangkat lunak, Penggunaan *Command Line Interface* (CLI) akan menjadi salah satu *point* penting. NPM memiliki tiga komponen utama dalam penulisan perintah CLI, Komponen ini akan berbentuk seperti:

**npm <command> [args]**

*NPM CLI* memiliki perintah-perintah yang dapat digunakan untuk membantu melakukan *maintenance* terhadap *library* pada projek *node.js*. Berikut perintah-perintah pada *npm cli*:

- Command untuk menginisialisasi *npm package* file dapat menggunakan perintah:

```
1  npm init [--force|-f|--yes|-y|--scope]
2  npm init <@scope> (same as 'npx <@scope>/create')
3  npm init [<@scope>/]<name> (same as 'npx [<@scope>/]create-<name>')
```

- Command untuk menambahkan *package / library* dapat menggunakan perintah:

```
1  npm install (with no args, in package dir)
2  npm install [<@scope>/]<name>
3  npm install [<@scope>/]<name>@<tag>
4  npm install [<@scope>/]<name>@<version>
5  npm install [<@scope>/]<name>@<version range>
6  npm install <alias>@npm:<name>
7  npm install <git-host>:<git-user>/<repo-name>
8  npm install <git repo url>
9  npm install <tarball file>
10  npm install <tarball url>
11  npm install <folder>
```

- Command untuk menghapus *package / library* dapat menggunakan perintah:

```
1  npm uninstall [<@scope>/]<pkg>[@<version>]
```

## 2.4 Expressjs

*Express.js* merupakan *web application framework* untuk *node.js*. *Express.js* menyediakan *robust feature* dalam pembuatan perangkat lunak berbentuk situs web maupun perangkat bgerak[5].

### 2.4.1 Instalasi

*Express.js* dapat diinstal dengan menggunakan *npm*. Untuk menambahkan *package express.js* dapat menggunakan perintah:

```
npm install express --save
```

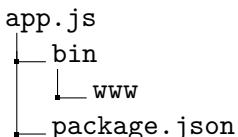
Ketika menjalankan perintah diatas maka secara otomatis akan menambahkan *library express.js* yang akan disimpan pada *folder package.json*.

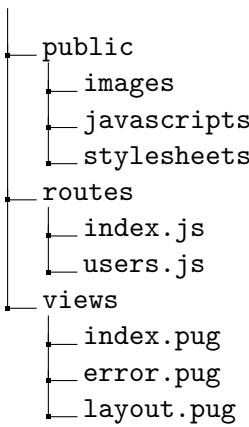
### 2.4.2 Struktur File Express.js

*Express.js* tidak memberikan aturan baku dalam penyusunan struktur file dalam pengembangan perangkat lunak. namun *express.js* menyediakan *application generator* yang disebut *express generator* untuk mempermudah pihak pengembang dalam pembuatan perangkat lunak[5]. Untuk dapat menggunakan *express generator* pengembang harus menambahkan *library express generator* melalui *npm* dengan menggunakan perintah:

```
npx express-generator
```

Secara otomatis *express* akan menghasilkan sebuah folder yang memiliki struktur file seperti:





### 2.4.3 Routing

*Routing* adalah proses untuk menentukan cara perangkat lunak merespon *input* melalui beberapa *endpoint*. Dalam pembuatan *routing* pada *express.js* terdapat empat komponen penting yaitu:

- Instansi dari *express.js* (app).
- *Method http method*.
- *Path*
- Perintah yang akan dijalankan jika *route* dijalankan *Handler*.

Pembuatan *routing* pada *express.js* memiliki struktur perintah sebagai berikut:

```
app.METHOD(PATH, HANDLER)
```

Berikut ini beberapa contoh perintah pembuatan *route* pada *express.js*

```

1 app.get('/', function (req, res) {
2   res.send('Hello World!')
3 }
4
5 app.post('/', function (req, res) {
6   res.send('Got a POST request')
7 }
8
9 app.put('/user', function (req, res) {
10   res.send('Got a PUT request at /user')
11 }
12
13 app.delete('/user', function (req, res) {
14   res.send('Got a DELETE request at /user')
15 })

```

### 2.4.4 Menampilkan File Statis

Untuk dapat menampilkan *file* yang bersifat statis seperti foto, *javascript*, dan *css*, *express.js* telah menyediakan objek

```
express.static(root, [options])
```

Parameter *root* menandakan *root directory* untuk menampilkan *file* statis. Contoh perintah untuk menampilkan *file* statis:

```
1 app.use('/static', express.static('public'))
```

Parameter '/static' berutujuan untuk membuat *virtual path* yang memiliki *prefix* '/static', sedangkan parameter 'public' menandakan bahwa *file* statis berada didalam *folder public*. Perintah diatas ketika dijalankan maka *express* akan secara otomatis membuat *route* seperti:

```
1 http://localhost:3000/static/images/kitten.jpg
2 http://localhost:3000/static/css/style.css
3 http://localhost:3000/static/js/app.js
4 http://localhost:3000/static/images/bg.png
5 http://localhost:3000/static/hello.html
```

yang dapat digunakan untuk mengakses *file* statis.

## 2.5 Google Maps Javascript API

Pada subbab ini akan menjelaskan tentang *google maps javascript api* beserta kelas-kelas yang dimilikinya. Google Maps adalah layanan pemetaan web yang dikembangkan oleh Google. Menawarkan citra satelit, foto udara, dan peta jalan yang interaktif, kondisi lalu lintas secara *real time*. Dalam pengembangannya *google maps* memiliki akses pendukung untuk bahasa pemrograman *javascript*. Berikut ini beberapa layanan yang telah disediakan oleh *google maps javascript api*[6]:

- *Maps*.
- *Drawing Object*.
- *Street Views*.
- *Routes*

### 2.5.1 Map

*Map* adalah sebuah objek pada *google maps javascript api* yang digunakan untuk membuat objek *map* didalam *html* elemen. Untuk dapat menginisialisasi objek *map* pengembang harus dapat menggunakan constructor yang memiliki perintah:

```
1 Map(mapDiv[, opts])
2 Parameters:
3   mapDiv: Element
4   opts: MapOptions optional
```

Pada *constructor* diatas terdapat dua paramete:

- *mapDiv*.
- *MapOptions*

*MapDiv* adalah elemen html dimana objek *map* akan diinisialisasi, parameter ini bersifat wajib. *MapOptions* adalah sebuah objek yang dapat digunakan untuk mengatur *property* yang dimiliki oleh objek *map*. Berikut ini adalah contoh menginisialisasi objek *map*:

```
1 function initMap() {
2     let mapOptions = {
3         center: {lat: -6.914744, lng: 107.609810},
4         zoom: 12,
5     }
6     let map = new google.maps.Map(document.getElementById("map"), mapOptions);
7     return map;
8 }
```

Ketika fungsi *initMap()* dijalankan maka *google maps javascript api* akan membuat objek *map* didalam *html dom* yang memiliki id='map' dan akan mengatur *property* yang dimiliki sesuai dengan objek *mapOptions*.

Objek *map* telah menyediakan fungsi-fungsi yang bisa langsung digunakan oleh pengembang, beberapa fungsi yang disediakan oleh objek *map* adalah:

- Fungsi *fitBounds* berguna untuk menentukan *view port* dari objek *map* sesuai dengan *bounds* yang diberikan. Fungsi ini memiliki struktur perintah seperti:

```

1     fitBounds(bounds[, padding])
2     Parameters:
3       bounds: LatLngBounds | LatLngBoundsLiteral
4       padding: number | Padding optional
5     Return Value: None

```

- Fungsi *getBounds* berguna untuk mendapatkan *view port* dari objek *map*. Fungsi ini memiliki struktur perintah seperti:

```

1     getBounds()
2     Parameters: None
3     Return Value: LatLngBounds

```

- Fungsi *getCenter* berguna untuk mendapatkan posisi yang ditunjukan pada objek *map* posisi yang didapatkan akan berbentuk *longitude* dan *latitude*. Fungsi ini memiliki struktur perintah seperti:

```

1     getCenter()
2     Parameters: None
3     Return Value: LatLang

```

- Fungsi *getClickableIcons* berguna untuk mendapatkan *clickable icon* setiap *clickable icon* merupakan *point of interest* pada *map*. Fungsi ini memiliki struktur perintah seperti:

```

1     getClickableIcons()
2     Parameters: None
3     Return Value: boolean

```

- Fungsi *getMapTypeId* beguna untuk mendapatkan *id* dari jenis *map* yang digunakan . Fungsi ini memiliki struktur perintah seperti:

```

1     getMapTypeId()
2     Parameters: None
3     Return Value: MapTypeId | string

```

*Google Maps Javascript API* menyediakan variabel konsanta / *constant* yang berfungsi untuk menentukan jenis peta yang akan digunakan pada objek *map*. Konsanta ini dapat memiliki empat nilai yaitu:

- HYBRID jenis peta ini akan menampilkan layar *transparan* pada jalan-jalan utama pada citra satelit.
- ROADMAP jenis peta ini akan menampilkan *street map*.
- SATELLITE jenis peta ini akan menampilkan citra satellite.
- TERRAIN jenis peta ini akan menampilkan bentuk nyata dari kondisi geologi suatu tempat.
- Fungsi *setMapTypeId* beguna untuk membuat atau mengubah *mapTypeId*. Fungsi ini memiliki struktur perintah seperti:

```

1     setMapTypeId(mapTypeId)
2     Parameters:
3       mapTypeId: MapTypeId | string

```

- Fungsi *setZoom* berguna untuk mengubah *zoom value* yang dimiliki oleh objek *map*.

```

1     setZoom(zoom)
2     Parameters:
3       zoom:  number

```

- Fungsi *setMapOption* berguna untuk mengubah *property mapOption* yang dimiliki oleh objek *map*.

```

1     setOptions(options)
2     Parameters:
3       options:  MapOptions

```

### 2.5.2 Sistem Kordinat Google Maps

*Google Maps Javascript API* menggunakan kelas *LatLng* untuk merepresentasikan kordinat secara geografis pada objek *map*. Kelas *LatLng* merupakan sebuah kelas yang merepresentasikan latitude dan longitude

- Latitude memiliki batas antara -90 sampai 90 derajat, jika ada nilai yang diluar batasan tersebut maka nilai tersebut akan dibulatkan kebatas terdekat.
- Longitude memiliki batas antara -180 sampai 180 derajat, jika ada nilai yang diluar batasan tersebut maka nilai tersebut akan dibulatkan kebatas terdekat.

Untuk dapat menginisialisasi kelas *LatLng* pada *google maps javascript api* pengembang perlu membuat *constructor* yang memiliki struktur:

```

1   LatLng(lat, lng[, noWrap])
2   Parameters:
3     lat:  number
4     lng:  number
5     noWrap:  boolean optional

```

Membuat objek *LatLng* yang mewakili titik geografis. *latitude* ditentukan dalam derajat dalam rentang [-90, 90]. *longitude* ditentukan dalam derajat dalam rentang [-180, 180]. Set *noWrap* *true* untuk mengaktifkan nilai di luar rentang ini. Contoh penggunaan kelas *LatLng*:

```

1   map.setCenter(new google.maps.LatLng(-34, 151));
2   map.setCenter({lat: -34, lng: 151});

```

*Google Maps Javascript API* telah menyediakan fungsi bawaan yang dapat diakses ketika menggunakan kelas *LatLng*. Fungsi tersebut adalah:

- Fungsi *equals* fungsi ini bertujuan untuk membandingkan posisi antara objek *map*.

```

1     equals(other)
2     Parameters:
3       other:  LatLng
4     Return Value:  boolean

```

- Fungsi *lat* fungsi ini bertujuan untuk mendapatkan posisi *latitude* dari objek *map*.

```

1     lat()
2     Parameters:  None
3     Return Value:  number
4     Returns the latitude in degrees.

```

- Fungsi *lng* fungsi ini bertujuan untuk mendapatkan posisi *longitude* dari objek *map*.

```

1     lng()
2     Parameters:  None
3     Return Value:  number
4     Returns the longitude in degrees.

```

- Fungsi `toJSON` fungsi ini akan mengembalikan format `json` terhadap posisi latlng pada objek `map`
- Fungsi `toUrlValue` Mengembalikan string dalam bentuk "lat, lng" untuk `LatLng` ini.

## 2.6 Marker



Gambar 2.2: Add Marker

*Marker* adalah sebuah kelas yang dapat memunculkan *mark* / tanda pada objek *map*. Untuk dapat menginisialisasi kelas *marker* pengembang dapat menggunakan *constructor* yang memiliki struktur seperti:

```
1  Marker([opts])
2  Parameters:
3    opts:  MarkerOptions optional
```

Pada *constructor* *marker* terdapat parameter *markeroptions* yang bersifat optional. *MarkerOptions* merupakan sebuah objek yang dapat digunakan pada objek *mark*. *MarkerOptions* memiliki properti yang dapat digunakan seperti:

Tabel 2.1: Tabel Properti Pada Objek Marker

Properti	Deksripsi
Properti <i>anchorPoint</i> .	Nilai offset dari objek marker terhadap info window.
Properti <i>animation</i> .	Animasi yang akan digunakan ketika objek marker diinisialisasikan.
Properti <i>clickable</i> .	Properti penanda jika objek marker diklik.
Properti <i>crossOnDrag</i> .	Properti penanda jika objek marker didrag oleh pengguna.
Properti <i>cursor</i> .	Properti yang akan menunjukkan <i>cursor</i> ketika objek marker di <i>hover</i> .
Properti <i>draggable</i> .	Properti bertipe <i>boolean</i> yang akan menandakan apakah objek marker dapat dilakukan operasi <i>drag</i> .
Properti <i>icon</i> .	Properti untuk memberikan icon pada objek marker.
Properti <i>label</i> .	Properti untuk memberikan label pada objek marker.
Properti <i>map</i> .	Properti untuk menentukan objek map yang akan dipakai oleh marker.
Properti <i>opacity</i> .	Properti untuk mengakses nilai opacity dari objek marker
Properti <i>position</i> .	Properti untuk mengakses nilai posisi dari objek marker.

Contoh penginisialisasi *marker* pada objek *maps* adalah:

```

1   return new google.maps.Marker({
2       position: location,
3       label: labels[i % labels.length],
4   });

```

Kelas *marker* memiliki fungsi bawaan yang telah disediakan oleh antara lain seperti:

Tabel 2.2: Tabel Fungsi Pada Kelas Marker

Package	Deksripsi
getAnimation	Fungsi untuk mendapatkan animasi yang digunakan oleh objek marker
getClickable	Fungsi untuk mendapatkan status <i>clickable</i>
getCursor	Fungsi untuk mendapatkan nilai <i>cursor</i>
getDraggable	Fungsi untuk mendapatkan nilai <i>draggable</i>
getIcon	Fungsi untuk mendapatkan nilai <i>icon</i>
getMap	Fungsi untuk mendapatkan objek <i>map</i>
getOpacity	Fungsi untuk mendapatkan objek <i>opacity</i>
getPosition	Fungsi untuk mendapatkan objek <i>position</i>
getShape	Fungsi untuk mendapatkan <i>shape</i> dari objek <i>marker</i>
getTitle	Fungsi untuk mendapatkan <i>title</i> dari objek <i>marker</i>

## 2.7 Marker Clusterer

*MarkerClustererPlus* merupakan sebuah *library* tambahan untuk dapat mengelompokan objek marker [2.6](#). Visualisasi dari *MarkerClustererPlus* dapat dilihat pada gambar [2.3](#)



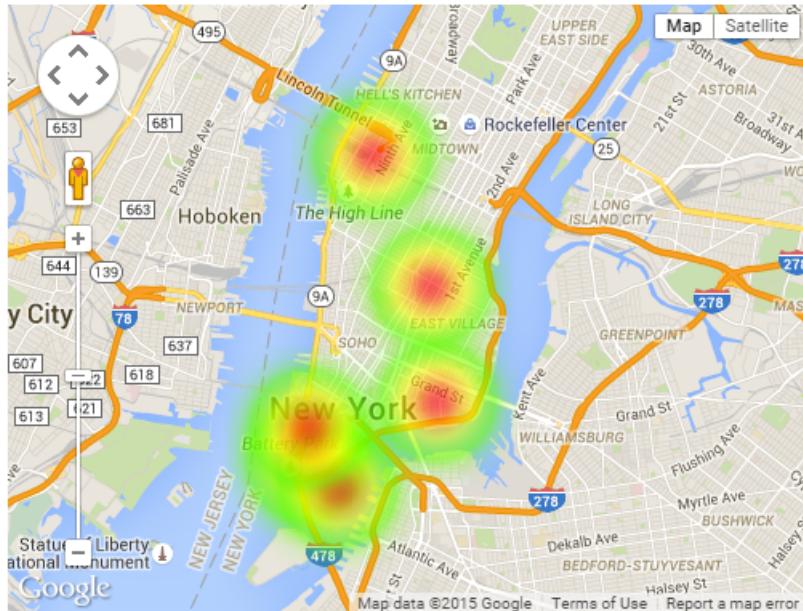
Gambar 2.3: Contoh Marker Clustering

Untuk dapat menginisialisasi kelas tersebut pihak pengembang perlu menuliskan perintah:

```
1 var markerCluster = new MarkerClusterer(map, markers,
2   {imagePath: '${path}/m'});
```

*Marker Clusterer* sendiri merupakan suatu library untuk kelas *mark* sehingga kelas ini dapat menggunakan fungsi turunan dari kelas *mark*.

## 2.8 HeatMap



Gambar 2.4: Contoh HeatMap

*Google Maps Javascript API* telah menyediakan kelas *heatmap* untuk menampilkan *heatmap* pada objek [maps 2.4](#). Untuk dapat menginisialisasi kelas ini pengembang perlu memanggil perintah *constructor* yang memiliki struktur seperti:

```
1     HeatmapLayer([opts])
2     Parameters:
3       opts:  HeatmapLayerOptions optional
```

Kelas *HeatMap* dilengkapi dengan parameter *HeatmapLayerOptions* yang bersifat opsional, objek ini bertujuan untuk dapat mengatur *property* dari kelas *HeatMap*. Parameter *HeatMapLayerOptions* merupakan sebuah objek yang memiliki atribut sebagai berikut:

- data *titik-titik* data yang diperlukan.
- dissipating variabel yang menentukan apakah *heatmap* akan menghilang jika *maps* diperbesar atau diperkecil.
- gradient gradient dari warna *heatmap*
- map attribute untuk menunjukkan peta dimana *heatmap* akan ditampilkan.
- maxIntensity nilai maximal dari intensitas warna pada *heatmap*.
- opacity nilai opacity dari *heatmap*.
- radius nilai radius dari *heatmap*.

Contoh penginisialisasi kelas *heatmap*:

```
1 let heatmap = new google.maps.visualization.HeatmapLayer({
2   data: heatmapData
3 });
```

Kelas *HeatMap* memiliki fungsi-fungsi bawaan yang telah disediakan oleh *google maps* beberapa fungsi tersebut adalah:

- *getData()* fungsi ini akan mengembalikan data point pada objek *heatmap*.
- *getMap()* fungsi ini akan mengembalikan objek *map*.
- *setData(data)* fungsi ini akan memasukan data pada objek *heatmap*.
- *setMap(map)* fungsi ini akan memasukan objek *map* pada objek *heatmap*.
- *setOption(option)* fungsi ini akan memasukan objek *HeatMapLayerOptions* pada objek *heatmap*.

## BAB 3

### ANALISIS

Pada bab ini dijelaskan mengenai deskripsi perangkat lunak, analisis data histori KIRI, analisis perangkat lunak, dan analisis *heat map*, dan *marker clustering* menggunakan *Google Maps Javascript API*.

#### 3.1 Analisis Data Histori KIRI

Perangkat lunak akan menggunakan data histori KIRI sebagai sumber data untuk melakukan visualisasi data. Data histori KIRI memiliki format *csv*. Data histori KIRI memiliki lima atribut yaitu:

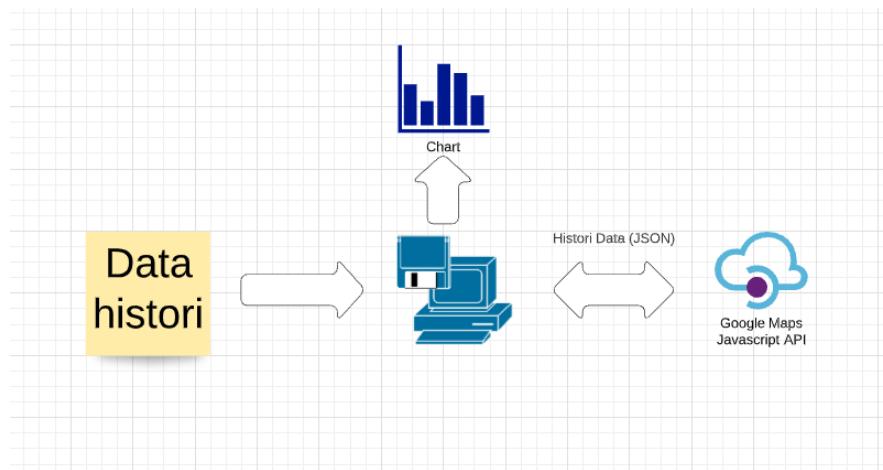
- LogId sebagai penanda satu record didalam data histori KIRI.
- APIKey sebagai atribut api key yang digunakan ketika melakukan perintah pada perangkat lunak KIRI.
- Timestamp (UTC) atribut untuk mencatat waktu perintah dilakukan format berbentuk *timestamp*.
- Action jenis action yang dilakukan user pada saat menggunakan perangkat lunak KIRI. Terdapat empat nilai action pada data histori KIRI yakni:
  - *PAGELOAD*.
  - *SEARCHPLACE*.
  - *WIDGETLOAD*.
  - *FINDROUTE*.
- AdditionalData atribut yang digunakan untuk mencatat informasi tambahan berdasarkan action yang dipilih. Nilai pada additionaldata akan bergantung pada action yang dipilih:
  - Jika action bernilai *PAGELOAD* maka additionaldata akan bernilai ip dari pengakses.
  - Jika action bernilai *SEARCHPLACE* maka additionaldata akan bernilai keyword yang dituliskan oleh pengakses.
  - Jika action bernilai *FINDROUTE* maka additionaldata akan bernilai posisi tempat dan tujuan dalam bentuk longitude dan latitude yang dicari oleh pengakses.
  - Jika action bernilai *WIDGETLOAD* maka additionaldata akan bernilai alamat url dari penyedia widget.

### 3.2 Deskripsi Perangkat Lunak

Pada skripsi ini akan dibangun aplikasi yang bertujuan untuk menemukan pola dengan tool visualisasi. Aplikasi ini akan menggunakan data histori perangkat lunak KIRI sebagai sumber data. Aplikasi ini akan dibangun menggunakan *tools nodejs* dan akan mengimplementasikan *google maps javascript api* sebagai tool untuk melakukan visualisasi data. Beberapa fitur yang dirancang dalam perangkat lunak ini:

- Perangkat lunak dapat memfilter data berdasarkan atribut *start/finish*.
- Perangkat lunak dapat memfilter data berdasarkan atribut waktu.
- Perangkat lunak dapat memfilter data berdasarkan atribut hari.
- Perangkat lunak dapat menampilkan data dalam bentuk *heat map*
- Perangkat lunak dapat menampilkan data dalam bentuk *marker clustering*.

Perangkat lunak ini akan melakukan filter data dan memproses data histori kiri dan akan menampilkan data tersebut kedalam bentuk *heat map* dan *marker clustering* untuk mendapatkan pola-pola tertentu berdasarkan data histori tersebut. Gambaran tentang alur komunikasi perangkat lunak dapat dilihat pada gambar 3.1



Gambar 3.1: Alur Komunikasi

1. Perangkat lunak akan mengolah data histori kiri sesuai dengan input yang diberikan.
2. Data yang sudah diolah akan diubah kedalam format *JSON*.
3. Data yang sudah diolah akan digunakan oleh *Google Maps Javascript API* untuk dapat dilakukan visualisasi data.
4. Perangkat lunak akan memvisualisasikan data kedalam bentuk *heat map* dan *marker clustering*.

### 3.3 Analisis Perangkat Lunak

Pada subbab ini akan dilakukan analisis untuk aplikasi visualisasi data histori KIRI.

### 3.3.1 Analisis Kebutuhan Perangkat Lunak

Berdasarkan latar belakang, rumusan masalah, dan tujuan maka dapat didefinisikan kebutuhan - kebutuhan perangkat lunak visualisasi data histori KIRI sebagai berikut:

- Data histori KIRI Aplikasi dapat melakukan proses filter pada data histori dan menggunakan data histori KIRI untuk dapat divisualisasikan oleh karena itu diperlukan raw data histori KIRI.
- Google Maps Javascript API Aplikasi dapat melakukan visualisasi data menggunakan *heat map* dan *marker clustering* yang merupakan fitur dari *google maps javascript api*.

### 3.3.2 Use Case Diagram

Interaksi antara pengguna dengan perangkat lunak yang dibangun pada skripsi ini digambarkan pada diagram *use case* 3.2



Gambar 3.2: Use Case Diagram

### 3.3.3 Use Case Skenario

Setiap fungsi yang ada pada diagram *use case* dijelaskan dengan skenario untuk memberi gambaran interaksi pengguna dengan sistem.

Tabel 3.1: Tabel Skenario Memfilter Data

Nama	Memfilter Data .
Deskripsi	Melakukan filter data berdasarkan kategori dari data histori kiri
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan sudah dapat mengolah raw data histori KIRI.
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem memuat aplikasi.</li> <li>2. Sistem menampilkan data.</li> <li>3. Pengguna dapat memfilter data berdasarkan waktu,hari, dan <i>start / finish</i>.</li> <li>4. Sistem melakukan proses filtering berdasarkan atribut yang telah dipilih pengguna.</li> </ol>

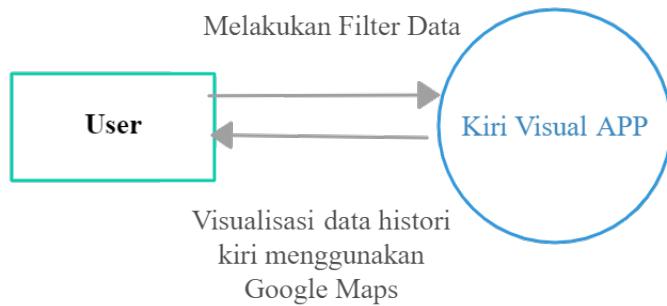
Tabel 3.2: Tabel Skenario Melihat Visualisasi Data Pada Map

Nama	Melihat Visualisasi Data Pada Map.
Deskripsi	Melihat hasil visualisasi data pada google map
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan sudah dapat menampilkan data histori KIRI.
Alur Skenario Utama	<ol style="list-style-type: none"> <li>1. Sistem menampilkan data yang telah difilter oleh pengguna.</li> <li>2. Pengguna dapat memilih akan menggunakan metode visualisasi <i>heat map</i> atau <i>marker clustering</i>.</li> <li>3. Sistem menampilkan hasil visualisasi data berdasarkan metode yang telah dipilih pengguna.</li> </ol>

### 3.3.4 Data Flow Diagram

Interaksi data dalam perangkat lunak yang dibangun pada skripsi ini digambarkan pada *data flow diagram* 3.3

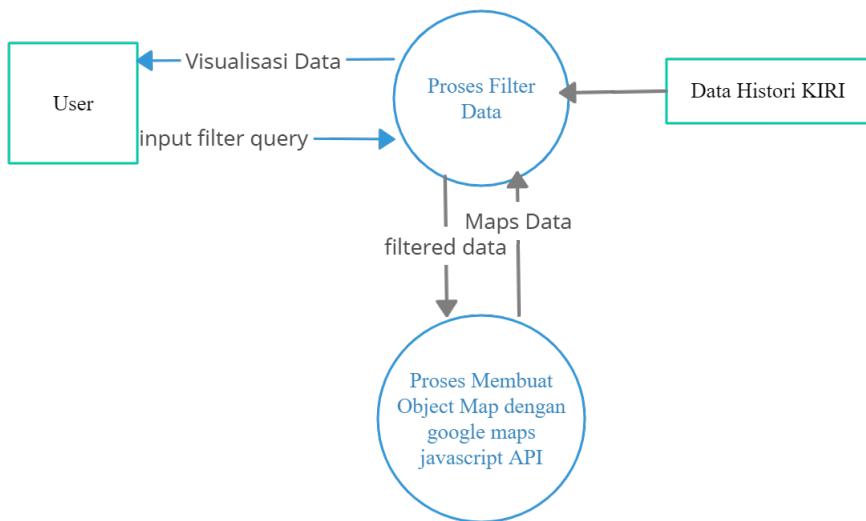
LEVEL 0:



Gambar 3.3: Data Flow Diagram Level 0

Pada level 0 ini dapat dilihat pada gambar 3.3, perangkat lunak yang akan dibuat memiliki dua buah flow utama yaitu user dapat melakukan penyaringan data, dan user dapat menerima hasil penyaringan data dalam bentuk visualisasi pada *google maps*. Setelah dilakukan pengujian lebih lanjut maka didapatkan *data flow diagram* level 1 3.4

LEVEL 1:



Gambar 3.4: Data Flow Diagram Level 1

Pada level 1 ini dapat dilihat pada gambar 3.4, perangkat lunak akan memiliki dua buah proses utama yaitu:

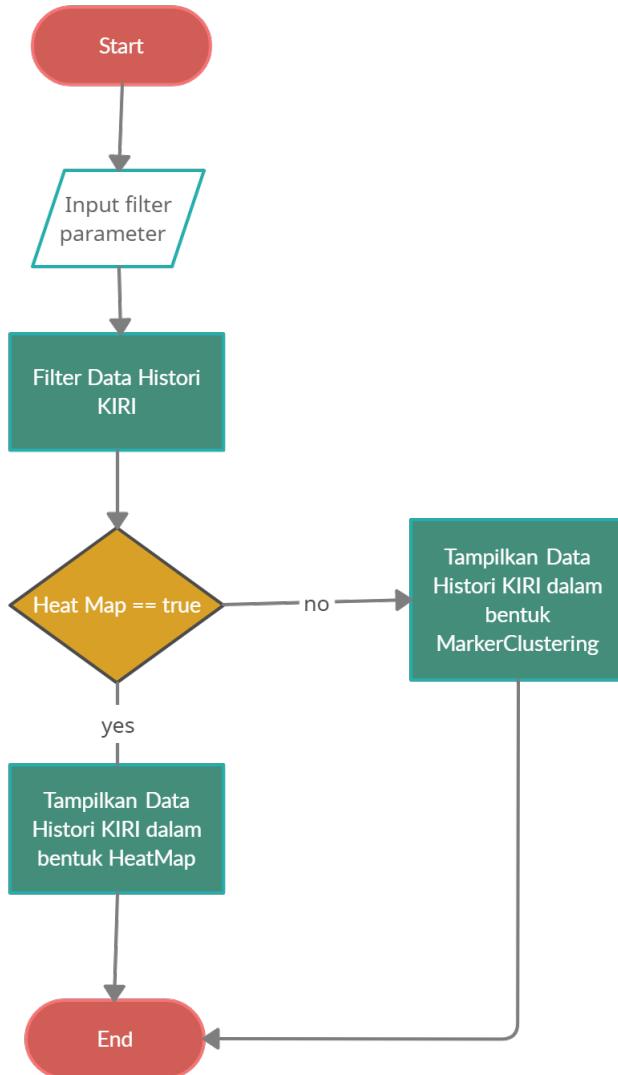
- Filter data proses untuk melakukan penyaringan data histori kiri berdasarkan *query input* dari user.
- Proses pembuatan object *maps* proses yang akan menerima data histori KIRI dan membuat objek map menggunakan *google maps javascript api*.

Berdasarkan *data flow diagram* pada gambar 3.4, pada perangkat lunak yang akan dibuat user dapat melakukan tiga buah aksi:

1. Filter data histori KIRI user dapat memasukan filter parameter untuk melakukan proses penyaringan data histori KIRI, dan perangkat lunak akan melakukan proses penyaringan berdasarkan filter parameter yang dipilih user.
2. Select mode visualisasi user dapat memilih mode visualisasi pada perangkat lunak ini, akan disediakan dua mode visualisasi *heatmap* dan *marker clustering*.
3. Visualisasi data dengan google maps pada perangkat ini user akan menerima output berupa data yang telah tervisualisasi dengan google map.

### 3.3.5 Flow Chart Diagram

Alur perangkat lunak yang akan dibuat dapat digambarkan kedalam bentuk *flow chart* 3.5



Gambar 3.5: KIRI Flow Chart

### 3.3.6 Input dan Output

Input dan output yang akan diterima oleh perangkat lunak yang akan dibuat adalah sebagai berikut.

- Input Filter Parameter filter parameter adalah semua parameter yang dapat dipilih user untuk dapat melakukan penyaringan data histori KIRI, pada perangkat lunak yang akan dibangun user dapat memilih tiga filter parameter.
- Input mode visualisasi pada perangkat lunak yang akan dibangun user dapat memilih dua mode visualisasi yaitu *mode heatmap* dan *mode markerclustering*.
- Output visualisasi data output pada perangkat lunak yang akan dibangun adalah hasil visualisasi data histori KIRI dalam bentuk *marker clustering* atau bentuk *heat map*.



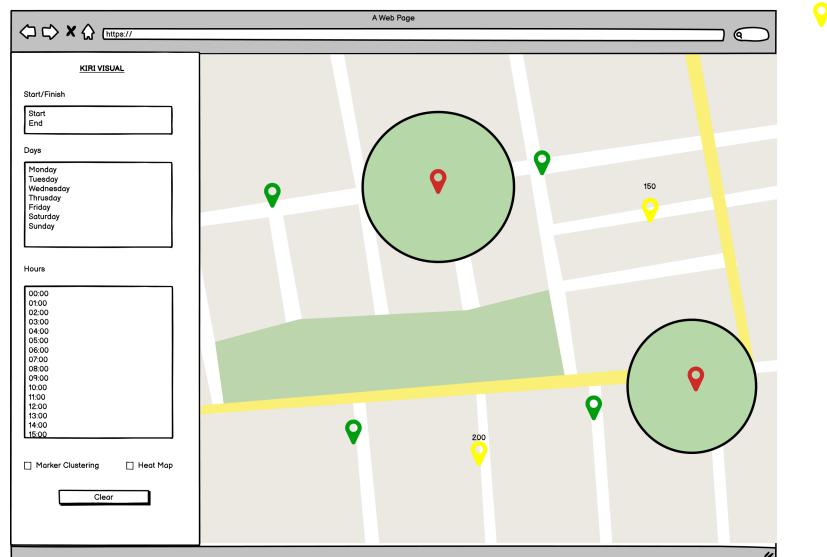
## BAB 4

# PERANCANGAN

Bab ini akan menjelaskan perancangan aplikasi visualisasi data histori KIRI pada google map

### 4.1 Perancangan Antarmuka

Pada aplikasi, disediakan antarmuka untuk memudahkan pengguna dalam berinteraksi dengan perangkat lunak 4.1.



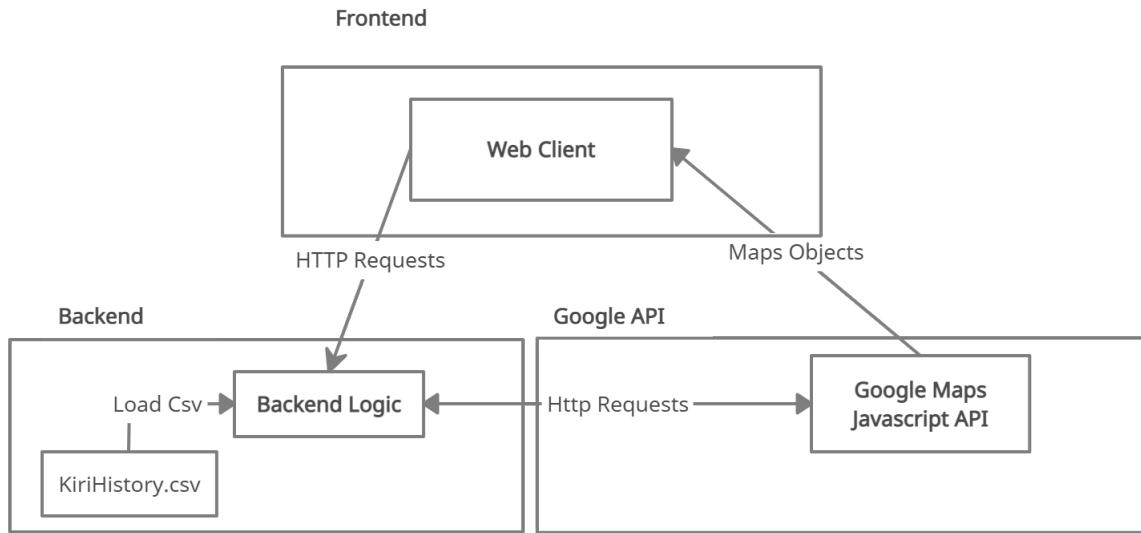
Gambar 4.1: Rancangan Antarmuka

Berdasarkan rancangan diatas, berikut adalah fungsi dari setiap komponen dalam antarmuka:

- *Map*: digunakan untuk menampilkan peta *google map*.
- *Checkbox Start*: untuk memfilter data berdasarkan tempat keberangkatan.
- *Checkbox End*: untuk memfilter data berdasarkan tempat tujuan.
- *Selection Box Hours*: untuk memfilter data berdasarkan jam.
- *Selection Box Days*: untuk memfilter data berdasarkan hari.
- *Checkbox Heat Map*: untuk menampilkan data dalam bentuk *heat map*.
- *Checkbox Marker Clustering* : untuk menampilkan data dalam bentuk *marker clustering*.
- Button *Clear*: menghapus seluruh *overlay* pada objek *map*.

## 4.2 Perancangan Diagram Fungsi

Berdasarkan hasil analisis pada 3.3.1, maka untuk aplikasi ini dapat dibuat diagram fungsi seperti 4.2



Gambar 4.2: Rancangan Diagram Fungsi

Aplikasi ini merupakan aplikasi berbasis website dimana akan memanfaatkan *node.js* yang akan berperan sebagai *asynchronous event-driven* 2.3 Sehingga backend logic dapat menggunakan *javascript* tanpa menggunakan *web browser*. Pada aplikasi ini terdiri dari tiga bagian yakni:

- Frontend pada bagian frontend terdiri dari *web client* yang berfungsi untuk menerima input dan mengeluarkan output dari *user*. Webclient akan berkomunikasi dengan *backend* dengan menggunakan *http request*.
- Backend akan menggunakan *node.js* akan menerima input dari web client pada bagian ini juga akan dilakukan *load* data histori kiri yang berbentuk csv untuk dapat diolah.
- Google API aplikasi ini akan memanfaat *google maps javascript api* untuk dapat memvisualisasikan data history KIRI.

## 4.3 Perancangan Diagram Kapabilitas

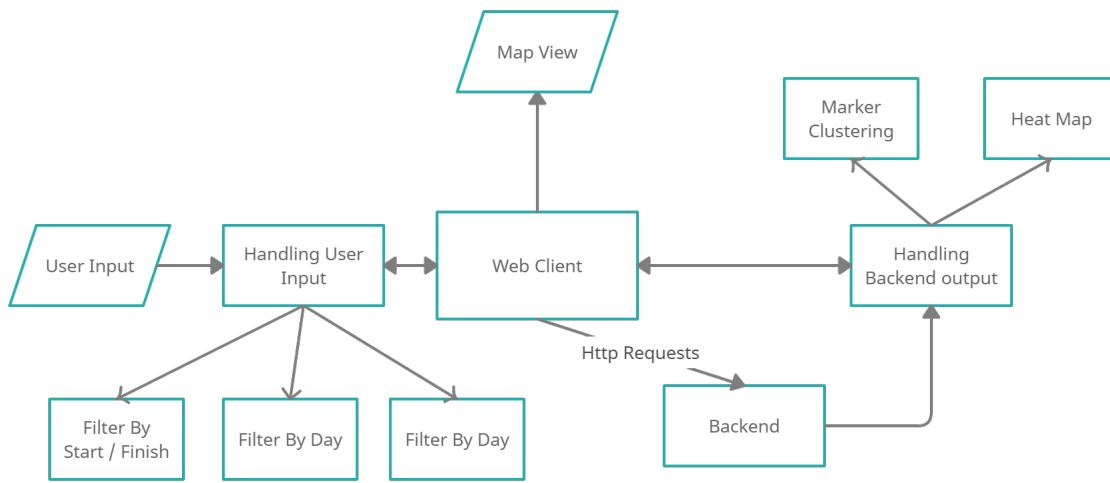
Berdasarkan kapabilitas dari masing masing komponen 4.2, maka dapat dibuat diagram kapabilitas.

### 4.3.1 Diagram Kapabilitas Web Client

Pada bagian *web client* akan memiliki kapabilitas:

- Menampilkan Google Map.
- Menampilkan *heatmap* atau *marker clustering* object.
- Menerima input dari user. Input dari user berupa parameter yang berguna untuk memfilter data histori kiri dan menampilkannya dalam bentuk *heatmap* atau *marker clustering*.

Berdasarkan kapabilitas 4.3.1 maka dapat dibuat diagram yang berbentuk seperti 4.3



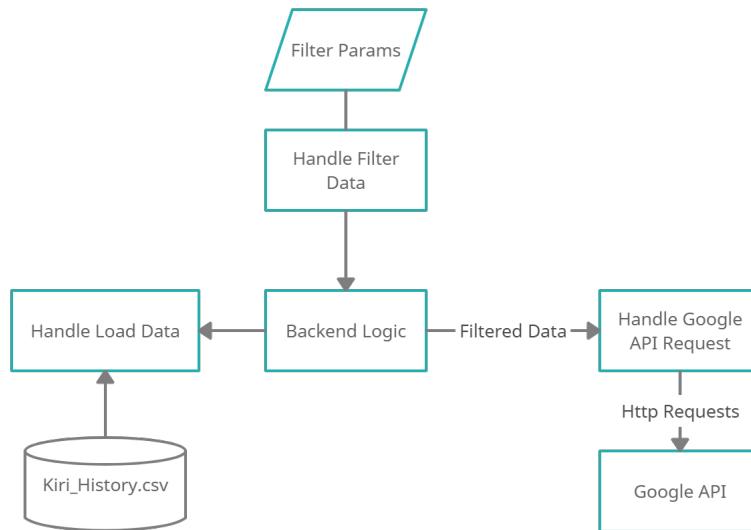
Gambar 4.3: Diagram Kapabilitas Web Client

#### 4.3.2 Diagram Kapabilitas Backend

Pada aplikasi yang akan dibangun backend memiliki kapabilitas:

- Menerima *input* dari *web client*. Input yang diterima berupa filter parameter yang telah diolah oleh *web client*.
- Melakukan filterisasi data berdasarkan *input parameter* dari *web client*.
- Melakukan komunikasi *via http request* terhadap *google api*.

Berdasarkan kapabilitas 4.3.2 maka dapat dibuat diagram yang berbentuk seperti 4.4



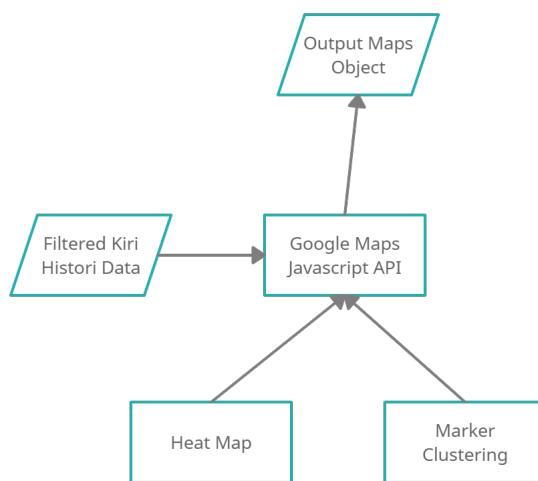
Gambar 4.4: Diagram Kapabilitas Backend

### 4.3.3 Diagram Kapabilitas Google API

Pada aplikasi ini akan menggunakan *google maps javascript api* sebagai *third party library* dalam menampilkan hasil visualisasi data yang dilakukan oleh *backend*. *Google Maps Javascript API* pada perangkat lunak ini memiliki kapabilitas antara lain:

- Menerima *input* dari *Backend*. Input yang diterima berupa data histori kiri yang telah disaring sesuai dengan *input* dari user.
- Mengeluarkan *output* berupa objek *map* yang akan dirender oleh *web client*. Objek *map* dapat berupa *marker cluster* objek atau *heatmap* objek tergantung pada input yang diterima dari *backend*.

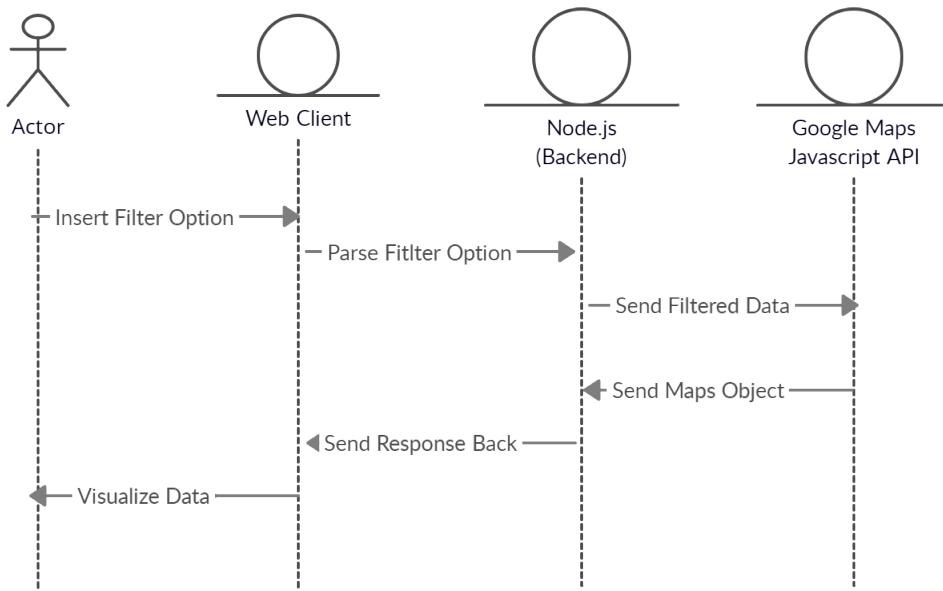
Berdasarkan kapabilitas 4.3.3 maka dapat dibuat diagram yang berbentuk seperti 4.5



Gambar 4.5: Diagram Kapabilitas Goolge API

### 4.4 Perancangan Diagram Interaksi

Pada subbab ini akan dijelaskan interaksi antar komponen yang sesuai dengan yang dijelaskan pada subbab 3.2, dengan diagram interaksi untuk setiap fitur pada aplikasi visualisasi data histori kiri.



Gambar 4.6: Diagram Interaksi Visualisasi Data Histori Kiri

Fitur pada perangkat lunak ini akan diawali dengan user dapat memilih *input filter*. *Input filter* yang dapat dipilih adalah filter berdasarkan atribut *start* atau *finish*, *days*, dan *time*. Setelah filter option dipilih oleh user maka *web client* akan mengubah hasil option tersebut menjadi *filter params*, *filter params* adalah sebuah objek yang akan digunakan oleh *backend logic* untuk melakukan proses *filter* terhadap data histori KIRI. Setelah *backend logic* menerima *filter params* maka proses *filter* akan dilakukan sesuai dengan isi dari *filter params*, aksi ini akan menghasilkan data histori kiri yang sudah disaring sesuai dengan isi dari *filter params*. *Backend logic* akan mengirimkan hasil data disaring tersebut ke *google maps javascript api* dengan lalu *google maps javascript api* akan mengeluarkan maps object berdasarkan data yang dikirim oleh *backend logic*. Data tersebut akan ditampilkan kembali oleh *web client* dalam bentuk data visualisasi menggunakan *google map*.

## 4.5 Perancangan Pseudocode

Pada subbab ini dirancang *pseudocode* untuk membuat aplikasi visualisasi data histori KIRI. *Pseudocode* yang dibuat akan mengacu pada kapabilitas setiap komponen 4.3.

#### 4.5.1 Perancangan Pseudocode Pada Web Client

Berdasarkan kapabilitasnya 4.3.1. Pseudocode dapat dibuat berdasarkan dua fungsi utama yakni:

- Handling User Input. Pada tahap ini web client akan menerima input dari user dan akan mengubah nya menjadi *filter params*.
- Menampilkan data histori kiri dengan menggunakan *google maps*.

Tabel 4.1: Handling User Input

Nama	Handling User Input.
Deskripsi	Fungsi yang bertugas untuk menerima input user dan mengubah inputan user menjadi <i>filter params</i>
Pre-kondisi	Aplikasi sudah menerima hasil input dari user.
Input	<p>Input yang akan diterima oleh aksi ini antara lain:</p> <ul style="list-style-type: none"> <li>• <i>Days</i> aksi akan menerima <i>input days</i> yang bertipe <i>array of number</i> dengan <i>range</i> antara 0 sampai 6 yang merepresentasikan hari.</li> <li>• <i>Hours</i> aksi akan menerima <i>input hours</i> yang bertipe <i>array of number</i> dengan <i>range</i> antara 0 sampai 23 yang merepresntasikan jam.</li> <li>• <i>isHeatMap</i> aksi akan menerima parameter bertipe <i>boolean</i> yang menandakan apakah user memilih menggunakan <i>map</i> bertipe <i>heat map</i>.</li> <li>• <i>isMarkerClustering</i> aksi akan menerima parameter bertipe <i>boolean</i> yang menandakan apakah user memilih menggunakan <i>map</i> bertipe <i>marker clustering</i>.</li> </ul>
Output	<p>Output yang akan dihasilkan oleh aksi ini adalah object <i>filter params</i>. <i>Filter params</i> merupakan sebuah objek yang memiliki dua buah atribut antara lain:</p> <ul style="list-style-type: none"> <li>• Atribut <i>day</i> atribut ini adalah atribut untuk menampung <i>array of number</i> yang merepresentasikan hari.</li> <li>• Atribut <i>hour</i> atribut ini adalah atribut untuk menampung <i>array of number</i> yang merepresentasikan jam.</li> </ul>

Pseudocode untuk aksi 4.5.1 dapat digambarkan seperti:

---

##### Algorithm 1 Handle User Input

```

1: function CREATEFILTEROBJ
2:   days ← array of days
3:   hours ← array of hours
4:   filterParams ← object of selected input
5:   filterParams.days → days
6:   filterParams.hours → hours
7:   return filterParams

```

---

Tabel 4.2: Menampilkan data histori kiri dengan menggunakan *google maps*

Nama	Handling Output.
Deskripsi	Fungsi yang bertugas untuk menampilkan <i>maps object</i> yang dikirimkan oleh <i>backend</i> .
Pre-kondisi	Aplikasi sudah menerima objek <i>map</i> dari <i>backend</i> .
Input	<p>Aksi memerlukan input antara lain</p> <ul style="list-style-type: none"> <li>• Selected <i>map type</i>. tipe <i>map</i> yang dapat dipilih user adalah <i>heat map</i> atau <i>marker clustering</i>.</li> <li>• Objek <i>map</i> yang telah dikirimkan oleh <i>google maps javascript api</i>.</li> </ul>
Output	Output yang akan dihasilkan oleh aksi ini adalah hasil visualisasi dari objek <i>map</i> yang diterima dari <i>backend</i> . Hasil visualisasi dapat berbentuk <i>Heat Map</i> atau <i>Marker Clustering</i>

**Algorithm 2** Handle Output

---

```

1: function SETMAP(isMarkerCluster,data)
2:   map ← google maps object
3:   data ← response.data
4:   if isMarkerCluster == true then
5:     markers ← setMarkers(data)
6:     markerCluster ← MarkerClusterer(map , data)
7:   else
8:     heatMap ← setHeatmap(data)
9:   end if
10:

```

---

**Perancangan Pseudocode Pada Backend Server**

Berdasarkan kapabilitasnya 4.3.2. Pseudocode dapat dibuat berdasarkan dua fungsi utama yakni:

- Handling load data histori KIRI.
- Handling filter data histori KIRI.

Tabel 4.3: Load data histori KIRI

Nama	Load Data.
Deskripsi	Fungsi ini adalah fungsi yang berperan untuk meload data histori KIRI.
Pre-kondisi	Data histori kiri sudah tersedia didalam folder asset dan harus bertipe <i>csv</i> .
Input	<p>aksi memerlukan input antara lain</p> <ul style="list-style-type: none"> <li>• Data histori KIRI berbentuk <i>csv</i>.</li> </ul>
Output	<i>Array of Object</i> data histori kiri.

**Algorithm 3** Load Data

---

```

1: function LOADDATA
2:   data  $\leftarrow$  array of objects
3:   data  $\leftarrow$  fs.readFile("KiriHistory.csv")
4:   return data
5:
6:
```

---

Tabel 4.4: Handling Filter Data

Nama	Filter Data.
Deskripsi	Fungsi ini bertugas untuk memfilter data berdasarkan filter parameter yang telah dikirimkan oleh <i>web client</i> .
Pre-kondisi	<i>Web client</i> telah mengirimkan <i>filter param</i> .
Input	Aksi memerlukan input antara lain <ul style="list-style-type: none"> <li>• Filter param.</li> </ul>
Output	<i>Array of Object</i> data histori kiri yang telah disaring.

**Algorithm 4** Filter Data

---

```

1: function FILTERDATA(filterParams)
2:   data  $\leftarrow$  array of objects
3:   data  $\leftarrow$  data.filter(filterParams)
4:   return data
5:
6:
```

---

## BAB 5

# IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas implementasi, pengujian, dan masalah yang dihadapi. Pada bagian implementasi akan dijelaskan mengenai lingkungan implementasi dan hasil dari implementasi. Pada bagian pengujian akan berisi hasil dari pengujian. Pada bagian masalah yang dihadapi akan dijelaskan masalah - masalah yang dihadapi pada saat implementasi.

### 5.1 Implementasi

#### 5.1.1 Lingkungan Implementasi

Berikut adalah spesifikasi *laptop* yang digunakan untuk implementasi:

1. *Processor* : AMD Ryzen 7
2. *Memory* : 16 GB DDR4 2400MHz SDRAM
3. *Storage* : 512 GB SSD
4. *VGA* : NVDIA GTX 1660TI
5. *OS* : Windows 10 64-bit

Berikut adalah spesifikasi *browser* yang digunakan:

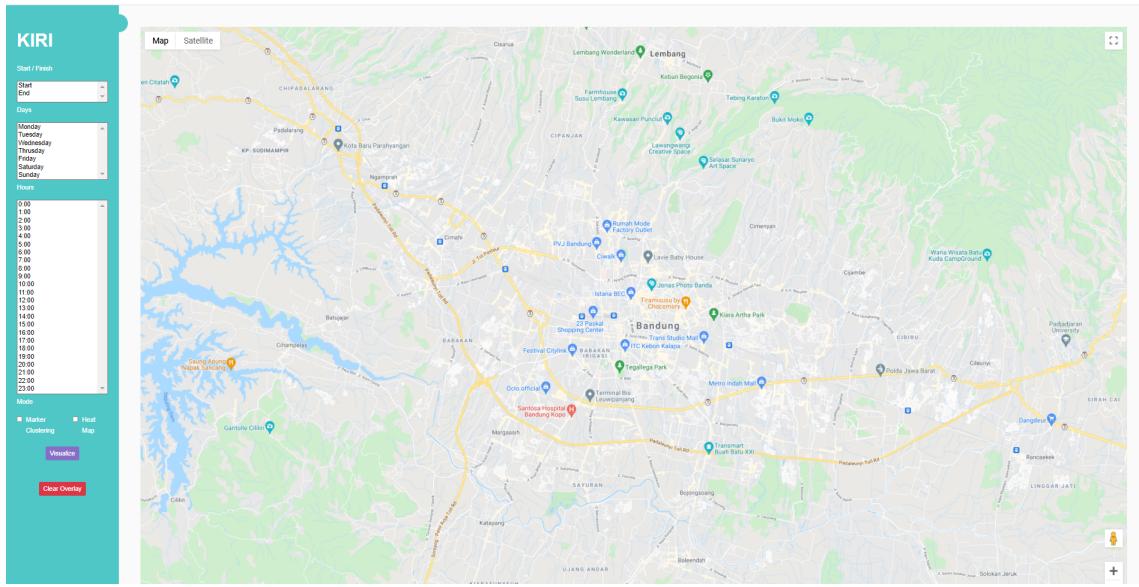
1. *Name* : Google Chrome
2. *Version* : 90.0.4430.212

Berikut adalah spesifikasi perangkat lunak yang digunakan untuk implementasi:

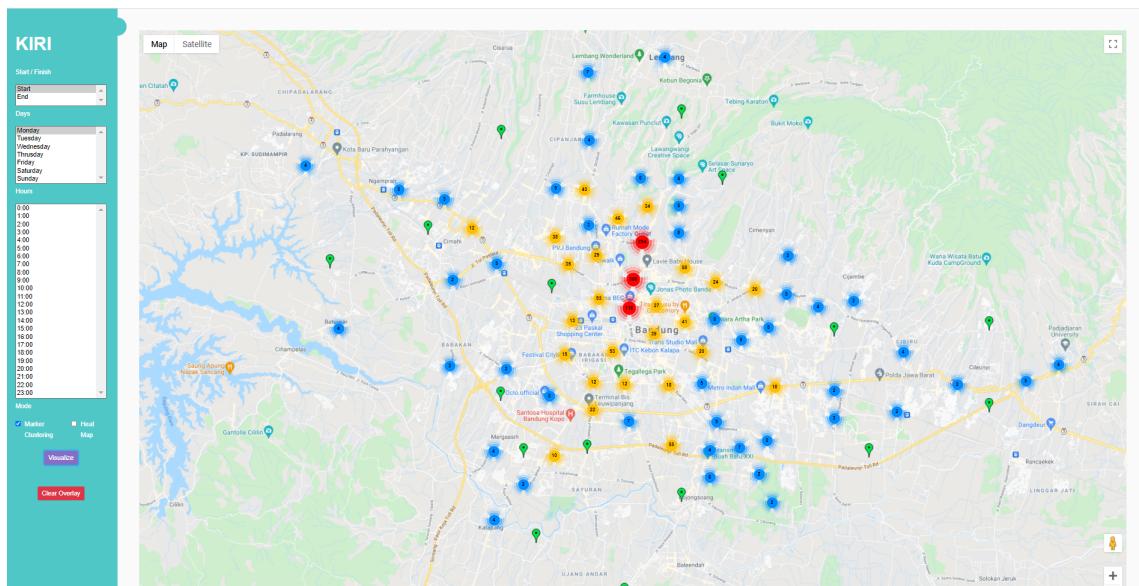
1. IDE : Webstrom 2021.1.1
2. Bahasa Pemrograman : Javascript
3. *Runtime Enviroment* : node.js 14.17.0 LTS

#### 5.1.2 Implementasi Antarmuka Perangkat Lunak

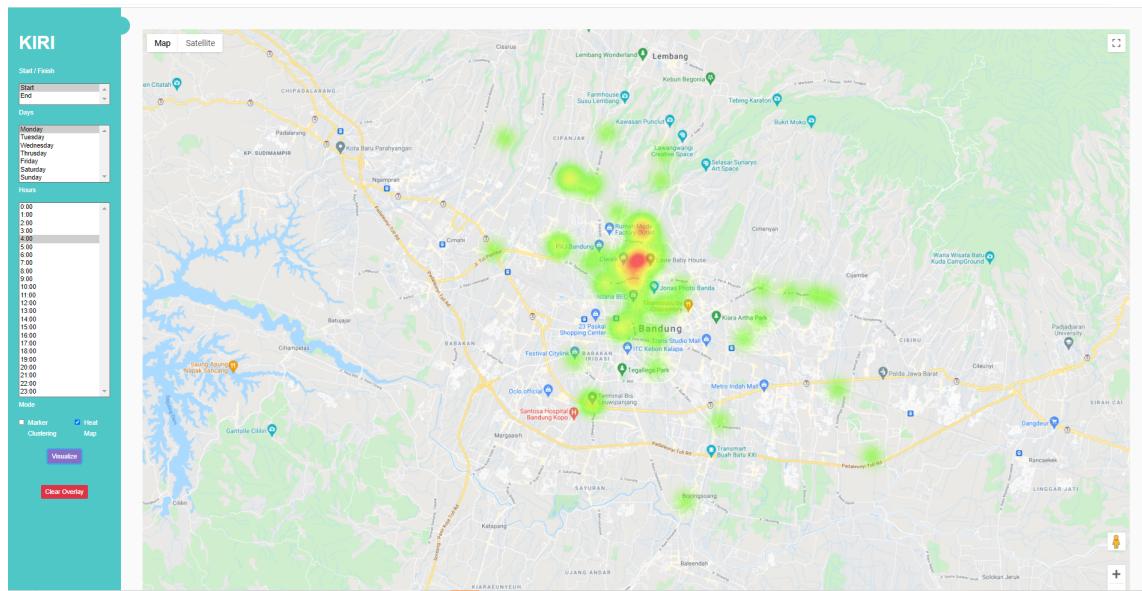
Pada implementasinya, antarmuka perangkat lunak telah berhasil dirancang sesuai dengan subbab 4.1. Berikut adalah hasil implementasinya:



Gambar 5.1: Tampilan Awal Antarmuka



Gambar 5.2: Tampilan setelah memilih marker clustering



Gambar 5.3: Tampilan setelah memilih Heat Map

### 5.1.3 Implementasi Perangkat Lunak

Perangkat lunak yang dibuat sesuai dengan perancangan pada Bab 4. Implementasi aplikasi ini menggunakan bahasa pemrograman Javascript. Terdapat tiga bagian utama dalam perangkat lunak yang dibuat.

- *Model* bagian ini adalah representasi dari data, bagian ini juga dapat disebut sebagai *backend*. Pada bagian ini terdapat kelas *model* untuk data histori KIRI. Pada bagian ini juga terdapat kelas *helper* untuk melakukan operasi pengolahan data.
- *Router* bagian ini adalah bagian yang menjadi *controller* dalam perangkat lunak. *Router* akan berguna sebagai *media* komunikasi antara *web client* dengan *backend server*
- *View* bagian ini adalah bagian tampilan atau yang bisa disebut *web client* karena pada perangkat lunak ini menggunakan platfrom *website*. Bagian ini bertugas untuk menampilkan hasil visualisasi dan mengolah input dari user.

#### Model

Bagian ini adalah representasi dari data, bagian ini juga dapat disebut sebagai *backend*. Pada bagian ini terdapat kelas *model* untuk data histori KIRI. Pada bagian ini juga terdapat kelas *helper* untuk melakukan operasi pengolahan data. Berikut ini adalah gambaran kelas *model* data histori KIRI:

```

1 const fs = require("fs")
2 const {csvToObject, buildFilter, filterData} = require("./Utils");
3
4 class KiriHistory {
5     constructor() {
6         fs.readFile(_dirname + "/data/KIRIStatistics.csv", "utf8", ((err, data) => {
7             if (err === null) {
8                 let arrCSV = data.split("\n")
9                 arrCSV.shift();
10                this.data = arrCSV.map(csvToObject).filter(item => item !== undefined);
11            }
12        }))
13    }
14
15    //return promise
16    getData = (filterParams) => {
17        let query = buildFilter(filterParams);
18        this.data = filterData(this.data, query);
19        return this.data;
20    }
21
22 }
23
24 module.exports = KiriHistory;

```

Listing 5.1: Metode Load Data

Pada kelas model ini memiliki dua action utama yaitu:

## Load Data

Action ini akan dijalankan begitu objek *model* dibuat. Action ini akan melakukan load dan normalisasi data histori kiri. Action ini akan menggunakan method *readFile* yang berasal dari *package filesystem*. Berikut ini adalah contoh implementasi dari action ini.

```

1 constructor() {
2     fs.readFile(_dirname + "/data/KIRIStatistics.csv", "utf8", ((err, data) => {
3         if (err === null) {
4             let arrCSV = data.split("\n")
5             arrCSV.shift();
6             this.data = arrCSV.map(csvToObject).filter(item => item !== undefined);
7         }
8     }))
9 }

```

Listing 5.2: Metode Load Data

Method *readFile* akan melakukan action load pada data histori KIRI [3.1](#). Data yang akan diload bertipe *csv* dan memiliki format data sebagai berikut.

```

1 logId, APIKey, Timestamp (UTC), Action, AdditionalData
2 113909, E5D9904F0A8B4F99, 2/1/2014 0:07, PAGELOAD, /5.10.83.30/
3 113914, A44EB361A179A49E, 2/1/2014 0:11, SEARCHPLACE, taman+fot/10
4 113915, A44EB361A179A49E, 2/1/2014 0:11, FINDROUTE, "
    -6.8972513,107.6385574/-6.91358,107.62718/1"
5 114256, 308201BB30820124, 2/1/2014 5:24, SEARCHPLACE, soekarno+hatta+643/10

```

Listing 5.3: Histori Data KIRI

Ketika *readFile* dijalankan *fungsi* ini akan memiliki *callback* yang akan mengembalikan data histori kiri, data yang telah diload oleh *readFile* akan berbentuk seperti:

```

1 [ 
2   'logId,APIKey,TimeStamp (UTC),Action,AdditionalData\r',
3   '113909,E5D9904F0A8B4F99,2/1/2014 0:07,PAGELOAD,/5.10.83.30\r',
4   '113910,E5D9904F0A8B4F99,2/1/2014 0:07,PAGELOAD,/5.10.83.49\r',
5   '113911,E5D9904F0A8B4F99,2/1/2014 0:09,PAGELOAD,/5.10.83.30\r',
6 ]

```

Listing 5.4: Raw Data

Data tersebut akan ditampung ke dalam variable yang diberinama *arrCSV*. Baris pertama dari data ini adalah *header* dari csv. Pada proses normalisasi data baris pertama ini tidaklah dibutuhkan oleh karena itu akan dihilangkan baris pertama menggunakan method *shift()*. Sehingga data akan berbentuk:

```

1 [ 
2   '113909,E5D9904F0A8B4F99,2/1/2014 0:07,PAGELOAD,/5.10.83.30\r',
3   '113910,E5D9904F0A8B4F99,2/1/2014 0:07,PAGELOAD,/5.10.83.49\r',
4   '113911,E5D9904F0A8B4F99,2/1/2014 0:09,PAGELOAD,/5.10.83.30\r',
5 ]

```

Listing 5.5: Raw Data 2

Data tersebut kemudian akan dipetakan ke dalam *map* dengan menggunakan fungsi mapper yang diberi nama *csvToObject*. Fungsi ini akan memetakan dan melakukan *formating* dari setiap value dari 5.5.

```

1 const csvToObject = item => {
2   let cols = item.split(",")
3   let action = cols[3];
4   if (action === "FINDROUTE") {
5     let startLng = cols[5].split("//")[0]
6     let endLat = cols[5].split("//")[1]
7     let fullDate = new Date(cols[2])
8     return {
9       apiKey: cols[0],
10      timestamp: fullDate,
11      action,
12      startCor: {lat: parseFloat(cols[4].substr(1)), lng: parseFloat(
13        startLng)},
14      endCor: {lat: parseFloat(endLat), lng: parseFloat(cols[6].split("//")
15        [0])},
16      day: fullDate.getDay(),
17      hour: fullDate.getHours()
18    }
19  }
20}

```

Listing 5.6: CSV Mapper

Fungsi ini akan akan memanfaatkan *method split* untuk membagi raw data 5.5 menjadi array yang berbentuk:

```

1 [ 
2   '115566',
3   'A44EB361A179A49E',
4   '2/2/2014 9:07',
5   'FINDROUTE',
6   '"-0.7819376',
7   '100.2871169/-6.90359',
8   '107.60040/1"\r'
9 ]

```

Listing 5.7: Data Split Result

Berdasarkan data [5.7](#) kita dapat mengekstrak *action type* [3.1](#) pada posisi ketiga dari array tersebut. Dalam aplikasi ini hanya akan digunakan data yang memiliki action *FINDROUTE* hal ini dikarenakan hanya pada *action* ini data mengandung value dari posisi start dan finish. Setelah function *csvToObject* ini dijalankan maka akan mengembalikan kumpulan objek data yang berbentuk:

```

1 [ 
2   {
3     apiKey: '113915',
4     timestamp: 2014-01-31T17:11:00.000Z,
5     action: 'FINDROUTE',
6     startCor: { lat: -6.8972513, lng: 107.6385574 },
7     endCor: { lat: -6.91358, lng: 107.62718 },
8     day: 6,
9     hour: 0
10    }
11 ]

```

Listing 5.8: Result Data

## Get Data

Fungsi ini akan dijalankan setiap ada request dari *web client*. Action ini akan mengembalikan data histori KIRI sesuai dengan filter parameter yang diberikan. Berikut ini adalah contoh implementasi dari action ini:

```

1   getData = (filterParams) => {
2     let query = buildFilter(filterParams);
3     this.data = filterData(this.data, query);
4     return this.data;
5   }

```

Listing 5.9: Function Get Data

Pada fungsi ini akan menerima parameter *filterParams* yang berasal dari *web client*. Fungsi ini akan menjalankan dua perintah yaitu *buildFilter* dan *filterData* dan akan mengembalikan data histori yang sesuai dengan *filterParams* yang diberikan. Fungsi *buildFilter* akan mengubah *filterParams* yang berbentuk:

```

1 {
2   day: [ 0 ],
3   hour: [ 8, 9, 10, 11 ]
4 }

```

Listing 5.10: Filter Parameter

Menjadi *query params*. Hal ini dilakukan karena fungsi *filter* yang dirancang pada perangkat lunak ini akan memanfaatkan *function filter* dari javascript.

## Filter Data

Berikut ini adalah hasil implementasi dari function filter:

```

1 const filterData = (data, query) => {
2     const keysWithMinMax = [];
3     const filteredData = data.filter((item) => {
4         for (let key in query) {
5             if (item[key] === undefined) {
6                 return false;
7             } else if (keysWithMinMax.includes(key)) {
8                 if (query[key]['min'] !== null && item[key] < query[key]['min']) {
9                     return false;
10                }
11                if (query[key]['max'] !== null && item[key] > query[key]['max']) {
12                    return false;
13                }
14                } else if (!query[key].includes(item[key])) {
15                    return false;
16                }
17            }
18            return true;
19        });
20        return filteredData;
21    };

```

Listing 5.11: Fungsi Filter

Fungsi ini akan memanfaatkan *method filter dan includes javascript*. Fungsi ini akan menghasilkan data yang telah disaring sesuai dengan query params yang diberikan.

### 5.1.4 Router

Bagian ini merupakan bagian yang bertugas sebagai *controller* antara *backend logic* dan *web client*. Implementasi router dapat dilihat sebagai berikut:

```

1 const express = require('express')
2 const app = express()
3 const Kiri = require('../model/KiriHistory');
4 const path = require('path');
5 const viewPath = "C:/KiriHistory/views"
6 let modelKiri = new Kiri();
7 app.use(express.json())
8 app.get("/", (req, res) => {
9     res.sendFile(path.join(`${viewPath}/index.html`));
10 })
11
12
13 app.post('/searchRoute', (req, res) => {
14     let filterParams = req.body;
15     let data = modelKiri.getData(filterParams);
16     res.status(200).json({
17         'status': 'OK',
18         'messages': 'Data',
19         'data': data,
20     })
21
22 })
23 module.exports = {app, express};

```

Listing 5.12: Router Implementation

Bagian ini memiliki dua fungsi utama yaitu melakukan render untuk *web client* dan mengatur *response* dan *request* antara *web client* dan *backend logic*.

## Redner HTML

Perangkat lunak ini menggunakan *node.js* sebagai *runtime manager*. Perangkat lunak ini memanfaatkan method *sendFile* milik *node.js* untuk dapat melakukan *rendering* pada *route* yang diinginkan. Implementasi pada bagian ini dapat dilihat sebagai berikut:

```
1 app.get("/", (req, res) => {
2     res.sendFile(path.join(`${viewPath}/index.html`));
3 })
```

Listing 5.13: Render HTML

Pada potongan kode diatas menunjukan bahwa *index.html* akan dirender pada *route* '/'.

### 5.1.5 Controller antara *Web client* dan *Backend Logic*

Implementasi pada bagian ini dapat dilihat sebagai berikut:

```
1 app.post('/searchRoute', (req, res) => {
2     let filterParams = req.body;
3     let data = modelKiri.getData(filterParams);
4     res.status(200).json({
5         'status': 'OK',
6         'messages': 'Data',
7         'data': data,
8     })
9
10})
```

Listing 5.14: Render HTML

Pada bagian ini perangkat lunak akan membuat *endpoint* yang akan menerima parameter berupa *filter parameter* dan akan mengembalikan hasil data histori yang telah di filter.

## View

Bagian ini adalah bagian disebut *web client* 4.3.1. Pada bagian ini terdapat fungsi untuk menerima *input user* dan menampilkan hasil visualisasi menggunakan *Google Maps Javascript API*. Pada bagian ini format *user interface* akan ditulis dalam format *Hypertext Markup Language (html)* hal ini dapat dilihat pada file *index.html*. Selain untuk tampilan bagian ini juga berfungsi untuk menerima input user dan mengubahnya menjadi *filter params*, hal ini dapat dilihat pada file *maps.js*. Bentuk implementasi dari bagian ini dapat dilihat:

```
1 function initMap() {
2     let map = new google.maps.Map(document.getElementById("map"), {
3         center: {lat: -6.914744, lng: 107.609810},
4         zoom: 13,
5
6     });
7     return map;
8 }
9
10 function setMarkers(data, isStart, isEnd) {
11     let locations = [];
12     if (isStart) {
13         data.forEach(item => {
14             let startMark = {
15                 name: "start",
16                 pos: item.startCor,
17                 icon: "http://maps.google.com/mapfiles/ms/icons/green-dot.png"
18             }
19             locations.push(startMark)
20         })
21     }
22 }
```

```
21     }
22
23     if (isEnd) {
24         data.forEach(item => {
25             let endMark = {name: "end", pos: item.endCor, icon: "http://maps.
26                 google.com/mapfiles/ms/icons/red-dot.png"}
27             locations.push(endMark)
28         })
29     }
30
31     // console.log("filtered markers size" , data.length)
32     let markers = locations.map((item) => {
33         return new google.maps.Marker({
34             position: {lat: parseFloat(item.pos.lat), lng: parseFloat(item.pos.lng
35                 )},
36             icon: item.icon
37         });
38     );
39     return markers;
40 }
41
42 function setHeatMap(arrData, map, isStart, isEnd) {
43     let heatmapData = [];
44     if (isStart) {
45         arrData.forEach(item => {
46
47             let startCoor = {location: new google.maps.LatLng(item.startCor.lat,
48                 item.startCor.lng)}
49             heatmapData.push(startCoor)
50         })
51     }
52     if (isEnd) {
53         arrData.forEach(item => {
54             // {location: new google.maps.LatLng(37.782, -122.447), weight: 0.5},
55             let endLocationObj = {location: new google.maps.LatLng(item.endCor.lat
56                 , item.endCor.lng)}
57             heatmapData.push(endLocationObj)
58         })
59     }
60
61     let heatmap = new google.maps.visualization.HeatmapLayer({
62         data: heatmapData,
63         radius: 50
64     );
65     heatmap.setMap(map);
66
67     return heatmap;
68 }
69
70
71 function docReady(fn) {
72     // see if DOM is already available
73     if (document.readyState === "complete" || document.readyState === "interactive
74         ") {
75         // call on next available tick
76         setTimeout(fn, 1);
77     } else {
78         document.addEventListener("DOMContentLoaded", fn);
```

```
79     }
80 }
81
82
83 createFilterObj = () => {
84     let days = Array.from(document.getElementById("day").selectedOptions).map(
85         option => parseInt(option.value))
86     let hours = Array.from(document.getElementById("hour").selectedOptions).map(
87         option => parseInt(option.value));
88     let filter = {day: days, hour: hours}
89     return filter
90 }
91
92 deleteMarkes = (markers) => {
93     markers.forEach(item => {
94         item.setMap(null)
95     })
96     return null
97 }
98
99 docReady(function () {
100     let map = initMap()
101     let markerCluster = null;
102     let markers;
103     let heatMap;
104     document.getElementById("send-btn").onclick = function (e) {
105         e.preventDefault();
106         let filterParams = createFilterObj()
107         let isMarkerCluster = document.getElementById("marker-cluster").checked
108         let isHeatMap = document.getElementById("heat-map").checked
109         let statusSelected = Array.from(document.getElementById("start-finish").selectedOptions).map(option => option.value)
110         let statusStartChecked = statusSelected.includes("start");
111         let statusEndChecked = statusSelected.includes("end")
112         sendRequest("http://localhost:3000/searchRoute", filterParams).then(res =>
113             {
114                 if (isMarkerCluster) {
115                     map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
116                     let data = res.data.data;
117                     if (!Array.isArray(markers)) {
118                         markers = setMarkers(data, statusStartChecked,
119                             statusEndChecked)
120                         markerCluster = new MarkerClusterer(map, markers, {
121                             imagePath:
122                                 "https://developers.google.com/maps/documentation/
123                                     javascript/examples/markerclusterer/m",
124                         });
125                     }
126                     if (isHeatMap) {
127                         map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
128                         if (!heatMap || heatMap.getMap() === null) {
129                             heatMap = setHeatMap(res.data.data, map, statusStartChecked,
130                                 statusEndChecked)
131                         }
132                     });
133
134         document.getElementById('clear-btn').onclick = function (e) {
```

```

135     e.preventDefault();
136     if (markerCluster) {
137         markerCluster.removeMarkers(markers);
138         markers = deleteMarkes(markers)
139     }
140     if (heatMap) {
141         heatMap.setMap(null)
142     }
143 }
144 });

```

Listing 5.15: Maps.js

Bagian ini memiliki tiga tugas utama yaitu:

- Menerima input user dan mengubah input user menjadi *filter params*.
- Mengirim dan menerima data dari *router*.
- Membuat google maps input dan berkomunikasi dengan *google maps javascript api*.

### Menerima input user dan mengubah input user menjadi *filter params*

Implementasi bagian ini dapat dilihat menjadi:

```

1 createFilterObj = () => {
2     let days = Array.from(document.getElementById("day").selectedOptions).map(
3         option => parseInt(option.value))
4     let hours = Array.from(document.getElementById("hour").selectedOptions).map(
5         option => parseInt(option.value));
6     let filter = {day: days, hour: hours}
7     return filter
}

```

Listing 5.16: Input User

Ketika fungsi ini dijalankan maka akan mengambil value dari element dengan id *day* dan *hour*. Kemudian data tersebut akan dibuat ke dalam bentuk *map* yang disebut sebagai *filter parameter*. Pada bagian ini juga terdapat fungsi *docReady* yang bertugas untuk berkomunikasi dengan *node.js* atau *google maps javascript api*. Hasil implementasi fungsi tersebut adalah sebagai berikut:

```

1 function docReady(fn) {
2     if (document.readyState === "complete" || document.readyState === "interactive")
3         setTimeout(fn, 1);
4     else {
5         document.addEventListener("DOMContentLoaded", fn);
6     }
7 }

```

Listing 5.17: docReady Method

Pada fungsi ini memiliki parameter sebuah fungsi dimana fungsi ini akan dijalankan setiap *tick*. Hal ini dilakukan agar koneksi antar *backend* bisa tetap terus dilakukan. Berikut ini implementasi fungsi *docReady* pada perangkat lunak ini:

```

1 docReady(function () {
2     let map = initMap()
3     let markerCluster = null;
4     let markers;
5     let heatMap;
6     document.getElementById("send-btn").onclick = function (e) {
7         e.preventDefault();
8         let filterParams = createFilterObj()
9         let isMarkerCluster = document.getElementById("marker-cluster").checked
10        let isHeatMap = document.getElementById("heat-map").checked
11        let statusSelected = Array.from(document.getElementById("start-finish").
12            selectedOptions).map(option => option.value)
13        let statusStartChecked = statusSelected.includes("start");
14        let statusEndChecked = statusSelected.includes("end")
15        sendRequest("http://localhost:3000/searchRoute", filterParams).then(res =>
16            {
17                if (isMarkerCluster) {
18                    map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
19                    let data = res.data.data;
20                    if (!Array.isArray(markers)) {
21                        markers = setMarkers(data, statusStartChecked,
22                            statusEndChecked)
23                    }
24                    markerCluster = new MarkerClusterer(map, markers, {
25                        imagePath:
26                            "https://developers.google.com/maps/documentation/
27                                javascript/examples/markerclusterer/m",
28                        });
29                }
30            });
31        }
32    );
33    });
34 }
35
36 document.getElementById('clear-btn').onclick = function (e) {
37     e.preventDefault();
38     if (markerCluster) {
39         markerCluster.removeMarkers(markers);
40         markers = deleteMarkes(markers)
41     }
42     if (heatMap) {
43         heatMap.setMap(null)
44     }
45 }
46 });

```

Listing 5.18: docReady Method

## Mengirim dan menerima data dari router

Implementasi untuk bagian ini dapat dilihat pada method docReady:

```

1  document.getElementById("send-btn").onclick = function (e) {
2      e.preventDefault();
3      let filterParams = createFilterObj()
4      let isMarkerCluster = document.getElementById("marker-cluster").checked
5      let isHeatMap = document.getElementById("heat-map").checked
6      let statusSelected = Array.from(document.getElementById("start-finish").
7          selectedOptions).map(option => option.value)
8      let statusStartChecked = statusSelected.includes("start");
9      let statusEndChecked = statusSelected.includes("end")
10     sendRequest("http://localhost:3000/searchRoute", filterParams).then(res =>
11         {
12             if (isMarkerCluster) {
13                 map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
14                 let data = res.data.data;
15                 if (!Array.isArray(markers)) {
16                     markers = setMarkers(data, statusStartChecked,
17                         statusEndChecked)
18                     markerCluster = new MarkerClusterer(map, markers, {
19                         imagePath:
20                             "https://developers.google.com/maps/documentation/
21                             javascript/examples/markerclusterer/m",
22                         });
23                 }
24             if (isHeatMap) {
25                 map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
26                 if (!heatMap || heatMap.getMap() === null) {
27                     heatMap = setHeatMap(res.data.data, map, statusStartChecked,
28                         statusEndChecked)
29                 }
30             }
31         });
32     }
33 }

```

Listing 5.19: communication method

Pada potongan kode diatas dapat dilihat jika tombol send-btn ditekan maka akan memanggil method *sendRequest* method ini akan bertujuan untuk berkomunikasi dengan *backend* untuk mendapatkan data yang histori yang telah disaring berdasarkan input dari user. Fungsi ini juga bertujuan untuk mendapatkan hasil visualisasi dari data yang telah disaring dengan menggunakan *google maps javascript api*.

### 5.1.6 Berkomunikasi dengan *google maps javascript api*

Perangkat lunak ini menggunakan *google maps javascript api* sebagai salah satu *third party library* untuk melakukan visualisasi data. Terdapat dua buah opsi yang disediakan oleh perangkat lunak ini dalam melakukan visualisasi data yaitu *Heat Map* dan *Marker Clustering*. Untuk dapat menggunakan opsi tersebut *web client* perlu membuat input parameter yang sesuai dengan kebutuhan *google maps javascript api 2.5*. Implementasi dari bagian ini dapat dilihat sebagai berikut:

```

1 function setMarkers(data, isStart, isEnd) {
2     let locations = [];
3     if (isStart) {
4         data.forEach(item => {
5             let startMark = {
6                 name: "start",
7                 pos: item.startCor,
8                 icon: "http://maps.google.com/mapfiles/ms/icons/green-dot.png"
9

```

```

9             }
10            locations.push(startMark)
11        })
12    }
13
14    if (isEnd) {
15      data.forEach(item => {
16        let endMark = {name: "end", pos: item.endCor, icon: "http://maps.
17          google.com/mapfiles/ms/icons/red-dot.png"}
18        locations.push(endMark)
19      })
20    }
21
22    // console.log("filtered markers size" , data.length)
23    let markers = locations.map((item) => {
24      return new google.maps.Marker({
25        position: {lat: parseFloat(item.pos.lat), lng: parseFloat(item.pos.lng
26          )},
27        icon: item.icon
28      });
29    );
30
31    return markers;
32  }
33
34  function setHeatMap(arrData, map, isStart, isEnd) {
35    let heatmapData = [];
36    if (isStart) {
37      arrData.forEach(item => {
38
39        let startCoor = {location: new google.maps.LatLng(item.startCor.lat,
40          item.startCor.lng)}
41        heatmapData.push(startCoor)
42
43      })
44    if (isEnd) {
45      arrData.forEach(item => {
46        // {location: new google.maps.LatLng(37.782, -122.447), weight: 0.5},
47        let endLocationObj = {location: new google.maps.LatLng(item.endCor.lat
48          , item.endCor.lng)}
49        heatmapData.push(endLocationObj)
50      })
51
52    let heatmap = new google.maps.visualization.HeatmapLayer({
53      data: heatmapData,
54      radius: 50
55    });
56    heatmap.setMap(map);
57
58    return heatmap;
59  }

```

Listing 5.20: Map Input

## **DAFTAR REFERENSI**

- [1] Nugroho, P. dan Natali, V. (2017) Open sourcing proprietary application case study: Kiri website. *International Journal of New Media Technology*, **4**, 82–86.
- [2] Shafranovich, Y. (2005) Common format and mime type for comma-separated values (csv) files. Technical Report 7111.
- [3] Dahl, R. nodejs. <https://nodejs.org/en/about/>.
- [4] Dahl, R. npmjs. <https://docs.npmjs.com/about-npm>.
- [5] Holowaychuk, T. expressjs. <https://expressjs.com/en/starter/installing.html>.
- [6] Google Google maps refrence. <https://developers.google.com/maps/documentation/javascript/reference>.



## LAMPIRAN A

### KODE PROGRAM

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4
5 #include<stdio.h>
6
7 void myFunction( int input, float* output ) {
8     switch ( array[i] ) {
9         case 1: // This is silly code
10            if ( a >= 0 || b <= 3 && c != x )
11                *output += 0.005 + 20050;
12            char = 'g';
13            b = 2^n + ~right_size - leftSize * MAX_SIZE;
14            c = ( -aaa + &daa ) / (bbb++ - ccc % 2 );
15            strcpy(a,"hello $@?");
16        }
17        count = ~mask | 0x00FF00AA;
18    }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.1: MyCode.c

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                         //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }

```

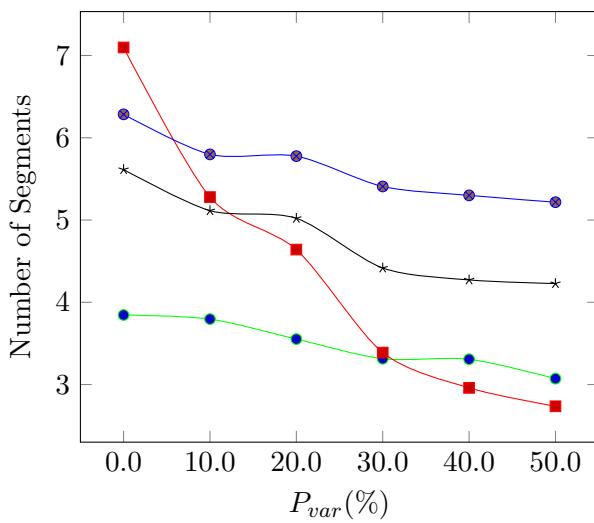
36 | P

Listing A.2: MyCode.java

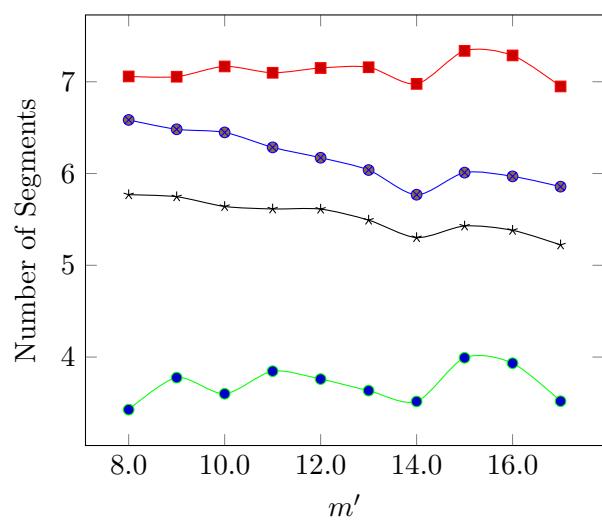
## LAMPIRAN B

### HASIL EKSPERIMENT

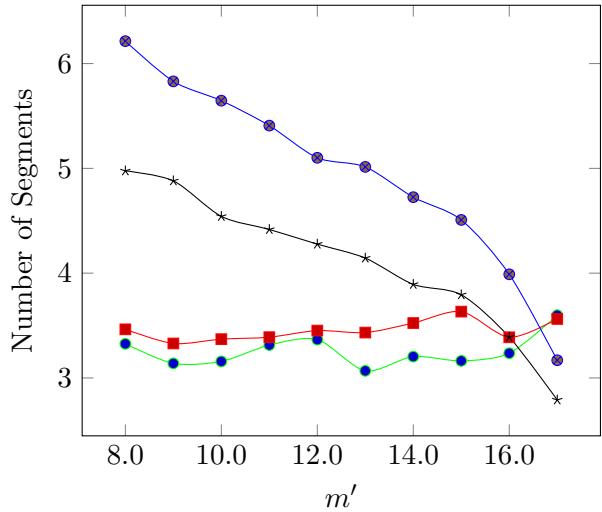
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



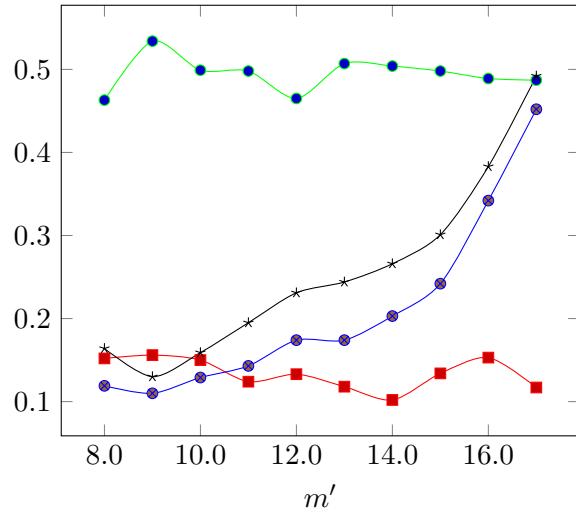
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4