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

DAFTAR GAMBAR

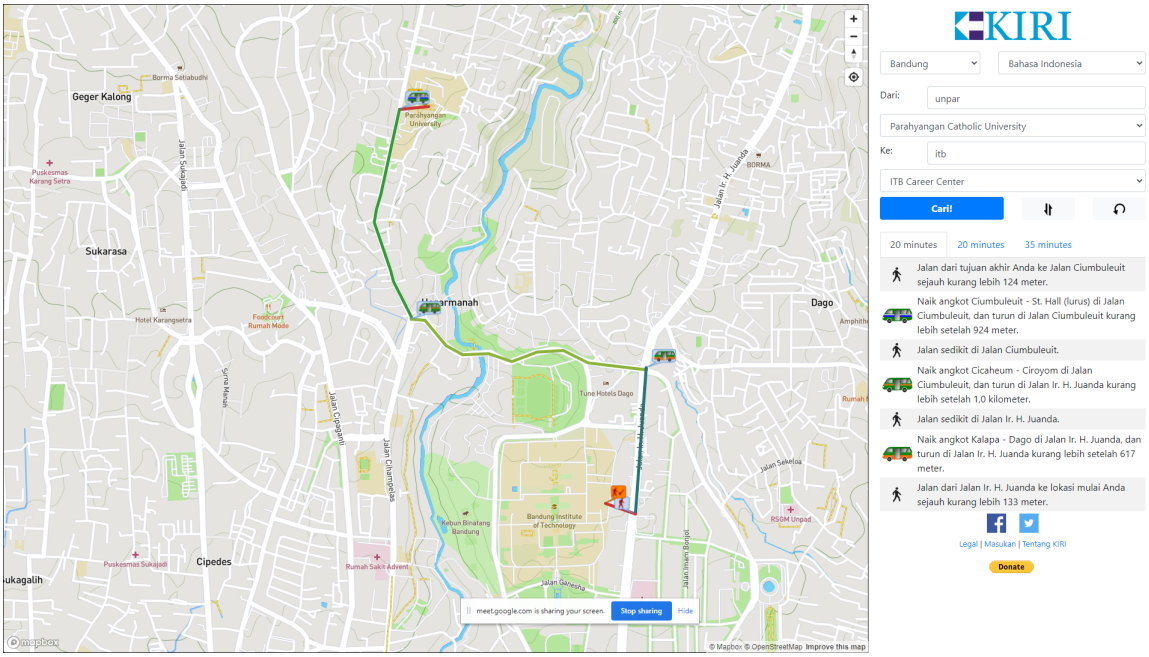
DAFTAR TABEL

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[**?**]. 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.



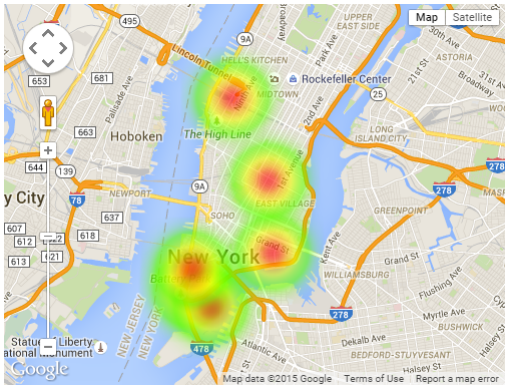
Gambar 1.1: Tampilan Utama Website KIRI

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 mengelompokan *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 pemograman *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.

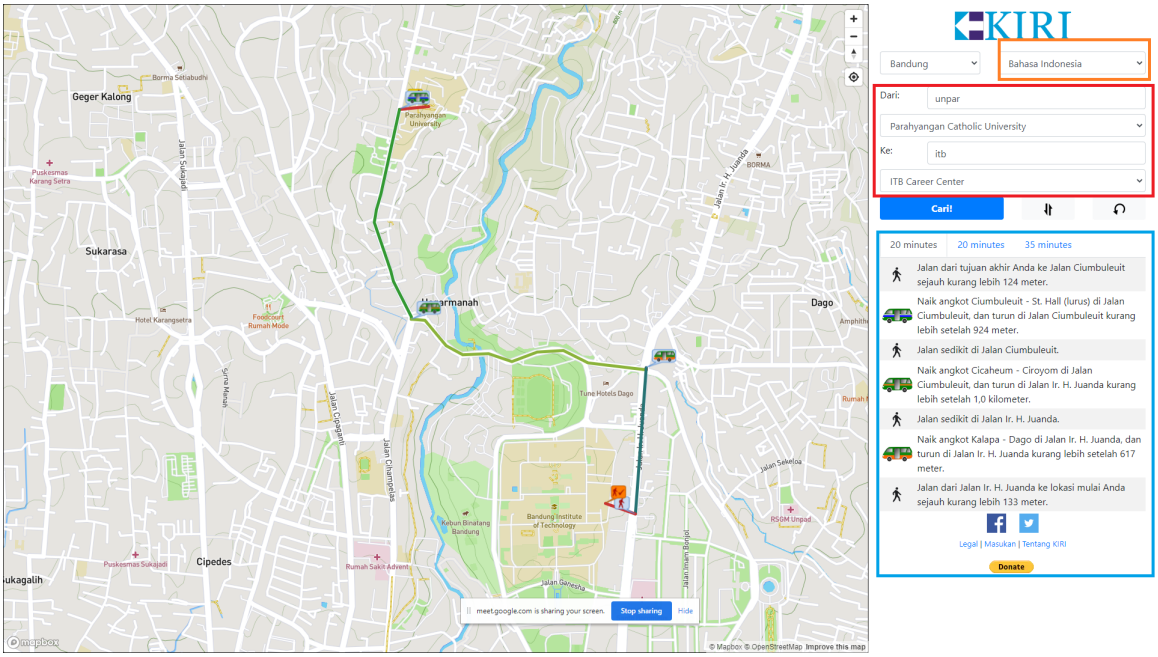
BAB 2

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.[**?**]. Pada awal pembuatannya KIRI dibuat untuk tujuan komersial. Namun karena dinilai kurang sukses projek KIRI sekarang menjadi open source projek yang dapat di akses. 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 *orange*.

3. Dapat menampilkan instruksi lengkap mencapai tujuan. Dapat dilihat pada gambar 2.1 pada kotak berwarna biru.

2.2 JSON

JSON (*Javascript Object Notation*) adalah format teks untuk melakukan serialisasi data terstruktur[**?**]. JSON berasal objek literal *javascript*, Seperti yang didefinisikan oleh oleh bahasa pemrograman *ECMAScript*, JSON dapat mewakili empat tipe data primitif (*strings*, *numbers*, *booleans*, *null*) dan dua tipe data terstruktur (*objects*, *arrays*).

2.2.1 JSON Grammar

JSON teks terdiri dari sekumpulan Token. Set Token mencakup enam *structural characters* , *strings*, *numbers* dan tiga nama literal. JSON teks merupakan *serialized value*. JSON membatasi JSON teks menjadi *object* atau *array*. Implementasi JSON hanya boleh menghasilkan *object* atau *array*. JSON memiliki enam *structural characters* yaitu :

- `begin-array = ws %x5B ws ; [ left square bracket.`
- `begin-object = ws %x7B ws ; { left curly bracket.`
- `end-array = ws %x5D ws ; ] right square bracket.`
- `end-object = ws %x7D ws ; } right curly bracket.`
- `name-separator = ws %x3A ws ; : colon.`
- `value-separator = ws %x2C ws ; , comma.`

Penggunaan *whitepace* sebelum atau setelah enam *structural characters* diperbolehkan.

```
ws = *(
            %x20 /              ; Space
            %x09 /              ; Horizontal tab
            %x0A /              ; Line feed or New line
            %x0D )              ; Carriage return
```

2.2.2 Values

JSON value harus berupa *object*, *array*, *number*, atau *string*, atau salah satu dari tiga *literal names*:

- `true`
- `false`
- `null`

Setiap *literal names* harus tersusun dari huruf kecil. Selain *literal names* yang disebutkan diatas tidak ada *literals names* lain yang izinkan.

```
value = false / null / true / object / array / number / string

false = %x66.61.6c.73.65    ; false

null  = %x6e.75.6c.6c       ; null

true  = %x74.72.75.65       ; true
```

2.2.3 Objects

Object direpresentasikan dengan sepasang kurung kurawal () kosong atau berisi name/*value pair* atau (*members).* Nama pada *objects* berupa *string.* Setelah nama akan dikuti oleh simbol *colon (:)* yang berfungsi untuk memisahkan nama dengan *value.* Setelah value akan diikuti dengan simbol koma (,). Nama pada *Object* harus bersifat unik.

```
object = begin-object [ member *( value-separator member ) ]
         end-object

member = string name-separator value
```

Object dengan susunan nama yang unik akan dapat beroperasi dan diimplementasikan oleh semua software. Jika susunan nama pada *object* tidak bersifat unik maka respon perangkat lunak yang menerima *object* tersebut tidak dapat diprediksi.

2.2.4 Arrays

Arrays direpresentasikan dengan sepasang kurung siku ([]) kosong atau berisi satu atau lebih element. Elements pada array akan dipisahkan dengan simbol koma (,).

```
array = begin-array [ value *( value-separator value ) ] end-array
```

Tidak ada persyaratan bahwa *value* pada array harus memiliki tipe data yang sama.

2.2.5 Numbers

Numbers direpresentasikan menggunakan basis 10 *decimal*, *Number* bisa memiliki *prefix* seperti simbol minus (-), *Number* juga biasanya dilengkapi dengan pecahan dan eksponen. Dalam penulisan *number leading zero* tidak diperbolehkan. *Pecahan* adalah *decimal point* yang diikuti oleh satu atau lebih digit. *Eksponen* biasanya diawali dengan *character* E yang bisa diikuti dengan simbol opsional *plus* atau *minus.* Huruf E dan simbol opsional akan diikuti oleh satu atau lebih digit. Nilai *Numeric* yang tidak diperbolehkan dalam aturan penulisan dibawah seperti (*NaN* dan *Infinity*).

```
  number = [ minus ] int [ frac ] [ exp ]

decimal-point = %x2E          ; .

digit1-9 = %x31-39            ; 1-9

e = %x65 / %x45               ; e E

exp = e [ minus / plus ] 1*DIGIT

frac = decimal-point 1*DIGIT

int = zero / ( digit1-9 *DIGIT )

minus = %x2D                  ; -

plus = %x2B                   ; +

zero = %x30                   ; 0
```

Spesifikasi ini memungkinkan implementasi untuk menetapkan batasan pada rentan dan ketepatan angka yang diterima. Sejak banyak *software* mengimplementasikan *IEEE 754-2008 binary64*

(double precision) numbers, Untuk mendapatkan performa yang baik maka *implementasi* tidak boleh melebihi *precision* atau *range* yang disediakan.

2.2.6 Strings

String direpresentasikan dengan simbol *quotation marks* semua *unicode* akan diletakan diantara simbol *quotation marks*, Kecuali beberapa *character* yang khusus yang harus melalui proses *escape*:

- Quotation mark.
- Reverse solidus.
- Control characters *(U+0000,U+001F).* .

Untuk melakukan *escape* pada *extended character* atau pada *character* yang tidak dalam bentuk penulisan standart, *character* dapat direpresentasikan kedalam *12-character sequence*.

```
string = quotation-mark *char quotation-mark

char = unescaped /
    escape (
        %x22 /          ; "    quotation mark  U+0022
        %x5C /          ; \    reverse solidus U+005C
        %x2F /          ; /    solidus         U+002F
        %x62 /          ; b    backspace       U+0008
        %x66 /          ; f    form feed       U+000C
        %x6E /          ; n    line feed       U+000A
        %x72 /          ; r    carriage return U+000D
        %x74 /          ; t    tab             U+0009
        %x75 4HEXDIG )  ; uXXXX                U+XXXX

escape = %x5C               ; \

quotation-mark = %x22       ; "

unescaped = %x20-21 / %x23-5B / %x5D-10FFFF
```

2.2.7 JSON Example

Berikut ini adalah contoh penulisan *JSON*

- Contoh JSON Object

```
{
"Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "View from 15th Floor",
    "Thumbnail": {
        "Url":    "http://www.example.com/image/481989943",
        "Height": 125,
        "Width":  100
    },
    "Animated" : false,
    "IDs": [116, 943, 234, 38793]
```

```
        }
    }
```

Image member adalah sebuah *object* yang memiliki *Thumbnail member* yang merupakan sebuah *object* dan *IDs Member* yang merupakan sebuah *array*.

- Contoh JSON *Array* dengan dua buah *elements* yang berupa *object*

```
[
  {
    "precision": "zip",
    "Latitude":  37.7668,
    "Longitude": -122.3959,
    "Address":   "",
    "City":      "SAN FRANCISCO",
    "State":     "CA",
    "Zip":       "94107",
    "Country":   "US"
  },
  {
    "precision": "zip",
    "Latitude":  37.371991,
    "Longitude": -122.026020,
    "Address":   "",
    "City":      "SUNNYVALE",
    "State":     "CA",
    "Zip":       "94085",
    "Country":   "US"
  }
]
```

- Contoh JSON teks yang hanya memiliki *values*

```
"hello world"
32
false
```

2.3 CSV

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

2.3.1 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 (CLRF)* contoh:

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

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

```
aaa,bbb,ccc CRLF
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:

```
field_name,field_name,field_name CRLF
aaa,bbb,ccc CRLF
zzz,yyy,xxx CRLF
```

- Memungkinkan ada satu atau lebih, *field* yang dipisahkab 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 terkahir sebuah baris tidak boleh diberikan simbol koma (,) contoh:

```
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*

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

- Jika simbol *double quotes* meruapakan salah satu elemn pada *field*, Maka simbol *double quotes* tersebut harus diescape dengan memberikan simbol *double quotes* lain didalam *field* tersebut contoh:

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

2.4 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*[**?**]. *Node.js* didesain untuk membuat *network applications* yang *scaleable*. Berikut ini contoh syntax *Node.js*:

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2.4.1 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 diinstal melalui npm. Semua *library* yang diinstall melalui npm akan dicatat pada file *package.json*, Hal ini bertujuan untuk mempermudah proses *maintence library* pada projek *node.js*.

Struktur dari *file-file* akan berbentuk seperti:

```
root
├── node_modules
│   ├── "file1"
│   └── "file2"
└── package.json
```

2.4.2 *Node Package Manager*

Node Package Manager(NPM) adalah *software registry* yang dimiliki oleh *Node.js*, NPM memungkinkan pengguna untuk mempublikasi atau menggunakan *software library*[**?**]. 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*.

2.4.3 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:

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

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

```
npm install (with no args, in package dir)
npm install [<@scope>/]<name>
npm install [<@scope>/]<name>@<tag>
npm install [<@scope>/]<name>@<version>
```

```
npm install [<@scope>/]<name>@<version range>
npm install <alias>@npm:<name>
npm install <git-host>:<git-user>/<repo-name>
npm install <git repo url>
npm install <tarball file>
npm install <tarball url>
npm install <folder>
```

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

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

2.5 *Expressjs*

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

2.5.1 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*.

2.5.2 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[**?**]. 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:

```
app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── index.pug
    ├── error.pug
    └── layout.pug
```

2.5.3 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*

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.post('/', function (req, res) {
  res.send('Got a POST request')
})

app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user')
})

app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user')
})
```

2.5.4 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:

```
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:

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

yang dapat digunakan untuk mengakses *file* statis.

2.6 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[?]*:

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

2.6.1 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:

```
Map(mapDiv[, opts])
Parameters:
mapDiv: Element
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*:

```
function initMap() {
    let mapOptions = {
        center: {lat: -6.914744, lng: 107.609810},
        zoom: 12,
}
let map = new google.maps.Map(document.getElementById("map"), mapOptions);
return map;
}
```

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:

```
fitBounds(bounds[, padding])
Parameters:
```

```
bounds:  LatLngBounds|LatLngBoundsLiteral
padding:  number|Padding optional
Return Value:  None
```

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

```
getBounds()
Parameters:  None
Return Value:  LatLngBounds
```

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

```
getCenter()
Parameters:  None
Return Value:  LatLng
```

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

```
getClickableIcons()
Parameters:  None
Return Value:  boolean
```

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

```
getMapTypeId()
Parameters:  None
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:

```
setMapTypeId(mapTypeId)
Parameters:
mapTypeId:  MapTypeId|string
```

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

```
setZoom(zoom)
Parameters:
zoom:  number
```

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

```
setOptions(options)
Parameters:
options:  MapOptions
```

2.6.2 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 longtitude

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

```
LatLng(lat, lng[, noWrap])
Parameters:
lat:  number
lng:  number
noWrap:  boolean optional
```

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

```
map.setCenter(new google.maps.LatLng(-34, 151));
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*.

```
equals(other)
Parameters:
other:  LatLng
Return Value:  boolean
```

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

```
lat()
Parameters:  None
Return Value:  number
Returns the latitude in degrees.
```

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

```
lng()
Parameters:  None
Return Value:  number
Returns the longitude in degrees.
```

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

2.7 Marker

Gambar 2.2: Add Marker

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

```
Marker([opts])
Parameters:
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

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

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

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

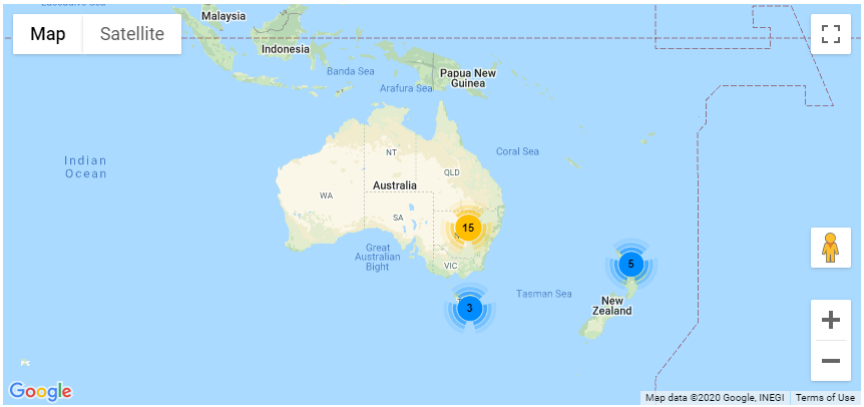
Kelas *marker* memiliki fungsi bawaan yang telah disediakan oleh antara lain seperti:

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 *clickable*
getCursor	Fungsi untuk mendapatkan nilai *cursor*
getDraggable	Fungsi untuk mendapatkan nilai *draggable*
getIcon	Fungsi untuk mendapatkan nilai *icon*
getMap	Fungsi untuk mendapatkan objek *map*
getOpacity	Fungsi untuk mendapatkan objek *opacity*
getPosition	Fungsi untuk mendapatkan objek *position*
getShape	Fungsi untuk mendapatkan *shape* dari objek *marker*
getTitle	Fungsi untuk mendapatkan *title* dari objek *marker*

2.8 Marker Clusterer

MarkerClustererPlus merupakan sebuah *library* tambahan untuk dapat mengelompokan objek marker 2.7. Visualisasi dari *MarkerClustererPlus* dapat dilihat pada gambar 2.3



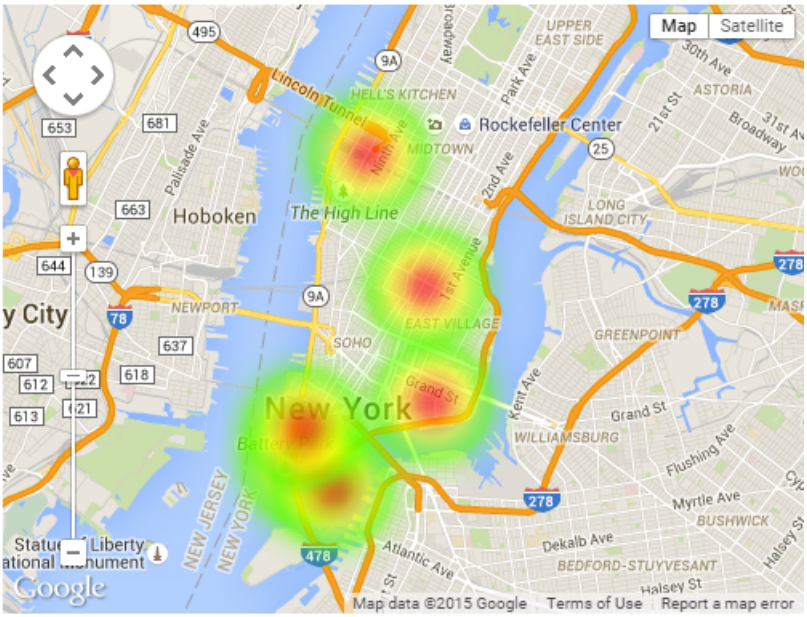
Gambar 2.3: Contoh Marker Clustering

Untuk dapat menginisialisasi kelas tersebut pihak pengembang perlu menuliskan perintah:

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

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

2.9 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:

```
HeatmapLayer([opts])
Parameters:
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.
- gardient gardient dari warna *heatmap*
- map atribute untuk menunjukan peta dimana *heatmap* akan ditampilkan.
- maxIntencity nilai maximal dari intensitas warna pada *heatmap*.
- opcity nilai opacity dari *heatmap*.
- radius nilai radius dari *heatmap*.

Contoh penginisialisasian kelas *heatmap*:

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

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 yang 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:

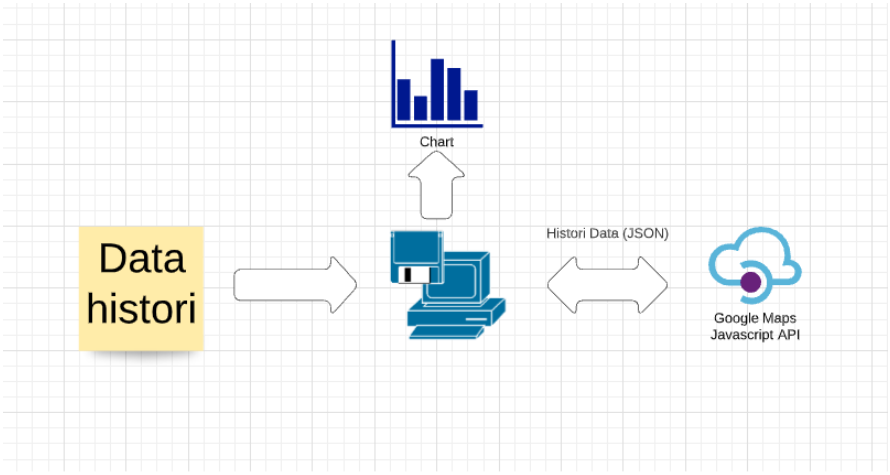
- LogId Id unik sebagai penanda satu record didalam data histori KIRI.
- APIKey atribut api key yang digunakan ketika melakukan perintah pada perangkat lunak KIRI.
- Timestamp (UTC) atribut untuk mencatat waktu perintah dilakukan format berbentuk *timestamp*.
- Action jenis action yang dilakukan user 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 bernlai keyword yang dituliskan oleh pengakses.
  - jika action bernilai *FINDROUTE* maka additionaldata akan bernilai posisi tempat dan tujuan dalam bentuk langtitude dan longtitude 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 melukan 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

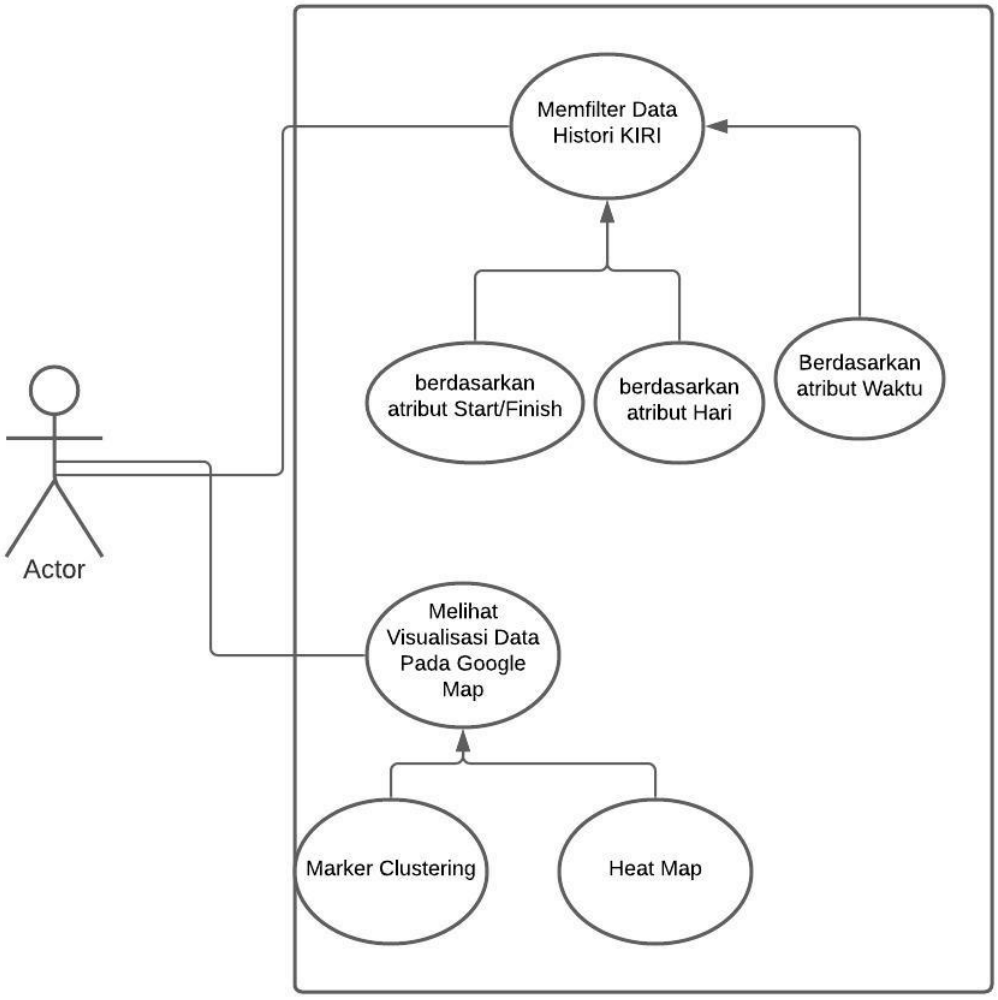
3.3.1 Analisis Kebutuhan Perangkat Lunak

Berdasarkan latar belakang, rumusan masalah, dan tujuan maka dapat didefinisikan kebutuhan - kebutuhan perangkat lunak visualisasi data histori KIRI sebagi 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* dan skenario berikut ini



Gambar 3.2: Alur Perangkat Lunak

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

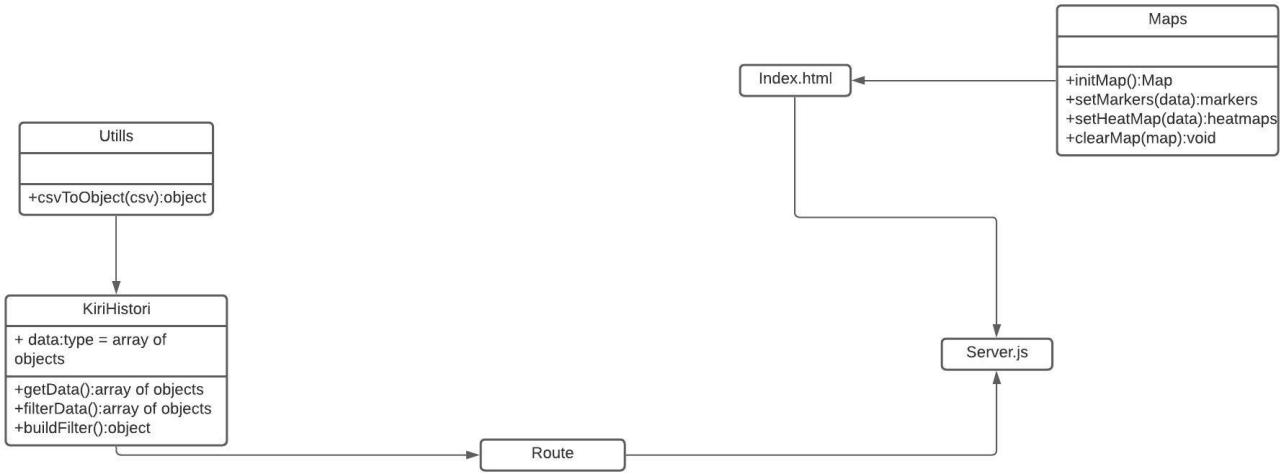
Nama	Memfilter Data .
Deskripsi	Melakukan filter data berdasarkan kategori dari data histori kiri
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan dan sudah dapat mengolah raw data histori KIRI.
Alur Skenario Utama	1. Sistem memuat aplikasi. 2. Sistem menampilkan data. 3. Pengguna dapat memfilter data berdasarkan waktu,hari, dan *start / finish*. 4. Sistem melakukan proses filtering berdasarkan atribut yang telah dipilih pengguna.

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	1. Sistem menampilkan data yang telah difilter oleh pengguna. 2. Pengguna dapat memilih akan menggunakan metode visulasisai *heat map* atau *marker clustering*. 3. Sistem menampilkan hasil visualisasi data berdasarkan metode yang telah dipilih pengguna.

3.3.4 Diagram Kelas

Diagram kelas sederhana ini akan memberi gambaran besar mengenai aplikasi visualisasi data histori KIRI pada *Google Maps*.



Gambar 3.3: Diagram Kelas Perangkat Lunak

Berikut adalah penjelasan dari kelas - kelas pada diagram kelas pada gambar3.3.

- Kelas KiriHistori
  Kelas ini adalah kelas yang merepresentasikan model pada data histori KIRI. Kelas ini akan berguna untuk mengolah data histori KIRI.
- Kelas Utils
  Kelas ini adalah *helper class* dimana akan berisi *function - function* yang dibutuhkan untuk mengolah data histori KIRI.
- Kelas Route
  Kelas ini akan mengatur routing pada *node.js*.
- Kelas Maps
  Kelas ini akan menggunakan *google maps javascript api* untuk dapat memvisualisasikan data histori kiri.
- Index
  File ini akan betujuan untuk menampilkan objek *map* dan hasil visualisasi data kedalam bentuk *html*.
- Kelas Server
  Kelas ini yang akan menjadi kelas *server* utama pada perangkat lunak.

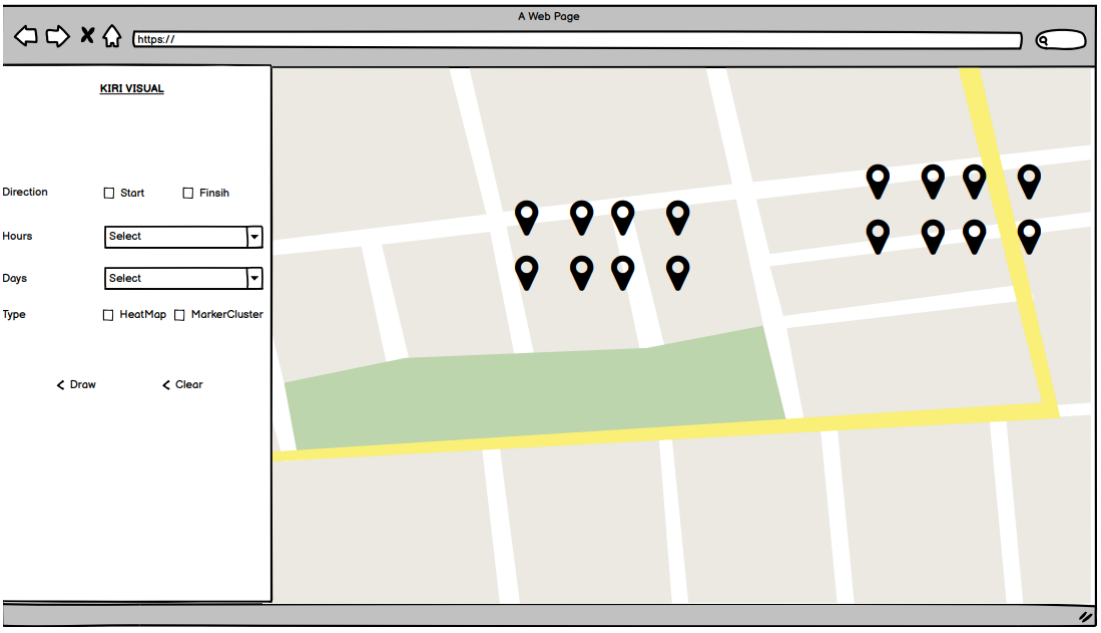
BAB 4

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 memilfter 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 *Draw*: menampilkan data yang telah diolah kedalam objek *map*.
- Button *Clear*: menghapus seluruh *overlay* pada objek *map*.

4.2 Perancangan Masukan dan Keluaran

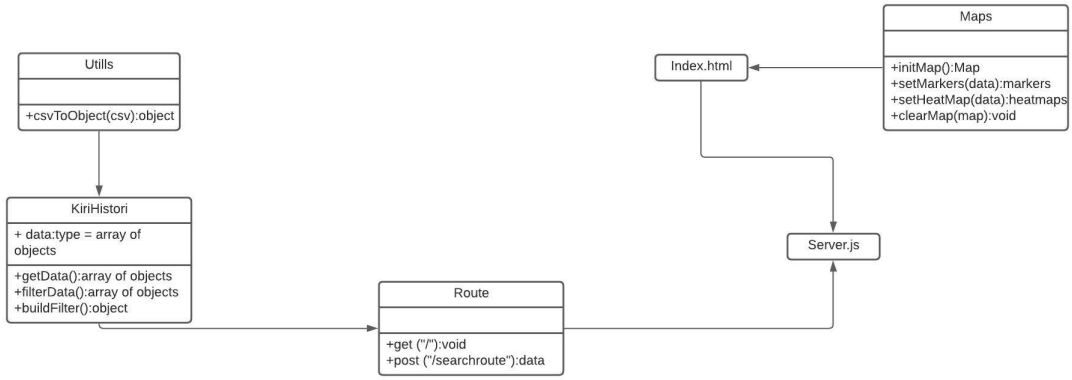
Aplikasi agregasi yang dibangun menggunakan antarmuka yang sudah dirancang seperti pada gambar 4.1. Dapat diketahui masukan pada aplikasi ini berupa *click* dari tombol dan *selection box* yang tersedia dimana ketika tombol-tombol tersebut ditekan maka akan memfilter data histori KIRI. Keluaran dari aplikasi ini berupa visualsisai data histori KIRI pada objek *map*.

4.3 Perancangan Kelas Aplikasi Visualisasi Data Histori KIRI

Pada subbab ini akan dijelaskan kelas diagram secara detail setelah sebelumnya dijelaskan diagram kelas sederhana pada subbab 3.3.4.

4.3.1 Diagram Kelas Perangkat Lunak

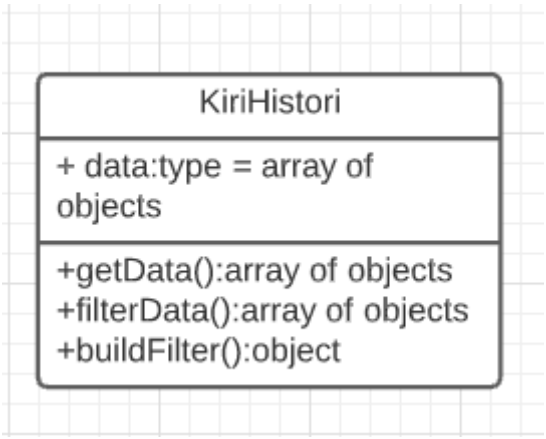
Pada penelitian ini dibuat satu diagram kelas lengkap, yaitu diagram kelas untuk perangkat lunak Visualisasi data histori KIRI.



Gambar 4.2: Rancangan Diagram Kelas

4.3.2 Detil Setiap Kelas

- **Kelas KIRIHistori**



Gambar 4.3: Kelas KIRI HISTORI

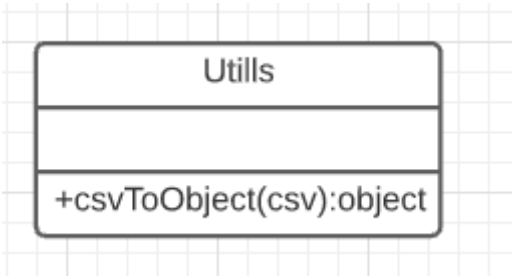
kelas ini berfungsi sebagai kelas model yang berfungsi untuk mengolah data histori KIRI. Berikut ini adalah atribut-atribut yang dimiliki kelas ini adalah:

- data atribut ini adalah atribut yang akan menampung data histori kiri

Kelas KIRIHistori juga memiliki *function-function* seperti:

- getData adalah fungsi yang akan menjadi getter untuk atribut data
- filterData adalah fungsi yang akan melakukan perintah filter
- buildFilter adalah fungsi yang akan membuat filter query

- **Kelas Utils**

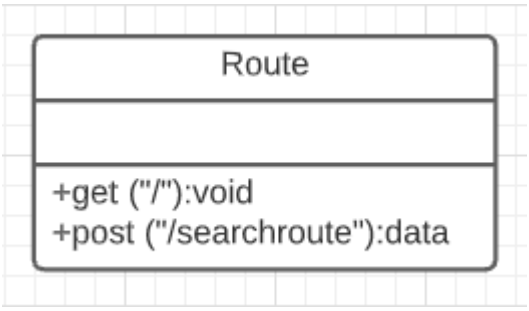


Gambar 4.4: Kelas utils

Kelas ini berfungsi sebagai kelas yang menyediakan *function - function* tambahan untuk kelas KIRIHistori. Kelas ini memiliki *function*:

- csvToObject *function* ini akan menerima parameter csv data dan akan mengolah nya menjadi javascript objek

- **Route**



Gambar 4.5: Kelas Route

Kelas ini berfungsi untuk mengatur *route path* pada aplikasi visualisasi data histori KIRI. Pada kelas ini akan terdapat dua route yaitu:

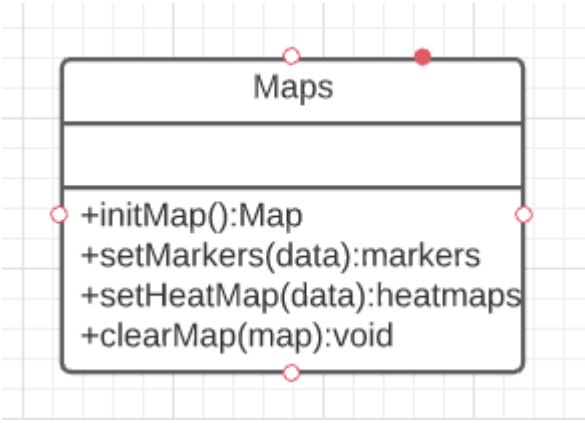
- route get ("/") route ini akan mengembalikan tampilan yang akan direnderpada halaman utama
- route post ("/searhroute") route ini akan mengembalikan data histori yang telah difilter

- **Server.js**

Gambar 4.6: Kelas Server

Kelas ini akan menjadi kelas utama yang akan melakukan *server side* rendering pada perangkat lunak ini.

- **Maps**



Gambar 4.7: Kelas Maps

Kelas ini akan menjadi kelas yang menerima input data histori dan menampilkannya menggunakan *google maps javascript api*. Pada kelas ini terdapat *function* - *function* yaitu:

- initMap *function* ini akan melakukan insiasi pada objek *map*

  - setHeatMap *function* ini akan mengolah data histori kiri menjadi objek *heat map* agar dapat divisualisasikan pada objek *map*
  - setMarkers *function* ini akan mengolah data histori kiri menjadi objek *marker* agar dapat divisualisasikan pada objek *map*

- **index**

Gambar 4.8: Index

Kelas ini akan menjadi kelas yang menampilkan tampilan utama pada perangkat lunak ini.

4.4 Perancangan Pseudocode Aplikasi Visualisasi data histori KIRI

Pada subbab ini dirancang *pseudocode* untuk membuat aplikasi visualisasi data histori KIRI. Pada perangkat lunak ini akan dijelaskan psudocode pada kelas-kelas KIRIHistori , Utils , Route , Server , Maps

4.4.1 Utils

Kelas ini merupakan kelas yang akan memberikan *function-function* bantuan untuk kelas KIRIHistori. Pada kelas ini terdapat *function* `csvToObject` yang akan berfungsi untuk mengolah data histori KIRI

Algorithm 1 csvToObject

1: $cols \leftarrow csv \rightarrow split('/')$
2: $action \leftarrow cols[3]$
3: **if** $action == FINDROUTE$ **then**
4: $\quad startLng \leftarrow cols[5] \rightarrow split(\text{"/"})[0]$
5: $\quad endLat \leftarrow cols[5] \rightarrow split(\text{"/"})[1]$
6: $\quad fullDate \leftarrow cols[2]$
7: $\quad$ **return** *object*
8: **end if**

4.4.2 KIRIHistori

Pada kelas ini terdapat *constructor* yang akan meload data menggunakan *javascript file system* pseudocode *constructor* ini akan berupa

Algorithm 2 Constructor KIRIHistori

1: $data \leftarrow array \rightarrow map(csvToObject(item))$

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

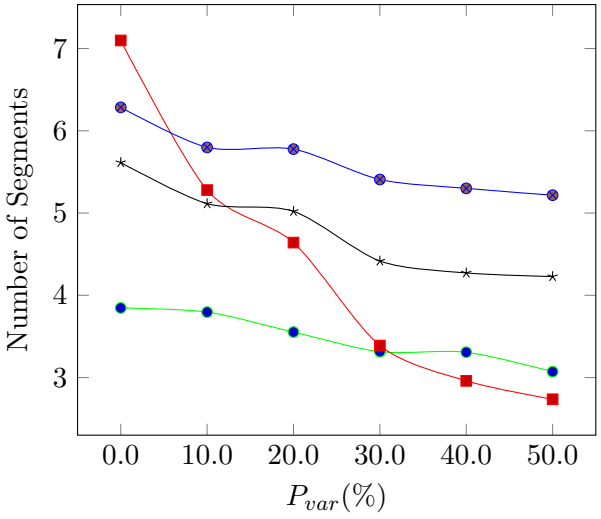
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

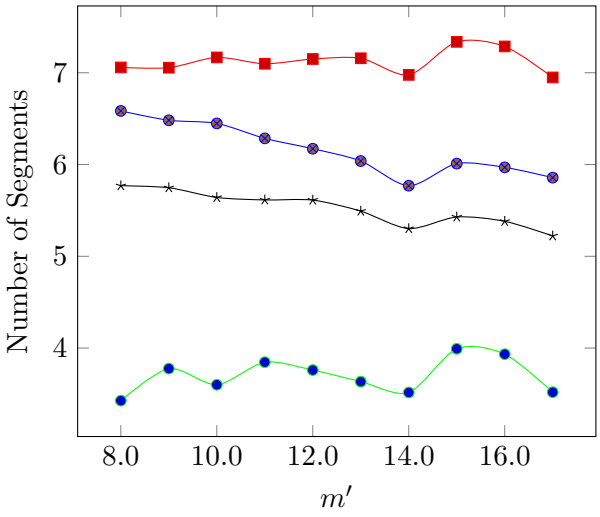
LAMPIRAN B

HASIL EKSPERIMEN

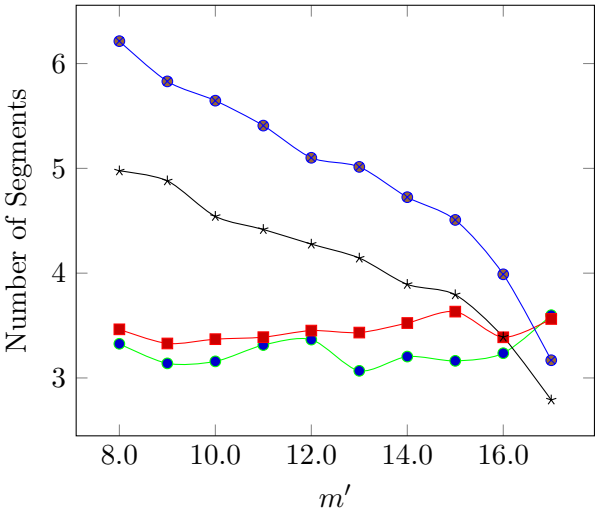
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



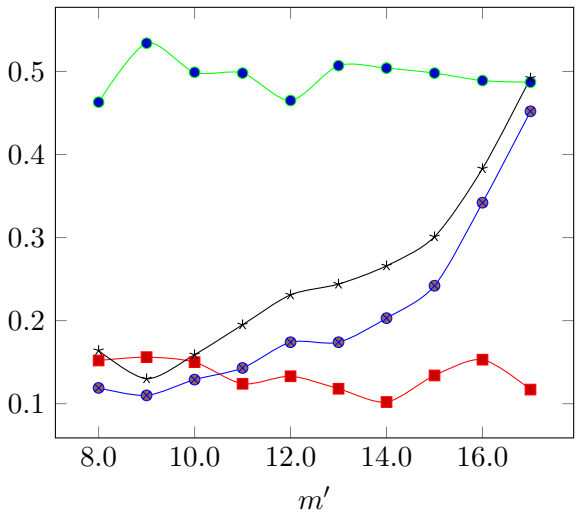
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4