

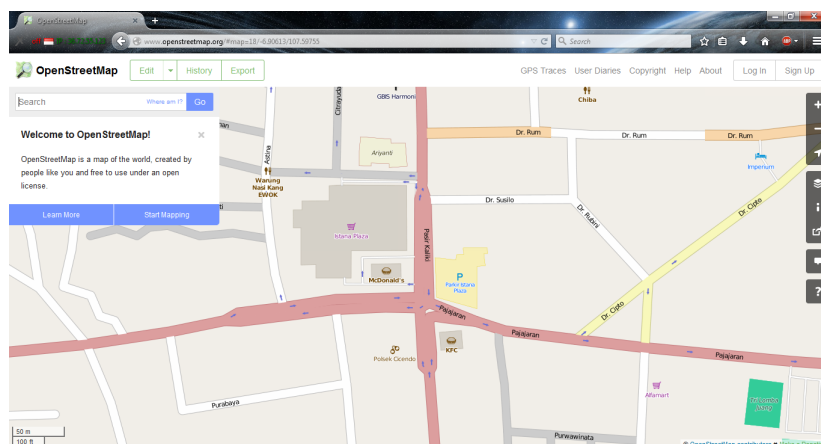
BAB 1

PENDAHULUAN

1.1 Latar Belakang

Mengemudi merupakan salah satu pilihan bagi masyarakat untuk bepergian dari suatu tempat ke tempat lain yang dituju. Contohnya adalah seorang wanita karir yang mengemudikan kendaraan pribadi dari rumah menuju kantor atau tempat kerjanya. Contoh lainnya adalah seorang sopir taksi yang mengemudikan kendaraannya untuk mengantar penumpang hingga sampai ke tujuan. Untuk dapat sampai ke titik tujuan, banyak rute yang dapat dilalui oleh seorang pengemudi. Seorang pengemudi, tentu saja akan mencari rute terdekat yang dapat dilalui, hal tersebut bertujuan untuk menghemat penggunaan bahan bakar dan juga waktu. Pemilihan rute terdekat untuk dapat sampai ke tujuan menjadi cukup penting, karena saat ini mobilitas masyarakat yang semakin tinggi. Aplikasi pencarian rute terdekat dapat membantu seorang pengemudi untuk menemukan rute terdekat untuk sampai ke tempat tujuan lebih cepat. Dengan cara menunjukkan rute menyetir terdekat dari satu tempat ke tempat lain.

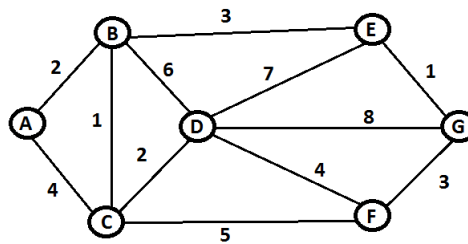
Aplikasi yang dibuat akan berbasis OpenStreetMap dan menggunakan algoritma Dijkstra. OpenStreetMap adalah portal peta terbuka yang menyediakan data dalam bentuk peta maupun XML, pengguna dapat mencari lokasi dan memilih area yang diinginkan. Setelah pengguna memilih area yang diinginkan, pengguna dapat menggunakan fitur export untuk mengunduh data XML pada area tersebut. Tampilan website OpenStreetMap dapat dilihat pada Gambar 1.1. Data yang disediakan oleh OpenStreetMap dalam bentuk XML biasa



Gambar 1.1: Tampilan website OpenStreetMap ¹

¹<http://www.openstreetmap.org>

1 disebut dengan OpenStreetMap XML dan disingkat menjadi OSMXML. OSMXML adalah
 2 dokumen XML yang berisi data-data peta OSM. Pada dasarnya, OSMXML berisi data pri-
 3 mitif (node, way, dan relation) yang merupakan arsitektur dari model OSM. Node dapat
 4 diartikan sebagai titik pada peta digital, way merupakan informasi garis pada peta yang
 5 melambangkan jalan atau elemen lain seperti rel kereta, dan relation memberikan informasi
 6 node-node yang bersinggungan, elemen relation dapat menggambarkan suatu area seperti
 7 lapangan, taman bermain, rute bus, dan lain-lain. Sedangkan algoritma Dijkstra adalah
 8 algoritma untuk mencari jarak terpendek pada sebuah graf berarah dengan bobot yang ber-
 9 nilai tidak negatif pada setiap sisinya [1]. Graf adalah himpunan objek yang terdiri dari
 10 simpul(node) dan sisi (edge), graf digambarkan sebagai kumpulan titik yang dihubungkan
 oleh garis. Contoh graf dapat dilihat pada Gambar 1.2.



Gambar 1.2: Contoh Graf

11
 12 Aplikasi yang dibangun akan mengolah data yang disediakan oleh OpenStreetMap dalam
 13 bentuk XML dan memodelkannya ke dalam bentuk graf. Selanjutnya akan diimplementa-
 14 sikan algoritma Dijkstra untuk mencari rute terdekat pada graf tersebut dan menunjukkan
 15 hasilnya secara visual.

16 1.2 Rumusan Masalah

17 Berdasarkan latar belakang, maka rumusan masalah berikut:

- 18 • Bagaimana cara memodelkan data OSMXML menjadi sebuah graf?
- 19 • Bagaimana cara menggunakan atau mengimplementasikan algoritma Dijkstra pada
- 20 sebuah graf untuk mencari rute terdekat?
- 21 • Bagaimana cara membuat visualisasi graf dan rute terdekat pada peta digital?

22 1.3 Tujuan

23 Berdasarkan rumusan masalah yang telah diuraikan di atas, maka tujuan dari penelitian
 24 yang dilakukan adalah:

- 25 • Mengetahui cara memodelkan data OSMXML menjadi sebuah graf.
- 26 • Mempelajari cara kerja algoritma Dijkstra dan mengimplementasikannya pada sebuah
- 27 graf.

- Mempelajari cara membuat visualisasi graf dan rute terdekat pada peta digital.

1.4 Batasan Masalah

Batasan permasalahan dari pembuatan aplikasi ini adalah :

- Aplikasi tidak mencari rute terdekat kedua dan seterusnya.

1.5 Metodologi Penelitian

Langkah-langkah yang akan dilakukan dalam melakukan penelitian adalah :

1. Melakukan studi pustaka untuk mengetahui teori-teori yang dapat mendukung proses pembuatan aplikasi pencarian rute terdekat.
2. Melakukan analisis teori-teori yang mendukung proses pembuatan aplikasi.
3. Membuat rancangan aplikasi.
4. Melakukan implementasi berdasarkan rancangan yang telah dibuat.
5. Melakukan pengujian aplikasi.
6. Melakukan pengambilan kesimpulan berdasarkan pengujian yang telah dilakukan.

1.6 Sistematika Pembahasan

Pada setiap bab akan dibahas beberapa hal sebagai berikut :

1. Bab Pendahuluan

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

2. Bab Dasar Teori

Bab 2 berisi teori-teori dasar mengenai OpenStreetMap, algoritma Dijkstra, Google Map Api, Graf, XML, dan beberapa teori lain yang mendukung pembuatan aplikasi.

3. Bab Analisis

Bab 3 berisi deskripsi sistem yang akan dibuat, analisis dasar teori, dan analisis cara kerja algoritma Dijkstra pada graf.

4. Bab Perancangan

Bab 4 berisi perancangan antarmuka aplikasi disertai beberapa gambar.

5. Bab Implementasi dan Pengujian

Bab 5 berisi hasil implementasi yang dilakukan disertai dokumentasi mengenai penjelasan aplikasi tersebut dan hasil pengujian yang dilakukan berupa *screenshot*

6. Bab Kesimpulan dan Saran

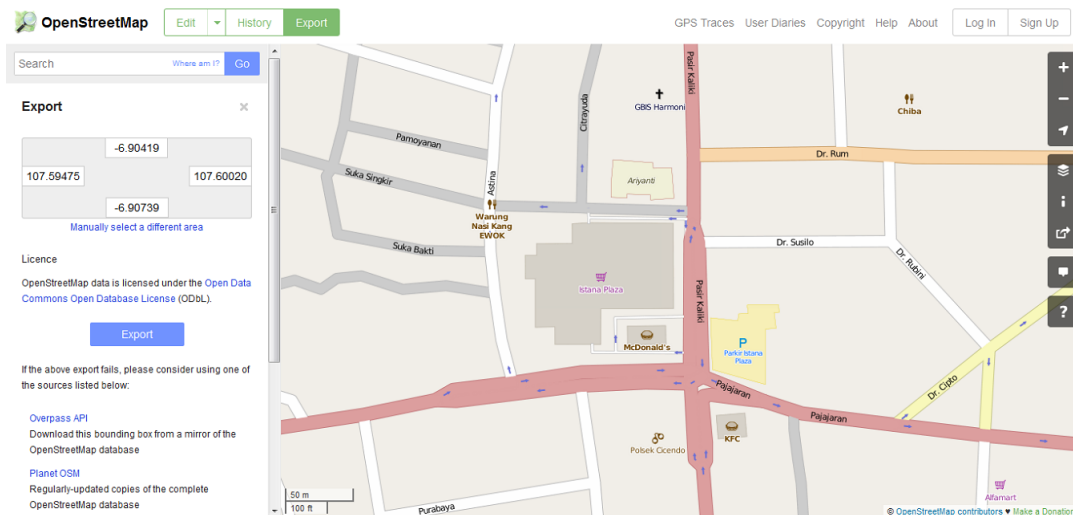
Bab 6 berisi kesimpulan dari seluruh hasil penelitian dan saran untuk pengembangan aplikasi yang akan datang.

BAB 2

DASAR TEORI

2.1 OpenStreetMap

OpenStreetMap (OSM) adalah portal peta terbuka yang menyediakan data dalam bentuk peta atau XML [2]. OSM menyediakan peta digital dan dapat diedit dari seluruh dunia, juga memungkinkan pengguna untuk mengakses gambar peta yang terdapat pada situs www.openstreetmap.org secara gratis. OSM terbentuk dan mendapatkan datanya dari berbagai sukarelawan yang bersedia untuk berkontribusi, misalnya para pengguna OSM yang menggunakan aplikasi untuk mengedit peta dan mengunggah data yang telah diedit ke situs OSM. Selain itu, OSM menyediakan beberapa aplikasi bagi para pengguna untuk mengedit peta, seperti iD online editor dan JOSM. Untuk mendapatkan gambar peta ataupun data peta dalam bentuk lain, pengguna dapat menggunakan fitur export pada situs OSM. Fitur export pada situs OSM dapat dilihat pada Gambar 2.1.



Gambar 2.1: Ekspor data pada situs OpenStreetMap

Berikut ini adalah beberapa data yang dapat diambil menggunakan fitur export [2]:

1. OpenStreetMap XML Data

OSM XML data dapat diperoleh dengan cara menggunakan tombol Export di bagian atas untuk membuka sidebar. Tombol Export mengarahkan langsung browser kepada OpenStreetMap API yang menyediakan data mentah OSM dalam bentuk XML.

2. Mapnik Image

Memungkinkan ekspor data OSM dalam bentuk PNG, JPEG, SVG, PDF dan peta PostScript.

3. *Embeddable* HTML

Fitur ini memungkinkan pengguna untuk mendapatkan kode HTML yang dapat disalin dan digunakan pada halaman web lain. Kode HTML tersebut akan menyisipkan peta dalam sebuah iframe lengkap dengan javascript.

2.2 XML

XML adalah singkatan dari eXtensible Markup Language, XML adalah bahasa markup yang dikembangkan oleh W3C (World Wide Web Consortium) [3]. Berikut ini adalah contoh dokumen XML:

```
<?xml version="1.0" encoding="utf-8"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
      with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
  </book>
</catalog>
```

Contoh di atas memberikan informasi mengenai katalog buku yang disimpan pada dokumen XML. Pada awal dokumen tertera versi XML dan *encoding* yang digunakan. Setelah itu, terdapat tag *catalog* yang memiliki *child* yaitu tag buku beserta informasinya. Terdapat informasi id buku yang tertera pada atribut tag buku, seperti `<book id="bk101">`. Dan juga informasi lain seperti judul buku, penulis, genre, harga, tanggal terbit, dan deskripsi.

XML dikembangkan terutama untuk mengatasi keterbatasan pada HTML (Hypertext Markup Language). HTML adalah salah satu bahasa markup yang paling populer dan terus dikembangkan, banyak tag baru yang diperkenalkan. Pada versi pertama, HTML memiliki satu lusin tag dan pada HTML pada versi 4.0 sudah hampir mencapai seratus

1 tag. Namun, pada aplikasi seperti *electronic commerce* dibutuhkan tag lebih untuk produk,
2 harga, nama, alamat, dan banyak lagi atau situs *streaming* memerlukan tag lebih untuk
3 mengontrol gambar dan suara.

4 HTML telah berkembang menjadi bahasa yang cukup kompleks, W3C memperkirakan
5 penggunaan komputer akan terus berkurang dan penggunaan gadget seperti smartphone
6 akan bertambah. Mesin tersebut tidak sekuat PC dan tidak bisa memproses bahasa yang
7 kompleks seperti HTML. Meskipun HTML adalah bahasa yang populer dan cukup suk-
8 ses, HTML memiliki beberapa kelemahan utama dan XML dikembangkan untuk mengatasi
9 kelemahan tersebut. XML adalah bahasa yang digunakan untuk menggambarkan dan me-
10 manipulasi dokumen terstruktur. Perubahan utama pada XML adalah tidak adanya tag
11 yang ditetapkan pada XML. Karena tidak ada tag yang ditetapkan, penulis dapat membuat
12 tag yang dibutuhkan. Beberapa ketentuan pada XML dapat dilihat pada uraian berikut:

13 1. Tag pada XML

14 Setiap elemen pada XML terdiri dari nama dan nilai, selain itu harus memiliki tag
15 pembuka dan tag penutup. Contoh:

16 `<tel> 513-555-7098 </ tel>`

17 Elemen untuk menyimpan nomor telepon memiliki nama tag tel, ditulis dengan `<tel>`
18 dan ditutup dengan `</tel>`.

19 2. Nama pada XML

20 Pemberian nama pada XML harus dimulai dengan huruf atau underscore (`_`) dan
21 sisanya diikuti huruf, angka, atau titik. Spasi tidak diperbolehkan pada pemberian
22 nama.

23 3. Atribut

24 Atribut memungkinkan untuk menyisipkan informasi tambahan, atribut juga memiliki
25 nama dan nilai. Contoh:

26 `<tel preferred="true">513-555-8889</tel>`

27 `<tel>513-555-7098</tel>`

28 Elemen tel dapat memiliki atribut *preferred*, memberikan informasi nomor telepon
29 yang lebih sering digunakan.

30 4. Elemen Kosong

31 Elemen yang tidak memiliki nilai atau isi disebut sebagai elemen kosong. Elemen
32 kosong biasanya memiliki atribut. Contoh:

33 `<email href="mailto:jdoe@emailaholic.com"></email>`

34 Elemen email tidak memiliki nilai atau isi.

35 5. *Nesting of Elements*

36 Sebuah elemen dapat memiliki elemen lain di dalamnya. Elemen yang berada di dalam
37 elemen lain disebut *child*, sedangkan elemen yang memiliki elemen lain disebut *parent*.
38 Contoh

```

1      <name>
2          <fname>Jack</fname>
3          <lname>Smith</lname>
4      </name>

```

Pada contoh berikut elemen name memiliki dua *child* yaitu fname dan lname dan elemen name merupakan *parent* dari kedua elemen tersebut.

6. Root

Root merupakan elemen level tertinggi, pada dokumen XML harus ada satu elemen pada level tertinggi. Dengan kata lain, elemen lain harus menjadi *child* dari *root*.

7. Deklarasi XML

Deklarasi XML dituliskan pada baris pertama dokumen. Pada deklarasi tersebut juga dituliskan versi XML yang digunakan. Contoh:

```

13      <?xml version="1.0"?>

```

2.2.1 OSMXML

OpenStreetMap XML atau biasa disingkat dengan OSMXML merupakan dokumen XML yang berisi data-data peta OSM. Pada dasarnya, OSMXML berisi data primitif (node, way, dan relation) yang merupakan arsitektur dari model OSM [2]. Berikut ini adalah contoh dokumen OSMXML:

```

19 <?xml version="1.0" encoding="UTF-8"?>
20 <osm version="0.6" generator="CGImap_0.0.2">
21   <bounds minlat="54.0889580" minlon="12.2487570" maxlat="
22     54.0913900" maxlon="12.2524800"/>
23   <node id="298884269" lat="54.0901746" lon="12.2482632" user="
24     SvenHRO" uid="46882" visible="true" version="1" changeset="
25     676636" timestamp="2008-09-21T21:37:45Z"/>
26   <node id="261728686" lat="54.0906309" lon="12.2441924" user="
27     PikoWinter" uid="36744" visible="true" version="1" changeset="
28     323878" timestamp="2008-05-03T13:39:23Z"/>
29   <node id="1831881213" version="1" changeset="12370172" lat="
30     54.0900666" lon="12.2539381" user="lafkor" uid="75625" visible
31     ="true" timestamp="2012-07-20T09:43:19Z">
32     <tag k="name" v="Neu_Broderstorf"/>
33     <tag k="traffic_sign" v="city_limit"/>
34   </node>
35   ...
36   <node id="298884272" lat="54.0901447" lon="12.2516513" user="
37     SvenHRO" uid="46882" visible="true" version="1" changeset="
38     676636" timestamp="2008-09-21T21:37:45Z"/>
39   <way id="26659127" user="Masch" uid="55988" visible="true"
40     version="5" changeset="4142606" timestamp="2010-03-16
41     T11:47:08Z">

```



```

1  <nd ref="292403538"/>
2  <nd ref="298884289"/>
3  ...
4  <nd ref="261728686"/>
5  <tag k="highway" v="unclassified"/>
6  <tag k="name" v="Pastower_Stra se"/>
7  <tag k="oneway" v="yes"/>
8  </way>
9  <relation id="56688" user="kmvar" uid="56190" visible="true"
10     version="28" changeset="6947637" timestamp="2011-01-12
11     T14:23:49Z">
12  <member type="node" ref="294942404" role=""/>
13  ...
14  <member type="node" ref="364933006" role=""/>
15  <member type="way" ref="4579143" role=""/>
16  ...
17  <member type="node" ref="249673494" role=""/>
18  <tag k="name" v="K ijstenbus_Linie_123"/>
19  <tag k="network" v="VWV"/>
20  <tag k="operator" v="Regionalverkehr_K ijste"/>
21  <tag k="ref" v="123"/>
22  <tag k="route" v="bus"/>
23  <tag k="type" v="route"/>
24  </relation>
25  ...
26 </osm>

```

Struktur OSMXML:

- Dokumen OSMXML diawali dengan tag xml yang menjelaskan versi xml dan encoding yang digunakan, pada contoh di atas digunakan xml versi 1.0 dan encoding UTF-8.
- Elemen osm memberikan informasi mengenai versi API dan generator yang digunakan. Generator adalah alat untuk membuat dokumen OSMXML pada saat fitur export digunakan.
- Elemen bound memberikan informasi mengenai cakupan area pada dokumen OSMXML tersebut. Dilengkapi dengan atribut koordinat yaitu latitude dan longitude. Data primitif pada OSM dibagi menjadi 3 bagian, yaitu node, way, dan relation.

1. Elemen Node merupakan informasi titik pada sebuah peta. Node memiliki beberapa atribut yaitu:

- id

Merupakan id dari node tersebut.

- user

Merupakan user yang melakukan editing pada node.

- uid
Id dari user.
- lat
berisi informasi koordinat pada garis lintang.
- lon
berisi informasi koordinat pada garis bujur.
- timestamp
Berisi informasi waktu saat node tersebut diperbaharui.

Node juga memiliki elemen tag sebagai *child* yang memberikan informasi tambahan pada node tersebut, contoh:

```
<tag k="name" v="Neu Broderstorf"/>
```

nama dari node tersebut adalah Neu Broderstorf.

2. Elemen Way merupakan informasi garis yang dapat diartikan sebagai jalan ataupun elemen lain seperti rel kereta pada peta OSM. Way menyimpan informasi node-node terurut yang dilalui oleh garis dan juga sama seperti node dilengkapi atribut seperti id, uid, user, changeset, timestamp. Elemen way memiliki *child* elemen nd, contoh:

```
<nd ref="292403538"/>
```

atribut ref pada elemen nd mengacu pada node yang memiliki id 292403538, dan elemen tag yang memberikan informasi tambahan pada elemen way. Selain itu, elemen way memiliki informasi lain yang disimpan pada elemen tag, elemen tag merupakan *child* dari elemen way dan menyimpan informasi jenis jalan yaitu *key highway* dan *key oneway* yang memberikan informasi arah jalan, contoh:

```
<tag k="highway" v="unclassified"/>
```

```
<tag k="oneway" v="yes"/>
```

Key oneway memiliki 4 jenis *value*. Informasi arah jalan mengikuti node-node yang telah terurut. Berikut ini adalah penjelasan dari keempat *value* tersebut:

- oneway=yes
Menunjukkan jalan satu arah
- oneway=no
Menujukkan jalan dua arah
- oneway=-1
Menunjukkan jalan satu arah dan berlawanan
- oneway=reversible
Menunjukkan jalan satu arah dan dapat berubah arah menjadi berlawanan.
Contoh, pengalihan jalan untuk mengatasi kemacetan.

3. Elemen relation menyimpan informasi node-node dan way yang bersinggungan. Elemen relation dapat menggambarkan suatu area seperti lapangan, taman bermain, atau pada contoh di atas menggambarkan rute bus.

2.3 Javascript

Javascript adalah bahasa pemrograman web yang mulai dikembangkan di perusahaan yang bernama Netscape. Javascript memiliki lisensi dari Sun Microsystems yang sekarang sudah berganti nama menjadi Oracle. Saat ini, mayoritas situs web sudah menggunakan javascript. Berikut ini adalah contoh penggunaan javascript pada dokumen HTML:

```

6 <!DOCTYPE html>
7 <html>
8 <head>
9 <script>
10 function myFunction() {
11     document.getElementById("demo").innerHTML = "Paragraph_changed
12     .";
13 }
14 </script>
15 </head>
16 <body>
17 <h1>JavaScript in Head</h1>
18 <p id="demo">A Paragraph.</p>
19 <button type="button" onclick="myFunction()">Try it</button>
20 </body>
21 </html>

```

Pada contoh di atas terdapat fungsi yang ditulis menggunakan javascript, fungsi tersebut akan mengubah string “A Paragraph” pada tag <p> menjadi “Paragraph changed” jika *button* atau tombol “Try it” di klik.

Seluruh browser yang terdapat pada komputer, konsol game, tablet, dan smartphone sudah disertai dengan javascript interpreter. Interpreter adalah suatu program yang berfungsi untuk menerjemahkan kode program ke dalam bahasa mesin. Javascript adalah bagian yang cukup penting pada sebuah halaman web, jika HTML berfungsi untuk menentukan isi dari halaman dan CSS untuk menentukan tampilan pada halaman, javascript berfungsi untuk menentukan “behavior” dari halaman web tersebut [4]. Berikut ini adalah uraian dari struktur javascript dan beberapa contoh sintaks:

1. Struktur

- *Character Set*

Javascript ditulis menggunakan karakter Unicode. Unicode adalah superset ASCII dan Latin-1 yang mendukung hampir seluruh bahasa di dunia.

- *Comments*

Javascript mendukung 2 jenis komentar yaitu komentar yang diletakkan setelah garis miring ganda // dan komentar yang diletakkan antara karakter /* dan */.

```
// This is a single-line comment.
```

```
/* This is also a comment */ // and here is another comment.
```

```
/*
```

```

1      * This is yet another comment.
2      * It has multiple lines.
3      */

```

- Literal

Literal adalah notasi untuk merepresentasikan nilai dan nilai yang dituliskan akan muncul secara langsung dalam program. Literal dapat berupa karakter, bilangan bulat, bilangan real, boolean. Berikut ini adalah contoh literal:

```

8      12 // The number twelve
9      1.2 // The number one point two
10     "hello world" // A string of text
11     'Hi' // Another string
12     true // A Boolean value
13     false // The other Boolean value
14     /javascript/gi // A "regular expression" literal (for pattern matching)
15     null // Absence of an object

```

- Identifier

Identifier pada javascript hanyalah nama yang digunakan untuk memberi nama pada variabel atau fungsi. Digit tidak diperbolehkan sebagai karakter pertama pada *identifier*.

- Reserved words

Reserved words adalah kata-kata yang tidak dapat digunakan sebagai identifier, karena digunakan oleh javascript sebagai keyword. Beberapa contoh keyword seperti break, delete, if, null, true, false, try, dan lain-lain.

- Optional Semicolons

Seperti banyak bahasa pemrograman lain, javascript menggunakan titik koma (;) untuk memisahkan perintah yang ditulis. Hal ini penting untuk membuat kode program menjadi jelas mengenai awal dan akhir. Pada javascript, titik koma dapat dihilangkan jika perintah ditulis pada baris yang berbeda, berikut adalah contoh penggunaan titik koma pada javascript:

```

30     a = 3;
31     b = 4;

```

titik koma pertama dapat dihilangkan, namun jika ditulis pada baris yang sama, titik koma tetap diperlukan

```

34     a = 3; b = 4;

```

35 2. Sintaks

- Deklarasi Variabel

Pembuatan variabel pada javascript menggunakan keyword var. Contoh deklarasi atau pembuatan variabel pada javascript:

```

39     var i;
40     var i, sum;

```

```

1      var message = "hello";
2      var i = 0, j = 0, k = 0;
3
4      • Fungsi
5      Fungsi adalah blok kode program yang hanya didefinisikan sekali, tapi dapat
6      dipanggil atau dijalankan berulang kali. Pada javascript, fungsi dapat dibuat
7      menggunakan keyword function. Sebuah fungsi harus memiliki nama, sepasang
8      tanda kurung untuk parameter, dan sepasang kurung kurawal. Berikut ini adalah
9      beberapa contoh fungsi:
10
11      // Print the name and value of each property of o. Return undefined.
12      function printprops(o) {
13          for(var p in o)
14              console.log(p + ": " + o[p] + "\n");
15      }
16
17      // Compute the distance between Cartesian points (x1,y1) and (x2,y2).
18      function distance(x1, y1, x2, y2) {
19          var dx = x2 - x1;
20          var dy = y2 - y1;
21          return Math.sqrt(dx*dx + dy*dy);
22      }

```

2.3.1 XMLHttpRequest

XMLHttpRequest adalah salah satu objek pada javascript yang dapat digunakan untuk mendapatkan *file* XML dari *server* secara *asynchronous* atau *synchronous* [5]. *Asynchronous* berarti bahwa pertukaran data dilakukan tanpa harus memuat ulang seluruh halaman *web*, sedangkan pertukaran data *synchronous* harus memuat ulang seluruh halaman *web*. Berikut ini adalah contoh penggunaan XMLHttpRequest:

```

26 var objXMLHTTP = new XMLHttpRequest();
27
28 objXMLHTTP.open('GET', 'books.xml', false);
29 objXMLHTTP.send(null);
30
31 var objXML = objXMLHTTP.responseXML;

```

Langkah pertama adalah dengan membuat objek XMLHttpRequest. Selanjutnya, dengan memanggil fungsi open("method", "url", asynchronous). Parameter *method* menentukan metode yang digunakan, contoh "GET" untuk menerima data dan "POST" untuk mengirim data, parameter url adalah alamat *file*, dan parameter boolean "false" menunjukkan bahwa permintaan tersebut dilakukan secara *synchronous*. Langkah terakhir adalah mendapatkan respon dari *server*. Berikut ini penjelasan dari setiap *method* yang digunakan:

1. open("method", "url", asynchronous, "username", "password")
Melakukan inisialisasi permintaan
Parameter:

- *method*

Method pada HTTP yang digunakan seperti “GET” dan “POST”.

- *url*

Alamat url tujuan

- *asynchronous*

boolean Opsional, secara default bernilai *true*. *True* menyatakan bahwa operasi yang dijalankan secara *asynchronous*. Nilai *false* menyatakan sebaliknya.

- *username*

Opsional, berisikan *username* yang digunakan untuk keperluan otentikasi. Secara default, berisi string kosong.

- *password*

Opsional, berisikan *password* yang digunakan untuk keperluan otentikasi. Secara default, berisi string kosong.

2. send(content)

Mengirimkan permintaan

Parameter:

- *content*

Opsional, *content* dapat berisi string atau data lainnya seperti Array, dokumen, dan lain-lain.

3. responseXML

Respon dari permintaan

Return:

DOM Object

2.3.2 XML DOM

DOM adalah singkatan dari *Document Object Model*, XML DOM adalah API umum untuk menangani dokumen XML [5]. API adalah singkatan dari *Application Programming Interface* merupakan fungsi atau perintah yang dapat digunakan untuk menangani masalah pemrograman tertentu. XML DOM menyediakan fungsi standar untuk mengakses, memodifikasi, dan menciptakan berbagai bagian dari sebuah dokumen XML. Contoh:

```
var myNodeset = objXML.getElementsByTagName('plant');
var name = myNodeset[0].getAttribute('name');
```

Pemanggilan fungsi `getElementsByTagName('plant')` akan mengembalikan satu set node yang memiliki nama tag 'plant'. Contoh lain, pemanggilan fungsi `getAttribute()` akan mengembalikan nilai atribut. Berikut ini penjelasan dari setiap *method* yang digunakan:

1. getElementsByTagName('tagName')

Mengembalikan elemen-elemen yang memiliki kesesuaian nama.

Parameter:

- *tagName*

String yang menentukan nama elemen yang dicari.

```

1      Return:
2      objek berisi elemen yang memiliki nama sesuai dengan yang dicari.

3      2. getAttribute('name')
4      Mengembalikan nilai atribut
5      Parameter:
6          • name
7          String yang menentukan nama atribut yang dicari.

8      Return:
9      Mengembalikan string jika atribut memiliki nilai, jika tidak mengembalikan null.

```

2.3.3 Google Maps Javascript API

Google Maps Javascript API memungkinkan untuk sebuah halaman web menampilkan peta dunia yang datanya didapat dari server google [6]. Google menyediakan fungsi atau perintah untuk menampilkan dan menyesuaikan peta sesuai dengan kebutuhan.

2.3.3.1 Elemen Dasar Google Maps

Google Maps Javascript API menyediakan fungsi dan kelas untuk memuat sebuah peta pada halaman html. Berikut ini adalah contoh halaman web yang menampilkan peta di lokasi Sydney, Australia:

```

18 <!DOCTYPE html>
19 <html>
20   <head>
21     <style type="text/css">
22       html, body, #map-canvas { height: 100%; margin: 0; padding:
23         0;}
24     </style>
25     <script type="text/javascript"
26       src="https://maps.googleapis.com/maps/api/js?key=API_KEY">
27     </script>
28     <script type="text/javascript">
29       function initialize() {
30         var mapOptions = {
31           center: { lat: -34.397, lng: 150.644},
32           zoom: 8
33         };
34         var map = new google.maps.Map(document.getElementById('map-
35           canvas'),
36           mapOptions);
37       }
38       google.maps.event.addDomListener(window, 'load', initialize)
39       ;
40     </script>

```

```

1  </head>
2  <body>
3  <div id="map-canvas"></div>
4  </body>
5  </html>

```

- Declaring

Google menyarankan untuk membuat deklarasi tipe dokumen pada awal dokumen yaitu dengan menulis `<!DOCTYPE html>`. Setelah itu diperlukan CSS yang bekerja untuk mengatur tampilan peta pada halaman web.

```

10  <style type="text/css">
11    html { height: 100% }
12    body { height: 100%; margin: 0; padding: 0 }
13    #map-canvas { height: 100% }
14  </style>

```

Kode CSS pada contoh menunjukkan tag yang memiliki id `map-canvas` akan memiliki tinggi 100% pada saat ditampilkan dan juga menunjukkan persentase yang sama pada `<html>` dan `<body>`.

- Loading Google Maps API

Untuk dapat menampilkan peta diperlukan juga melakukan *load* javascript. URL yang terdapat pada tag script adalah lokasi file javascript yang akan memuat seluruh simbol dan definisi yang dibutuhkan untuk menggunakan Google Maps API ini. Paramater key berisi API key yang dimiliki oleh pengguna.

```

23  <html>
24    <head>
25      <script type="text/javascript"
26        src="https://maps.googleapis.com/maps/api/js?key=API_KEY">
27      </script>

```

- Initialize

Setelah melakukan load javascript, diperlukan pemanggilan fungsi initialize. Di dalam fungsi tersebut dapat ditambahkan beberapa variabel yang dibutuhkan.

```

31  function initialize() {}

```

Untuk inisialisasi peta, diperlukan variabel *map options*

```

33  var mapOptions = {};

```

Selanjutnya diperlukan koordinat pusat peta yang akan ditampilkan, sedangkan zoom menunjukkan level zoom yang ingin ditampilkan

```

36  center: new google.maps.LatLng(-34.397, 150.644),
37  zoom: 8

```


- 1 • Map Object
- 2 objek peta perlu dibuat dengan cara melakukan inisialisasi kelas `google.maps.Map`.
- 3 Pada contoh, peta diletakkan pada `<div>` yang memiliki id `map-canvas`.

```
4 var map = new google.maps.Map(document.getElementById("map-canvas"),  
5     mapOptions);
```

- 6 • Loading the Map
- 7 Google Maps API menyediakan fungsi untuk memuat peta. Pada potongan kode di
- 8 bawah, fungsi *listener* akan memanggil fungsi *initialize* ketika halaman dimuat.

```
9 google.maps.event.addDomListener(window, 'load', initialize);
```

10 Berikut ini adalah penjelasan kelas dan fungsi yang digunakan:

11 1. `google.maps.Map` class

12 Membuat peta baru pada halaman html.

13 Konstruktor:

- 14 • `mapDiv:Node`
15 node yang digunakan untuk membuat peta.
- 16 • `opts?:MapOptions`
17 Opsi dari *map* yang akan dibuat.

18 2. `google.maps.LatLng` class

19 Membuat objek `LatLng` yang merepresentasikan titik geografis.

20 Konstruktor:

- 21 • `lat:number`
22 *Latitude* dalam derajat.
- 23 • `lng:number`
24 *Longitude* dalam derajat.
- 25 • `noWrap?:boolean`
26 *Latitude* ditentukan dalam rentang derajat -90 hingga 90 dan *longitude* diten-
27 tukan dalam rentang derajat -180 hingga 180. Nilai *true* pada boolean `noWrap`
28 untuk mengaktifkan nilai di luar batas tersebut.

29 3. `google.maps.event.addDomListener(instance:Object, eventName:string, handler:Function)`

30 Menambahkan fungsi *listener*

31 Parameter:

- 32 • `instance:Object`
33 Objek yang ditambahkan *listener*.
- 34 • `eventName:string`
35 Nama *Event*.
- 36 • `handler:Function`
37 Fungsi yang dipanggil ketika *event* terjadi.

38 Return:

39 `MapsEventListener`

2.3.3.2 Menggambar pada Peta

Peta pada Google Maps API dapat ditambahkan objek seperti titik, garis, area, atau objek lainnya. objek tersebut dinamakan *overlay*. Terdapat beberapa jenis *overlay* yang dapat ditambahkan pada peta yaitu *marker* dan *polyline*. Berikut ini adalah penjelasan kelas dan fungsi yang digunakan:

1. google.maps.Marker class

Membuat *marker* pada peta dengan opsi tertentu.

Konstruktor:

- `opts?:MarkerOptions`

Opsi dari *marker* yang dibuat.

2. google.maps.Polyline class

Membuat *polyline* pada peta dengan opsi tertentu.

Konstruktor:

- `opts?:PolylineOptions`

Opsi dari *polyline* yang dibuat.

3. setMap(map:Map)

Menyisipkan *marker* atau *polyline* pada peta tertentu.

Parameter:

- `map:Map`

Peta yang disisipkan *marker* atau *polyline*.

4. setIcon(icon:string|Icon|Symbol)

Mengubah *icon* pada *marker*.

Parameter:

- `icon:string|Icon|Symbol`

Icon yang digunakan.

Berikut ini adalah contoh penggunaan *marker* dan *polyline* pada peta:

1. Marker

Lokasi tunggal pada peta ditunjukkan oleh *Marker*.

- Menambahkan *Marker*

Untuk menampilkan *marker* pada peta harus membuat objek `google.maps.Marker`.

Berikut ini adalah atribut penting pada saat membuat objek *marker*:

(a) *position*

atribut *position* diperlukan untuk mengatur letak *marker* pada peta.

(b) *map*

atribut *map* bersifat opsional, untuk menentukan *marker* tersebut akan diletakkan pada peta. Jika atribut *map* tidak diatur, maka *marker* akan tetap dibuat tetapi tidak akan ditampilkan pada peta.

Berikut ini adalah contoh kode program untuk menambahkan *marker* pada peta:

```

1      var myLatLng = new google.maps.LatLng(-25.363882,131.044922);
2      var mapOptions = {
3          zoom: 4,
4          center: myLatLng
5      }
6      var map = new google.maps.Map(document.getElementById
7      ("map-canvas"), mapOptions);
8
9      // To add the marker to the map, use the 'map' property
10     var marker = new google.maps.Marker({
11         position: myLatLng,
12         map: map,
13         title:"Hello World!"
14     });

```

Pada contoh, objek `google.maps.Marker` yang dibuat disimpan pada variabel *marker*, terdapat atribut *position* menggunakan variabel *myLatLng* yang berisi koordinat (-25.363882,131.044922), atribut *map* menunjukkan bahwa *marker* akan ditampilkan pada objek map yang tersimpan pada variabel *map*, dan atribut yang menunjukkan judul *marker*.

- Mengubah *icon marker*

Untuk mengubah *icon*, diperlukan pengaturan pada konstruktor *marker* tersebut. Pada contoh, *icon marker* diubah menjadi *beachflag.png*.

```

23     var image = 'images/beachflag.png';
24     var myLatLng = new google.maps.LatLng(-33.890542, 151.274856);
25     var beachMarker = new google.maps.Marker({
26         position: myLatLng,
27         map: map,
28         icon: image
29     });

```

Selain pengaturan pada konstruktor, pengubahan *icon* juga dapat dilakukan dengan cara memanggil fungsi `setIcon()`

```

32     beachMarkers.setIcon('images/beachflag.png');

```

- Menghapus *Marker* pada peta

Untuk menghapus *marker* pada peta, hanya diperlukan pemanggilan fungsi `setMap()` dan mengisi parameter fungsi tersebut dengan *null*. Contoh:

```

36     marker.setMap(null);

```

Pada contoh di atas hanya menghilangkan *marker* dari peta dan tidak menghapus objek *marker*.

- Animasi *Marker*

Menambahkan animasi pada *marker*, hanya memerlukan pengaturan atribut pada konstruktor `google.maps.Marker`. Contoh:

```

1      var marker = new google.maps.Marker({
2          position: myLatLng,
3          map: map,
4          animation: google.maps.Animation.BOUNCE,
5          title:"Hello World!"
6      });

```

Pada contoh, menambahkan animasi *bounce* pada marker sehingga *marker* bergerak melompat-lompat pada peta.

- Mengubah Ikon

Gambar *marker* pada peta dapat diubah sesuai keinginan, hanya memerlukan pengaturan atribut pada konstruktor `google.maps.Marker`. Contoh:

```

12     var image = 'images/beachflag.png';
13     var myLatLng = new google.maps.LatLng(-33.890542, 151.274856);
14     var beachMarker = new google.maps.Marker({
15         position: myLatLng,
16         map: map,
17         icon: image
18     });

```

Pada contoh, ikon *marker* akan ditampilkan menggunakan *file* gambar *beachflag.png*

- *Draggable*

Draggable memungkinkan pengguna untuk menyeret marker ke lokasi yang berbeda, hanya memerlukan pengaturan atribut pada konstruktor `google.maps.Marker`. Contoh:

```

24     var marker = new google.maps.Marker({
25         position: myLatLng,
26         map: map,
27         draggable: true,
28         title:"Hello World!"
29     });

```

30 2. *Polyline*

Objek *polyline* adalah serangkaian garis pada peta, *polyline* berguna untuk menunjukkan dari satu titik ke titik lain. *Polyline* memiliki atribut yang dapat diubah sesuai kebutuhan seperti warna, *opacity*, dan *weight*. Berikut ini penjelasan dari beberapa atribut tersebut:

- *strokeColor*

Atribut *strokeColor* menentukan warna dalam format heksadesimal, contoh `"#FFFFFF"`.

- *strokeOpacity*

Atribut *strokeOpacity* menentukan *opacity* dalam nilai antara 0.0 dan 1.0.

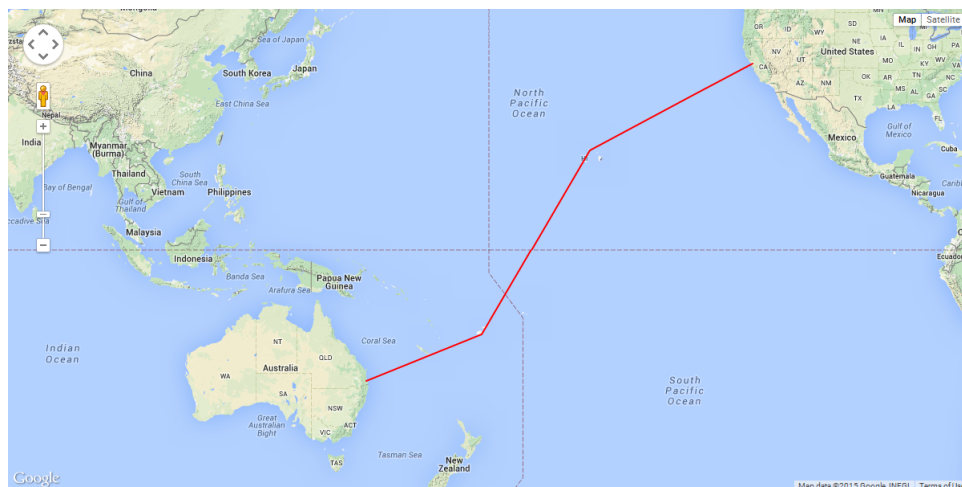
- *strokeWeight*

Atribut *strokeWeight* menentukan lebar garis dalam piksel.

Berikut ini adalah contoh potongan kode program untuk menampilkan *polyline* pada peta:

```
var flightPlanCoordinates = [  
    new google.maps.LatLng(37.772323, -122.214897),  
    new google.maps.LatLng(21.291982, -157.821856),  
    new google.maps.LatLng(-18.142599, 178.431),  
    new google.maps.LatLng(-27.46758, 153.027892)  
];  
var flightPath = new google.maps.Polyline({  
    path: flightPlanCoordinates,  
    strokeColor: '#FF0000',  
    strokeOpacity: 1.0,  
    strokeWeight: 2  
});  
  
flightPath.setMap(map);
```

Pada contoh, akan menampilkan polyline pada peta yang akan menghubungkan setiap koordinat yang terdapat pada variabel *flightPlanCoordinates*. *Polyline* yang ditampilkan pada peta dapat dilihat pada Gambar 2.2.



Gambar 2.2: Polyline pada Peta

2.3.3.3 Geometry Library

Gambar pada Google Maps adalah dua dimensi, sedangkan bumi adalah tiga dimensi yang menyerupai bentuk bola. Hal ini tentu akan berbeda ketika mengukur suatu jarak dari satu titik ke titik lain, misalnya jarak terpendek antara dua titik pada bola bukanlah garis lurus, tetapi menyerupai lingkaran besar atau busur. Karena perbedaan tersebut, diperlukan *spherical geometry* untuk menghitung data geometris pada permukaan bumi seperti sudut, jarak, dan area yang berdasarkan garis lintang dan garis bujur. Google Maps JavaScript API menyediakan *geometry library* yang memiliki fungsi utilitas tersebut, fungsi utilitas

tersebut dinamakan `google.maps.geometry.spherical`. Untuk menghitung jarak antara dua titik dapat memanggil fungsi `computeDistanceBetween()`.

1. `google.maps.geometry.spherical` namespace
Fungsi utilitas untuk menghitung sudut, jarak, dan area. Secara *default*, radius bumi yang digunakan adalah 6378137 meter.
2. `computeDistanceBetween(from:LatLng, to:LatLng, radius?:number)`
Menghitung jarak antara dua titik.
Parameter:
 - `from:LatLng`
Koordinat titik pertama.
 - `to:LatLng`
Koordinat titik kedua.
 - `radius?:number`
Radius yang digunakan.

Berikut ini adalah contoh penggunaan fungsi `computeDistanceBetween()` untuk menghitung jarak antara koordinat Kota Jakarta dan koordinat Kota Bandung:

```
var jakarta = new google.maps.LatLng(-6.1745,106.8227);
var bandung = new google.maps.LatLng(-6.9167,107.6000);

var distance = google.maps.geometry.spherical
.computeDistanceBetween(jakarta, bandung);
```

setelah menggunakan fungsi `computeDistanceBetween()`, didapatkan jarak antara dua titik koordinat tersebut adalah 119231.23264342443 meter atau lebih kurang 119,2 kilometer.

2.3.3.4 Info Window

Info window adalah kelas yang disediakan Google Maps untuk menampilkan konten (biasanya berupa teks atau gambar) pada jendela *popup*. *Info window* memiliki ujung yang melekat ke lokasi tertentu pada peta. Biasanya *info window* diletakkan pada *marker* yang ada pada peta, tetapi *info window* juga dapat diletakkan pada koordinat peta tertentu. Berikut ini adalah contoh potongan kode program yang menampilkan *marker* beserta *info window*:

```
var contentString = 'Info Window';

var infowindow = new google.maps.InfoWindow({
  content: contentString
});

var marker = new google.maps.Marker({
  position: myLatLng,
  map: map,
```

```

1      title: 'Uluru (Ayers Rock)'
2    });
3    google.maps.event.addListener(marker, 'click', function() {
4      infowindow.open(map,marker);
5    });

```

6 Pada contoh, terdapat variabel *contentString* yang berisi teks yang akan dimuat pada *info*
 7 *window*. Selanjutnya, diperlukan variabel *infowindow* yang menginisialisasi *info window*
 8 dan variabel *marker* yang menginisialisasi *marker*. Dan *listener* yang memanggil fungsi
 9 *open* ketika *marker* tersebut diklik. Berikut ini adalah penjelasan dari kelas dan fungsi yang
 10 digunakan:

11 1. google.maps.InfoWindow class

12 Membuat *overlay* yang berbentuk seperti gelembung dan memuat konten seperti teks
 13 atau gambar.

14 Konstruktork:

15 • *opts?:InfoWindowOptions*

16 Opsi dari *info window* yang dibuat.

17 2. *open(map?:Map|StreetViewPanorama, anchor?:MVCObject)*

18 Membuka *info window* pada peta.

19 Parameter

20 • *map?:Map|StreetViewPanorama*

21 Membuka *info window* pada peta yang diberikan.

22 • *anchor?:MVCObject*

23 Objek yang berasosiasi dengan *info window*, contoh: *marker*.

Info Window yang ditampilkan pada peta dapat dilihat pada Gambar 2.3.

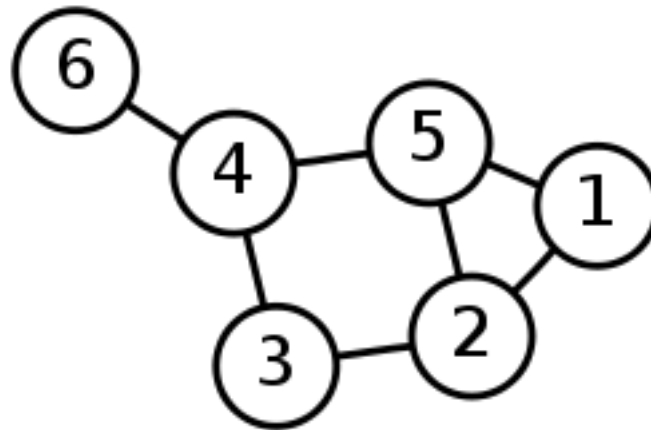


Gambar 2.3: Info Window pada Peta

2.4 Graf

Graf adalah himpunan objek yang terdiri dari node (simpul) dan edge (sisi), graf digambarkan sebagai node yang dihubungkan oleh edge. Terdapat berbagai macam jenis graf, tetapi pada subbab ini akan dibahas beberapa jenis graf seperti graf tidak berarah, graf berarah, dan graf berbobot. Contoh graf dapat dilihat pada Gambar 2.4. Graf mengikuti aturan²:

1. Graf terdiri dari dua bagian yang disebut node dan edge.
2. Node digambarkan berdasarkan tipenya dan nilainya mungkin terbatas atau tidak terbatas.
3. Setiap node menghubungkan dua buah edge.
4. Node digambarkan sebagai kotak atau lingkaran dan edge digambarkan sebagai garis atau busur.



Gambar 2.4: Contoh Graf

Berdasarkan contoh pada Gambar 2.4 didapatkan informasi tipe dari node adalah bilangan bulat

Himpunan node = 1,2,3,4,5,6

Himpunan edge = (6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)

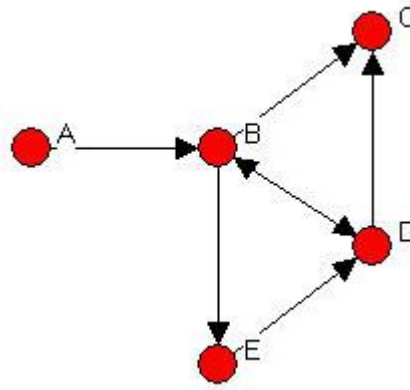
2.4.1 Graf Tidak Berarah

Graf tidak berarah adalah graf yang tidak memiliki arah pada setiap edgenya, sehingga setiap node tidak memiliki urutan. Graf tidak berarah digambarkan dengan garis lurus antara node. Contoh graf berarah dapat dilihat pada Gambar 2.4.

2.4.2 Graf Berarah

Graf berarah memiliki arah pada setiap edgenya. Pada graf berarah, edge biasanya digambarkan dengan panah sesuai arahnya. Contoh graf berarah dapat dilihat pada Gambar 2.5. Berdasarkan contoh pada Gambar 2.5 didapatkan informasi tipe dari node adalah huruf

²<http://web.cecs.pdx.edu/sheard/course/Cs163/Doc/Graphs.html>

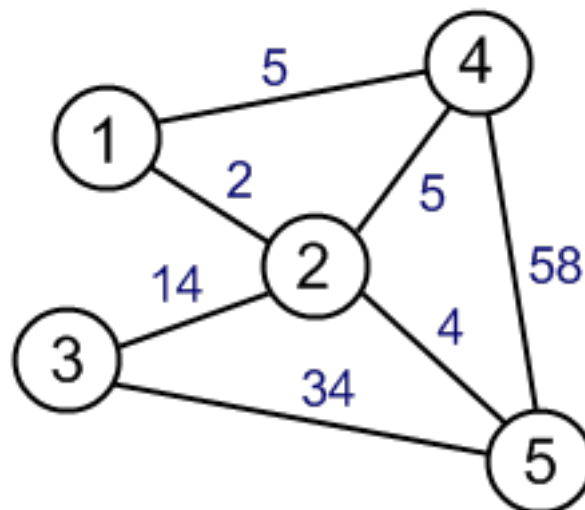


Gambar 2.5: Contoh Graf Berarah

- 1 kapital.
- 2 Himpunan node = A, B, C, D, E
- 3 Himpunan edge = (A, B), (B, C), (D, C), (B, D), (D, B), (E, D), (B, E)

2.4.3 Graf Berbobot

- 5 Graf berbobot adalah graf yang memiliki nilai pada setiap edgenya. Nilai tersebut dapat
- 6 berupa bilangan bulat ataupun bilangan pecahan desimal. Nilai tersebut dapat digunakan
- 7 untuk menyimpan jarak dari suatu node ke node lain. Contoh graf berbobot dapat dilihat
- pada Gambar 2.6. Berdasarkan contoh pada Gambar 2.6 didapatkan informasi tipe dari



Gambar 2.6: Contoh Graf Berbobot

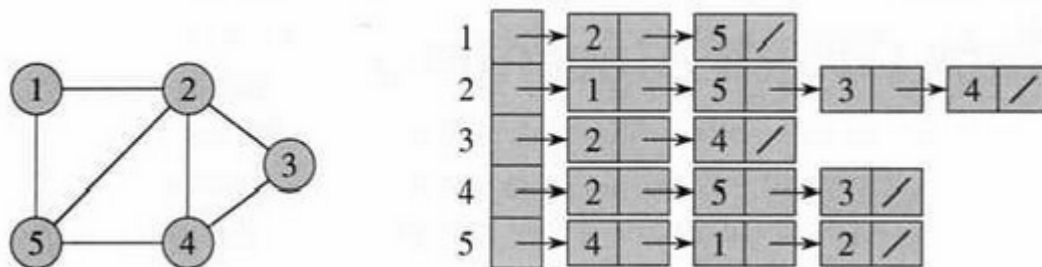
- 8
- 9 node adalah bilangan bulat dan tipe dari bobot adalah bilangan bulat.
- 10 Himpunan node = 1,2,3,4,5
- 11 Himpunan edge = (1,4,5), (4,5,58), (3,5,34), (2,4,5), (2,5,4), (3,2,14), (1,2,2)

2.4.4 Representasi Graf

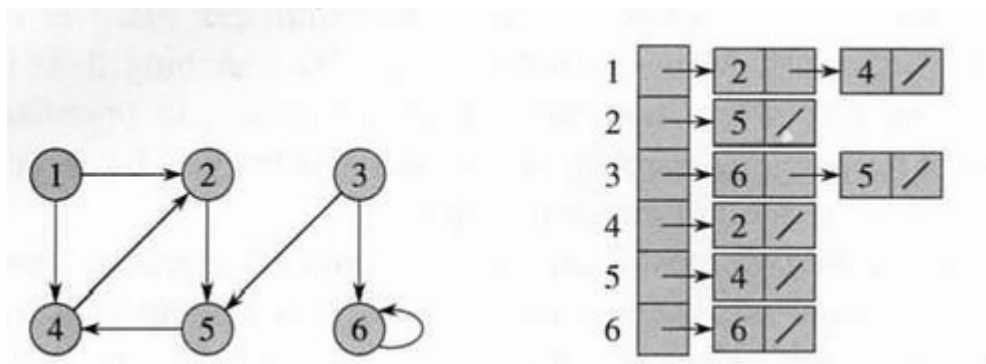
Terdapat dua cara untuk merepresentasikan graf yaitu dengan *adjacency list* dan *adjacency matrix* [1]. Keduanya dapat merepresentasikan graf berarah ataupun graf tidak berarah. *Adjacency list* merepresentasikan graf ke dalam bentuk array, sedangkan *adjacency matrix* merepresentasikan graf ke dalam bentuk matriks.

- *Adjacency List*

Adjacency List merupakan representasi graf ke dalam bentuk array, panjang array sesuai dengan jumlah node pada graf. Setiap index pada array mengacu pada setiap node graf, setiap index array tersebut memiliki list yang merepresentasikan hubungan dengan node-node lainnya. Contoh representasi graf tidak berarah dalam bentuk *adjacency list* dapat dilihat pada Gambar 2.7 dan representasi graf berarah dalam bentuk *adjacency list* dapat dilihat pada Gambar 2.8.



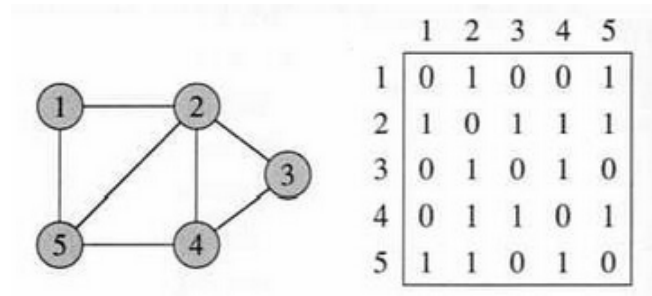
Gambar 2.7: Contoh Adjacency List (Graf Tidak Berarah)



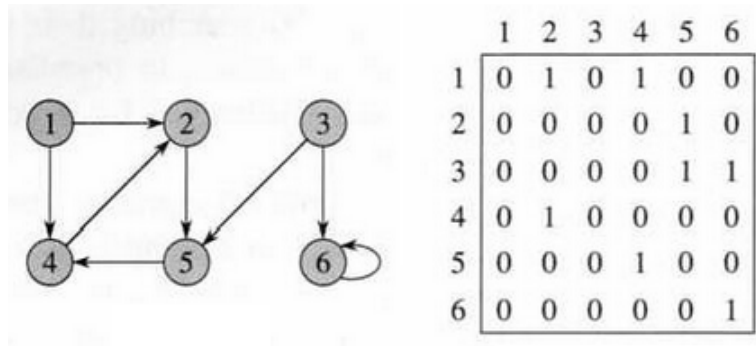
Gambar 2.8: Contoh Adjacency List (Graf Berarah)

- *Adjacency Matrix*

Adjacency Matrix merupakan representasi graf ke dalam bentuk matriks $n \times n$, pada matriks tersebut menyatakan hubungan antar node atau pada graf. Nilai n pada matriks $n \times n$ sesuai dengan jumlah node pada graf. Nilai 1 pada matriks menandakan terdapat hubungan pada node dan sebaliknya jika bernilai 0. Contoh representasi graf tidak berarah dalam bentuk *adjacency matrix* dapat dilihat pada Gambar 2.9 dan representasi graf berarah dalam bentuk *adjacency matrix* dapat dilihat pada Gambar 2.10.



Gambar 2.9: Contoh Adjacency Matrix (Graf Tidak Berarah)



Gambar 2.10: Contoh Adjacency Matrix (Graf Berarah)

2.5 Algoritma Dijkstra

- Algoritma dijkstra adalah algoritma yang dapat mencari jalur terpendek pada graf bera-
rah dengan persamaan $G=(V,E)$ untuk kasus pada setiap sisinya bernilai tidak negatif [1].
Algoritma ini menggunakan prinsip greedy. Prinsip greedy pada algoritma dijkstra adalah
memilih sisi yang memiliki bobot paling kecil dan memasukannya dalam himpunan solusi.
Berikut ini adalah *pseudocode* dari algoritma dijkstra:

Algorithm 1 *Dijkstra*

```

 $dist[s] \leftarrow 0$ 
for all  $v \in V$  do
   $dist[v] \leftarrow \infty$ 
end for
 $S \leftarrow \emptyset$ 
 $Q \leftarrow V$ 
while  $Q \neq \emptyset$  do
   $u \leftarrow minDistance(Q, dist)$ 
   $S \leftarrow u$ 
  for all  $v \in neighbors[u]$  do
    if  $dist[v] > dist[u] + w(u, v)$  then
       $d[v] \leftarrow d[u] + w(u, v)$ 
    end if
  end for
end while
return  $dist$ 

```

2.6 Haversine *Formula*

Haversine *formula* adalah persamaan yang dapat memberikan jarak antara dua titik berdasarkan *latitude* atau garis lintang dan *longitude* atau garis bujur [7]. Haversine *formula* dinyatakan dalam persamaan berikut ini:

$$d = 2r \sin^{-1} \left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cos(\phi_2) \sin^2\left(\frac{\psi_2 - \psi_1}{2}\right)} \right)$$

dimana:

- d : jarak antara dua buah titik
- r : radius bumi (6371 km)
- ϕ_1, ϕ_2 : *latitude* dari titik 1 and *latitude* dari titik 2
- ψ_1, ψ_2 : *longitude* dari titik 1 and *longitude* dari titik 2

BAB 3

ANALISIS

Pada bab ini akan dipaparkan analisis yang dilakukan dalam pembuatan aplikasi ini. Bagaimana data XML didapatkan dari situs www.openstreetmap.org yang akan dibahas pada subbab 3.1 dan membacanya menggunakan beberapa fungsi javascript yang akan dibahas pada subbab 3.2. Setelah itu, data tersebut disimpan atau dikonversi ke dalam bentuk graf sehingga dapat diimplementasikan algoritma dijkstra untuk mencari rute terpendek dari satu node ke node lain. Berdasarkan informasi yang telah diolah, maka dapat dibuat visualisasi data atau informasi tersebut menggunakan Google Maps Javascript API. Pada akhir bab ini, juga akan dibahas mengenai diagram *use case* dan skenario untuk memperjelas apa saja yang dapat dilakukan oleh *user* pada aplikasi ini.

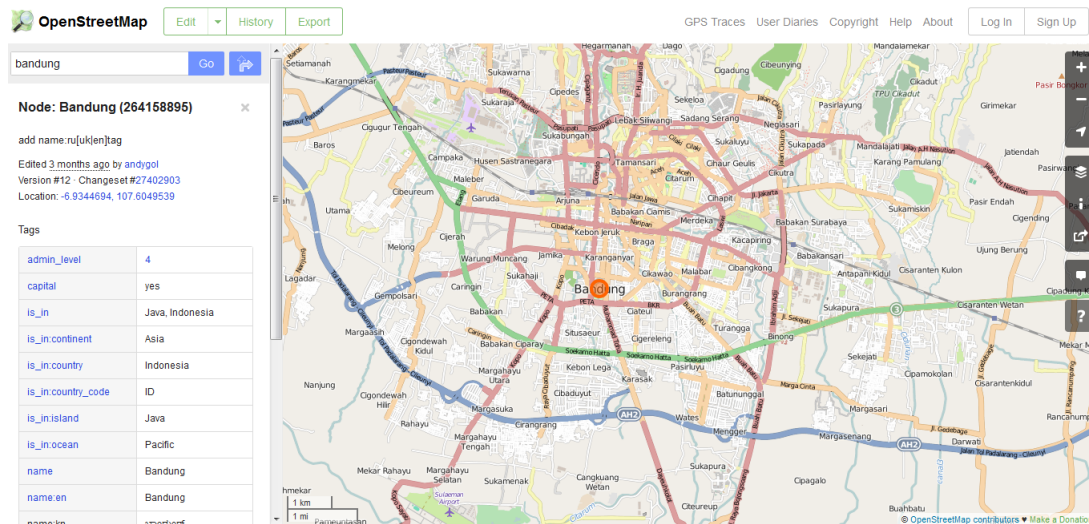
3.1 Analisis OpenStreetMap

OpenStreetMap adalah portal peta terbuka yang menyediakan data dalam bentuk peta ataupun dokumen XML. Aplikasi yang dibuat akan berbasis OpenStreetMap, hal ini berarti aplikasi yang dibuat akan menggunakan data yang diperoleh dari situs www.openstreetmap.org. Untuk mendapatkan data peta pada situs OpenStreetMap, user harus mengunjungi situs tersebut dan menggunakan fitur *export*. Data yang digunakan adalah data peta yang berbentuk dokumen XML atau biasa disebut dengan OSMXML. Selanjutnya, informasi yang terkandung di dalam dokumen OSMXML tersebut akan diproses untuk mengetahui node dan edge pada peta. Informasi tersebut akan diubah ke dalam bentuk graf yang akan diproses lebih lanjut menggunakan algoritma dijkstra untuk mengetahui jarak terpendek dari satu node ke node lain.

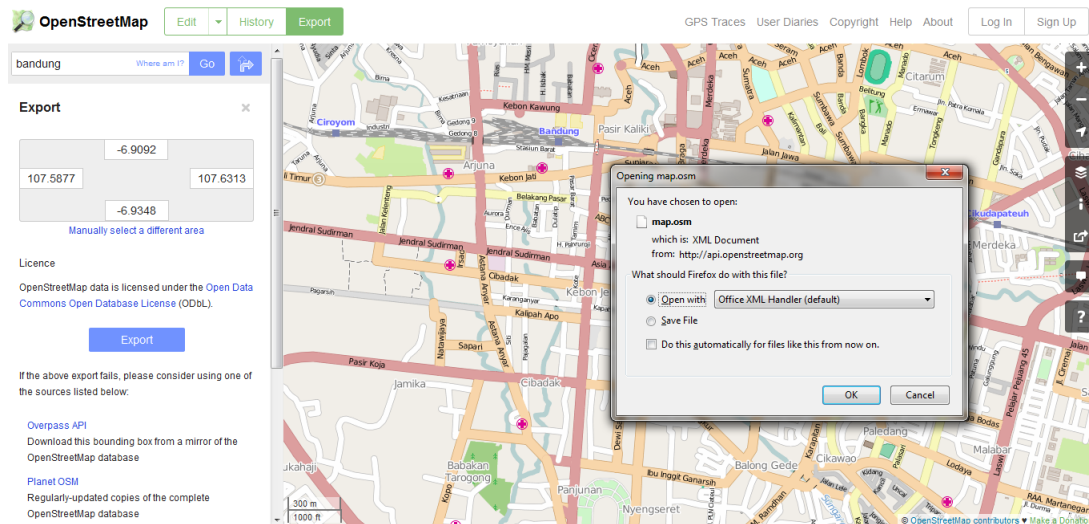
3.1.1 Langkah-Langkah Pengambilan Data OSMXML

Berikut ini adalah langkah-langkah pengambilan data OSMXML yang akan digunakan:

1. Mengunjungi situs www.openstreetmap.org.
2. Menggunakan fitur *search* untuk mencari area lokasi yang diinginkan. penggunaan fitur ini dapat dilihat pada Gambar 3.1 .
3. Menggunakan fitur *export* untuk mengunduh data dalam bentuk dokumen OSMXML. penggunaan fitur ini dapat dilihat pada Gambar 3.2.



Gambar 3.1: Fitur search pada situs OpenStreetMap

Gambar 3.2: Fitur *export* pada situs OpenStreetMap

3.1.2 OSMXML

- 2 Sesuai dengan pembahasan pada subbab 2.2.1, OSMXML merupakan dokumen XML yang
- 3 mengandung data-data peta OpenStreetMap. Berikut ini adalah dokumen OSMXML yang
- 4 sudah diunduh dan digunakan pada proses analisis.

```

5 <?xml version="1.0" encoding="UTF-8"?>
6 <osm version="0.6" generator="CGImap_0.3.3_(29805_thorn-03.
7   openstreetmap.org)" copyright="OpenStreetMap_and_contributors"
8   attribution="http://www.openstreetmap.org/copyright" license="
9   http://opendatacommons.org/licenses/odbl/1-0/">
10 <bounds minlat="-6.9076500" minlon="107.5961800" maxlat="
11   -6.9044500" maxlon="107.6016300"/>
12 <node id="25418868" visible="true" version="6" changeset="
13   27915808" timestamp="2015-01-04T17:54:58Z" user="isonpurba"
14   uid="2552445" lat="-6.9064389" lon="107.5976351"/>

```

```

1 <node id="25433683" visible="true" version="3" changeset="839915"
2   timestamp="2009-03-21T14:18:48Z" user="adhitya" uid="7748"
3   lat="-6.9067659" lon="107.5989458"/>
4 ...
5 <node id="25433687" visible="true" version="2" changeset="839915"
6   timestamp="2009-03-21T14:18:36Z" user="adhitya" uid="7748"
7   lat="-6.9040267" lon="107.5969508"/>
8 <node id="25433688" visible="true" version="2" changeset="839915"
9   timestamp="2009-03-21T14:18:58Z" user="adhitya" uid="7748"
10  lat="-6.9039393" lon="107.5963723"/>
11 <node id="25500626" visible="true" version="3" changeset="839915"
12  timestamp="2009-03-21T14:22:17Z" user="adhitya" uid="7748"
13  lat="-6.9070329" lon="107.6019401"/>
14 ...
15 <node id="3030289971" visible="true" version="1" changeset="
16   24892866" timestamp="2014-08-20T18:40:31Z" user="albahrimaraxsa
17   " uid="2162153" lat="-6.9066710" lon="107.5982569"/>
18 <node id="2325451442" visible="true" version="4" changeset="
19   27916144" timestamp="2015-01-04T18:06:33Z" user="isonpurba"
20   uid="2552445" lat="-6.9045011" lon="107.6024922"/>
21 <way id="4567626" visible="true" version="4" changeset="15861148"
22   timestamp="2013-04-25T13:56:12Z" user="mrdoggie94" uid="
23   1331966">
24   <nd ref="25433682"/>
25   <nd ref="25433681"/>
26   <nd ref="25433680"/>
27   <tag k="avgspeed" v="15"/>
28   <tag k="highway" v="residential"/>
29   <tag k="name" v="Dr. J. Rubini"/>
30 </way>
31 <way id="4567634" visible="true" version="2" changeset="7821743"
32   timestamp="2011-04-10T11:15:30Z" user="evo2mind" uid="234610">
33   <nd ref="25433681"/>
34   <nd ref="28802396"/>
35   <tag k="avgspeed" v="15"/>
36   <tag k="highway" v="residential"/>
37   <tag k="name" v="Dr. J. Susilo"/>
38 </way>
39 ...
40 </osm>

```

Node dan way memiliki informasi penting yang akan digunakan pada aplikasi. Pada tag node terdapat atribut id yang menunjukkan id pada setiap node, kemudian terdapat atribut lat dan lon yang memberikan informasi titik koordinat (*latitude* dan *longitude*) pada node tersebut. Informasi yang didapatkan akan disimpan ke dalam bentuk node pada graf. Tag

1 way akan menunjukkan hubungan pada node-node yang terdapat pada dokumen, dan akan
 2 disimpan sebagai edge pada graf. Selain itu, tag way tidak hanya memberikan informasi
 3 jalan raya atau jalan besar saja, tetapi juga beberapa elemen peta lain seperti area sekeli-
 4 ling bangunan atau area sekitar tempat parkir. Maka dari itu, diperlukan *filter* pada tag
 5 way, karena hanya informasi jalan raya atau jalan besar saja yang diperlukan oleh aplikasi.
 6 Data atau dokumen OSMXML yang telah diperoleh, selanjutnya akan dibaca menggunakan
 7 javascript yang akan dibahas pada subbab 3.2.

8 3.2 Analisis Javascript

9 Javascript diperlukan untuk membaca dokumen OSMXML, sehingga seluruh informasi yang
 10 diperlukan dapat diubah ke dalam bentuk graf yang akan diproses lebih lanjut. XMLHttpRequest
 11 adalah salah satu objek pada javascript yang dapat digunakan untuk mendapatkan
 12 *file* XML. Berikut ini adalah langkah-langkah yang dilakukan untuk membaca dokumen
 13 OSMXML menggunakan javascript:

- 14 1. Membuat Objek XMLHttpRequest.

```
15 xmlhttp=new XMLHttpRequest();
16 xmlhttp.open("GET","map.xml",false);
17 xmlhttp.send();
18 xmlDoc=xmlhttp.responseXML;
```

19 Objek XMLHttpRequest digunakan untuk mendapatkan *file* XML yang telah diunduh
 20 sebelumnya.

- 21 2. Menampilkan informasi node yang didapat pada layar.

```
22 document.write("<div style='float: left'>");
23 document.write("<table><tr><th>Node</th><th>Id</th>
24 <th>Latitude</th><th>Longitude</th></tr>");
25 document.write("<caption>Node</caption>");
26 var node=xmlDoc.getElementsByTagName("node");
27 for (ct=0;ct<node.length;ct++)
28 {
29     document.write("<tr><td>");
30     document.write(ct);
31     document.write("</td><td>");
32     document.write(node[ct].getAttribute('id'));
33     document.write("</td><td>");
34     document.write(node[ct].getAttribute('lat'));
35     document.write("</td><td>");
36     document.write(node[ct].getAttribute('lon'));
37     document.write("</td></tr>");
38 }
39 document.write("</table>");
40 document.write("</div>");
```


1 Kode diatas menampilkan informasi node pada dokumen OSMXML dalam bentuk
2 tabel.

3 3. Membuat fungsi untuk melakukan *filter* pada elemen way.

```
4 function isHighway(way,index){
5     var tag = way[index].getElementsByTagName("tag");
6     for (hg=0;hg<tag.length;hg++)
7     {
8         if(tag[hg].getAttribute('k') == "highway"){
9             return true;
10        }
11    }
12    return false;
13 }
```

14 *Filter* dilakukan karena hanya elemen way yang berjenis *highway* saja yang akan di-
15 gunakan.

16 4. Menampilkan informasi way yang didapat pada layar.

```
17 document.write("<div style='margin-left: 20px;float: left'>");
18 document.write("<table><tr><th>Way</th><th>Id Way</th>
19 <th>Edge</th><th>Id Node 1</th><th>Id Node 2</th></tr>");
20 document.write("<caption>Edge</caption>");
21 var way = xmlDoc.getElementsByTagName("way");
22 var nd;
23 for (i=0;i<way.length;i++)
24 {
25     nd = way[i].getElementsByTagName("nd");
26     if(isHighway(way,i)){
27         for (j=0;j<nd.length-1;j++)
28         {
29             document.write("<tr><td>");
30             document.write(i);
31             document.write("</td><td>");
32             document.write(way[i].getAttribute('id'));
33             document.write("</td><td>");
34             document.write(j);
35             document.write("</td><td>");
36             document.write(nd[j].getAttribute('ref'));
37             document.write("</td><td>");
38             document.write(nd[j+1].getAttribute('ref'));
39             document.write("</td></tr>");
40         }
41     }
```

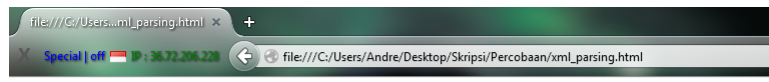
```

1      }
2      document.write("</div>");

```

3 Kode diatas menampilkan informasi way pada dokumen OSMXML dalam bentuk ta-
 4 bel.

Hasil dari kode program di atas dapat dilihat pada Gambar 3.3.



Node				Edge			
Node	Id	Latitude	Longitude	Way	Id Way	Edge	Id Node 1
0	25418868	-6.9064389	107.5976351	0	4567626	0	25433682
1	25433683	-6.9067659	107.5989458	0	4567626	1	25433681
2	25433687	-6.9040267	107.5969508	1	4567634	0	25433681
3	25433688	-6.9039393	107.5963723	2	4625182	0	29376826
4	25433690	-6.9052824	107.5961768	2	4625182	1	364364242
5	25433685	-6.9049404	107.5975738	2	4625182	2	29376827
6	25433678	-6.9039784	107.5985467	3	4627111	0	29356177
7	25433679	-6.9049265	107.5985843	3	4627111	1	29356178
8	25433680	-6.9062500	107.5995945	3	4627111	2	29356179
9	25433681	-6.9055235	107.5989193	3	4627111	3	29356180
10	25434115	-6.9097812	107.5978508	4	4628057	0	29392373
11	25500626	-6.9070329	107.6019401	4	4628057	1	29392374
12	28802396	-6.9055299	107.5976092	6	247058985	0	2325451442
13	29356177	-6.9102898	107.5994584	6	247058985	1	364364086
14	29356178	-6.9090157	107.5994925	6	247058985	2	364364087
15	29356374	-6.9081596	107.5987289	6	247058985	3	2325451578
16	29356381	-6.9082496	107.5977288	9	32388779	0	364364184
17	29356499	-6.9099650	107.5978505	9	32388779	1	364364194

Gambar 3.3: Parsing XML menggunakan Javascript

5 Pada Gambar 3.3 dapat dilihat terdiri dari dua tabel yang menunjukkan informasi node
 6 dan edge yang sudah dibaca. Tabel node menunjukkan atribut penting yang akan digunakan
 7 yaitu id node, *latitude*, dan *longitude*. Sedangkan tabel edge menunjukkan informasi yang
 8 didapatkan dari tag way yang sudah dilakukan *filter*, yaitu hanya tag way yang memiliki
 9 *child* highway saja yang akan digunakan. Pada tabel edge terdapat informasi penting yang
 10 akan digunakan, yaitu id way, id node pertama, dan id node kedua. Informasi yang sudah
 11 didapatkan, selanjutnya akan dimodelkan ke dalam bentuk graf yang akan dibahas pada
 12 subbab 3.4.

3.3 Menghitung Jarak Antara Dua Titik

15 Untuk menghitung jarak antara dua titik dapat menggunakan rumus haversine atau dike-
 16 nal dengan haversine *formula*. Analisis rumus haversine dilakukan dengan implementasi
 17 rumus tersebut dengan contoh kasus perhitungan jarak antara koordinat Kota Bandung (-
 18 6.9167,107.6000) dan koordinat Kota Jakarta (-6.1745,106.8227). Berikut ini adalah rumus
 19 haversine yang telah diimplementasikan:

```

20 function getDistance(lat1,lon1,lat2,lon2) {
21     var R = 6371;
22     var dLat = deg2rad(lat2-lat1);

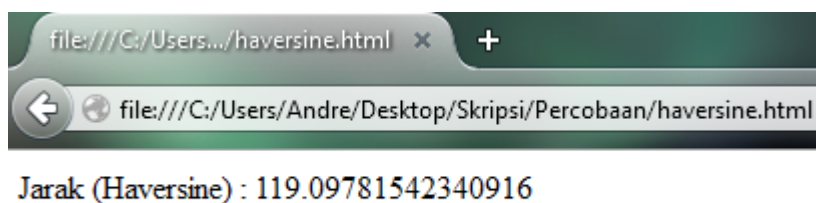
```

```

1   var dLon = deg2rad(lon2-lon1);
2   var a =
3     Math.sin(dLat/2) * Math.sin(dLat/2) +
4     Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
5     Math.sin(dLon/2) * Math.sin(dLon/2)
6     ;
7   var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
8   var d = R * c;
9   return d;
10  }
11
12  function deg2rad(deg) {
13    return deg * (Math.PI/180)
14  }

```

Hasil yang ditunjukkan dari rumus haversine adalah 119.09781542340916 km dapat dilihat pada Gambar 3.4.



Gambar 3.4: Perhitungan Jarak Dengan Haversine Formula

Saat proses analisis, ternyata Google Maps Javascript API menyediakan suatu *library* untuk mengukur jarak antara dua titik. *Library* tersebut adalah *geometry library* yang menyediakan fungsi utilitas yaitu `google.maps.geometry.spherical`. Untuk menghitung jarak antara dua titik digunakan fungsi `computeDistanceBetween()`. Sama seperti rumus haversine, analisis fungsi `computeDistanceBetween()` dilakukan dengan implementasi contoh kasus perhitungan jarak antara koordinat Kota Bandung dan Jakarta. Berikut ini adalah fungsi `computeDistanceBetween()` yang telah diimplementasikan:

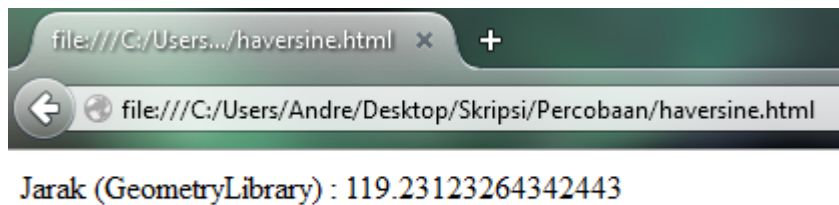
```

24  var jakarta = new google.maps.LatLng(-6.1745,106.8227);
25  var bandung = new google.maps.LatLng(-6.9167,107.6000);
26  var distance = google.maps.geometry.spherical.
27  computeDistanceBetween(jakarta, bandung);

```

Hasil yang ditunjukkan dari fungsi `computeDistanceBetween()` adalah 119.23123264342443 km dapat dilihat pada Gambar 3.5.

Terdapat perbedaan atau selisih yang dihasilkan oleh rumus haversine dan fungsi `computeDistanceBetween()` sebesar 0.133417220015275 km. Fungsi `computeDistanceBetween()` yang akan digunakan untuk pembuatan aplikasi, bukan rumus haversine. Hal ini karena fungsi tersebut lebih mudah digunakan.



Gambar 3.5: Perhitungan Jarak Dengan Geometry Library

1 3.4 Pemodelan OSMXML Menjadi Graf

2 Pada subbab 3.2, dokumen OSMXML sudah dibaca dan tahap selanjutnya adalah memo-
 3 delkan data OSMXML tersebut ke dalam bentuk graf. Seperti yang sudah diketahui, graf
 4 terdiri dari node dan edge. Informasi node dan edge yang sudah didapatkan akan dimo-
 5 delkan ke dalam bentuk graf berarah, hal ini karena jalan yang menghubungkan node-node
 6 tersebut memiliki arah, baik searah maupun dua arah, arah jalan diketahui dengan melihat
 7 informasi tag oneway. Untuk merepresentasikan graf tersebut akan digunakan *adjacency list*.
 8 Berikut ini adalah langkah-langkah yang dilakukan untuk memodelkan OSMXML menjadi
 9 graf:

- 10 1. Membuat kelas Node dan kelas Neighbor.

```

11     function Node(id, neighbors){
12         this.id = id;
13         this.adjList = neighbors;
14     }
15
16     function Neighbor(vnum, nbr, weight){
17         this.vertexNum = vnum;
18         this.weight = weight;
19         this.next = nbr;
20     }
  
```

21 Kedua kelas tersebut digunakan untuk menyimpan informasi id node dan jarak.

- 22 2. Membaca seluruh informasi node dan menyimpan pada kelas node.

```

23     for(v=0; v < node.length; v++){
24         adjLists.push(new Node(node[v].getAttribute('id'),null));
25     }
  
```

- 26 3. Membuat fungsi untuk menentukan arah pada setiap node yang terhubung dengan
 27 node lain.

```

28     function wayDirection(way,index){
29         var tag = way[index].getElementsByTagName("tag");
30         for (hg=0;hg<tag.length;hg++)
  
```

```

1      {
2          if(tag[hg].getAttribute('k') == "oneway"){
3              return tag[hg].getAttribute('v');
4          }
5      }
6      return false;
7  }

```

8 4. Membuat fungsi untuk mendapatkan informasi koordinat node (*latitude* dan *longitu-*
9 *de*).

```

10 function getLatByAtt(str)
11 {
12     for (n=0;n<node.length;n++)
13     {
14         if(node[n].getAttribute('id') == str)
15         {
16             return node[n].getAttribute('lat');
17         }
18     }
19 }
20
21 function getLonByAtt(str)
22 {
23     for (m=0;m<node.length;m++)
24     {
25         if(node[m].getAttribute('id') == str)
26         {
27             return node[m].getAttribute('lon');
28         }
29     }
30 }

```

31 Fungsi tersebut digunakan sebagai parameter *input* untuk mencari jarak dari satu
32 node ke node lain.

33 5. Membaca seluruh informasi edge yang terdapat pada tag way

```

34 for (i=0;i<way.length;i++)
35 {
36     nd = way[i].getElementsByTagName("nd");
37     if(isHighway(way,i)){
38         oneway = wayDirection(way,i);
39         for (j=0;j<nd.length-1;j++)
40         {
41             v1 = indexForId(adjLists,nd[j].getAttribute('ref'));

```

```

1      v2 = indexForId(adjLists,nd[j+1].getAttribute('ref'));
2
3      lat1 = getLatByAtt(nd[j].getAttribute('ref'));
4      lon1 = getLonByAtt(nd[j].getAttribute('ref'));
5      lat2 = getLatByAtt(nd[j+1].getAttribute('ref'));
6      lon2 = getLonByAtt(nd[j+1].getAttribute('ref'));
7
8      point1 = new google.maps.LatLng(lat1,lon1);
9      point2 = new google.maps.LatLng(lat2,lon2);
10     distance = google.maps.geometry.spherical.
11     computeDistanceBetween(point1,point2);
12
13     if(oneway == "yes"){
14         adjLists[v1].adjList = new Neighbor(v2,
15         adjLists[v1].adjList,distance);
16     }else if(oneway == "no"){
17         adjLists[v1].adjList = new Neighbor(v2,
18         adjLists[v1].adjList,distance);
19         adjLists[v2].adjList = new Neighbor(v1,
20         adjLists[v2].adjList,distance);
21     }else if(oneway == "-1"){
22         adjLists[v2].adjList = new Neighbor(v1,
23         adjLists[v2].adjList,distance);
24     }else{
25         adjLists[v1].adjList = new Neighbor(v2,
26         adjLists[v1].adjList,distance);
27         adjLists[v2].adjList = new Neighbor(v1,
28         adjLists[v2].adjList,distance);
29     }
30 }
31 }
32 }

```

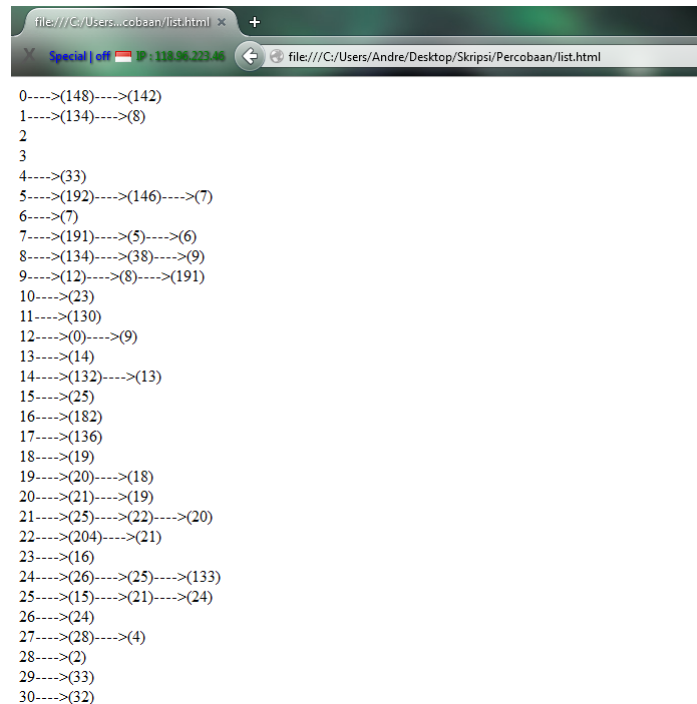
33 6. Membuat fungsi untuk menampilkan graf ke layar

```

34 function print(list){
35     for (j=0; j < list.length; j++){
36         document.write(j);
37         for (nbr=list[j].adjList; nbr != null;nbr=nbr.next) {
38             document.write("---->");
39             document.write('(' +nbr.vertexNum+')');
40         }
41         document.write("<br>");
42     }
43 }

```

Hasil dari kode program di atas dapat dilihat pada Gambar 3.6. Pada Gambar 3.6, data pada



Gambar 3.6: Pemodelan OSMXML menjadi Graf

- 1
- 2 OSMXML sudah dimodelkan ke dalam bentuk graf menggunakan representasi *adjacency list*.
- 3 Graf tersebut selanjutnya akan divisualisasikan menggunakan Google Maps Javascript API
- 4 yang akan dibahas pada subbab 3.5.

5 3.5 Visualisasi Graf

6 Data OSMXML yang sudah dimodelkan ke dalam bentuk graf, selanjutnya akan divisualisa-
 7 sikan menggunakan Google Maps Javascript API. Peta yang akan ditampilkan dalam bentuk
 8 *roadmap*, karena peta jenis ini memberikan informasi mengenai nama jalan, sehingga peta
 9 jenis ini lebih cocok untuk aplikasi yang akan dibangun. Berikut ini langkah-langkah yang
 10 dilakukan untuk membuat visualisasi graf:

- 11 1. Melakukan *load* Google Maps Javascript API.

```
12 <script src="https://maps.googleapis.com/maps/api
13 /js?v=3.exp&libraries=geometry"></script>
```

- 14 2. Membuat elemen div sebagai wadah atau tempat untuk peta.

```
15 <div id="googleMap" style="width:75%;height:600px;float:left"></div>
```

- 16 3. Membuat kelas `google.maps.Map` dengan parameter *input map options* (variabel `map-`
 17 `Prop`). Peta disisipkan pada elemen div yang memiliki id `googleMap`.

```
18 var mapProp = {
19   center:new google.maps.LatLng(-6.906845432118958,107.59851515293121),
```

```

1      zoom:17,
2      mapTypeId:google.maps.MapTypeId.ROADMAP
3  };
4
5  var map=new google.maps.Map(document.getElementById("googleMap"),mapProp);

```

6 4. Membuat objek *marker* untuk setiap node pada peta.

```

7  for (i=0;i<way.length;i++)
8  {
9      nd = way[i].getElementsByTagName("nd");
10     if(isHighway(way,i))
11     {
12         id = uniqueId();
13         marker = new google.maps.Marker({
14             id: id,
15             position: new google.maps.LatLng(getLatByAtt(nd[0].getAttribute('ref')),
16                 getLonByAtt(nd[0].getAttribute('ref'))),
17             map: map,
18             icon: image,
19         });
20         markers[id] = marker;
21         for (j=0;j<nd.length-1;j++)
22         {
23             id = uniqueId();
24             marker = new google.maps.Marker({
25                 id: id,
26                 position: new
27                     google.maps.LatLng(getLatByAtt(nd[j+1].getAttribute('ref')),
28                         getLonByAtt(nd[j+1].getAttribute('ref'))),
29                 map: map,
30                 icon: image,
31             });
32             markers[id] = marker;

```

33 5. Membuat objek *polyline* yang menghubungkan setiap node pada peta.

```

34     line = new google.maps.Polyline({
35         path: [new google.maps.LatLng(getLatByAtt(nd[j].getAttribute('ref')),
36             getLonByAtt(nd[j].getAttribute('ref'))), new
37             google.maps.LatLng(getLatByAtt(nd[j+1].getAttribute('ref')),
38                 getLonByAtt(nd[j+1].getAttribute('ref')))],
39         strokeColor: "#000000",
40         strokeOpacity: 1.0,
41         strokeWeight: 3, map: map

```



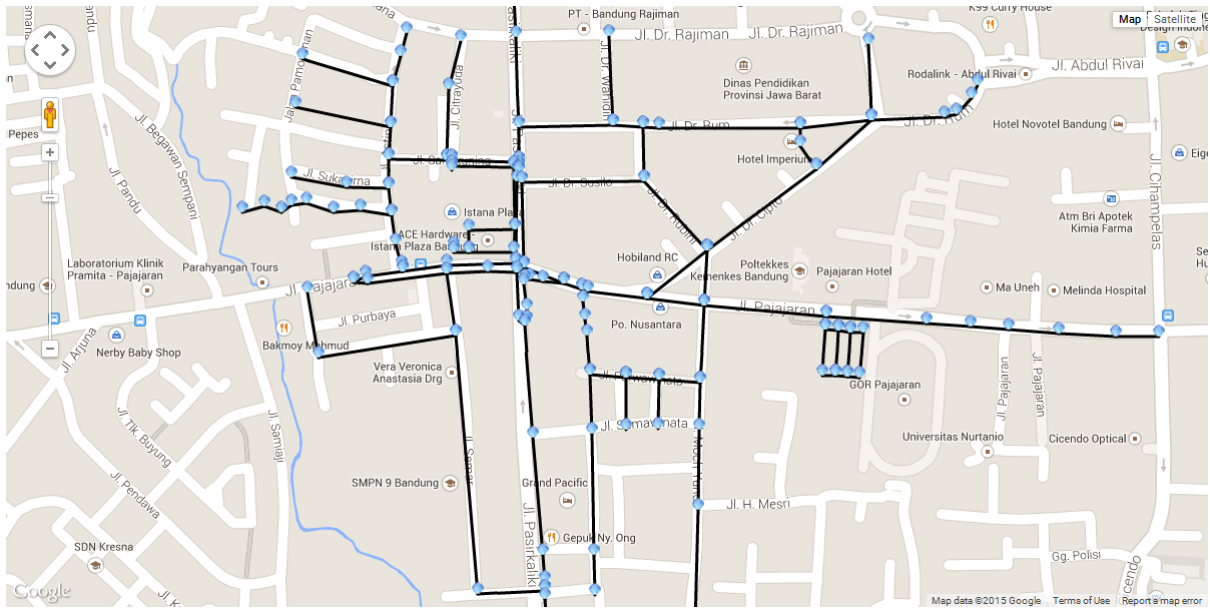
```
1         });  
2     }  
3 }  
4 }
```

- 5 6. Membuat fungsi untuk menambahkan *info window* pada setiap node untuk membe-
6 rikan informasi id dan index node.

```
7     function addInfoWindow(marker, message) {  
8         var infoWindow = new google.maps.InfoWindow({  
9             content: message  
10        });  
11  
12        google.maps.event.addDomListener(marker, 'click', function () {  
13            infoWindow.open(map, marker);  
14        });  
15    }
```

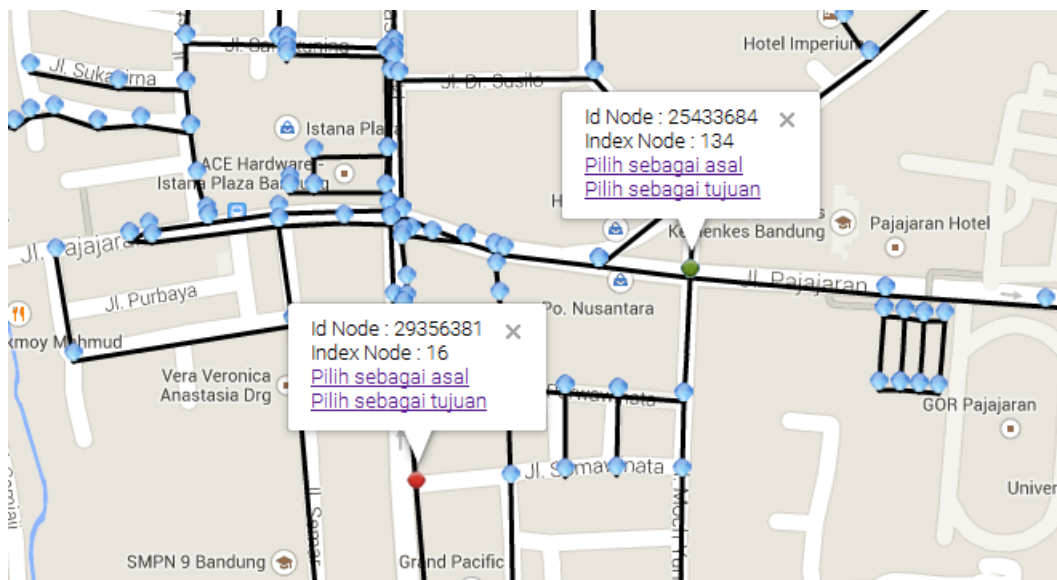
- 16 7. Membuat fungsi untuk menandai *marker* sebagai titik asal atau titik tujuan.

```
17     function setAsal(id,idnode){  
18         if(!isAsalClicked){  
19             markers[id].setIcon('icon/dot_green.png');  
20             isAsalClicked = true;  
21             asal = id;  
22         }else{  
23             markers[asal].setIcon('icon/dot_blue.png');  
24             markers[id].setIcon('icon/dot_green.png');  
25             asal = id;  
26         }  
27         return asal;  
28     }  
29  
30     function setTujuan(id,idnode){  
31         if(!isTujuanClicked){  
32             markers[id].setIcon('icon/dot_red.png');  
33             isTujuanClicked = true;  
34             tujuan = id;  
35         }else{  
36             markers[tujuan].setIcon('icon/dot_blue.png');  
37             markers[id].setIcon('icon/dot_red.png');  
38             tujuan = id;  
39         }  
40         return tujuan;  
41     }
```



Gambar 3.7: Visualisasi Graf

- 1 Hasil dari langkah-langkah di atas dapat dilihat pada Gambar 3.7.
 - 2 Setiap node pada graf yang sudah dilakukan *filter* akan diwakili oleh marker. Setiap
 - 3 marker tersebut akan memiliki *info window* yang akan memberikan informasi seperti id
 - 4 node, index node pada graf, dan dua buah *hyperlink* yang berfungsi untuk menjadikan
 - 5 marker yang dipilih menjadi asal atau tujuan. Contoh *info window* yang ditampilkan pada
 - 6 peta dapat dilihat pada Gambar 3.8. Setiap edge pada graf akan menjadi garis pada peta
- yang dibuat menggunakan *polyline*.



Gambar 3.8: Info Window

7

8 3.6 Algoritma Dijkstra

- 9 Untuk pencarian rute terdekat, aplikasi menggunakan algoritma dijkstra. Algoritma terse-
- 10 but diimplementasikan pada graf yang sudah dimodelkan sebelumnya. Berikut ini adalah

1 langkah-langkah algoritma dijkstra yang digunakan:

2 1. Membuat kelas Dijkstra dan inisialisasi variabel.

```
3     function Dijkstra(){
4         var INFINITY = 1/0;
5         this.vertices = {};
6     }
```

7 2. Membuat fungsi untuk menambahkan node.

```
8     this.addVertex = function(name,edges){
9         this.vertices[name] = edges;
10    }
```

11 3. Implementasi algoritma dijkstra

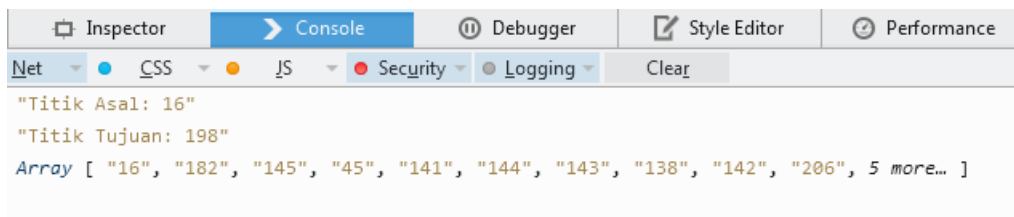
```
12    this.shortestPath = function(asal, tujuan){
13        var nodes = new PriorityQueue();
14        var distances = {};
15        var previous = {};
16        var path = [];
17        var smallest, vertex, neighbor, alt;
18
19        for(vertex in this.vertices){
20            if(vertex === asal){
21                distances[vertex] = 0;
22                nodes.enqueue(0, vertex);
23            }
24            else{
25                distances[vertex] = INFINITY;
26                nodes.enqueue(INFINITY, vertex);
27            }
28            previous[vertex] = null;
29        }
30
31        while(!nodes.isEmpty()){
32            smallest = nodes.dequeue();
33            if(smallest === tujuan){
34                while(previous[smallest]){
35                    path.push(smallest);
36                    smallest = previous[smallest];
37                }
38                break;
39            }
40            if(!smallest || distances[smallest] === INFINITY){
```

```

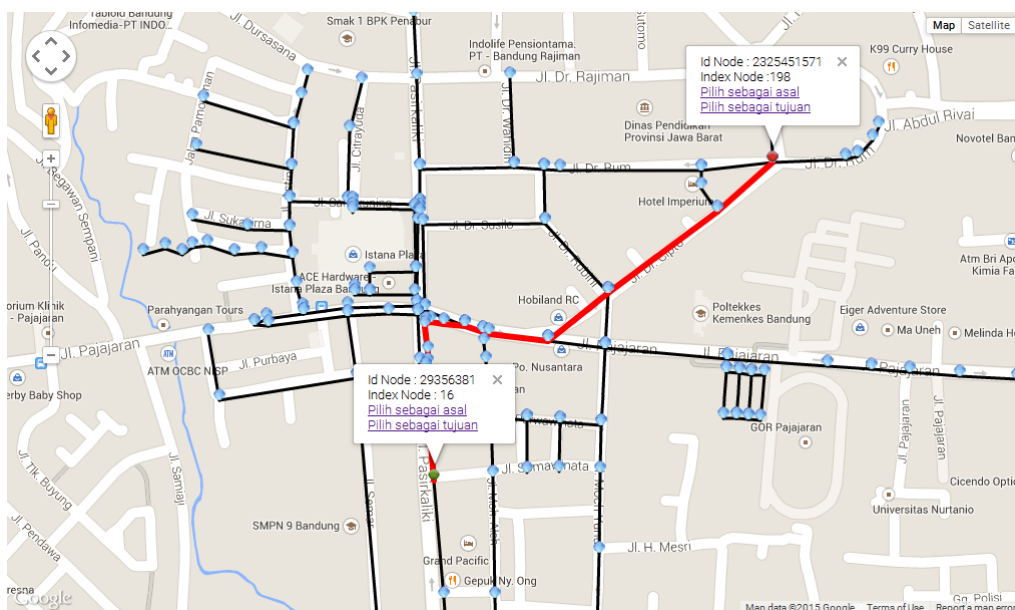
1      continue;
2  }
3  for(neighbor in this.vertices[smallest]){
4      alt = distances[smallest] + this.vertices[smallest][neighbor];
5      if(alt < distances[neighbor]) {
6          distances[neighbor] = alt;
7          previous[neighbor] = smallest;
8          nodes.enqueue(alt, neighbor);
9      }
10 }
11 }
12 return path;

```

13 Fungsi dijkstra akan menerima *input* berupa objek “edge” yang berisi informasi dari graf,
 14 selanjutnya fungsi akan mengeluarkan *output* yaitu jalur terpendek dari satu titik ke titik
 15 lain dalam bentuk array. Contoh kasus yang digunakan, yaitu mencari rute terdekat dari
 16 titik asal “16” dan titik tujuan “198”, output yang dihasilkan dapat dilihat pada Gambar 3.9.
 Visualisasi rute terdekat dapat dilihat pada Gambar 3.10.



Gambar 3.9: Keluaran fungsi Dijkstra



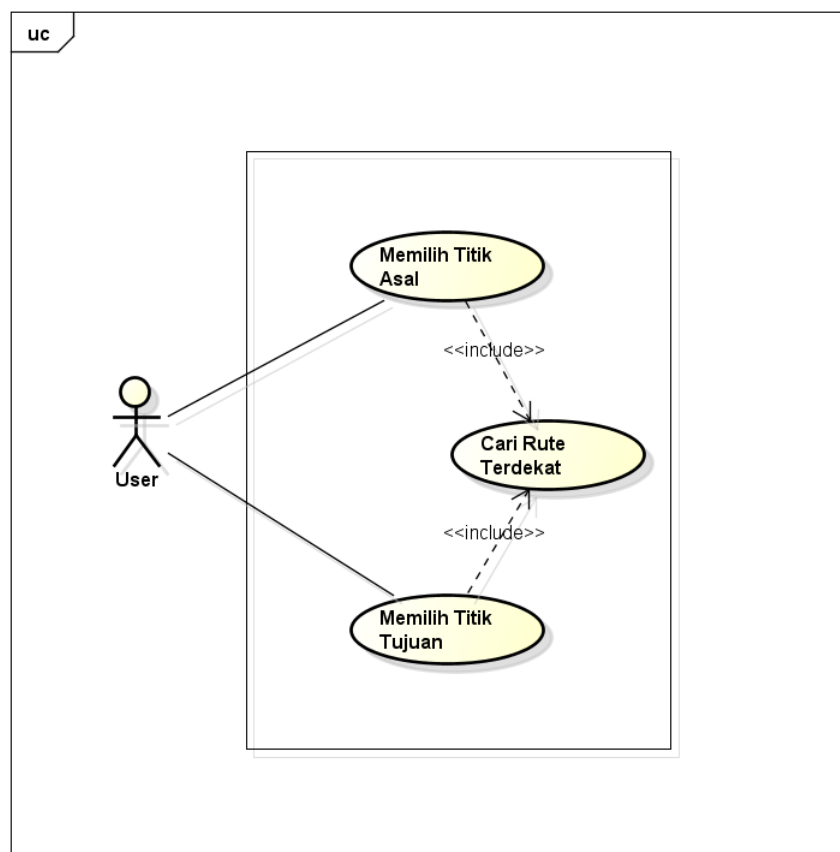
Gambar 3.10: Visualisasi Rute Terdekat

3.7 Analisis Berorientasi Objek

Aplikasi pencarian rute terdekat yang dibangun akan mengolah data yang disediakan oleh OpenStreetMap dalam bentuk XML dan memodelkannya ke dalam bentuk graf. *User* dapat memilih titik asal dan titik tujuan, selanjutnya akan diimplementasikan algoritma Dijkstra untuk mencari rute terdekat antara kedua titik tersebut dan menunjukkan hasilnya secara visual menggunakan Google Maps Javascript API. Pada subbab ini akan memperjelas interaksi antara *user* dengan sistem yaitu menggunakan diagram *use case* dan skenario. Dan juga diagram kelas untuk menunjukkan kelas-kelas yang ada pada sistem dan hubungannya.

3.7.1 Diagram *Use Case*

Diagram *use case* adalah pemodelan yang berfungsi memperjelas interaksi antara aktor atau *user* dengan sistem. Diagram *use case* aplikasi pencarian rute terdekat dapat dilihat pada Gambar 3.11.



powered by Astah

Gambar 3.11: Diagram *use case*

12

13 Berdasarkan analisis yang telah dilakukan, maka *user* dapat melakukan interaksi sebagai
14 berikut:

15 1. Memilih titik asal.

16 *User* dapat menekan salah satu *marker* yang ada pada peta dan menekan *link* “pilih
17 titik asal”. Selanjutnya, akan ditampilkan informasi titik yang sudah dipilih pada sisi
18 kanan layar.

2. Memilih titik tujuan.

User dapat menekan salah satu *marker* yang ada pada peta dan menekan *link* “pilih titik tujuan”. Selanjutnya, akan ditampilkan informasi titik yang sudah dipilih pada sisi kanan layar.

3. Mencari rute terdekat dari titik asal ke titik tujuan.

User dapat menekan tombol “Cari” untuk mencari rute terdekat dari kedua titik yang telah dipilih sebelumnya.

3.7.2 Skenario

Berikut ini adalah skenario untuk setiap *use case*:

1. Skenario memilih titik asal dapat dilihat pada Tabel 3.1.

Tabel 3.1: Skenario memilih titik asal

Nama	Memilih titik asal
Aktor	User
Kondisi Awal	Titik asal belum terpilih
Kondisi Akhir	Titik asal sudah terpilih dan ditampilkan di sisi kanan layar
Skenario	User menekan marker pada peta dan menekan link pilih titik asal
Deskripsi	User memilih titik pada peta untuk dijadikan titik asal
Eksepsi	-

2. Skenario memilih titik tujuan dapat dilihat pada Tabel 3.2.

Tabel 3.2: Skenario memilih titik tujuan

Nama	Memilih titik tujuan
Aktor	User
Kondisi Awal	Titik tujuan belum terpilih
Kondisi Akhir	Titik tujuan sudah terpilih dan ditampilkan di sisi kanan layar
Skenario	User menekan marker pada peta dan menekan link pilih titik tujuan
Deskripsi	User memilih titik pada peta untuk dijadikan titik tujuan
Eksepsi	-

3. Skenario cari rute terdekat dapat dilihat pada Tabel 3.3.

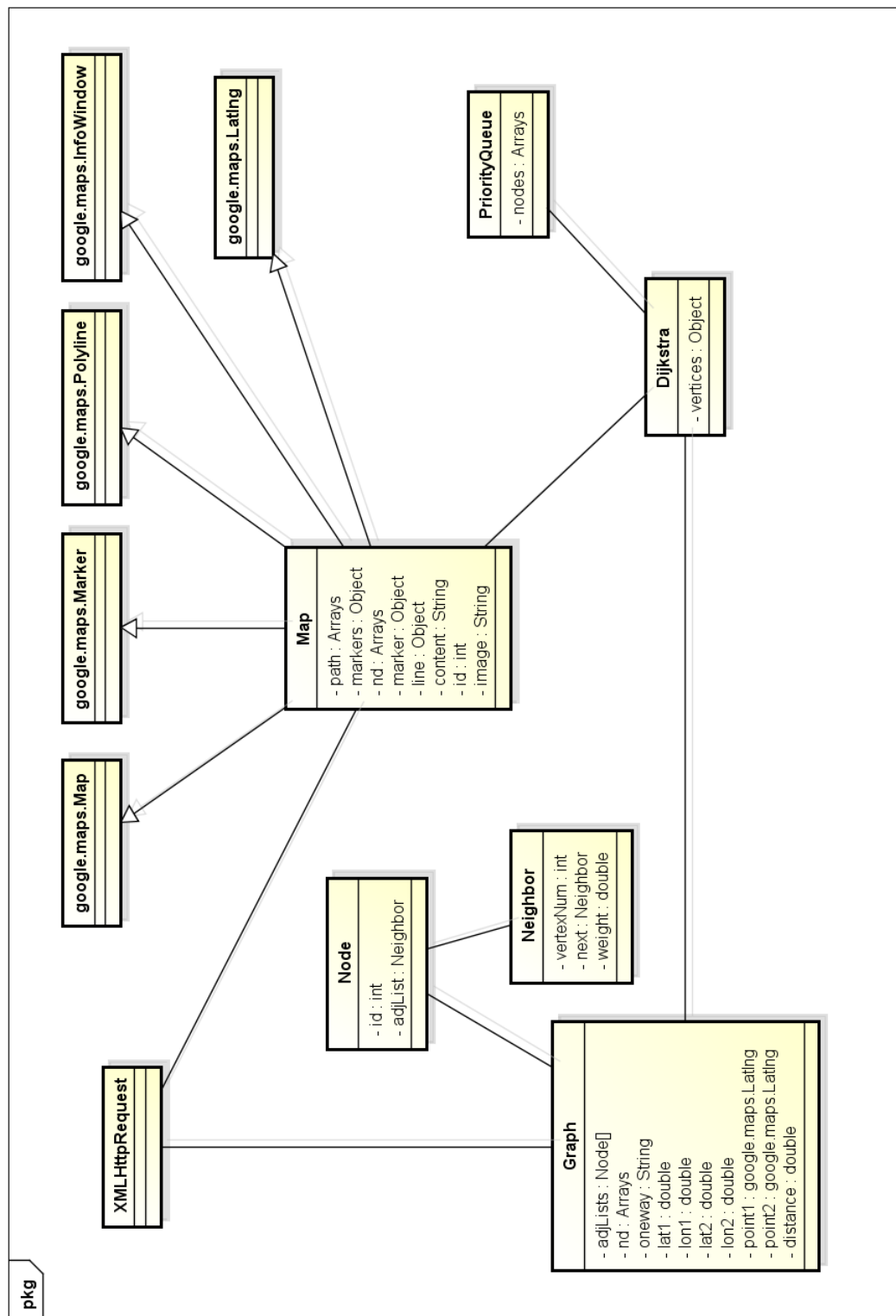
Tabel 3.3: Skenario cari rute terdekat

Nama	Cari rute terdekat
Aktor	User
Kondisi Awal	Titik asal dan titik tujuan sudah terpilih
Kondisi Akhir	Sistem menampilkan rute terdekat menggunakan polyline
Skenario	User menekan tombol Cari!
Deskripsi	User menekan tombol Cari! dan sistem menampilkan rute terdekat
Eksepsi	Jika user belum memilih titik asal atau tujuan akan ditampilkan alert atau peringatan

3.7.3 Diagram Kelas Sederhana

Pada bagian ini, akan dijelaskan diagram kelas yang digunakan untuk memenuhi kebutuhan *user* yang sudah dijelaskan pada bagian diagram *use case* dan skenario. Berikut ini adalah diagram kelas sederhana yang dapat dilihat pada Gambar 3.12. Berikut ini adalah penjelasan dari setiap kelas yang terdapat pada diagram kelas sederhana:

- Kelas XMLHttpRequest
Kelas ini berfungsi untuk melakukan *load* dokumen OSMXML.
- Kelas Node dan Neighbor
Kedua kelas ini akan menyimpan informasi yang didapatkan dari OSMXML sebagai representasi dari graf yaitu *adjacency list*.
- Kelas Graph
Kelas ini berfungsi untuk mengubah informasi yang didapatkan dari OSMXML menjadi graf.
- Kelas Map
Kelas ini berfungsi untuk melakukan *generate* peta, visualisasi graf, dan visualisasi rute terdekat.
- Kelas google.maps.Map
Kelas ini berfungsi untuk membuat objek peta.
- Kelas google.maps.Marker
Kelas ini berfungsi untuk membuat objek *marker* yang akan digunakan sebagai visualisasi node pada peta.
- Kelas google.maps.Polyline
Kelas ini berfungsi untuk membuat objek *polyline* yang akan digunakan sebagai visualisasi edge pada peta.
- Kelas google.maps.InfoWindow
Kelas ini berfungsi untuk membuat objek *InfoWindow* yang akan disisipkan pada setiap objek *marker*.
- Kelas google.maps.LatLng
Kelas ini berfungsi untuk membuat objek *LatLng*. *LatLng* merupakan objek yang berisi informasi koordinat (*latitude* dan *longitude*).
- Kelas Dijkstra
Kelas ini berfungsi untuk mencari rute terdekat berdasarkan *input* titik asal dan titik tujuan.
- Kelas PriorityQueue
Kelas ini merupakan struktur data *queue* yang digunakan pada kelas *dijkstra*.



Gambar 3.12: Diagram Kelas Sederhana

BAB 4

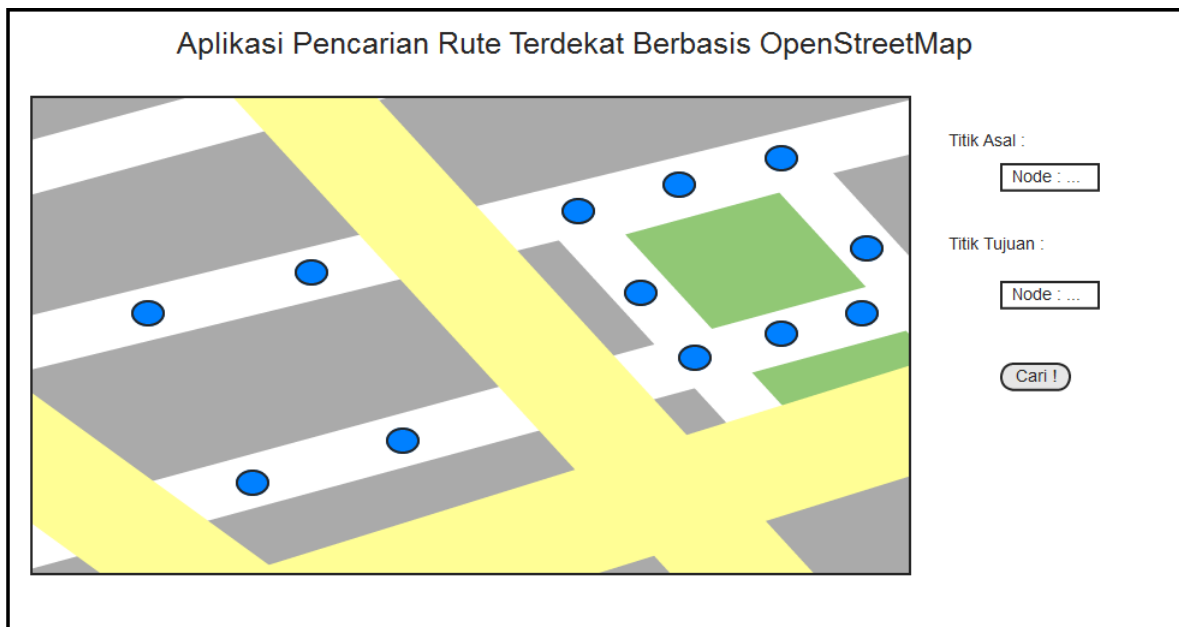
PERANCANGAN

Pada bab ini akan dijelaskan perancangan aplikasi pencarian rute terdekat berbasis OpenStreetMap. Pada bagian pertama akan dijelaskan perancangan antar muka untuk aplikasi yang akan dibangun. Diawali *user* dengan membuka aplikasi, selanjutnya *user* dapat melihat titik-titik yang terdapat pada peta sebagai titik asal ataupun titik tujuan. Dan mencari rute terdekat antara kedua titik tersebut.

Pada bagian kedua akan dijelaskan rancangan untuk aplikasi agar dapat menjalankan fungsinya melalui diagram kelas. Dan pada bagian ketiga akan dijelaskan bagaimana alur aplikasi dari *user* hingga mengeluarkan *output* yaitu rute terdekat menggunakan diagram *sequence*.

4.1 Perancangan Antar Muka

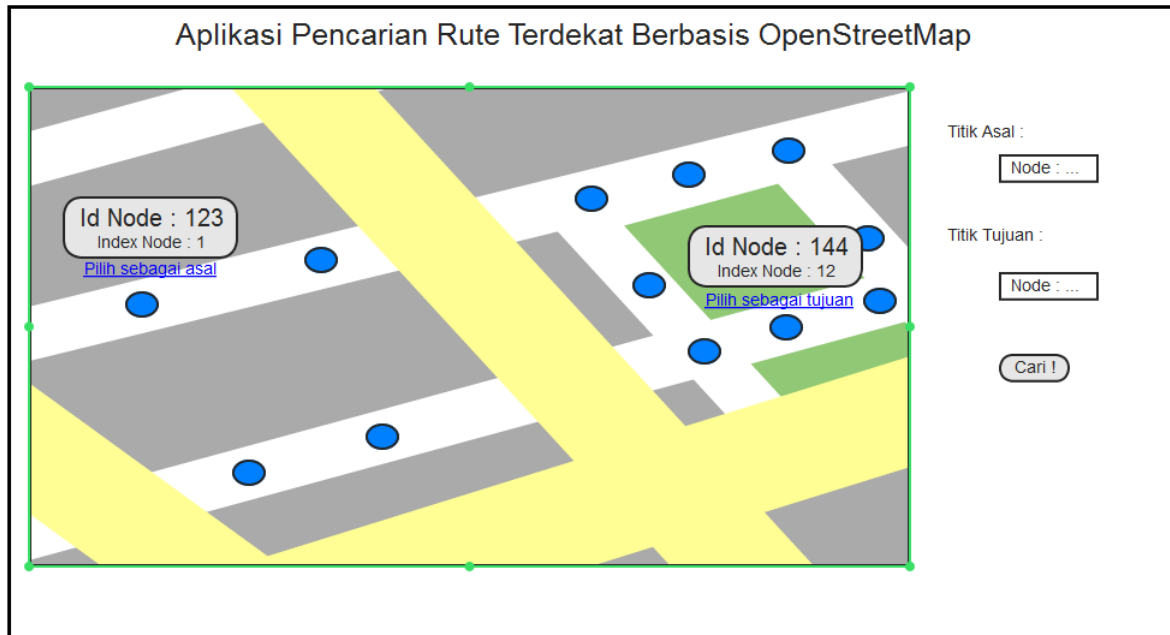
Pada subbab ini akan dijelaskan rancangan antar muka untuk aplikasi pencarian rute terdekat. Aplikasi yang dibangun akan memiliki tampilan awal seperti pada Gambar 4.1.



Gambar 4.1: Rancangan Antar Muka Awal Aplikasi

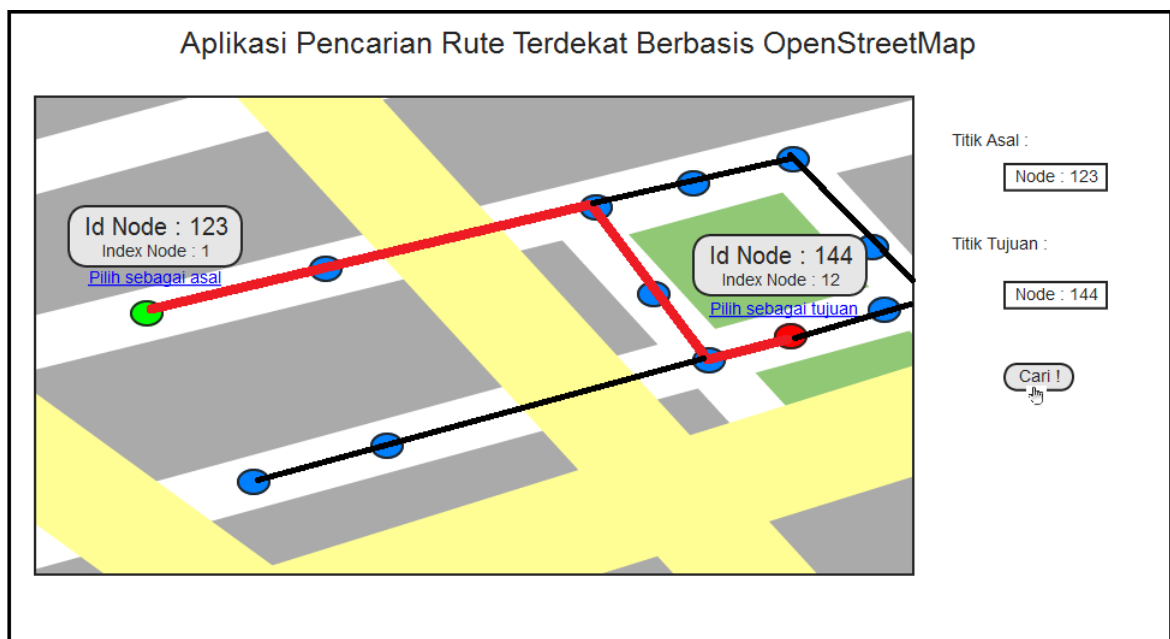
Setelah halaman awal terbuka, *User* dapat menekan setiap titik tersebut dan akan menampilkan *info window* yang memberikan informasi titik beserta *link*. *User* dapat menekan *link* tersebut untuk menjadikannya sebagai titik asal ataupun titik tujuan. Rancangan antar

muka saat user memilih titik, dapat dilihat pada Gambar 4.2.



Gambar 4.2: Rancangan Pemilihan Titik

- 1
- 2 Setelah user memilih titik asal dan titik tujuan, user dapat menekan tombol “Cari!”
- 3 untuk melihat rute terdekat antara kedua titik tersebut. Rute tersebut divisualisasikan
- 4 dengan menggunakan *polyline* yang berwarna merah. Rancangan pada tahap ini dapat dilihat pada Gambar 4.3.



Gambar 4.3: Rancangan Cari Rute

5

6 4.2 Perancangan Kelas

- 7 Pada bagian ini akan dibahas perancangan kelas yang dilakukan untuk aplikasi yang akan
- 8 dibangun. Perancangan kelas bertujuan untuk menjelaskan seluruh atribut dan fungsi yang

1 akan diimplementasikan pada setiap kelas yang sudah dibahas pada bab analisis. Peran-
2 cangan kelas yang dibuat dapat dilihat pada Gambar 4.4. Berikut ini adalah penjelasan
3 dari setiap kelas beserta atribut dan fungsi yang digunakan:

4 • Kelas XMLHttpRequest

5 Kelas ini berfungsi untuk melakukan *load* dokumen OSMXML.

6 – Fungsi

7 * open()

8 Fungsi open() digunakan untuk membuka dokumen OSMXML.

9 * send()

10 Fungsi send() digunakan untuk mengirimkan dokumen OSMXML yang sudah
11 dibuka.

12 • Kelas Node dan Neighbor

13 Kedua kelas ini akan menyimpan informasi yang didapatkan dari OSMXML sebagai
14 representasi dari graf yaitu *adjacency list*.

15 – Atribut Kelas Node

16 * id : int

17 Atribut id menyimpan id node yang didapatkan dari dokumen OSMXML.

18 * adjList : Neighbor

19 Atribut adjList menyimpan objek Neighbor.

20 – Atribut Kelas Neighbor

21 * vertexNum : int

22 Atribut vertexNum menyimpan index dari node.

23 * next : Neighbor

24 Atribut next menyimpan objek Neighbor.

25 * weight : double

26 Atribut weight menyimpan jarak antar node.

27 • Kelas Graph

28 Kelas ini berfungsi untuk mengubah informasi yang didapatkan dari OSMXML men-
29 jadi graf.

30 – Atribut

31 * adjLists : Array of Node

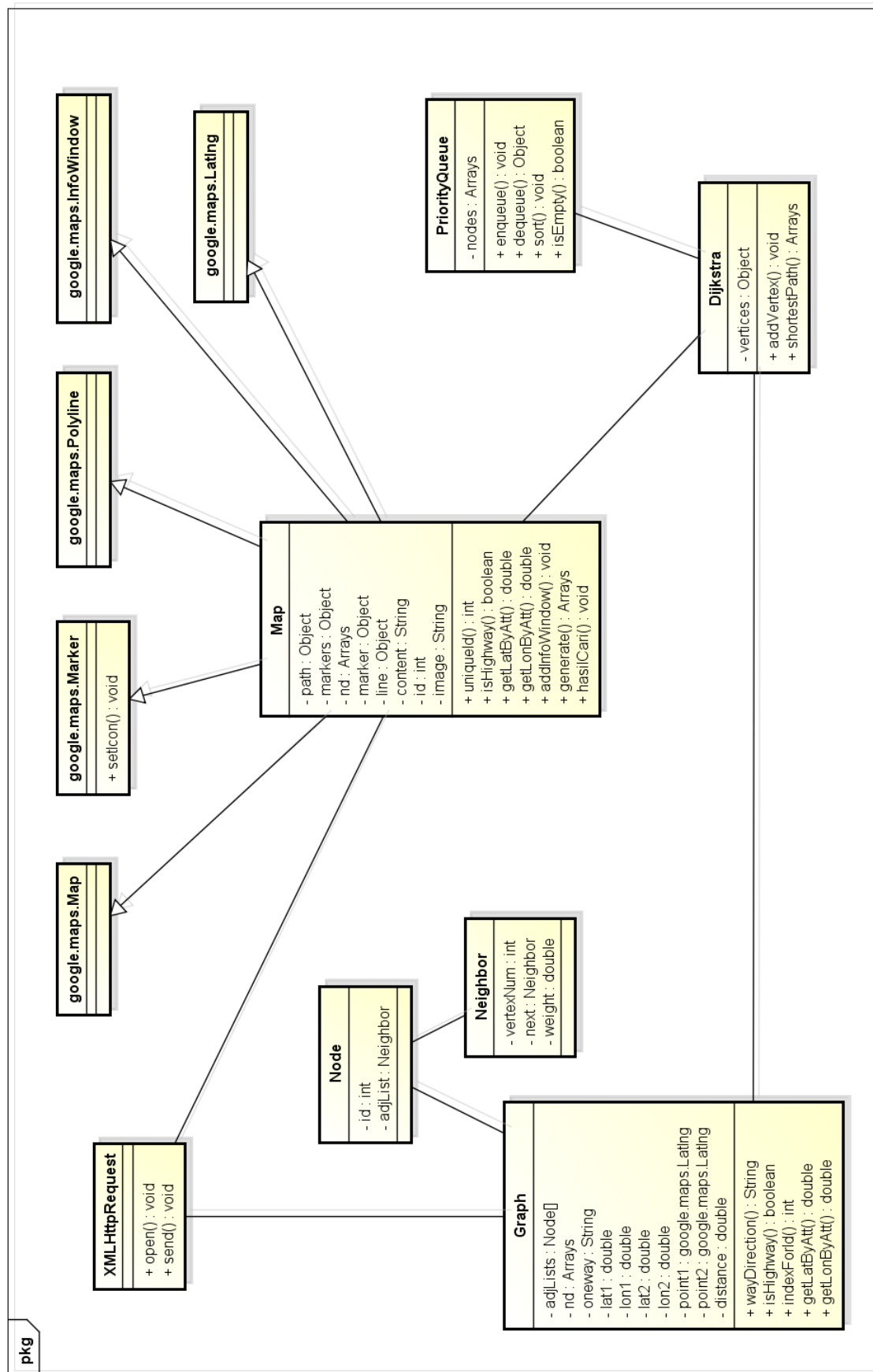
32 Atribut adjLists merupakan *array* yang menyimpan objek node.

33 * nd : Array

34 Atribut nd menyimpan informasi berupa id node yang terdapat pada tag way
35 di dalam dokumen OSMXML, atribut nd bertipe array.

36 * oneway : String

37 Atribut nd menyimpan informasi berupa *value* dari *key* oneway yang terdapat
38 pada tag way di dalam dokumen OSMXML, atribut oneway bertipe String.



Gambar 4.4: Diagram Kelas

```
1      * lat1 : double
2      Atribut lat1 menyimpan informasi latitude untuk point1.
3      * lon1 : double
4      Atribut lon1 menyimpan informasi longitude untuk point1.
5      * lat2 : double
6      Atribut lat2 menyimpan informasi latitude untuk point2.
7      * lon2 : double
8      Atribut lon2 menyimpan informasi longitude untuk point2.
9      * point1 : google.maps.LatLng
10     Atribut point1 untuk menyimpan informasi latitude dan longitude pada node
11     pertama, berupa objek google.maps.LatLng.
12     * point2 : google.maps.LatLng
13     Atribut point2 untuk menyimpan informasi latitude dan longitude pada node
14     kedua, berupa objek google.maps.LatLng.
15     * distance : double
16     Atribut distance untuk menyimpan jarak antara point1 dan point2.

17     – Fungsi
18     * wayDirection() : String
19     Fungsi ini digunakan untuk mengetahui arah pada setiap node berdasarkan
20     value yang terdapat pada OSMXML.
21     * isHighway() : boolean
22     Fungsi ini digunakan untuk melakukan filter pada tag way, yaitu hanya way
23     yang bertipe “Highway” saja yang akan digunakan pada pemodelan graf.
24     * indexForId() : int
25     Fungsi ini digunakan untuk mengetahui index node pada graf berdasarkan id
26     node.
27     * getLatByAtt() : double
28     Fungsi ini digunakan untuk mendapatkan informasi latitude pada sebuah
29     node berdasarkan atributnya, yaitu id node.
30     * getLonByAtt() : double
31     Fungsi ini digunakan untuk mendapatkan informasi longitude pada sebuah
32     node berdasarkan atributnya, yaitu id node.

33     • Kelas Map
34     Kelas ini berfungsi untuk melakukan generate peta, visualisasi graf, dan visualisasi
35     rute terdekat.

36     – Atribut
37     * path : Object
38     Atribut path adalah variabel yang menyimpan informasi titik koordinat da-
39     lam bentuk objek, atribut path digunakan untuk pembuatan marker dan
40     polyline.
```

- 1 * markers : Object
- 2 Atribut ini menyimpan seluruh objek *marker*.
- 3 * nd : Arrays
- 4 Atribut nd menyimpan informasi berupa id node yang terdapat pada tag way
- 5 di dalam dokumen OSMXML, atribut nd bertipe array.
- 6 * marker : Object
- 7 Atribut ini menyimpan objek *marker* untuk ditampilkan pada peta.
- 8 * line : Object
- 9 Atribut ini menyimpan objek *polyline* untuk ditampilkan pada peta.
- 10 * content : String
- 11 Atribut ini merupakan isi dari *info window* yang disisipkan pada setiap mar-
- 12 ker. Atribut ini berisi id node, index node, *link* titik asal, dan *link* titik
- 13 tujuan.
- 14 * id : int
- 15 Atribut ini merupakan id yang digunakan untuk melakukan *generate* id pada
- 16 fungsi *uniqueId*.
- 17 * image : String
- 18 Atribut ini menyimpan alamat dari *image* atau gambar yang digunakan oleh
- 19 *marker*.
- 20 – Fungsi
- 21 * *uniqueId()* : id
- 22 Fungsi ini digunakan untuk melakukan *generate* id, id tersebut digunakan
- 23 oleh *marker*.
- 24 * *isHighway()* : boolean
- 25 Fungsi ini digunakan untuk melakukan *filter* pada tag way, hanya way yang
- 26 bertipe “Highway” saja yang digunakan. Fungsi ini digunakan pada saat
- 27 pembuatan objek *marker* pada peta.
- 28 * *getLatByAtt()* : double
- 29 Fungsi ini digunakan untuk mendapatkan informasi *latitude* pada sebuah
- 30 node berdasarkan atributnya, yaitu id node.
- 31 * *getLonByAtt()* : double
- 32 Fungsi ini digunakan untuk mendapatkan informasi *longitude* pada sebuah
- 33 node berdasarkan atributnya, yaitu id node.
- 34 * *addInfoWindow()*
- 35 Fungsi ini digunakan untuk menambahkan objek *info window* pada setiap
- 36 *marker*.
- 37 * *generate()* : Arrays
- 38 Fungsi ini digunakan untuk menampilkan secara keseluruhan *marker* dan
- 39 *polyline* pada peta. Fungsi akan mengembalikan objek *marker* di dalam
- 40 bentuk array.
- 41 * *hasilCari()*
- 42 Fungsi ini digunakan untuk melakukan visualisasi rute terdekat menggunakan
- 43 *polyline*.

- 1 • Kelas `google.maps.Map`
2 Kelas ini berfungsi untuk membuat objek peta.
- 3 • Kelas `google.maps.Marker`
4 Kelas ini berfungsi untuk membuat objek *marker* yang akan digunakan sebagai visu-
5 alisasi node pada peta.
 - 6 – Fungsi
 - 7 * `setIcon()`
8 Fungsi ini digunakan untuk mengubah icon dari *marker*.
- 9 • Kelas `google.maps.Polyline`
10 Kelas ini berfungsi untuk membuat objek *polyline* yang akan digunakan sebagai visu-
11 alisasi edge pada peta.
- 12 • Kelas `google.maps.InfoWindow`
13 Kelas ini berfungsi untuk membuat objek *InfoWindow* yang akan disisipkan pada
14 setiap objek *marker*.
- 15 • Kelas `google.maps.LatLng`
16 Kelas ini berfungsi untuk membuat objek `LatLng`. `LatLng` merupakan objek yang berisi
17 informasi koordinat (*latitude* dan *longitude*).
- 18 • Kelas `Dijkstra`
19 Kelas ini berfungsi untuk mencari rute terdekat berdasarkan *input* titik asal dan titik
20 tujuan.
 - 21 – Atribut
 - 22 * `vertices : Object`
23 Atribut ini menyimpan informasi node dalam bentuk objek.
 - 24 – Fungsi
 - 25 * `addVertex()`
26 Fungsi ini digunakan untuk menambahkan satu node beserta edgenya.
 - 27 * `shortestPath() : Arrays`
28 Fungsi ini digunakan untuk pencarian rute terdekat, fungsi akan mengemba-
29 likan rute atau jalur terpendek dalam bentuk array.
- 30 • Kelas `PriorityQueue`
31 Kelas ini merupakan struktur data *queue* yang digunakan pada kelas `dijkstra`.
 - 32 – Atribut
 - 33 * `nodes : Array`
34 Atribut ini merupakan *queue* dari kelas `PriorityQueue`.
 - 35 – Fungsi
 - 36 * `enqueue()`
37 Fungsi ini digunakan untuk menambahkan objek ke dalam *queue*.

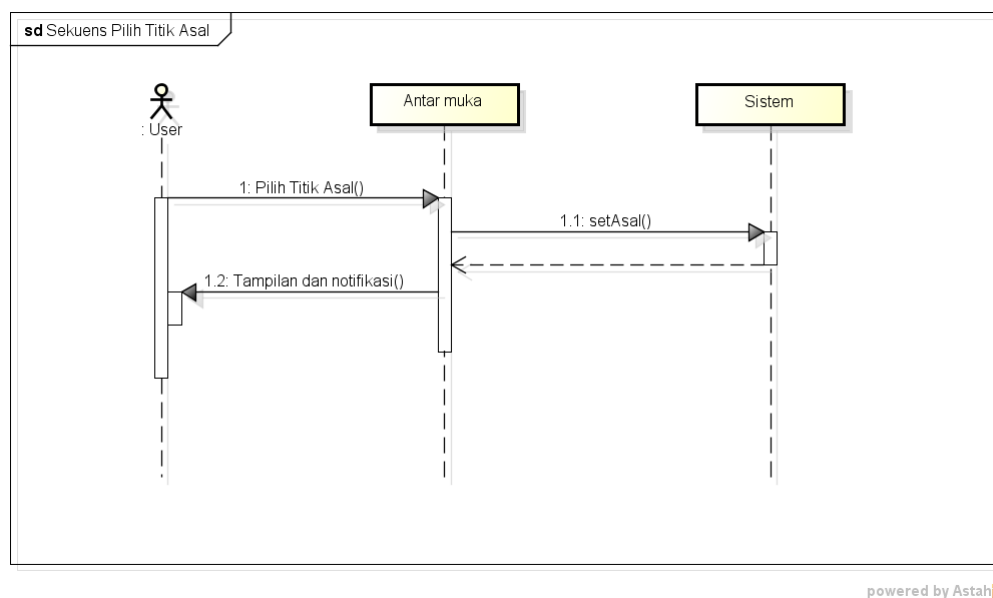
- 1 * dequeue() : Object
- 2 Fungsi ini digunakan untuk mengeluarkan objek dari dalam *queue*.
- 3 * sort()
- 4 Fungsi ini digunakan untuk menyusun *queue*.
- 5 * isEmpty() : boolean
- 6 Fungsi ini digunakan untuk pengecekan isi dari *queue*. jika *queue* kosong,
- 7 fungsi akan mengembalikan “true” dan sebaliknya “false” jika *queue* tidak
- 8 kosong.

9 4.3 Diagram Sekuens

10 Diagram sekuens adalah diagram yang digunakan untuk menggambarkan interaksi antara
 11 objek dengan waktu, interaksi tersebut digambarkan dengan grafik dua dimensi. Kedua
 12 dimensi tersebut adalah dimensi horizontal dan dimensi vertikal. Dimensi horizontal meng-
 13 gambarkan objek-objek yang berperan, sedangkan dimensi vertikal menggambarkan waktu.
 14 Pesan yang dikirimkan oleh suatu objek digambarkan dengan panah dan panah dengan garis
 15 putus-putus menggambarkan pesan balasan. Diagram sekuens pada bagian ini mengacu ke-
 16 pada diagram *use case* yang terdapat pada bab 3, dapat dilihat pada Gambar 3.11. Berikut
 17 ini adalah diagram sekuens berdasarkan diagram *use case* tersebut:

18 1. Pemilihan Titik Asal

Diagram sekuens untuk pemilihan titik asal dapat dilihat pada Gambar 4.5.



Gambar 4.5: Diagram Sekuens Pemilihan Titik Asal

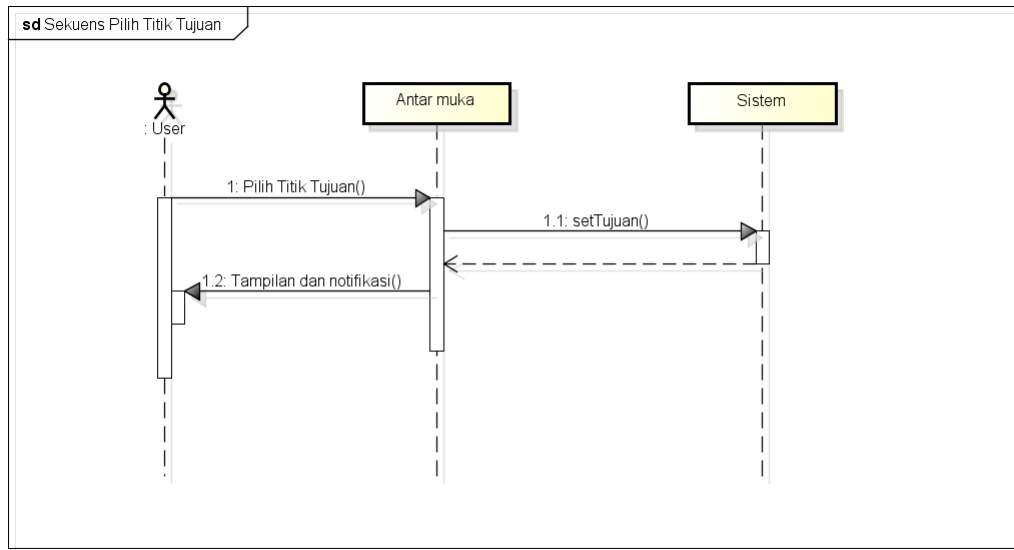
19

20 2. Pemilihan Titik Tujuan

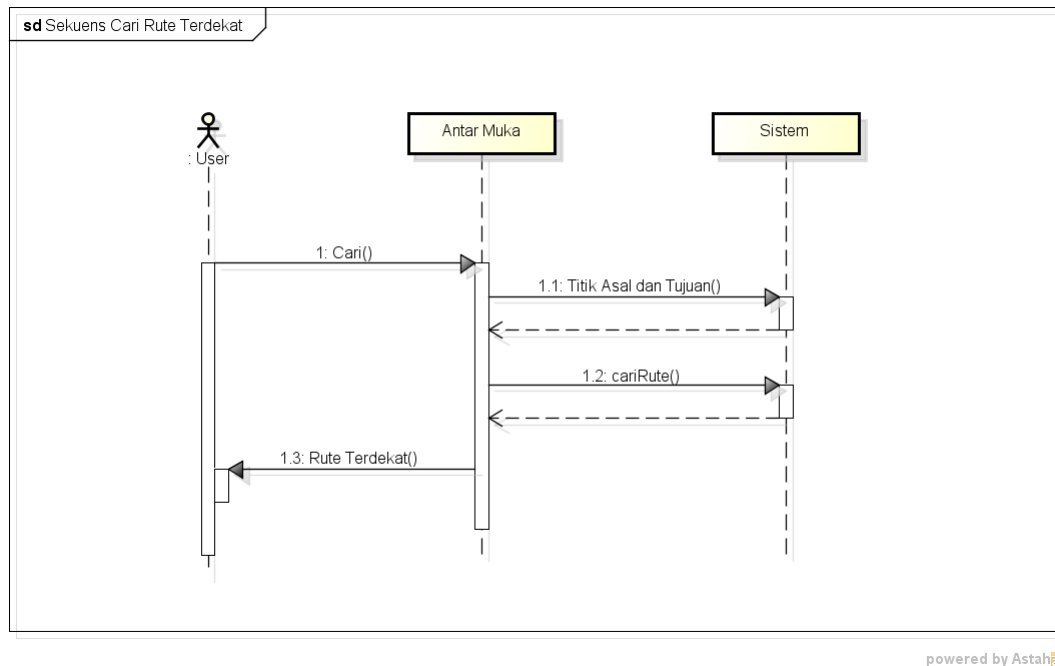
21 Diagram sekuens untuk pemilihan titik asal dapat dilihat pada Gambar 4.6.

22 3. Pencarian Rute Terdekat

23 Diagram sekuens untuk pencarian rute terdekat dapat dilihat pada Gambar 4.7.



Gambar 4.6: Diagram Sekuens Pemilihan Titik Tujuan



Gambar 4.7: Diagram Sekuens Pencarian Rute Terdekat

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Pada bagian ini akan dijelaskan tentang implementasi dan pengujian yang dilakukan. Implementasi adalah tahapan setelah proses perancangan selesai, rancangan yang sudah dibuat, diimplementasikan menggunakan kode program. Bahasa pemrograman yang digunakan adalah javascript. Setelah implementasi dilakukan, akan dilakukan juga beberapa pengujian. Pengujian dilakukan untuk mengetahui apakah fungsi-fungsi utama aplikasi sudah berjalan dengan baik dan juga untuk mengetahui kekurangan yang dimiliki aplikasi tersebut.

5.1 Implementasi

Implementasi dilakukan menggunakan bahasa pemrograman javascript. Berikut ini adalah lingkungan pembangunan aplikasi pada saat implementasi dilakukan:

1. Processor

Intel Core i5-2410M CPU @ 2.30Ghz (4 CPU)

2. Memory

4096MB RAM

3. Display

AMD Radeon HD 6470M

4. Operating System

Windows 7 Ultimate 64-bit

5. Browser

Mozilla Firefox Version 37.0 dan Google Chrome Version 41.0.2272.118 m

6. Bahasa Pemrograman

Javascript

7. Teks Editor

Notepad++

DAFTAR REFERENSI

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. MIT Press and McGrawHill, 2001.
- [2] OpenStreetMap, "OpenStreetMap Wiki." <http://wiki.openstreetmap.org/>, 2014. [Online; accessed 30-January-2015].
- [3] B. Marchal, *XML by Example*. John Pierce, 2000.
- [4] D. Flanagan, *JavaScript: The Definitive Guide, Sixth Edition*. O'Reilly Media, Inc, 2011.
- [5] E. Woychowsky, *Ajax: Creating Web Pages with Asynchronous JavaScript and XML*. Prentice Hall, 2006.
- [6] Google, "Google Maps JavaScript API v3." <https://developers.google.com/maps/documentation/javascript/>, 2015. [Online; accessed 31-January-2015].
- [7] N. R.Chopde and M. K. Nichat, "Landmark based shortest path detection by using a* and haversine formula," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, 2013.