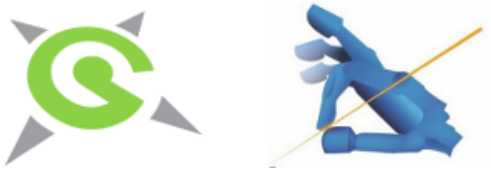
HAMK University of Applied Sciences
and Ramboll Group

Project Report

Transportation modeling and Web content

František Kolovský
Pierre-Vincent Vrot

Erasmus Exchange from
University of West Bohemia
and IMERIR



7th December 2015
version 0.1

Abstract

The Geographic Information Systems (GIS) allow, among others, to acquire, to treat, to organize and to present geographical data, to produce clear, accurate and intuitive maps, via an accessible web component from any browser.

The TraMap project, proposed by HAMK and supervised by Ramboll, is a simple and flexible tool that allows a user to view the transportation habits of information from open source geographic data and show informations in an application (web or mobile).

It in this context, of consultation tool and operating data, that our project takes place. Our role is to look for open data sources available on the Internet, to extract the information, to build an usual transport model and then develop a consultation multiplatform web application.

Keywords: GIS, Transport Modeling, Web, Algorithm, Open Sources

Acknowledgments

We are pleased to meet our gratitude to all those whose involvement in this project has promoted its outcome and all those involved, from near or far, to our training.

Initially, we would like to present our sincere thanks to Mr. Jari Mustajärvi, responsible for ICT training (Information Technology and Communication), for its welcome in HAMK and for allowing us to live a very rewarding experience.

Our biggest thanks go to Mr. Janne Rautio, our project supervisor, for his availability and encouragement throughout this project and whose valuable guidance and relevant advice was considerable support us in our efforts ; Ms. Taina Haapamaki to its guidelines, expertise and involvement in our project.

We also wish to extend our thanks to our respective schools, University of West Bohemia and IMERIR, they proposed to have this enriching Erasmus, and this in a pleasant complicity and respect.

Finally, all those who contributed directly or indirectly to the accomplishment of this work are the expression of our heartfelt thanks.

Contents

Introduction

In Finland, cycling, walking and public transport are increasingly used, it becomes necessary to bring the most useful information possible means in order to anticipate the movements.

In the current era, companies are always on the lookout for new products, particularly in the GIS domain, either issue new tools or technology. Each companies develop and market their traffic information tools. That is why our project is centered on the sharing of information and free access to sources and data.

Today, open source and open data sources of information are increasingly used and allow to evolve in cooperative community. Our main objective for this project is to recover the data and open source tools on which anyone could build or develop a data model of consultation on transportation habits of a city. Our research work will focus on the city of Hyvinkää and bicycle habits data.

The work is divided into three parts: the first is to determine the theoretical and transportation model data to be consumed; the second explains how an application can be developed for the retrieval of data; and the third and final section combines the previous parts in order to get a result viewed on different platforms.

1 Transportation Modeling

1.1 Transportation model

A transportation modelling is method for determining a traffic on road in focused area. Our goal is determine number of trip per road in road network from socio-economic and demographic data.

Traditional transportation modelling have a several independent step (e.g trip generation, trip destination, ...). Our implementation of transportation modelling consist from 3 steps.

trip generation - I this part we want to determine number of trip origin, destination in zones. This step is depend on data (e.g demographic data, socio-economic data, weather, local habits, ...). Quality of model depend a lot on these data.

trip destination - In this section we want determine *transportation matrix* T (OD matrix). It say how many people travel from zone i to zone j. For it was used Gravity model.

counting traffic - This is last step. We compute traffic for every road link (edge of graph).

In upcoming section following expressions will be used:

T	transportation matrix (number of trips between zones)
C	travel cost matrix (between zones)
T_i, T_j	sum value in row/column in T
n	number of zones (size of matrix)

1.1.1 Trip destination

For determine transportation matrix T was used Gravity model:

$$T_{ij} = K_i K_j T_i T_j f(C_{ij})$$

where

$$T_i = \sum_{j=1}^{n} T_{ij}$$

$$T_j = \sum_{i=1}^{n} T_{ij}$$

$$K_i = \frac{1}{\sum_j K_j T_j f(C_{ij})}$$

$$K_j = \frac{1}{\sum_i K_i T_i f(C_{ij})}$$

where in our case

$$f(x) = x^{-2}$$

T_i is number of trips outcoming from the zone (origin in the zone) i, T_j is number of trips incoming to the zone (destination in zone) j. So sometimes transportation matrix is called *OD Matrix*.

Now we must determine K_i and K_j. We used iterative proportional fitting. It is iterative solution. First we compute T^1 with $K_i, K_j = 1$. After we can use iteration equation for T:

$$T_{ij}^p = \frac{Z_i}{T_i^{m-1}} T_{ij}^{k-1}$$

$$T_{ij}^k = \frac{Z_j}{T_j^{m-1}} T_{ij}^p$$

where Z_i and Z_j are origin and destination trips (we know), k is iteration.

For example:

$$Z = \begin{pmatrix} 24 & 34 & 15 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 10 & 20 \\ 10 & 0 & 15 \\ 20 & 15 & 0 \end{pmatrix} \Rightarrow T^0 = \begin{pmatrix} 0 & 8.16000 & 0.90000 \\ 8.16000 & 0 & 2.26667 \\ 0.90000 & 2.26667 & 0 \end{pmatrix} \Rightarrow$$

$$T^0 = \begin{pmatrix} 0 & 8.16000 & 0.90000 \\ 8.16000 & 0 & 2.26667 \\ 0.90000 & 2.26667 & 0 \end{pmatrix} \Rightarrow T^1 = \begin{pmatrix} 0.00000 & 22.71648 & 3.65832 \\ 20.68579 & 0.00000 & 11.34168 \\ 3.31421 & 11.28352 & 0.00000 \end{pmatrix}$$

After 1 iteration coefficients are:

$$\frac{Z_i}{\sum_j T_{ij}} = \begin{pmatrix} 0.90996 \\ 1.06159 \\ 1.02756 \end{pmatrix} \quad \frac{Z_j}{\sum_i T_{ij}} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

1.1.2 Count traffic

Now we know how many people travel from zone i to zone j, so we can find path from i to j and record this value into every edges in path. So we have to compute z^2 shortest paths (z is number of zones). Because we have to compute shortest path between all pair of zones.

In our solution we compute N paths for every pair of zones (finally will be compute Nz^2 paths). Every path is based on another cost. Cost is based on length, time and vertical distance. Final cost is linear combination these partition cost.

$$c = \begin{pmatrix} k_t & k_l & k_h \end{pmatrix} \begin{pmatrix} t \\ l \\ h \end{pmatrix}$$

where c is cost, t is time, l is length and h is vertical distance. Number of trip (traffic between two zones) is splitted between those paths. In Fig. ?? you can see multipath (same pareto optimal paths) between two zones.

1.2 Implementation

In upcoming section following expressions will be used:

- n number of nodes
- m number of edges
- z number of zones
- N number of path computed for one pair of zone

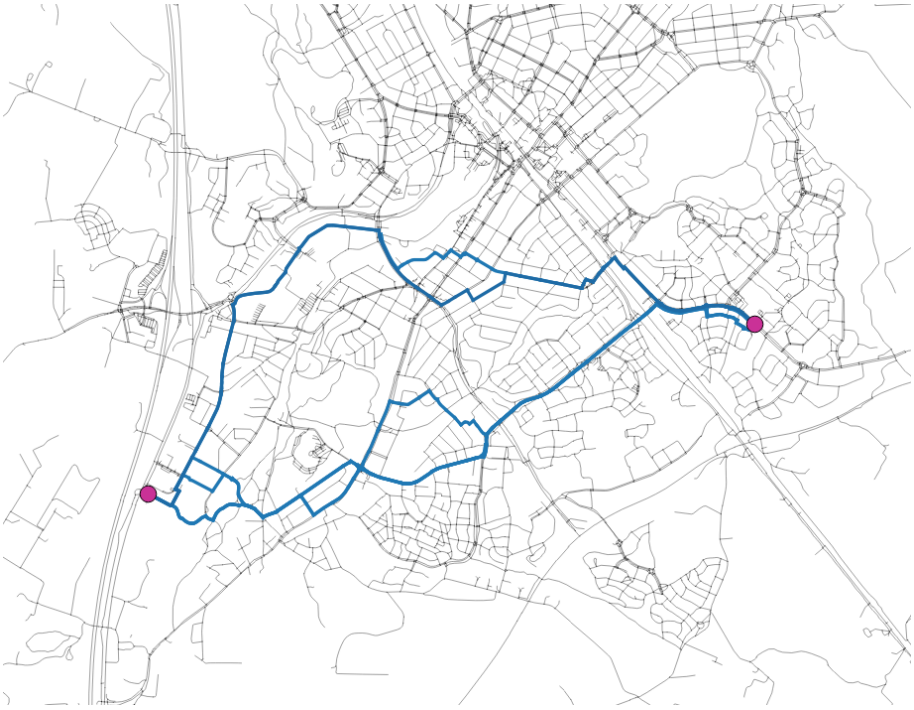


Figure 1.1: Possible paths between one pair of zone (multipath)

Transportation modelling described in previous section was implemented in Python programming language using NumPy library.

1.2.1 Static shortest path search

For determined C we use Dijkstra's algorithm (complexity $O(m + nlog(n))$). So final complexity is

$$O(z(m + nlog(n)))$$

For traffic count we used also Dijkstra's algorithm. Final complexity for traffic count is

$$O(Nz(m + nlog(n)))$$

For Dijkstra's algorithm we used Python library iGraph. iGraph is written in C programming language so it is quite fast. For example one Dijkstra running 30 ms ($n = 8000$ $m = 18000$).

1.2.2 Data store

All data for transportation modelling are stored in relation database PostgreSQL with extension PostGIS. PostGIS is spatial extension. Using it you can manipulate with line, point, polygon, raster.

In database there are 6 main table:

Table name	
roads	road links (edges)
nodes	nodes (vertexes)
zones	list of zones
traffic	
general_area_information	contains interested area geometry
od_pair	Database implementation for T matrix

In Fig 1.2 there are all table in database model and relationships between them (arrow is FOREIGN KEY).

More details about DB you can find in project documentation on GitHub.

1.2.3 Future work

This method of transportation modelling is very hard ($O(Nz(m+nlog(n)))$), so it is suitable compute this problem in some parallel environment (shared (Threads) or distributed memory (MapReduce or MPI)).

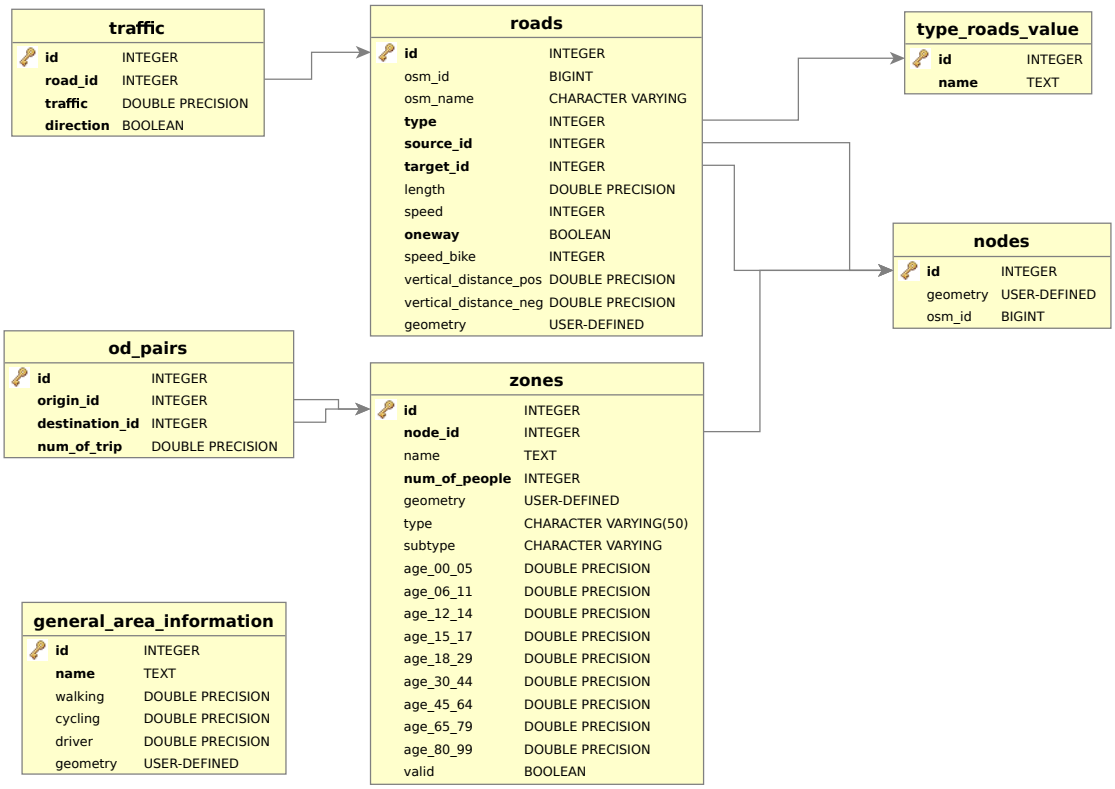


Figure 1.2: Database model

2 Web content

Before starting the development of the application, it is necessary to make a statement of needs and expectations. Thus, by structuring our interface will make it easier to conduct its graphic development, anticipate the classes and functions necessary for treatment. Initially, we will structure our interfaces in order to be closer to the users expectations, then we will detail the tools available and how they work, and then we will proceed to the developement of the client application.

2.1 Prototypes

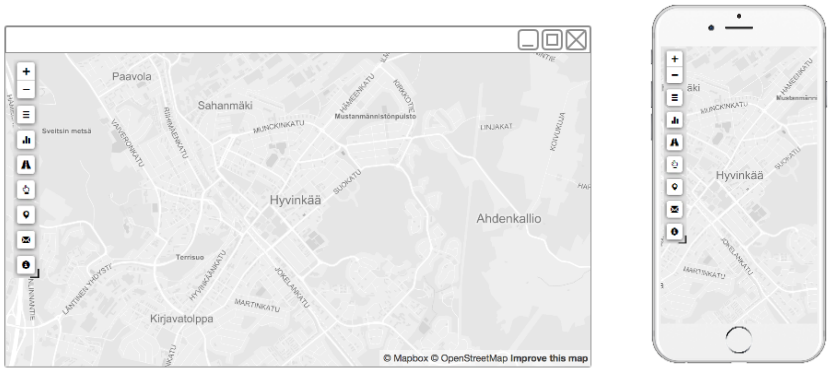
In this section we will detail each of the interfaces that will be implemented in the application. The application will be built in two different ways:

Map content :
containing a basemap and whose tools are over the map

Website content :
containing independent map tools

2.1.1 Map content

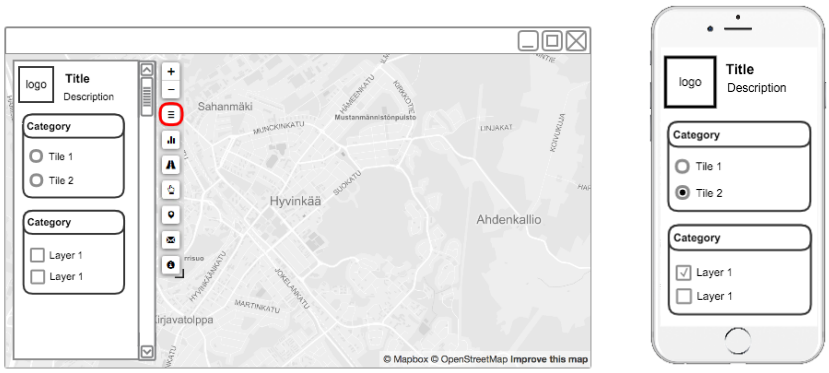
The map content determines the basis of our application. Indeed, the latter consists of three major components: the base map, table of contents and the interraction with map buttons.



(a) Web (b) Mobile

Figure 2.1: Application: Basemap

The base map allows you to locate geographically in a recognizable environment. It shall be set up raster image on which we will be able to overlay vector data from the server. For our application we will build on the maps tiles proposed by MapBox.



(a) Web (b) Mobile

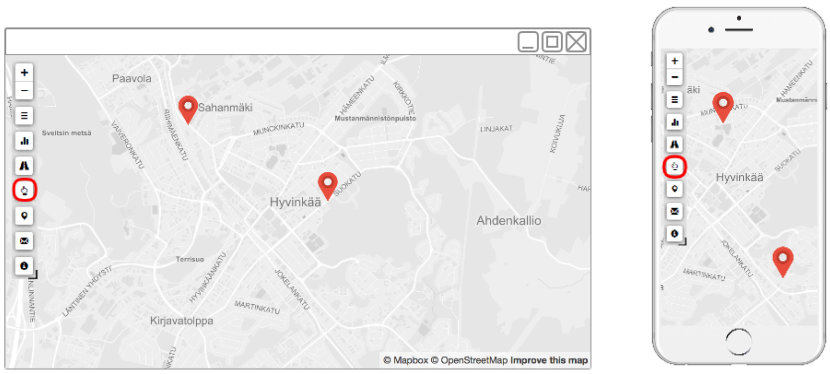
Figure 2.2: Application: Table Of Content

The table of contents (abbreviated TOC) can list the elements of the map and interact with them. It will contain the different layers component of the application. The checkbox or radio button should be there to show or hide items.

The interractions buttons are of three types: the onset of action over the map (popup), the onset of action with interraction on the map (marker), or

open a specific page of the mini-website.

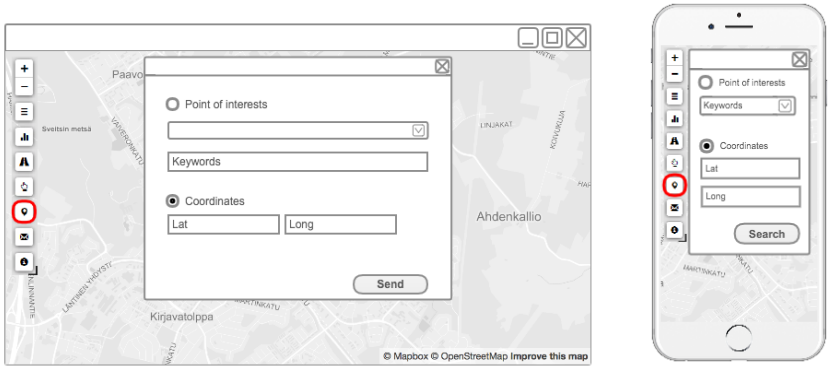
Among the action buttons on the map, an indispensable tool is the route search. This button turns the marker removal by hand directly on the map to place two points that serve as departure and arrival. After dropping the markers, the shortest route is calculated and displayed on the map.



(a) Web (b) Mobile

Figure 2.3: Application: SearchPointer

Another of the tools necessary for the application is a button to geotag. This button opens a popup over the map to choose to locate by keyword or by coordinates (latitude and longitude).

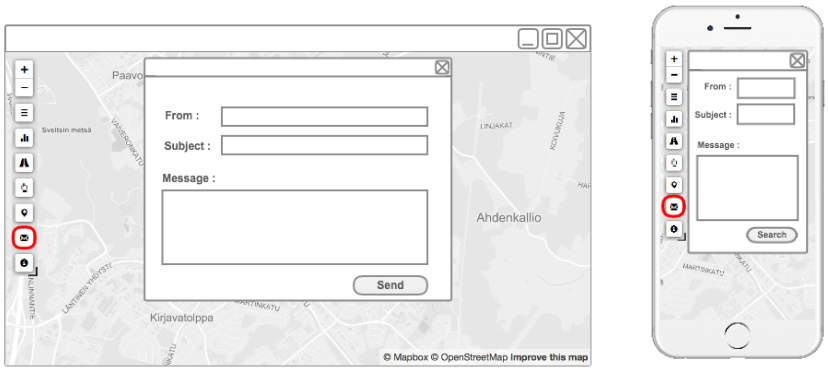


(a) Web (b) Mobile

Figure 2.4: Application: Focus

The last two buttons that need to be added are not necessary for the application but merely useful and informative. These are contact buttons

and information about the application. These two buttons open a popup over the map. The contact button provides a form to send a message directly to the developers of the application. As for the information button is a simple overlay that displays information about the developers and the sources of the application.



(a) Web (b) Mobile

Figure 2.5: Application: Contact

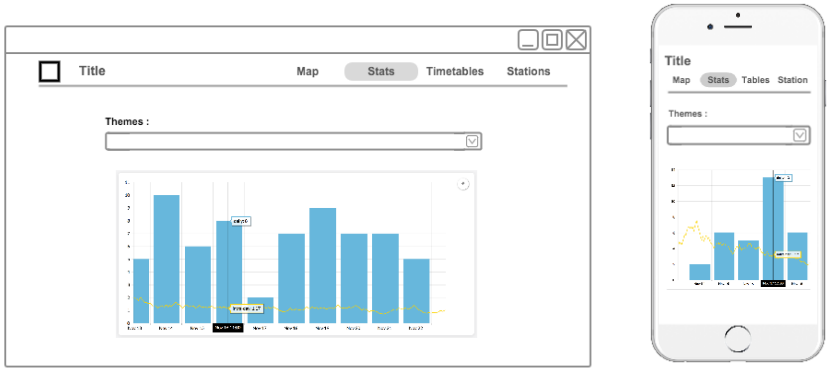
2.1.2 Website content

The last buttons that were not detailed in the previous section are direct links to the Internet website content. Indeed, it is easier to go directly to the desired content from the map rather than browse in the site. Among the non-explicit buttons, we have a button access to statistics and trains timetables.

The website consists of a small top navigation bar and a central content. A link in the navigation bar allows us to go back on the map. The central content of the page changes depending on the desired chart.

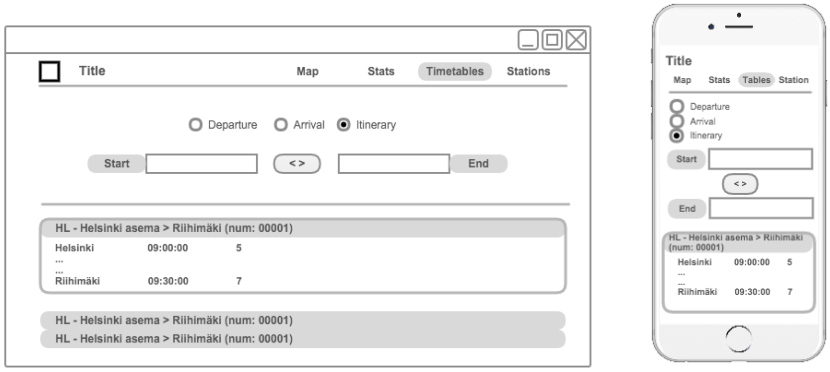
The central content of the statistical part is a charts list and an overall chart that is updated as the selected object on the list.

Regarding train schedules, it is an interface to retrieve the trains that interest the user. The user then has the choice of listing all train departures of a city, all arrivals to a city or to determine the route between two cities. The choice made, the user fills in the desired towns in the affected areas. The application then retrieves the data and signs the user in a list to open.



(a) Web (b) Mobile

Figure 2.6: Application: Statistics



(a) Web (b) Mobile

Figure 2.7: Application: Timetables

2.2 Application's Development

For an efficient, scalable and maintainable code, it is necessary to have to think about the flexibility of it. Indeed, it is possible to add functionality, tools and customize its environment, the tool is more versatile.

2.2.1 Environment

Before beginning the core's development, it is necessary to remind the environment in which the application is developed.

The application must be used on a web environment, desktop and mobile. In fact, the application can be viewed from a web browser. It is also interesting to think now in protabilité to a mobile application (Android, iOS, Windows Phone, FirefoxOS). A modular code is interesting, but even more if it can be adapted as desired platforms.

Server side, we will have all the map information needed. For that, we need to build an server open on the internet and can respond to web requests from the application.

The selected technologies are:

Server

- Apache2 : Free software Apache HTTP Server (Apache) is an HTTP server created and maintained within the Apache Foundation. It will serve as Web server.
- Tomcat7 : Apache Tomcat is a web container free of servlets and JSP Java EE. It will be used to execute the GeoServer's sources.
- GeoServer : GeoServer is an open-source server written in Java - allows users to share, process and edit geospatial data.
- REST Services : The REST web services (Representational state transfer) fully expose functionality as a set of resources (URI) identifiable and accessible by the syntax and semantics of HTTP. This type of service is performed by a Python script Deamon.

Application

- HTML : HyperText Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages.
- CSS : Cascading Style Sheets, CSS, is a style sheet language used for describing the presentation of a document written in a markup language. This language will give style to our application.
- Javascript : JavaScript is a high-level, dynamic, untyped, and interpreted programming language. This language will allow us to

interactions between our HTML content and our GIS data content on the server. It will manage interractions.

Frameworks and Tools

A software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solutions. We will use different frameworks and utilities for our application:

- Bootstrap : Bootstrap is a free and open-source collection of tools in HTML, CSS and Javascript for creating websites and web applications.
- Leaflet : Leaflet is a widely used open source JavaScript library used to build web mapping applications.
- JQuery : jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.
- AmCharts : JavaScript / HTML5 charts and maps library for web sites and web applications. Fast and responsive.
- Cordova : Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies such as HTML5, CSS3, and JavaScript for cross-platform development, avoiding each mobile platforms' native development language.

2.2.2 Application architecture

The server does not require a special structure (a simple REST script executes in Deamon and three self-managed servers) we will detail the application's structure.

The application is based on a pseudo-architecture MVC (model, view, controller). A single HTML page is required for the base mapping, JavaScript will serve as controller to load the data and HTML pages contain information Popup in the form of views.

In fact, we can imagine the structure of the application like this:

```
/sources
├── /config
│   └── (json configuration files)
├── /css
│   └── (app's style files)
├── /img
│   └── (pictures ressources)
├── /js
│   └── (javascript class, loader and event files)
├── /lib
│   └── (frameworks and tools sources)
├── /views
│   └── (HTML sources for differents views)
└── index.html
```

JSON Configuration

In the previous paragraph and the proposed scheme, there is talk of a JSON configuration file. Indeed, to avoid duplication of functions and greater flexibility, it is easier to use a JSON configuration file coupled with a JavaScript class containing Getters. We will use four configuration files:

- configMap : This file contains the default configurations of the map to load (zoom, center, TOC position)
- configContent : This file contains the names of HTML tags that will receive information Popup but also links to popup views, their name and icon.
- configServer : This file contains the server access. Including configurations for GeoServer and the REST API
- configLayerStyle : This file is one of the most important, it contains graphic styles to be applied to each layer superimposed on the map. In particular, the name to be displayed to the user, the minimum and maximum zoom levels, colors and visual styles of the layers.

Classes

Each of the configuration files JSON Javascript has at least one class in the " /js " associated with it. Indeed, as explained in the previous paragraph, it is easier to use objects and tools Getters and Setters like to access a JSON configuration file. In fact, we have the following classes:

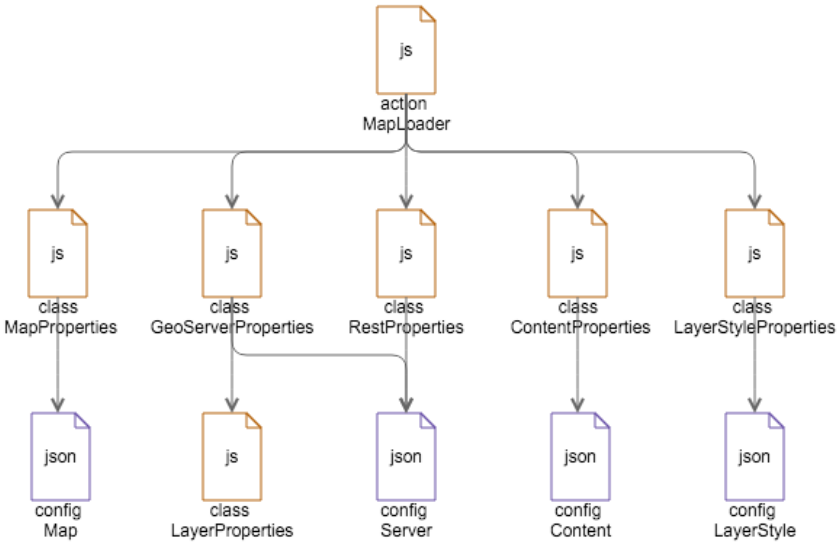


Figure 2.8: Loading JSON and Classes

- classMapProperties: Retrieves all information about the map contained in the file configMap.json
- classGeoServerProperties: Retrieves all information relating to GeoServer contained in the file configServer.json
- classRestProperties: Retrieves all information related to the REST API contained in the file configServer.json
- classContentProperties: Retrieves all information about Popup content to be displayed above the map in the file configContent.json
- classLayerStyleProperties: Retrieves all information about layers contained in the file configLayerStyle.json
- classLayerProperties: This class does not depend on a configuration file, it allows to structure a Layer with attributes allowing interractions

between the map and the table of contents (TOC). An object from this class will then have a name, an alias, a position, a type layer, a URL to the layer on the GeoServer and map item to display.

Events

The core of the application is concentrated in Javascript files prefixed with "action...". Each of the data sorted by theme, to generate the contents of the map. In our sources, the files are loaded in the following order to obtain all the necessary functions before loading the entire map:

- Javascript files of different frameworks (jQuery, Bootstrap, bootstrap extensions, Leaflet, Leaflet Extensions).
- Class files associated with JSON configurations of the preceding paragraph.
- actionPopup: This file contains the related functions indicated in the popup JSON configuration file. Indeed, every popup has its functions, its interractions with the map and its specificities. Among these functions, we also take care of the HTML content that will show popup over the map.
- actionGeoServerLayers: This file contains the functions relating to GeoServer access to layers, their contents and all that it brings, including the recovery of layers of the TOC, management styles and tooltip information.
- actionMapLoader : This file is the main file from our application, it loads all the necessary components. At first it should test the server connection, then load all classes associated with JSON files, generates the popup buttons and actions, recovers GeoServer layers to generate the TOC and display them on the map. The file also contains the refresh map for each move and global variables used in other scripts.

3 Merge

This last part allows for the binder between the transport model calculated in the first paragraph and the application developed in the second paragraph. The ultimate goal is to view information bicycle transportation habits in the city of Hyvinkää through a web browser.

To do this, we will build on the concepts explained in the preceding paragraphs: the transport model acquired and calculated, the application developed and configured.

3.1 REST Services

For the application to interact with the server, we need REST services described in chapter 2. The Python script running on the server in Deamon offers REST services. Each service is called by an HTTP request from the application and returns a JSON containing the requested data.

This service is based on python micro-framework Flask. Backend environment you can see in Fig. 3.1

The rest of this section describes each of the necessary services and used by the application and explains returned JSON formats.

/api

Return an HTML list of all services on the REST API, with an example.

```
1 <h2>Api info for TraMap</h2>
2 <ul>
```

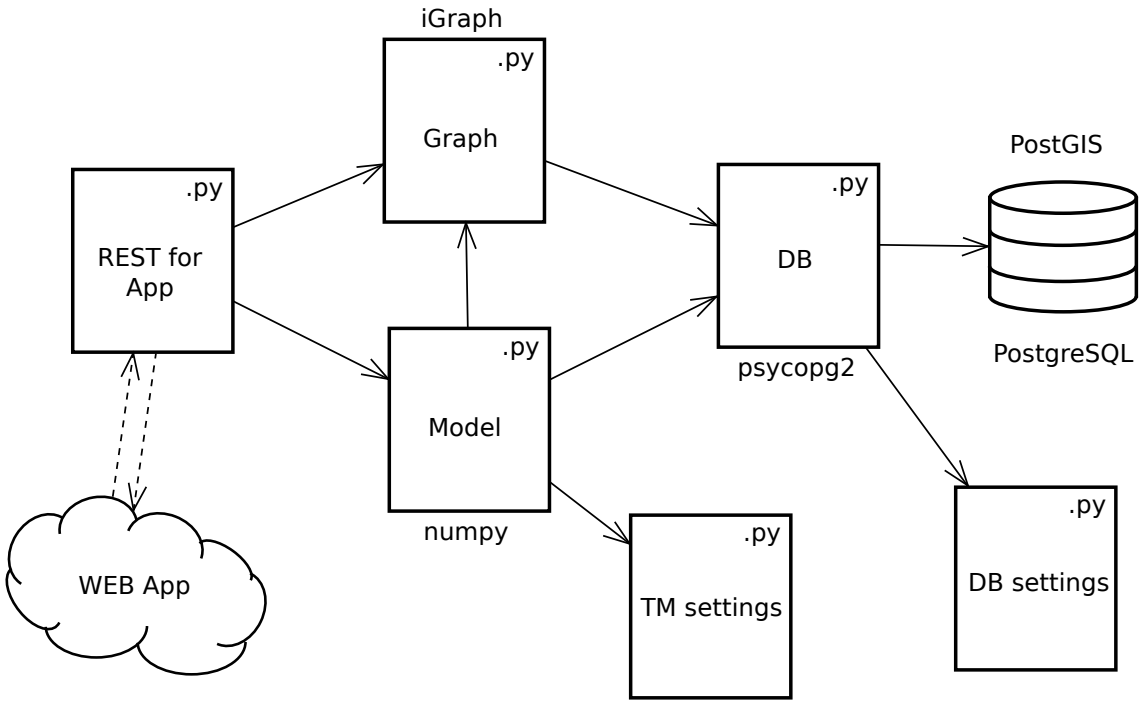


Figure 3.1: Backend (REST)

```
3   <li><b>/api/interests</b> - return type for tables (don
        't need any parameters)</li>
4   <li><b>/api/ssp</b> - return Shortest path from A to B
        e.g
5     <a href="api/ssp?lon1=24.87078&lat1=60.61663&lon2
          =24.85747&lat2=60.63003">
6       api/ssp?lon1=24.87078&lat1=60.61663&lon2
            =24.85747&lat2=60.63003
7     </a></li>
8   <li><b>/api/compute_traffic</b> - start compute traffic
        (return info about start compute)</li>
9   <li><b>/api/compute_traffic_progress</b> - progress for
        compute traffic</li>
10 <ul>
```

/api/interests

Return a JSON object with all the interests point by table.

```
1   {
2     "status" : "ok",
3     "result" : [
4       {
```

```
        "table" : "roads",
        "interests" : [
          "motorway",
          "footway",
          "cycleway",
          "..."
        ]
      },{...}
    ]
  }
```

/api/ssp

Return a JSON object with the geometry shortest path, time (in seconds) and distance (in meters)

```
{
  "status":"ok",
  "result":{
    "distance":1941.2000656999999,
    "features":[
      {
        "type":"LineString",
        "coordinates":[
        [
          24.8714912,
          60.6171716
        ],
        [
          24.8706363,
          60.6168433
        ],
        [
          24.870383,
          60.6167319
          ]
        ]
      },
      {...}
    ]
  }
}
```

/api/compute_traffic

Recompute traffic take a lot of time, so REST must be asynchronous. In our approach we made two service. First service start compute and

second return progress. If you call this service, server start recomputing traffic and return information about start of recompute.

```
{
"status": "ok",
"result": "calculation began"
}
```

/api/compute_traffic_progress

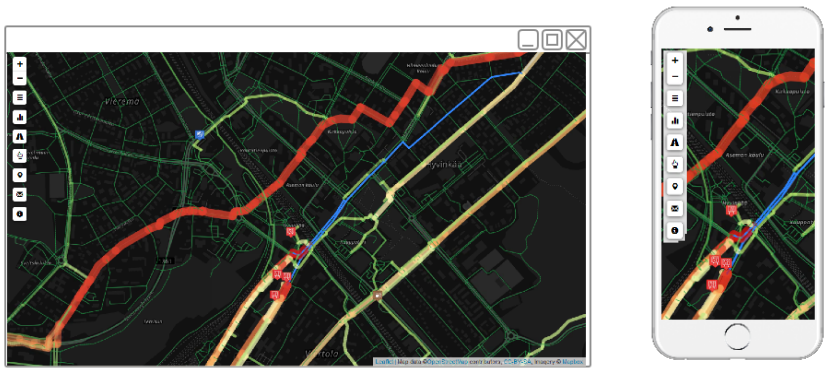
This service is for view progress of recomputing traffic. Service return progress in %.

```
{
"status": "ok",
 "result": {
      "progress": 34,
      "isrun": true
       }
}
```

3.2 Visualization

Now, the REST services are available, we know what is returned and how to analyze it in the application. It remains for us to run the application to see the results.

To do this, simply open a web browser (Mozilla Firefox, Google Chrome, Safari ...) and write the site's URL. For the mobile, we can choose between the web browser or directly by the application. We can see the result :



(a) Web (b) Mobile

Figure 3.2: Combined display

Conclusion

conclusion.tex

Bibliography

[REF] `http://leafletjs.com/`
Leaflet website. 2015 Vladimir Agafonkin. Maps - OpenStreetMap contributors.

[REF] `https://github.com/Turbo87/leaflet-sidebar`
Leaflet extension Sidebar. leaflet-sidebar is free software, and may be redistributed under the MIT license.

[REF] `https://github.com/CliffCloud/Leaflet.EasyButton`
Leaflet extension EasyButton. Leaflet-EasyButton maintained by CliffCloud

[REF] `http://getbootstrap.com/`
Bootstrap CSS Framework website. Code licensed under MIT, documentation under CC BY 3.0.

[REF] `http://silviomoreto.github.io/bootstrap-select/`
Bootstrap extension Select. Bootstrap-select maintained by caseyjhol

[REF] `https://cordova.apache.org/`
Cordova website. Copyright 2012, 2013, 2015 The Apache Software Foundation, Licensed under the Apache License, Version 2.0

[REF] `https://github.com/jsdoc3/jsdoc`
JsDoc3 : Generate Javascript documentation. JSDoc 3 is free software, licensed under the Apache License, Version 2.0.

List of Figures

List of Tables