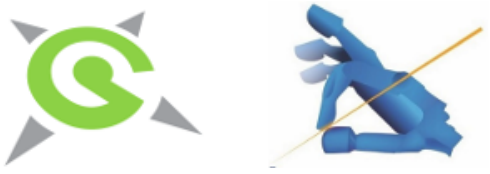

HAMK University of Applied Sciences
et Ramboll Group

Rapport de Projet

# Modèle de Transport et Consultation web

*František Kolovský*
*Pierre-Vincent Vrot*

Erasmus Exchange from
University of West Bohemia
and IMERIR



8 décembre 2015
version 1.0

# Résumé

Les Systèmes d'Information Géographique (SIG) permettent, entre autres, d'acquérir, de traiter, d'organiser et de présenter des données géographiques, produisant ainsi des plans clairs, précis et intuitifs, et ce, à travers une composante web accessible depuis n'importe quel navigateur.

Le projet TraMap, proposé par HAMK et supervisé par Ramboll, est un outil simple et flexible permettant à un utilisateur de consulter des informations d'habitudes de transport issues de données géographiques en open source et d'en obtenir les informations sous forme d'application web et mobile.

C'est dans le cadre de ce besoin d'outil de consultation et d'exploitation des données que s'inscrit notre projet. Notre rôle est de rechercher des données open sources disponibles sur internet, d'en extraire les informations pour contruire un modèle d'habitude de transport, puis de développer une application de consultation des données multiplateformes.

**Keywords :** SIG, Modèle de Transport, Web, Algorithme, Sources Libres

# Remerciements

Il nous est agréable de nous acquitter d'une dette de reconnaissance auprès de toutes les personnes dont l'intervention au cours de ce projet a favorisé son aboutissement ainsi que toutes les personnes ayant participé, de près ou de loin, à notre formation.

Dans un premier temps, nous tenons à présenter nos sincères remerciements à Mr. Jari Mustajärvi, responsable de la formation TIC (Technologies de l'information et de la Communication), pour son accueil chaleureux au sein de HAMK et pour nous avoir permis de vivre une expérience très enrichissante.

Nos plus grands remerciements vont à Mr Janne Rautio, notre responsable de projet, pour sa disponibilité et ses encouragements tout au long de ce projet et dont les directives précieuses et les conseils pertinents nous ont été d'un appui considérable dans nos démarches ; Mme. Taina Haapämaki pour ses directives, son expertise et son implication dans notre projet.

Nous tenons également à adresser nos remerciements à nos écoles respectives, University of West Bohemia et l'IMERIR, pour nous avoir proposé cet Erasmus enrichissant, et ce, dans un cadre agréable de complicité et de respect.

Enfin, que tous ceux et celles qui ont contribué de près ou de loin à l'accomplissement de ce travail trouvent l'expression de nos remerciements les plus chaleureux.

# Table des matières

# Introduction

Dans une Finlande où le vélo, la marche à pieds et les transports en communs sont de plus en plus utilisés, il devient nécessaire de se munir de moyens d'informations les plus utiles possibles afin de pouvoir anticiper les déplacements.

A l'ère actuelle, les entreprises sont toujours à l'affût de nouveautés, notamment dans le domaine SIG, que ce soit question de nouveaux outils ou de technologie. Chaque entreprises développent et commercialisent leurs outils d'information traffic. C'est dans une optique de partage de l'information, et du libre accès que va s'axer notre projet.

Aujourd'hui, open source et open data sont des sources d'information de plus en plus utilisées et permettent d'évoluer en communauté coopérative. Notre objectif principal, pour la réalisation de ce projet, est de récupérer les données et les outils open sources sur lesquels n'importe qui pourrait s'appuyer pour développer un modèle de consultation de données sur les habitudes de transport d'une ville. Notre cadre de recherche s'axera sur la ville de Hyvinkää et des données de déplacement à vélo.

Le travail réalisé se découpe en trois grandes parties : la première consiste à déterminer le modèle de transport théorique et mathématique des données qui seront consommées ; la seconde explique la façon dont une application peut être développée pour la consultation des données ; et enfin la troisième, et dernière partie, permet de combiner les précédentes parties afin d'obtenir un résultat consultable sur différentes plateformes.

# 1 Modèle de Transport

## 1.1 Modèle de Transport

Un modèle de transport est une méthode déterminant un traffic sur les routes ciblé sur une zone décidée. Notre objectif est de déterminer un élément quantitatif d'afflux par route sur un réseau routier à partir de données socio-économiques et démographiques.

La modélisation de transport traditionnels possède plusieurs étape indépendantes (par exemple la génération de voyage, voyage destination, ...). Notre mise en oeuvre de la modélisation du transport se composent de 3 étapes :

**la génération des déplacements** - Dans cette partie, nous voulons déterminer le nombre d'origine du voyage, la destination dans les zones. Cette étape est dépendent de données (par exemple des données démographiques, des données socio-économiques, météorologiques, les habitudes locales, ...). La qualité du modèle dépendra beaucoup de ces données.
**la destination des déplacements** - Dans cette section nous voulons déterminer *transportation matrix* $T$ (OD matrix). Cette matrice permet de dénombrer les voyages entre une zone $i$ vers une zone $j$. Pour ce faire, nous utiliserons un modèle de gravité (Gravity Model).
**Dénombrement du traffic** - Cette étape est la dernière. Nous calculons le trafic pour chaque liaison routière (bord du graphique).

Dans la suite de ce chapitre, les expressions suivantes vont être utilisées :

| | |
|---|---|
| $T$ | transportation matrix (nombre de voyage entre deux zones) |
| $C$ | travel cost matrix (le coup du voyage entre deux zones) |
| $T_i, T_j$ | somme des valeurs en ligne/colonne dans $T$ |
| $n$ | nombre de zone (taille de la matrice) |

### 1.1.1 Destinations des déplacements

Pour déterminer la matrice de transport $T$ nous avons utilisé la méthode Gravity Model :

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

où

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

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

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

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

où dans notre cas

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

$T_i$ est le nombre de voyage sortant de la zone (origine dans la zone) $i$, $T_j$ est le nombre de voyage entrant dans la zone (destination dans la zone) $j$. Quelques fois, la matrice de transpor est appellée *OD Matrix* (Origine - Destination).

Maintenant nous devons déterminer $K_i$ et $K_j$. Pour ce faire nous avons utilisé un ajustement proportionnel itératif. Premièrement nous calculons $T^1$ avec $K_i, K_j = 1$. Ensuite nous pouvons utiliser une équation itérative pour $T$ :

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

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

où $Z_i$ et $Z_j$ sont les voyages d'origine et de destination, $k$ est une itération.

Exemple :

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

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

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

Après une itération, les coefficients sont :

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

### 1.1.2 Dénombrement du traffic

Maintenant, nous savons combien de personnes voyagent de la zone $i$ à la zone $j$, afin que nous puissions trouver le chemin le plus court de $i$ à $j$ et enregistrer cette valeur dans tous bords de la matrice nous devons calculer $z^2$, où $z$ est le nombre de zones. Nous devons donc calculer plus court chemin entre les deux zones OD.

Dans notre solution, nous calculons N chemins pour toutes les paires OD (finallement, nous calculons $Nz^2$ chemins). Chaque chemin est basé sur un autre coût. Le coût est basé sur la longueur, le temps et la destination verticale. Le coût final est combinaison linéaire de ces coûts de partition.

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

où $c$ est le coût, $t$ le temps, $l$ la longueur et $h$ la distance verticale. Nombre de voyage (traffic entre deux zones) est découpée entre ces parties. Sur la figure **??** on peut voir les multi-chemins (même chemins optimaux) entre deux zones.

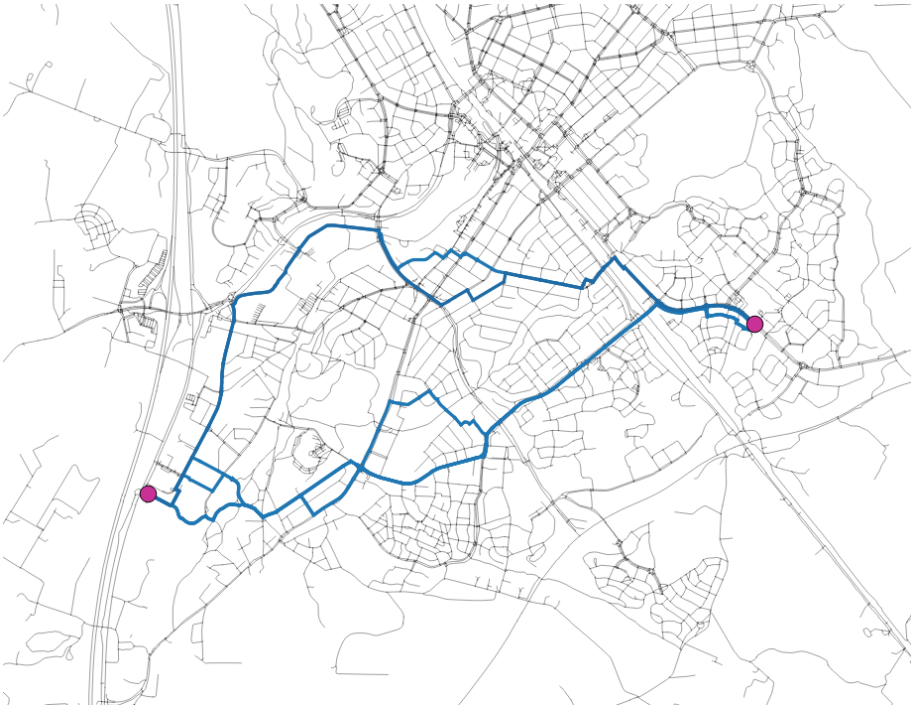


FIGURE 1.1 – Possible paths between one pair of zone (multipath)

## 1.2 Implémentation

Dans cette section, les expressions suivantes vont être utilisées :

| | |
|---|---|
| $n$ | nombre de noeuds |
| $m$ | nombre de bords |
| $z$ | nombre de zone |
| $N$ | nombre de chemin calculé pour une paire de zone |

La modélisation de transport décrits dans la section précédente a été mise en œuvre dans le langage de programmation Python en utilisant la bibliothèque NumPy.

### 1.2.1 Recherche du plus court chemin statique

Pour déterminer $C$ nous utilisons l'algorithme de Dijkstra's (de complexité $O(m + nlog(n))$). De fait, la complexité finale est de

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

Pour le calcul du traffic nous avons aussi utilisé l'algorithme de Dijkstra. De fait, la complexité finale du calcul du traffic est de

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

Pour l'algorithm de Dijkstra nous avons utilisé la bibliothèque Python iGraph. iGraph est écrit en C, c'est donc très rapide. Par exemple, un algorithme Dijkstra est exécuté en 30 ms ($n = 8000$ $m = 18000$).

### 1.2.2 Stockage des données

Toutes les données pour la modélisation des transports sont stockées dans une base de données PostgreSQL avec l'extension PostGIS. PostGIS est une extension spatiale pour les données géométrique. En l'utilisant, on pout manipuler les lignes, les points, les polygones, les annotations et les rasters.

Dans cette base de données on retrouve six tables principales :

| Nom de table | Description |
|---|---|
| **roads** | Les routes (edges) |
| **nodes** | Les noeuds (vertexes) |
| **zones** | La liste des zones |
| **traffic** | |
| **general_area_information** | l'aire géométrique contenant la zone d'intérêt |
| **od_pair** | Implémentation en base de données de la matrice $T$ |

La figure 1.2 schématise le modèle de la base de données avec les relations entre les tables.

Plus de détails sont disponibles sur la documentation de notre dépot de code Github.

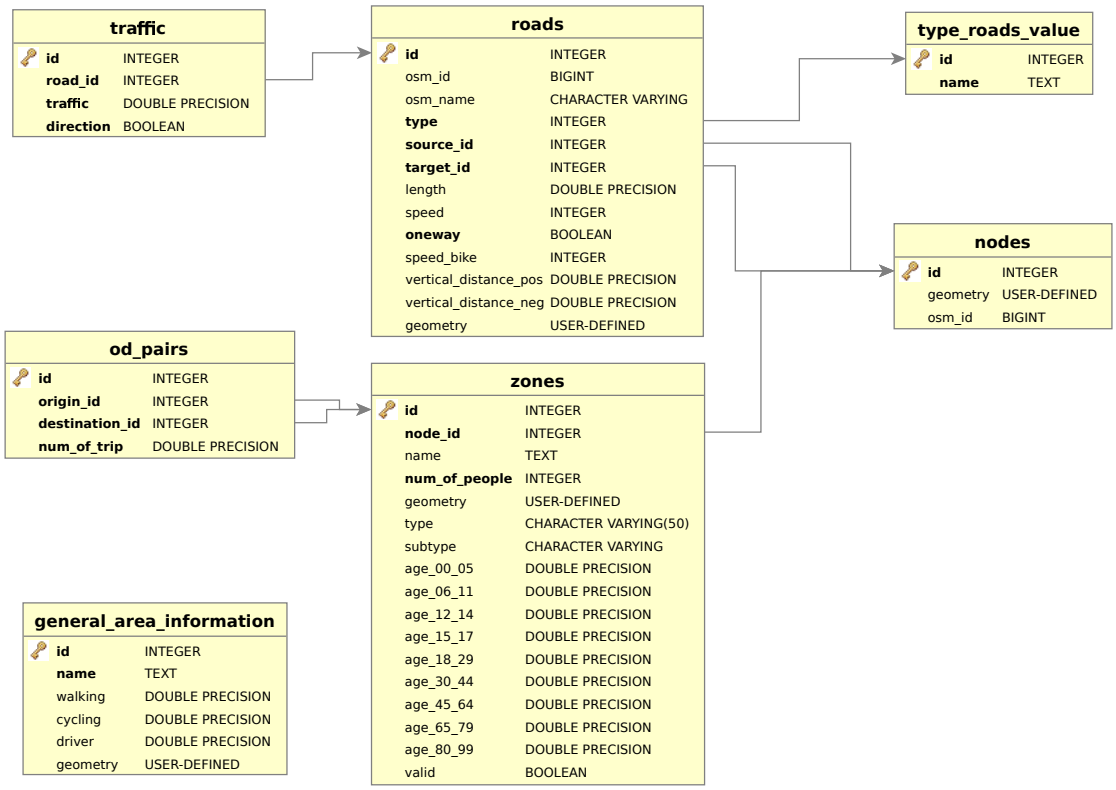



FIGURE 1.2 – Database model

### 1.2.3 Prochains travaux

Cette méthode de modélisation des transports est très difficile ($O(Nz(m+nlog(n))))$, il serait donc préférable d'adapter l'algorithme pour calculer ce problème dans un environnement parallèle (partagé par des threads, entre autre) ou de la mémoire distribuée (MapReduce ou MPI)).

# 2 Consultation web

Avant d'entamer le développement de l'application, il est nécessaire de faire un état du besoin et des attentes. Ainsi, en structurant notre interface il sera plus facile de procéder à son développement graphique, d'anticiper les classes et fonctions nécessaires aux traitements. Dans un premier temps, nous allons structurer nos interfaces afin d'être au plus proche des attentes utilisateurs, puis nous détaillerons les outils à notre disposition et comment ils fonctionnent, et enfin nous procèderons au dévelopement de l'application cliente.

## 2.1 Prototypes

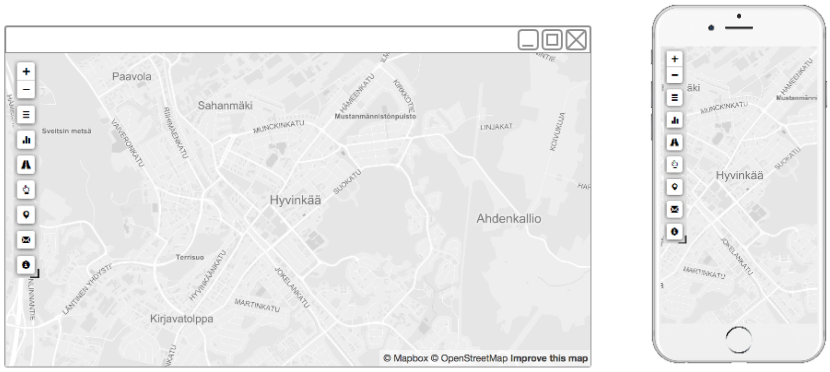
Dans ce paragraphe nous allons détailler chacune des interfaces qui seront implémentées dans l'application. L'application sera construite sous deux angles différents :

**La partie cartographique :**
contenant une carte de base et dont les outils se superposent à cette dernière

**La partie site web :**
contenant des outils indépendants de la carte

### 2.1.1 Partie cartographique

Le contenu cartographique détermine la base de notre application. En effet, cette dernière est constituée de trois éléments important : le fond de carte, la table des matières et les boutons d'interraction avec la carte.




(a) Web (b) Mobile

FIGURE 2.1 – Application : Basemap

Le fond de carte permet de se repérer géographiquement dans un environnement reconnaissable. Il est consitué d'image raster sur lesquelles on va pouvoir superposer les données vectorielles issues du serveur. Pour notre application nous allons nous appuyer sur les fonds de carte proposé par MapBox.

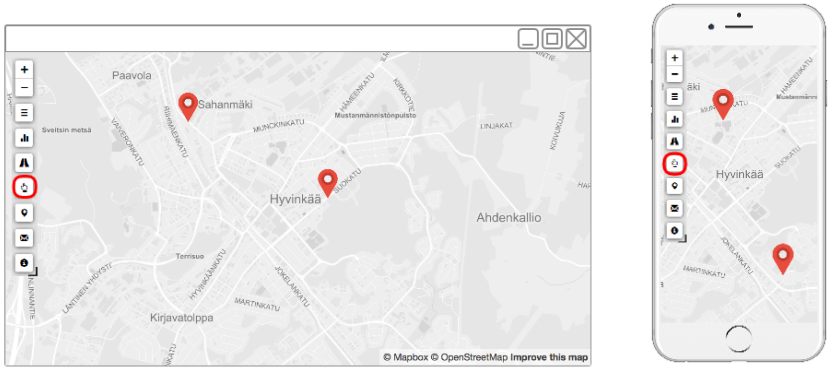

(a) Web (b) Mobile

FIGURE 2.2 – Application : Table Of Content

La table des matières, ou *Table of Content* (en anglais, abrégé TOC), permet de lister les élements de la carte et de pouvoir interagir avec eux. Elle contiendra les différentes couches, ou *Layer* en anglais, constitutives de l'application. Des checkbox ou bouton radio devraient s'y trouver pour afficher ou masquer des éléments.

Les boutons d'interractions sont de trois types : le déclenchement d'action par dessus la carte (popup), le déclenchement d'action avec une interraction sur la carte, ou l'ouverture d'une page spécifique du minisite.

Parmis les boutons d'action sur la carte, un outils indispensable est la recherche d'itinéraire. Ce bouton permet d'activer la dépose de marker à la main directement sur la carte pour placer deux points qui serviront de départ et d'arrivée. Après avoir déposé les markers, un itinéraire le plus court est calculé et affiché sur la carte.

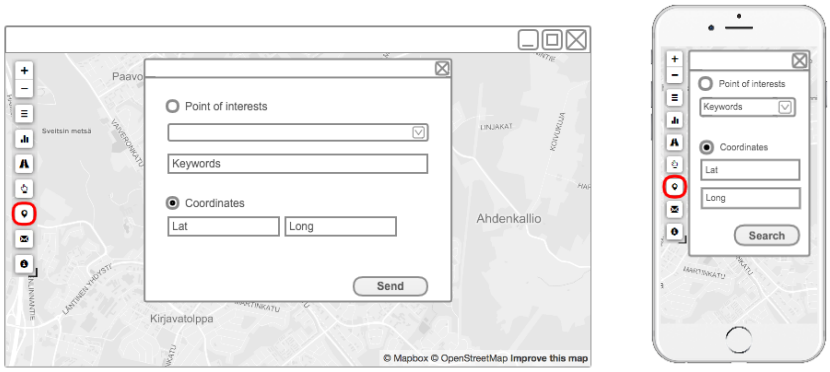



(a) Web (b) Mobile

FIGURE 2.3 – Application : SearchPointer

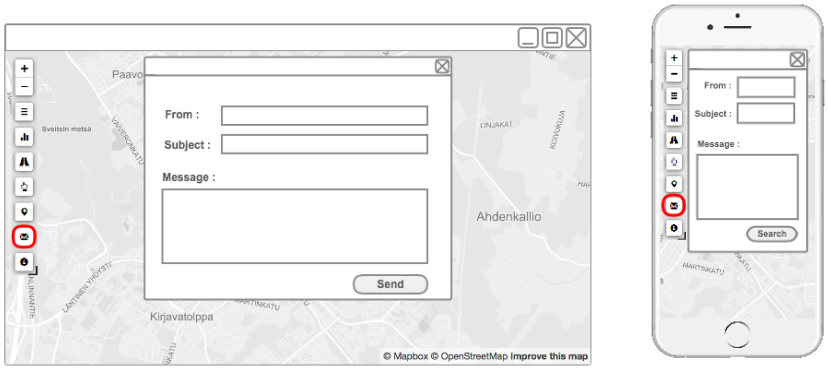
Un autre des outils nécessaires au bon fonctionnement de l'application est un bouton permettant de se géolocaliser. Ce bouton ouvre une popup par dessus la carte permettant de choisir de se localiser par mot clef ou bien par coordonnées (latitude et longitude).

Les deux derniers boutons qui doivent être ajoutés ne sont pas nécessaires pour l'application mais simplement utiles et informatifs. Il s'agit des boutons de contact et d'information à propos de l'application. Ces deux boutons ouvre une popup par dessus la carte. Le bouton de contact propose un formulaire permettant de directement envoyer un message aux développeurs de l'application. Quant au bouton d'information, c'est une simple surcouche permettant d'afficher des informations sur les développeurs et les sources de l'application.




(a) Web (b) Mobile

FIGURE 2.4 – Application : Focus




(a) Web (b) Mobile

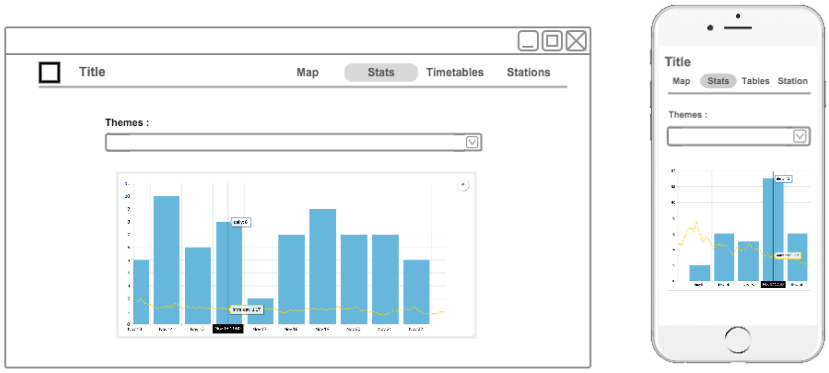
FIGURE 2.5 – Application : Contact

### 2.1.2 Partie site web

Les derniers boutons qui n'ont pas été détaillés dans la partie précédent sont les liens directs vers le mini-site internet. En effet, Il est plus aisé d'aller directement sur le contenu souhaité depuis la carte plutôt que de naviguer sur le site. Parmis les boutons non explicités, on retrouve un bouton d'accès aux statistiques et aux horaires de trains.

Le site web est composé d'une petite barre de navigation en haut de la page et d'un contenu central. Un lien dans la barre de navigation nous permet de revenir sur la carte. Le contenu central de la page change selon la page désirée.

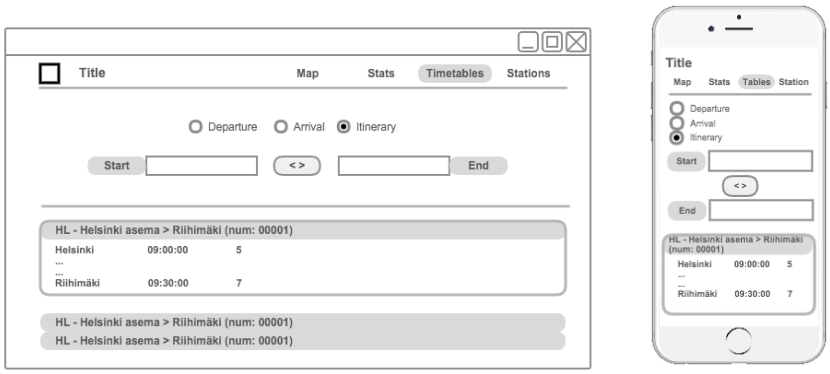
Le contenu central de la partie statistique est une liste de schéma et d'un schéma global qui s'actualise selon l'objet sélectionné sur la liste.




(a) Web (b) Mobile

FIGURE 2.6 – Application : Statistics

En ce qui concerne les horaires de train, il s'agit d'une interface permettant de récupérer les trains qui intéressent l'utilisateur. L'utilisateur à alors le choix de lister tous les départs de train d'une ville, toutes les arrivées d'une ville ou bien de déterminer l'itinéraire entre deux villes. Une fois le choix fait, l'utilisateur renseigne les villes désirées dans les zones concernées. L'application récupère alors les données et les affiches à l'utilisateur sous forme de liste à ouvrir.




(a) Web (b) Mobile

FIGURE 2.7 – Application : Timetables

## 2.2 Développement de l'application

Pour obtenir un code efficace, maintenable et évolutif, il est nécessaire de devoir penser à la fléxibilité de celui-ci. En effet, plus il est possible d'ajouter des fonctionnalité, des outils et de personnaliser son environnement, plus l'outil est polyvalent.

### 2.2.1 Environnement

Avant d'entamer le coeur du développement, il est nécessaire de rappeler l'environnement dans lequel l'application est développée.

L'application doit être utilisable sur un environnement web, sur bureau et sur mobile. De fait, l'application peut être consultable depuis un navigateur web. Il est également intéressant de penser dès maintenant à la protabilité vers une application mobile (Android, iOS, Windows Phone, FirefoxOS). Un code modulable est intéressant, mais encore plus s'il peut être adapté selon les plateformes désirées.

Coté serveur, nous disposerons de toutes les informations cartographiques nécessaires. Pour celà, il faut pouvoir construire un serveur ouvert sur internet et pouvant répondre aux requêtes web venant de l'application.

Les technologies choisies sont donc :

**Serveur**

- Apache2 : Le logiciel libre Apache HTTP Server (Apache) est un serveur HTTP créé et maintenu au sein de la fondation Apache. Il servira de serveur Web.
- Tomcat7 : Apache Tomcat est un conteneur web libre de servlets et JSP Java EE. Il servira à faire tourner les sources GéoServer.
- GéoServer : GéoServer est un serveur informatique open source et libre écrit en Java qui permet aux utilisateurs de partager et modifier des données géographiques. Il contiendra nos informations cartographiques.
- REST Services : Les services web de type Representational state transfer (REST) exposent entièrement des fonctionnalités comme un ensemble de ressources (URI) identifiables et accessibles par la

syntaxe et la sémantique du protocole HTTP. Ce type de service est exécutée par un script Python en Deamon.

**Application**

- HTML : L'Hypertext Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web.
- CSS : Les feuilles de style en cascade est un langage informatique qui décrit la présentation des documents HTML et XML. Ce langage permettra de donner un style à notre application.
- Javascript : JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs. Ce langage nous permettra de faire les interractions entre notre contenu HTML et nos données SIG contenu sur le serveur. Il permettra de gérer les interractions.

**Frameworks et Utilitaires**

En programmation informatique, un framework ou structure logicielle est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel. Nous allons nous appuyer sur différents frameworks et utilitaires pour notre application :

- Bootstrap : Twitter Bootstrap est une collection d'outils HTML, CSS et JavaScript utiles à la création de sites et d'applications web.
- Leaflet : Leaflet est une bibliothèque logicielle libre en JavaScript de cartographie interactive.
- JQuery : jQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.
- AmCharts : amCharts est une bibliothèque Javascript de graphiques qui répond à tous les besoins de visualisation de données.
- Cordova : Apache Cordova est un framework de développement mobile open-source. Il permet d'exploiter les technologies Web courantes telles que HTML5, CSS3 et JavaScript pour développer des applications multi-plateformes, évitant ainsi l'utilisation des langages natifs propres aux différentes plates-formes mobiles.

### 2.2.2 Architecture de l'application

Le serveur ne nécessitant pas de structure particulière (un simple script REST excuté en Deamon et trois serveurs auto-gérés) nous allons détailler

la structure de l'application.

L'application s'articule sur une pseudo-architecture MVC (model, view, controller). Une seule page HTML est nécessaire pour obtenir la cartographie de base, des scripts JavaScript vont servir de controller pour charger les données et des pages HTML contiendront les informations des Popup sous forme de views.

De fait, nous pouvons imaginer la structure de l'application de la manière suivante :

```
/sources
├── /config
│   └── (fichier de configuration json)
├── /css
│   └── (style de l'application)
├── /img
│   └── (les sources d'imagerie)
├── /js
│   └── (fichier de chargement et d'évènement javascript)
├── /lib
│   └── (sources des frameworks et outils)
├── /views
│   └── (sources HTML des différents affichages)
└── index.html
```

### Configuration JSON

Dans le paragraphe précédent et sur schéma proposé, il est question d'un dossier de configuration JSON. En effet, pour éviter la redondance des fonctions et pour une meilleure fleibilité, il est plus facile d'utiliser un fichier de configuration JSON couplé à une classe JavaScript contenant des Getters. Nous utiliserons ainsi, quatre fichiers de configurations :

— configMap : Ce fichier contient les configurations par défaut de la carte à charger (zoom, centre, position de la TOC)
— configContent : Ce fichier contient les noms des balises HTML qui recevront les informations des Popup, mais également les liens vers les views des popup, leur nom et leur icon.
— configServer : Ce fichier contient les accès au serveur. Notamment les

configurations pour le GéoServer et pour l'API REST
— configLayerStyle : Ce fichier est l'un des plus important, il contient les styles graphiques à appliquer pour chaque couches à superposer sur la carte. Notamment, le nom à afficher à l'utilisateur, les niveaux de zoom minimum et maximum, les couleurs et les styles visuels des couches.

### Classes

Chacun des fichiers JSON de configuration possède au moins une classe Javascript dans le répertoire "/js" qui lui est associé. En effet, comme expliqué dans le paragraphe précédent, il est plus facile d'utiliser des objets et des outils comme des Getters et Setters pour accéder à un fichier de configuration JSON. De fait, nous avons les classes suivantes :

— classMapProperties : Permet de récupérer toutes les informations relatives à la carte contenu dans le fichier configMap.json
— classGeoServerProperties : Permet de récupérer toutes les informations relatives au GeoServer contenu dans le fichier configServer.json
— classRestProperties : Permet de récupérer toutes les informations relatives à l'API REST contenu dans le fichier configServer.json
— classContentProperties : Permet de récupérer toutes les informations relatives aux contenus des Popup à afficher au dessus de la carte dans le fichir configContent.json
— classLayerStyleProperties : Permet de récupérer toutes les informations relatives aux layers contenus dans le fichier configLayerStyle.json
— classLayerProperties : Cette classe ne dépend pas d'un fichier de configuration, elle permet de structurer un Layer avec des attributs permettant des interractions entre la carte et la table des matières (TOC). Un objet issu de cette classe possèdera alors un nom, un alias, une position, un type de couche, une URL vers la couche sur le GeoServer ainsi que l'objet cartographique à afficher.

### Evènements

Le coeur de l'application est concentrée dans des fichiers Javascript préfixés par "action...". Chacun de ces fichiers, triés par thème, permet de générer le contenu de la carte. Dans nos sources, les fichiers sont chargés dans l'ordre

suivant afin d'obtenir toutes les fonctions nécessaires avant de charger la totalité de la carte :

- Les fichiers Javascript des différents Frameworks (jQuery, Bootstrap, Extensions de Bootstrap, Leaflet, Extensions de Leaflet).
- Les fichiers de classes associés aux configurations JSON du paragraphe précédent.
- actionPopup : Ce fichier contient les fonctions relatives aux popup renseignées dans le fichier de configuration JSON des Popup. En effet, chaque popup dispose de ses fonctions, de ses interractions avec la carte et de ses spécificités. Parmis ces fonctions, nous chargeons également le contenu HTML des popup qui seront au-dessus de la carte.
- actionGeoServerLayers : Ce fichier contient les fonctions relatives aux accès GeoServer, aux layers, à leur contenus et tout ce qui s'y rapproche, notamment la récupération des couches de la TOC, la gestion des styles et des bulles d'informations.
- actionMapLoader : Ce fichier est le fichier central de toute l'application, il charge tout les composants nécessaires. Dans un premier temps il se doit de tester la connexion serveur, puis de charger toutes les classes associés aux fichiers JSON, génère les boutons de popup et d'action, récupère les couches GéoServer pour générer la TOC et les afficher sur la carte. Le fichier contient également le rafraichissement de la carte pour chaque déplacement et les variables globales utilisées dans les autres scripts.

# 3 Combinaison

Cette dernière partie permet de faire le liant entre le modèle de transport calculé dans le premier paragraphe et l'application développée dans le second paragraphe. L'objectif final étant de visualiser les informations des habitudes de transport à vélo dans la ville de Hyvinkää au travers un navigateur web.

Pour ce faire, nous nous appuierons sur les notions expliqués dans les paragraphes précédents : le modèle de transport aquis et calculé, l'application développée et configurée.

## 3.1 Services REST

Pour que l'application puisse interagir avec le serveur, nous avons besoin de services REST expliqué dans le chapitre 2. Le script Python exécuté en Deamon sur le serveur propose des services REST. Chaque service est appellé par une requête HTTP depuis l'application puis retourne un JSON contenant les données demandées.

Ce service est basé sur un micro-framework python nommé Flask. L'environnement de la partie Backend est schématisé sur la figure 3.1.

La suite de ce paragraphe décrit chacun des services nécessaires et utilisés par l'application et explique les formats de JSON retournés.

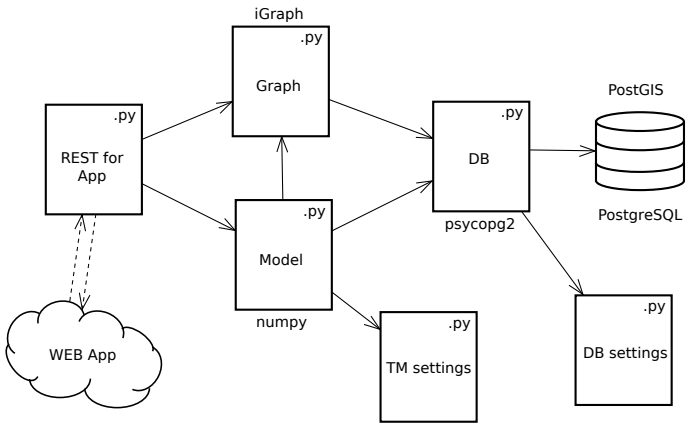



FIGURE 3.1 – Backend (REST)

**/api**

Retourne une liste HTML des services REST disponibles, avec un exemple pour chacun.

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

Retourne un object JSON avec tous les points d'intérêts classés par tables.

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

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

**/api/ssp**

Retourne un object JSON avec la géométrie du chemin le plus court, le temps (en seconde) et la distance (en mètres)

```
{
  "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**

Recalculer le traffic prend beaucoup de temps, ce service REST permet de le faire en asynchone. Dans notre approche, nous avons développé deux services : l'un pour lancer le calcul, et le second pour suivre la progression. Si ce service est appelé, le serveur lance le recalcul du traffic et retourne l'information d'un démarrage avec succès (ou non)

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

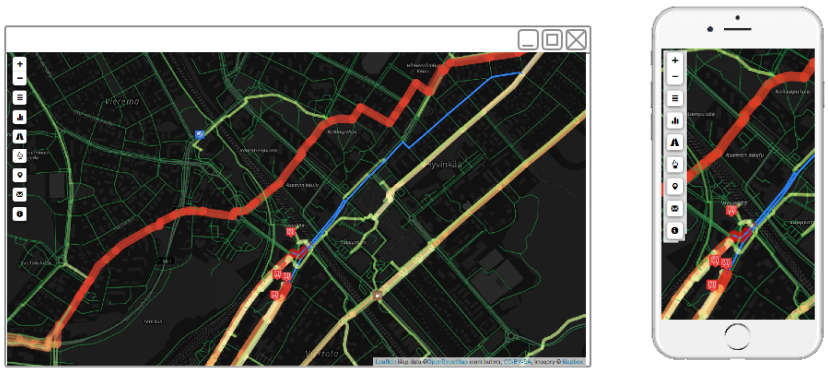
**/api/compute_traffic_progress**
Ce service permet d'obtenir un JSON décrivant la progression du recalcul du traffic. Le service retourne une progression en pourcentage (%).

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

## 3.2 Visualisation

Désormais, les services REST sont accessibles, nous savons ce qui est retournés et comment l'analyser avec l'application. Il nous reste plus qu'à faire tourner l'application afin de consulter les résultats.

Pour ce faire, il suffit d'ouvrir un navigateur web (Mozilla Firefox, Google Chrome, Safari...) et de renseigner l'URL du site ou bien d'ouvrir directement l'application depuis son mobile. On peut consulter le résultat :



(a) Web (b) Mobile

FIGURE 3.2 – Combined display

# Conclusion

L'information représente un actif important lié aux systèmes d'information géographique, dans le contexte actuel, la concurrence effrénée à laquelle se livrent les entreprises entraîne la multiplication des SIG comme atout majeur lors des décisions stratégiques.

Durant tout notre projet nous avons travaillé sur une application open source basée sur des données ouvertes sur les informations traffic des déplacements à vélo sur la ville de Hyvinkää, au sud de la Finlande. Notre application se centre sur la visualisation des informations géographiques ainsi que des fonctions additionnelles comme la recherche du chemin le plus court.

Notre logiciel peut être découpé en deux parties : la partie frontend et la partie backend. La partie frontend, écrite en Javascript utilisant Leaflet et Bootstrap, permet la consultation. La partie backend, écrite en Python utilisant Flask et le moteur de base de données PostgreSQL avec son extension PostGIS, permet le stockage et l'envoi d'information des données.

Pour calculer les informations traffic, nous avons utilisé un modèle de transport standard (Gravity model), ainsi la qualité de nos résultat dépend des données d'entrées (démographie et socio-économiques).

Notre projet peut encore être poussé bien plus loin et continuer d'évoluer. Par l'ajout d'outils applicatif, tout comme sur le calcul des chemins. A court terme, la partie frontend peut être doté d'une partie statistique informant sur les données de l'application ; la partie backend peut se concentrer sur d'autres paramètres de calcul pour le flux de traffic (notamment la capacité de la route) et être optimisé en utilisant des calculs parallèles.

A long terme, il serait intéressant de pouvoir constituer une application basée sur des données en temps réel, couplant ainsi les données des transports

en commun, des conditions météos, des fluctiations de traffics des voitures, des cyclistes et des piétons.

# Bibliographie


[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.

[REF] `http://postgis.net/docs/manual-2.0/`
*PostGIS 2.0 Manual*. This work is licensed under a Creative Commons Attribution-Share Alike 3.0 License

[REF] `http://igraph.org/python/doc/igraph-module.html`
*python-igraph manual*.

[REF] `http://flask.pocoo.org/docs/0.10/`
*Flask's documentation*.

# Table des figures