Introduction
- La société:
- Le logiciel
- Présentation de la mission

Configuration du serveur

Serveur cartographique
- Installation de postgresql
  - Méthode recommandée
- Installation de PostGis
- Installation d'Osm2pgsql
  - Caractéristiques
  - Installation
- Installation de geoserver

Les API
- Introduction
- Mise en place des API avec Docker
- Le routing
  - Les algorithmes utilisés
    - A* (A star)
    - Dijkstra
  - Les outils
  - Installation de OSRM
  - Mise en oeuvre de OSRM
    - Requête:
    - Réponses:
    - Les Services rendu par l'API
      - Nearest:
      - Route
      - Trip
  - Tile service
- Le geocoding
  - Pré-requis
  - Installation
    - Tester l'instance
  - Mise en oeuvre de l'API
    - /search/
    - /reverse/
    - /search/csv/
      - Exemples
    - /reverse/csv/

Bibliographie et liens

Introduction

La société:

En 2016 la société Terra Nova rejoint Sysoco et est spécialisée dans les solutions de collecte d'informations à distance dans des domaines liés à la Propreté Urbaine, au Transport et à la Viabilité Hivernale. Cette société prend le nom de SYSOCO mobility.

Les solutions logicielles SYGETRACK de SYSOCO mobility sont utilisées par des clients importants comme Montpellier Méditerranée Métropole, la Métropole Aix-Marseille, Nice Côte d'Azur Métropole ou le Syndicat Mixte du Bois de l'Aumône. La Police Municipale de la Ville de Nice figure également dans la liste de références de SYSOCO mobility ainsi que le Conseil Départemental du Loir-et-Cher. Dans la région Rhône Alpes, des clients comme les Villes de Saint-Etienne, d'Echirolles et Chambéry Métropole utilisent quotidiennement des solutions déployées par SYSOCO mobility. Des entreprises privées comme, SILIM & BRONZO Environnement, Derichebourg PolyUrbaine ou le Groupe Nicollin ont choisi les solutions développées et déployées par la société.

SYSOCO mobility compte prés de 150 clients a travers la France.

Depuis mars 2019 SYSOCO mobility est intégré au sein de Vinci énergies.

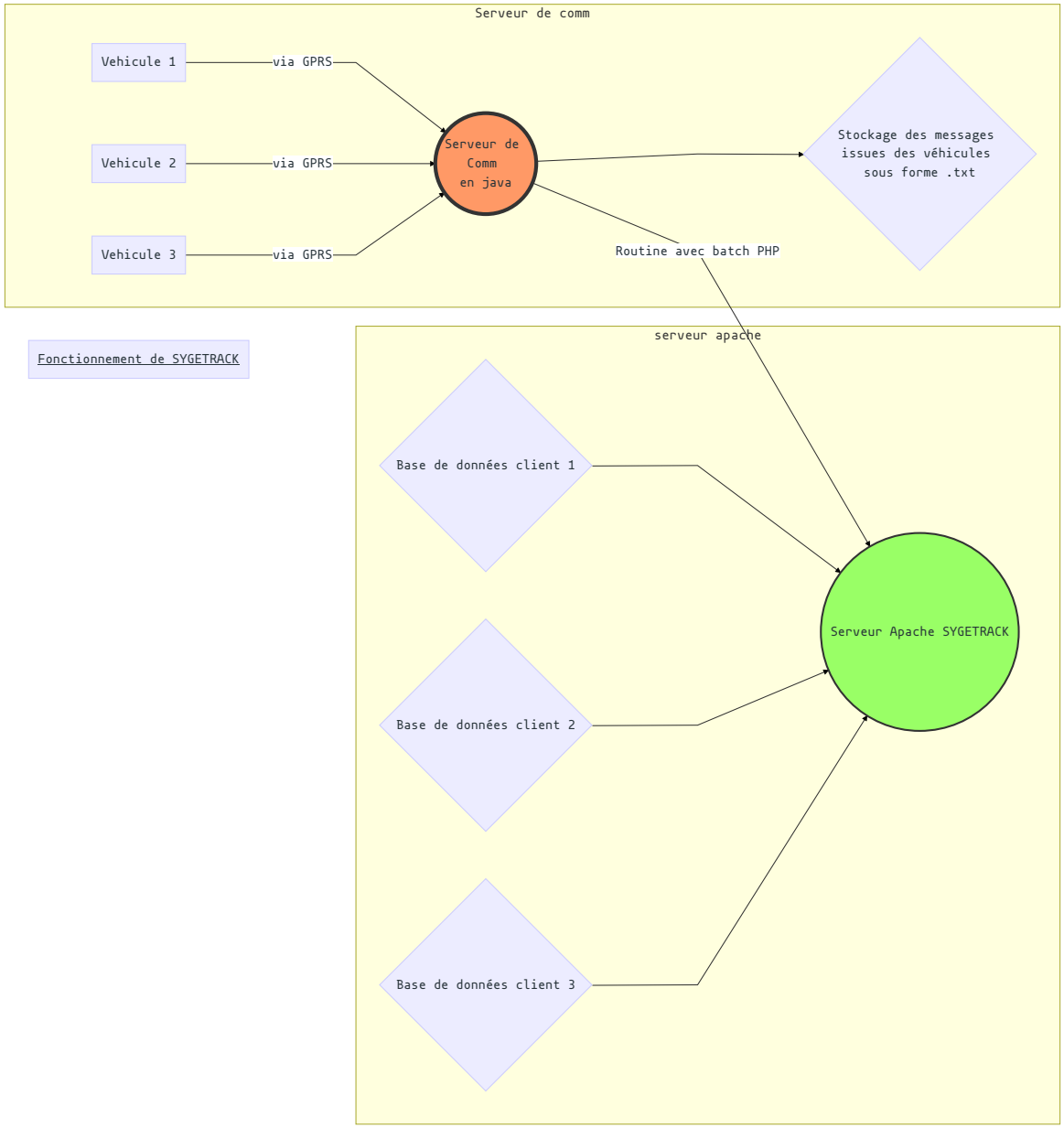
SYSOCO mobility est la société au sein de Vinci qui assure la mise en oeuvre et le développement des solutions SYGETRACK. Elle est composée d'un directeur, de quatre ingénieurs développeurs, deux secrétaires qui assurent le SAV et les contrats et de deux responsables commerciaux.

Le logiciel

Les solutions mise en œuvre par SYGETRACK permettent aux opérateurs de programmer les tournées de ramassage des ordures ménagères. Chaque camion benne à ordures ménagères ou BOM est équipé de multiples capteurs; puces GPS, capteurs de levée de bac, lecteurs de puces RFID permettant de remonter vers les serveurs toutes les informations de poids, d'identification, de chaque mouvement de bac. De plus chaque information est géolocalisée.

SYGETRACK est développé en php 4 avec un framework créé par les développeurs de SYSOCO avec un serveur Web apache. La base de données est une solution firebird.

La réception des informations issues des camions benne par GPRS transit par un serveur de communication développé en java en interne.



SYGETRACK permet pour l'opérateur des remontées de statistiques sur l'ensemble des tournées, d'effectuer des calculs de carbone consommée pour l'établissement des taxes carbone, etc… Ces taxes carbone sont ensuite payer par l'organisme. Cela montre l'importance d'optimiser les tournées.

Comme chaque informations est géolocalisée par une latitude et une longitude il est nécessaire d'effectuer un « reverse-geocoding » de ces positions pour retrouver une adresse avec numéro de rue, nom de rue et ville. Il existe des services payants ou gratuits qui permettent ce reverse-geocoding mais avec des limites.

Le gouvernement français à mis en place une API permettant de faire du geocoding et reverse-geocoding à l'adresse suivante:

`https://adresse.data.gouv.fr/api` .

Cette API est gratuite mais avec les limites suivantes:

- L'API unitaire est disponible à hauteur de 10 appels par seconde et par adresse IP.
- Le géocodage de masse (CSV) est disponible à hauteur d'un appel simultané par adresse IP.

SYGETRACK à besoin de retrouver plusieurs centaines d'adresses par jour. Cette solution n'est donc pas possible. Il existe aussi des sociétés privés qui vendent des services de geocoding et de geocoding inverse. C'est la solution actuelle retenue par SYGETRACK. Mais elle à un coût.

Pour l'optimisation des tournées de ramassage SYSOCO fait appel à une société qui calcule la meilleurs tournée entre plusieurs points, c'est la société benomad: `https://benomad.com` .

Présentation de la mission

La mission qui m'est confiée comporte deux parties.

La première partie concerne les serveurs de tuiles cartographiques. Il s'agit de faire un état de l'art des technologies existantes puis de mettre en place ce serveur.

La deuxième partie concerne les API de **géocodage** et **routage**. Il faut après étude des meilleurs technologies du moment mettre en place ces API.

Le géocodage consiste à affecter des coordonnées géographiques (longitude/latitude) à une adresse postale. Son pendant est le géocodage inversé. Le géocodage inversé (ou en anglais : *reverse geocoding*) consiste à effectuer l'opération inverse du géocodage, c'est-à-dire d'attribuer une adresse à des coordonnées géographiques. L'adresse ainsi retrouvée peut être utilisée dans des applications de géolocalisation capables d'indiquer l'adresse où se trouve la personne utilisatrice.

Le **routage** est le mécanisme par lequel des chemins sont sélectionnés dans un réseau routier pour acheminer un voyageur d'un point de départ vers un ou plusieurs point de destination. Il existe traditionnellement le routage entre deux points ou la routage sur une multitude de point pour organiser au mieux une tournée.

Configuration du serveur

Pour créer, tester les différentes configuration il sera fait le choix d'un serveur sous VM avec virtualBox. Cette VM sera montée sur un PC tournant sous windows 7.

Vu les besoins de mémoire la configuration minimale sera la suivante:

- disque dur de 75 Go.
- 4 coeurs virtuels
- Un minimum de 35 Go de RAM
- Si la RAM est inférieur mettre en place une swap de 35 Go

Les prérequis logiciels sont:

- docker
- Docker-compose
- git
- wget
- Unzip
- postgresql
- Postgis

Serveur cartographique

Actuellement SYSOCO utilise un serveur de tuiles cartographique ayant été mis en place il y a une dizaine d'année sous Flash/Adobe en actionscript 2. Ce serveur n'a jamais été mis à jour et se retrouve bloqué par les politiques des navigateurs internet refusant la technologie flash.

SYSOCO utilise aussi un serveur de tuiles cartographique sur internet mise en place par OpenStreetMap et soumis à des mise à jour non maitrisée par SYSOCO.

SYSOCO désire maitriser les mises à jour du serveur en mettant en place son propre serveur de tuiles cartographiques.

Après recherche des différentes technologies existantes pour mettre en place un serveur de tuiles cartographiques il s'avère que deux serveurs dominent le marché open source des serveurs cartographiques: Mapserver et Geoserver.

Mapserver ne possède pas d'interface graphique et l'intégralité de la configuration se fait avec des fichiers.map.

Geoserver se configure grâce à une interface graphique ou l'on peut mettre en place les différentes services avec leurs sources de données. Geoserver est écrit en java et nécessite l'installation d'un serveur TOMCAT.

La difficulté sur ces deux serveurs est de mettre en place des fichiers de configuration de styles permettant la visualisation des informations cartographiques aux couleurs habituelles pour les clients, à savoir OpenStreetMap.

Ces deux serveurs sont connectés à un serveur de base de données.

En terme de gestion de la données physique, nous utiliserons un serveur de base de données Postgresql avec PostGis. Le plugin Postgis est doté de toutes les fonctionnalités d'un système d'information géographique (SIG). Postgresql+Postgis est basé sur une architecture client-serveur accessible depuis le web grâce à pgAdmin.

Pour télécharger les données issue de OpenStreetMap il est nécessaire d'utiliser un outils supplémentaire: **osm2pgsql**

Installation de postgresql

Méthode recommandée

Installez simplement le paquet postgresql, par ligne de commande :

```
sudo apt install postgresql
```

PostgreSQL est un serveur qui permet de se connecter à différentes bases de données. Par défaut, seul l'utilisateur *postgres* peut se connecter.

Toutes les opérations d'administration se font, au départ, avec l'utilisateur *postgres*. À la fin de l'installation, celui-ci ne possède pas de mot de passe : c'est un utilisateur bloqué et le mieux est qu'il le reste. La première chose à faire sera de créer un nouvel utilisateur, mais pour ce faire, il faut se connecter au moins une fois en tant qu'utilisateur *postgres*. Pour devenir *postgres* et faire les opérations d'administration qui suivent, utilisez sudo :

```
$ sudo -i -u postgres
Password:
```

exit permettra, à la fin de cette session d'administration dans PostgreSQL, de reprendre la main en tant qu'utilisateur du système.

Désormais, l'invite de commande doit mentionner que vous êtes actif en tant que *postgres*. Pour lancer l'invite de commande SQL de PostgreSQL, tapez simplement :

```
psql
```

Installation de PostGis

```
    sudo apt-get install postgis
```

Pour créer une base de données nommée webmapping avec l'extension PostGis il faut:

```
sudo -u postgres createdb -O postres webmapping
sudo -u postgres psql -c "CREATE EXTENSION PostGIS; CREATE EXTENSION
PostGis_topology;" webmapping
```

Cette extension permettra de mettre en place les différentes tables

Installation d'Osm2pgsql

Osm2pgsql est un outil de chargement de données **OpenStreetMap** dans une base de données **PostgreSQL** / **PostGIS** appropriée pour des applications telles que le rendu dans une carte.

Pour télécharger les données OSM dans la base de données postGIS voir le lien https://github.com/openstreetmap/osm2pgsql.

Caractéristiques

- Convertit les fichiers OSM en une base de données PostgreSQL
- La conversion des étiquettes en colonnes est configurable dans le fichier de style
- Capable de lire les fichiers .gz, .bz2, .pbf et .osm directement
- Peut appliquer des différences pour maintenir la base de données à jour
- Soutenir le choix de la projection en sortie

- Noms de table configurables
- Prise en charge du type de champ hstore pour stocker l'ensemble complet de balises dans un champ de base de données si nécessaire

Installation

Prerequis:

```
sudo apt-get install make cmake g++ libboost-dev libboost-system-dev \
  libboost-filesystem-dev libexpat1-dev zlib1g-dev \
  libbz2-dev libpq-dev libproj-dev lua5.2 liblua5.2-dev
```

Pour installer de manière simple osm2pgsql on peut passer par **apt-get**:

```
apt-get install osm2pgsql
```

Pour installer osm2pgsql sous Ubuntu en compilant les sources il faut télécharger sur github la dernière version :

```
git clone git://github.com/openstreetmap/osm2pgsql.git
```

Une fois les dépendances installées il faut générer le MakeFile:

```
mkdir build && cd build
cmake ..
```

Une fois le makeFile créé, exécuter:

```
make
```

Les fichiers compilés peuvent être installés avec:

```
sudo make install
```

Pour vérifier que osm2pgsql est installé il faut exécuter:

```
osm2pgsql --help
```

La réponse devra être:

```
osm2pgsql version 0.96.0 (64 bit id space)

Usage:
```

```
  osm2pgsql [options] planet.osm
  osm2pgsql [options] planet.osm.{pbf,gz,bz2}
  osm2pgsql [options] file1.osm file2.osm file3.osm

This will import the data from the OSM file(s) into a PostgreSQL database
suitable for use by the Mapnik renderer.

    Common options:
       -a|--append      Add the OSM file into the database without removing
                        existing data.
       -c|--create      Remove existing data from the database. This is the
                        default if --append is not specified.
       -l|--latlong     Store data in degrees of latitude & longitude.
       -m|--merc        Store data in proper spherical mercator (default).
       -E|--proj num    Use projection EPSG:num.
       -s|--slim        Store temporary data in the database. This greatly
                        reduces the RAM usage but is much slower. This switch is
                        required if you want to update with --append later.
       -S|--style       Location of the style file. Defaults to

/usr/local/Cellar/osm2pgsql/0.96.0_1/share/osm2pgsql/default.style.
       -C|--cache       Use up to this many MB for caching nodes (default: 800)
       -F|--flat-nodes  Specifies the flat file to use to persistently store node
                        information in slim mode instead of in PostgreSQL.
                        This file is a single > 40Gb large file. Only recommended
                        for full planet imports. Default is disabled.

    Database options:
       -d|--database    The name of the PostgreSQL database to connect to.
       -U|--username    PostgreSQL user name (specify passsword in PGPASS
                        environment variable or use -W).
       -W|--password    Force password prompt.
       -H|--host        Database server host name or socket location.
       -P|--port        Database server port.

A typical command to import a full planet is
    osm2pgsql -c -d gis --slim -C <cache size> -k \
      --flat-nodes <flat nodes> planet-latest.osm.pbf
where
    <cache size> is 20000 on machines with 24GB or more RAM
      or about 75% of memory in MB on machines with less
    <flat nodes> is a location where a 19GB file can be saved.

A typical command to update a database imported with the above command is
    osmosis --rri workingDirectory=<osmosis dir> --simc --wxc - \
      | osm2pgsql -a -d gis --slim -k --flat-nodes <flat nodes> -r xml -
where
    <flat nodes> is the same location as above.
    <osmosis dir> is the location osmosis replication was initialized to.

Run osm2pgsql --help --verbose (-h -v) for a full list of options.
```

Un appel pour charger les données dans la base de données gis pour le rendu serait

```
osm2pgsql --create --database gis data.osm.pbf
```

Cela chargera les données de data.osm.pbf dans les tables

- planet_osm_point,
- planet_osm_line,
- planet_osm_roads et
- planet_osm_polygon.

Pour importer l'extraction complète de OpenStreetMap pour la France la commande utilisée est:

```
wget http://download.geofabrik.de/europe/france-latest.osm.pbf
// une fois le téléchargement terminé on peut importer le fichier dans la base de
données.

osm2pgsql -c -d webmapping -U postgres -W  -C 20000 france-latest.osm.pbf
```

Où

- -d => webmapping est le nom de la base de données
- -c donne l'instruction create
- -W force la demande de mot de passe
- -U postrgres est le nom d'utilisateur
- _C = vaut 75% de la mémoire en MiB, avec un maximum de 30000. 20000 semble correct.

Installation de geoserver

Les API

Introduction

Il existe un certain nombre de site proposant des API de routing ou geocoding sur internet. Les services gratuits sont bien souvent assortis de limite ne permettant pas d'utiliser ces services de manière industrielle. Quant aux services sans limite leur coût est bien souvent très important.

Toutes les routes des API sont testées avec le logiciel POSTMAN qui est un client WEB/API.

Mise en place des API avec Docker

Le routing

Les algorithmes utilisés

De nombreux algorithmes de recherche de chemin dans un graphe existent, et leur utilisation se limite pas à du calcul d'itinéraire routier. Une application fréquente de ces algorithmes est par exemple le routage dans un réseau informatique, pour sélectionner les meilleurs chemins permettant d'acheminer des données. Plusieurs algorithmes courants de recherche du plus court chemin dans un graphe sont présentés ci-après.

A* (A star)

L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique. C'est un algorithme simple, ne nécessitant pas de prétraitement, et ne consommant que peu de mémoire.

Dijkstra

L'algorithme de Dijkstra permet de résoudre un problème algorithmique : le problème du plus court chemin. Ce problème a plusieurs variantes. La plus simple est la suivante : étant donné un graphe non-orienté, dont les arêtes sont munies de poids, et deux sommets de ce graphe, trouver un chemin

entre les deux sommets dans le graphe, de poids minimum. L'algorithme

de Dijkstra permet de résoudre un problème plus général : le graphe peut

être orienté,

et l'on peut désigner un unique sommet, et demander d'avoir la liste

des plus courts chemins pour tous les autres nœuds du graphe.

Les outils

Pour le routing une bibliothèque est plébiscité par l'ensemble de la communauté des utilisateurs des services géographiques. Il s'agit d'OSRM.

L'**Open Source Routing Machine** ou **OSRM** est l'implémentation C++ d'un moteur de recherche d'itinéraire haute performance afin d'obtenir les plus courts chemins dans un réseau routier. OSRM est en développement actif. Sur Github le dernier commit date de 13 jours. OSRM est actuellement maintenu par Dennis Luxen docteur en recherche informatique au sein du KIT le Karlruhe Institute of Technology.

Ce moteur de routage est prévu de fonctionner avec les données de OpenStreetMap.

Les services suivants sont disponible via une API en HTTP:

- Nearest - Enregistre les coordonnées sur le réseau de rues et renvoie les correspondances les plus proches.
- Route - Trouve l'itinéraire le plus rapide entre les coordonnées
- Table - Calcule la durée ou les distances de l'itinéraire le plus rapide entre toutes les paires de coordonnées fournies
- Trip - Résout le problème du voyageur de commerce en utilisant une heuristique gloutonne
- Tiles - Génère des tuiles vectorielles Mapbox avec des métadonnées de routage internes

Contrairement à la plupart des serveurs de routage, OSRM n'utilise pas l'algorithme A* pour calculer le plus court chemin, mais mais le théorème de la contraction des hiérarchies. Il en résulte des temps de requêtes très rapides, généralement inférieure à 1 ms pour les ensembles de données comme l'Europe, faisant OSRM un bon candidat pour les applications et sites sensibles routage sur le Web.

Installation de OSRM

Première chose à faire pour tester OSRM il faut télécharger des données de open street map au format pbf.

Ces données sont librement téléchargeable sur le site de

`http://download.geofabrik.de/`

On utilise la commande suivante dans le répertoire de travail:

```
wget http://download.geofabrik.de/europe/france-latest.osm.pbf
```

OSRM ayant était conteneurisée avec docker il faut initialiser un conteneur docker avec la commande suivante:

```
docker run -t -v "${PWD}:/data" osrm/osrm-backend osrm-extract -p /opt/car.lua
/data/france-latest.osm.pbf
```

Cette commande va extraire du fichier natif de OSM les fichiers contenant les notes, way et relations permettant d'effectuer le calcul de routing.

Cette extraction se fait en utilisant un profil de véhicule . Par défaut il existe trois profiles dans OSRM:

- Véhicule standard (voiture)
- Piéton
- Cycliste

Ces profiles sont utilisé lors de préprossessing c'est à dire lors de l'extraction des données du fichier OSM. C'est profiles sont écrit en LUA car ce sont de véritables script de configuration. Dans ces profiles les poids, vitesses et tailles des véhicules peuvent être particularisés.

L'opération d'extraction nécessite beaucoup de mémoire et de temps.

A titre d'exemple pour le fichier paca-latest.osm.pbf de 260 Mo le serveur consomme en RAM: *1 Go*

Apres l'extraction on exécute:

```
docker run -t -v "${PWD}:/data" osrm/osrm-backend osrm-partition /data/france-
latest.osrm
```

Ensuite:

```
docker run -t -v "${PWD}:/data" osrm/osrm-backend osrm-customize /data/france-
latest.osrm
```

Il faut ensuite lancer une instance du conteneur docker avec le serveur:

```
docker run -t -i -p 5000:5000 -v "${PWD}:/data" osrm/osrm-backend osrm-routed --
algorithm mld /data/france-latest.osrm
```

Une fois le serveur en fonction on peut tester une requête GET avec PostMan:

```
http://localhost:5000/route/v1/driving/5.9501,43.1194;5.9655,43.113?
steps=true&geometries=geojson
```

La réponse (tronquée) du serveur sera du type:

La réponse en JSON sera:

```
{
    "code": "Ok",
    "routes": [
        {
            "geometry": {
                "coordinates": [
                    [
                        5.950025,
                        43.119348
                    ],
                    [
                        5.950102,
                        43.119252
                    ],
                    [
                        5.950346,
                        43.119312
                    ...
                      ...
                      ...
                      ]


                            },
                            "mode": "driving",
                            "duration": 63,
                            "maneuver": {
                                "bearing_after": 120,
                                "type": "turn",
                                "modifier": "right",
                                "bearing_before": 70,
                                "location": [
                                    5.950346,
                                    43.119312
                                ]
                            },
                            "weight": 63,
                            "distance": 684.6,
                            "name": "Avenue François Nardi"
                        },
                        {
                            "intersections": [
                                {
                                    "out": 1,
                                    "location": [
                                        5.957094,
                                        43.11611
                                    ],
                                    "bearings": [
                                        90,
                                        210,
                                        315
                                    ],
                                    "entry": [
                                        true,
```

```
                                    true,
                                    false
                                ],
                                "in": 2
                            },

                    ],
                    "distance": 1638.3,
                    "duration": 172.1,
                    "summary": "Avenue François Nardi, Avenue de la Résistance",
                    "weight": 172.1
                }
            ],
            "distance": 1638.3,
            "duration": 172.1,
            "weight_name": "routability",
            "weight": 172.1
        }
    ],
    "waypoints": [
        {
            "hint":
"a5AAgMWRAIBPAAAABwAAAAsAAAAFAAAAgcmuQmFz9UC910ZBN32mQE8AAAAHAAAACwAAAAUAAAD4AgAAScpaAPTykQKUyloAKPORAgEAjwj51xz2",
            "distance": 8.403752,
            "name": "Boulevard Glassendi",
            "location": [
                5.950025,
                43.119348
            ]
        },
        {
            "hint":
"5JAAgDuQAIAZAAAAEAAAADEAAACHAAAAIGzmQdp5hkGQj1lChrwWQxkAAAAQAAAAMQAAAIcAAAD4AgAA0AZbAEDakQK8BlsAKNqRAgMAnwf51xz2",
            "distance": 3.123755,
            "name": "Avenue de la Résistance",
            "location": [
                5.96552,
                43.113024
            ]
        }
    ]
}
```


Mise en oeuvre de OSRM

Toutes les requêtes HTTP OSRM utilisent une structure commune.

```
GET/{service}/{version}/{profile}/{coordinates}[.{format}]?
option=value&option=value
```

Requête:

Paramètres	Description
Service	L' une des valeurs suivantes: route , nearest , table , match , trip , tile
version	Version du protocole implémenté par le service. v1 pour toutes les installations OSRM 5.x
profile	Le mode de transport, est déterminé statiquement par le profil Lua utilisé pour préparer les données à l'aide de osrm-extract . Généralement car , bike ou foot si vous utilisez l'un des profils fournis.
coordinates	Chaîne de format {longitude},{latitude};{longitude},{latitude}[;{longitude},{latitude} ...] ou polyline({polyline}) or polyline6({polyline6}) .
format	Seul json est pris en charge pour le moment. Ce paramètre est optionnel et vaut par défaut json .

Réponses:

Voici les réponses possibles:

Type	La description
Ok	La demande pourrait être traitée comme prévu.
InvalidUrl	La chaîne d'URL n'est pas valide.
InvalidService	Le nom du service est invalide.
InvalidVersion	La version est introuvable.
InvalidOptions	Les options ne sont pas valides.
InvalidQuery	La chaîne de requête est incorrectement synchronisée.
InvalidValue	Les paramètres de requête analysés avec succès ne sont pas valides.
NoSegment	L'une des coordonnées d'entrée fournies n'a pas pu s'aligner sur le segment de rue.
TooBig	La taille de la demande enfreint l'une des restrictions de taille de la demande spécifique au service.

Les Services rendu par l'API

Nearest:

A partir d'une coordonnées en latitude et longitude OSRM recherche et renvoie les n correspondances les plus proche.

Le couple `coordinates` prend uniquement comme valeur `{longitude},{latitude}` .

L'option pour ce service :

Option	Valeurs	Description
number	`integer >= 1` (par défaut `1`)	Nombre de réponses renvoyée.

Response

- si la demande aboutie `code: Ok` est renvoyé.
- `waypoints` tableau d'objet de type `Waypoint` triés par distance croissance par rapport aux coordonnées entrées. Chaque objet comporte aussi l'objet `nodes` qui est un tableau des `id` de OpenStreetMap.

GET

`http://{server}/nearest/v1/{profile}/{coordinates}.json?number={number}`

Exemple d'une requête

```
# Querying nearest three snapped locations of `13.388860,52.517037` with a bearing
between `20° – 340°`.
curl 'http://router.project-osrm.org/nearest/v1/driving/13.388860,52.517037?
number=3&bearings=0,20'
```

Exemple d'une réponse


```
{
   "waypoints" : [
      {
         "nodes": [
            2264199819,
            0
         ],
         "hint" :
"KSoKADRYroqUBAEAEAAAABkAAAAGAAAAAAAAABhnCQCLtwAA_0vMAKlYIQM8TMwArVghAwEAAQH1a66g",
         "distance" : 4.152629,
         "name" : "Friedrichstraße",
         "location" : [
            13.388799,
            52.517033
         ]
      },
      {
         "nodes": [
            2045820592,
            0
         ],
         "hint" :
"KSoKADRYroqUBAEABgAAAAAAAAAAAAAAKQAAABhnCQCLtwAA7kvMAAxZIQM8TMwArVghAwAAAQH1a66g",
         "distance" : 11.811961,
         "name" : "Friedrichstraße",
         "location" : [
            13.388782,
            52.517132
         ]
      },
      {
         "nodes": [
            0,
            21487242
         ],
         "hint" :
"KioKgDbbDgCUBAEAAAAAABoAAAAAAAAAPAAAABlnCQCLtwAA50vMADJZIQM8TMwArVghAwAAAQH1a66g",
         "distance" : 15.872438,
         "name" : "Friedrichstraße",
         "location" : [
            13.388775,
            52.51717
         ],
      }
   ],
   "code" : "Ok"
}
```

Route

Trouver la route la plus rapide .

Les différentes options possibles sont:

Option	Valeurs	Description
alternatives	`true` , `false` (par défaut), or Number	Cherche des routes alternatives. Si `alternatives=n` la recherche renvoie le nombre de route spécifié.
steps	`true` , `false` (défaut)	Retourne pour chaque segment toutes les étapes en latitude et longitude
annotations	`true` , `false` (défaut), `nodes` , `distance` , `duration` , `datasources` , `weight` , `speed`	Retourne des metadata supplémentaire pour chaque segment.
geometries	`polyline` (défaut), `polyline6` , `geojson`	Format des géometries
overview	`simplified` (défaut), `full` , `false`	Affiche un paramètre supplémentaire d'affichage en fonction du zoom
continue – straight	`default` (défaut), `true` , `false`	Force la route au point de passage à rester droite, même s'il y a perte de temps.
waypoints	`{index};{index};{index}...`	Permet de modifier l'ordre du cheminement

La réponse

- `code: OK` Si la requête est validée.
- `waypoints` : Tableau d'objet `Waypoint` :
- `routes` : Tableau d'objet `Route` , classé dans l'ordre du rank.

En cas d'erreur en plus du `code` l'information `no route` est ajoutée:

GET

```
/route/v1/{profile}/{coordinates}?alternatives={true|false|number}&steps={true|false}&geometries={polyline|polyline6|geojson}&overview={full|simplified|false}&annotations={true|false}
```

Exemple de requête

```
# Query on Berlin with three coordinates and no overview geometry returned:
curl 'http://router.project-
osrm.org/route/v1/driving/13.388860,52.517037;13.397634,52.529407;13.428555,52.5232
19?overview=false'
```

Trip

Le service **TRIP** permet de résoudre le problème du voyageur de commerce. Il est a noter que toutes les coordonnées d'entrée doivent être connectées pour que le service de déplacement fonctionne

En plus des options générales les options suivantes sont supportée par le service `TRIP` :

Option	Values	Description
roundtrip	`true` (défaut), `false`	L'itinéraire retourné est un aller-retour (l'itinéraire retourne au premier emplacement)
source	`any` (défaut), `first`	L'itinéraire retourné commence à n'importe quelle coordonnées `any` ou à lapremière coordonnée `first`
destination	`any` (défaut), `last`	L'itinéraire retourné se termine à la coordonnée `any` ou `last`
steps	`true` , `false` (défaut)	Pour chaque segment retourne les instructions
annotations	`true` , `false` (défaut), `nodes` , `distance` , `duration` , `datasources` , `weight` , `speed`	Retourne des metadata supplémentaire pour chaque segment.
geometries	`polyline` (default), `polyline6` , `geojson`	Format des géometries.
overview	`simplified` (default), `full` , `false`	Affiche un paramètre supplémentaire d'affichage en fonction du zoom.

Imposer les points de départ et d'arrivé

Il est possible de définir explicitement les coordonnées de début ou de fin de la tournée. Lorsque `source` est défini sur `first` , la première coordonnée est utilisée comme coordonnée de début du trajet dans la sortie. Lorsque la destination est définie sur `last` , la dernière coordonnée sera utilisée comme destination dans la sortie renvoyée. Si on spécifie `any` , n'importe laquelle des coordonnées peut être utilisée comme première ou dernière coordonnée dans la sortie.

Cependant, si `source=any&destination=any` la tournée débutera à la première coordonnée par défaut.

Actuellement toutes les combinaison de `roundtrip` , `source` and `destination` ne sont pas supportées.

Seules les combinaisons suivantes peuvent être utilisées:

roundtrip	source	destination	supported
true	first	last	**oui**
true	first	any	**oui**
true	any	last	**oui**
true	any	any	**oui**
false	first	last	**oui**
false	first	any	Non
false	any	last	Non
false	any	any	Non

- `` `code: OK `` Si la requête est validée.
- `waypoints` : Tableau d'objet `Waypoint` :
  - `waypoint_index` : index des point de passage.
- `trips` : tableau des objets `Route` formant la route

En cas d'erreur en plus du `code` les informations suivantes seront ajoutées:

Type	Description
`NoTrips`	Pas de tournée possible.
`NotImplemented`	Cette demande n'est pas supportée

GET

```
 /trip/v1/{profile}/{coordinates}?roundtrip=
{true|false}&source{any|first}&destination{any|last}&steps=
{true|false}&geometries={polyline|polyline6|geojson}&overview=
{simplified|full|false}&annotations={true|false}'
```

Exemple de requête

```
# Round trip in Berlin with three stops:
curl 'http://router.project-
osrm.org/trip/v1/driving/13.388860,52.517037;13.397634,52.529407;13.428555,52.52321
9'
# Round trip in Berlin with four stops, starting at the first stop, ending at the
last:
curl 'http://router.project-
osrm.org/trip/v1/driving/13.388860,52.517037;13.397634,52.529407;13.428555,52.52321
9;13.418555,52.523215?source=first&destination=last'
```

Tile service

Ce service génère des [tuiles vectorielles Mapbox] (https://www.mapbox.com/developers/vector-tiles/) qui peuvent être visualisées avec une bibliothèque prenant en charge les tuiles vectorielles. Les tuiles contiennent des géométries de route et des métadonnées qui peuvent être utilisées pour afficher le graphe de routage. Les mosaïques sont générées directement à partir des données en mémoire. Elles sont donc synchronisées avec les résultats de routage réels.

Les valeurs `x` , `y` et `zoom` sont identiques à celles décrites à l'adresse https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames.

L'objet de réponse est soit un blob codé binaire avec un `Content-Type` de `application / x-protobuf` , soit une erreur `404` .

OSRM est codé en dur pour ne renvoyer que les tuiles à partir du niveau de zoom 12 ou supérieur (pour éviter de renvoyer accidentellement des tuiles vectorielles extrêmement grandes).

GET

`/tile/v1/{profile}/tile({x},{y},{zoom}).mvt`

Exemple de requête

```
# This fetches a Z=13 tile for downtown San Francisco:
curl 'http://router.project-osrm.org/tile/v1/car/tile(1310,3166,13).mvt'
```

Le geocoding

Le gouvernement français mets à la disposition des développeurs un répertoire github (https://github.com/etalab) dans lequel on trouvera l'ensemble des produits et API développé par ETALAB.

La mission Etalab fait partie de la Direction interministérielle du numérique et du système d'information et de communication de l'Etat (DINSIC), dont les missions et l'organisation sont fixées par les décrets du 20 Novembre 2017.

Etalab est une mission placée au sein de la Direction interministérielle du numérique et du système d'information et de communication de l'État (DINSIC).

Etalab améliore le service public :

- en menant une politique de la donnée (ouverture et partage des données publiques ou "open data", exploitation des données et intelligence artificielle...). Etalab met notamment en œuvre les missions de l'Administrateur général des données (AGD), fixées par le décret n° 2014-1050 du 16 septembre 20141.
- en promouvant une action publique plus transparente et collaborative grâce au numérique, pour tendre vers un gouvernement ouvert.

Etalab développe et maintient le portail des données ouvertes du gouvernement français data.gouv.fr.

Etalab contribue à la promotion des sciences des données (« datasciences ») et de l'intelligence artificielle dans la sphère publique, en menant des projets, en facilitant l'expérimentation et le partage de bonnes pratiques, en animant des communautés de datascientists et porteurs de projet d'IA.

Pour palier aux limites imposées sur leur API ETALAB propose en téléchargement libre le code source de leur moteur de coding à l'adresse suivante:

`https://github.com/etalab/addok`

Addok fonctionne avec Redis en back-end.

- il importe et indexe les données (et peut aussi être importé depuis une base de données Nominatim) en ligne de commande
- il sert une API basée sur du GeoJSON (en utilisant Werkzeug et Gunicorn)
- il fait du géocodage inversé ("reverse geocoding")
- il fait du géocodage par batch (via du CSV pour le moment)
- il fournit une ligne de commande de debug pour inspecter l'index
- il est configurable sur de nombreux détails
- il est open source.

Pré-requis

- Au moins 6 Go de RAM disponible (à froid)
- 8 Go d'espace disque disponible (hors logs)Installer une instance avec les données de la Base Adresse Nationale en ODbL

Installation

Tout d'abord il faut se placer dans un dossier de travail par exemple `ban` .

```
mkdir ban && cd ban
```

Télécharger les données pré-indexées

```
wget https://adresse.data.gouv.fr/data/geocode/ban_odbl_addok-latest.zip
```

Décompresser l'archive

```
mkdir addok-data
unzip -d addok-data ban_odbl_addok-latest.zip
```

Télécharger le fichier Compose

```
wget https://raw.githubusercontent.com/etalab/addok-docker/master/docker-
compose.yml
```

Démarrer l'instance

Suivant votre environnement, `sudo` peut être nécessaire pour les commandes suivantes.

```
# Attachée au terminal
docker-compose up

# ou en arrière-plan
docker-compose up -d
```

Suivant les performances de la machine, l'instance mettra entre 30 secondes et 2 minutes à démarrer effectivement, le temps de charger les données dans la mémoire vive.

- 90 secondes sur une VPS-SSD-3 OVH (2 vCPU, 8 Go)
- 50 secondes sur une VM EG-15 OVH (4 vCPU, 15 Go)

Par défaut l'instance écoute sur le port `7878` .

Tester l'instance

```
curl "http://localhost:7878/search?q=1+rue+de+la+paix+paris"
```

Mise en oeuvre de l'API

/search/

Point d'entrée pour le géocodage.

Utiliser le paramètre **q** pour faire une recherche plein texte:

```
curl https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port
```

Avec **limit** on peut contrôler le nombre d'éléments retournés:

```
curl https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port&limit=15
```

Avec **autocomplete** on peut désactiver les traitements d'auto-complétion:

```
curl https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port&autocomplete=0
```

Avec **lat** et **lon** on peut donner une priorité géographique:

```
curl https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port&lat=48.789&lon=2.789
```

Les filtres **type**, **postcode** (code Postal) et **citycode** (code INSEE) permettent de restreindre la recherche:

```
curl https://api-adresse.data.gouv.fr/search/?q=8+bd+du+port&postcode=44380
curl https://api-adresse.data.gouv.fr/search/?q=paris&type=street
```

Le retour est un geojson *FeatureCollection* respectant la spec GeoCodeJSON:


```
{
  "attribution": "BAN",
  "licence": "ODbL 1.0",
  "query": "8 bd du port",
  "type": "FeatureCollection",
  "version": "draft",
  "features": [
    {
      "properties": {
        "context": "80, Somme, Picardie",
        "housenumber": "8",
        "label": "8 Boulevard du Port 80000 Amiens",
        "postcode": "80000",
        "citycode": "80021",
        "id": "ADRNIVX_0000000260875032",
        "score": 0.3351181818181818,
        "name": "8 Boulevard du Port",
        "city": "Amiens",
        "type": "housenumber"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          2.29009,
          49.897446
        ]
      },
      "type": "Feature"
    },
    {
      "properties": {
```

```
        "context": "34, Hérault, Languedoc-Roussillon",
        "housenumber": "8",
        "label": "8 Boulevard du Port 34140 Mèze",
        "postcode": "34140",
        "citycode": "34157",
        "id": "ADRNIVX_0000000284423783",
        "score": 0.3287575757575757,
        "name": "8 Boulevard du Port",
        "city": "Mèze",
        "type": "housenumber"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          3.605875,
          43.425232
        ]
      },
      "type": "Feature"
    }
  ]
}
```


Les attributs retournés sont :

- *id* : identifiant de l'adresse (non stable: actuellement identifiant IGN)
- *type* : type de résultat trouvé
  - *housenumber* : numéro « à la plaque »
  - *street* : position « à la voie », placé approximativement au centre de celle-ci
  - *locality* : lieu-dit
  - *municipality* : numéro « à la commune »
- *score* : valeur de 0 à 1 indiquant la pertinence du résultat
- *housenumber* : numéro avec indice de répétition éventuel (bis, ter, A, B)
- *name* : numéro éventuel et nom de voie ou lieu dit
- *postcode* : code postal
- *citycode* : code INSEE de la commune
- *city* : nom de la commune
- *context* : n° de département, nom de département et de région
- *label* : libellé complet de l'adresse

/reverse/

Point d'entrée pour le géocodage inverse.

Les paramètres **lat** et **lon** sont obligatoires:

```
curl https://api-adresse.data.gouv.fr/reverse/?lon=2.37&lat=48.357
```

Le paramètre **type** permet forcer le type de retour:

```
curl https://api-adresse.data.gouv.fr/reverse/?lon=2.37&lat=48.357&type=street
```

Même format de réponse que pour le point d'entrée **/search/**.

/search/csv/

Point d'entrée pour le géocodage de masse à partir d'un fichier CSV.

Le fichier csv, encodé en UTF-8 et limité actuellement à 8Mo, doit être passé via le paramètre **data**:

```
curl -X POST -F data=@path/to/file.csv https://api-adresse.data.gouv.fr/search/csv/
```

Par défaut, toutes les colonnes sont concaténées pour constituer l'adresse qui sera géocodée. On peut définir les colonnes à utiliser via de multiples paramètres **columns**:

```
curl -X POST -F data=@path/to/file.csv -F columns=voie -F columns=ville
https://api-adresse.data.gouv.fr/search/csv/
```

Il est possible de préciser le nom d'une colonne contenant le **code INSEE** ou le **code Postal** pour limiter les recherches, exemple :

```
curl -X POST -F data=@path/to/file.csv -F columns=voie -F columns=ville -F
citycode=ma_colonne_code_insee https://api-adresse.data.gouv.fr/search/csv/
curl -X POST -F data=@path/to/file.csv -F columns=voie -F columns=ville -F
postcode=colonne_code_postal https://api-adresse.data.gouv.fr/search/csv/
```

Exemples

```
http -f POST http://localhost:7878/search/csv/ columns='voie' columns='ville'
data@path/to/file.csv
http -f POST http://localhost:7878/search/csv/ columns='rue' postcode='code postal'
data@path/to/file.csv
```

/reverse/csv/

Point d'entrée pour le géocodage inverse de masse à partir d'un fichier CSV.

Le fichier csv, encodé en UTF-8 et limité actuellement à 8Mo, doit être passé via le paramètre **data**. Il doit contenir les colonnes **latitude** (ou *lat*) et **longitude** (ou *lon* ou *lng*).

```
curl -X POST -F data=@path/to/file.csv https://api-
adresse.data.gouv.fr/reverse/csv/
```

Bibliographie et liens

Editions papier

- COLLADO David : Géomatique, Webmapping en Open Source aux éditions *ellipse*
- LACOVELLA Stefano: GeoServer Beginner's Guide - Second Edition: Share geospatial data using Open Source standards (English Edition): Édition Kindle
- YOUNGBLOOD Brian: GeoServer Beginner's Guide (English Edition): Édition Kindle

Sites internets

- http://project-osrm.org/
- https://adresse.data.gouv.fr/
- https://adresse.data.gouv.fr/api
- https://forum.etalab.gouv.fr/
- https://github.com/Project-OSRM/
- https://wiki.openstreetmap.org/wiki/FR:Open_Source_Routing_Machine
- https://fr.wikipedia.org/wiki/Géocodage_inversé
- https://fr.wikipedia.org/wiki/Géocodage
- https://fr.wikipedia.org/wiki/Routage
- https://github.com/openstreetmap/osm2pgsql