



# Documentation technique

Planificateur intelligent – Backend

Par

Robin CAVALIERI

1. Environnement de développement .....	3
2. Déployer le serveur back .....	3
3. Déployer le serveur MongoDB.....	3
4. L'architecture .....	4
5. Les API .....	4
6. Lancer Spark (Standalone & Cluster Nodes) .....	5
7. Reporting.....	6
8. Problèmes rencontrés.....	7
9. Améliorations .....	7

## 1. Environnement de développement

Pour développer la partie backend de ce projet, nous avons utilisé l'IDE PyCharm que vous pouvez télécharger à l'adresse ci-dessous :

<https://www.jetbrains.com/pycharm/>

Pour réaliser des tests sur notre API, nous avons utilisé POSTMAN :

<https://www.getpostman.com/>

Cet utilitaire vous permettra de requêter votre API et de visualiser les JSON que celle-ci vous retournera.

La base de données utilisée est MongoDB. C'est une base de donnée orientée documents que nous possédons en local, les VM sont actuellement indisponibles. Le nom de la base est **smartplanner\_users** et la collection dans laquelle sont classés les utilisateurs de l'application est nommée **user**. Initialement nous possédions une base de données distante hébergée sur le webservice MLab : <https://mlab.com/>. Cependant, le proxy nous empêche de réaliser des requêtes sur cette base distante. ITS nous a donc demandé d'installer une base locale.

## 2. Déployer le serveur back

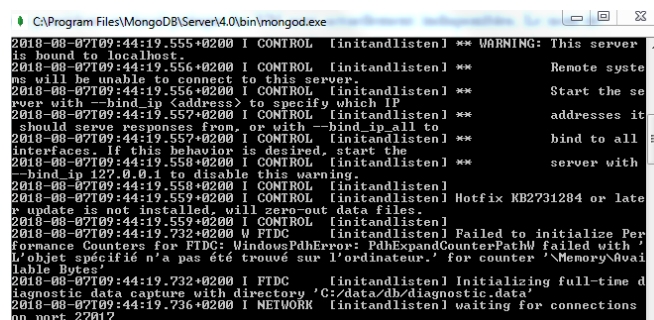
Pour déployer l'application back, il faut exécuter le script python *routes.py* présent dans le dossier **/backend/app** présent à la racine du projet.

Vous pouvez lancer ce script à partir d'un IDE ou par ligne de commande dans une console :

```
C:\Users\rcavalieri\Desktop\Projet Arch\backend\app>python routes.py
* Restarting with stat
* Debugger is active!
* Debugger PIN: 279-609-187
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

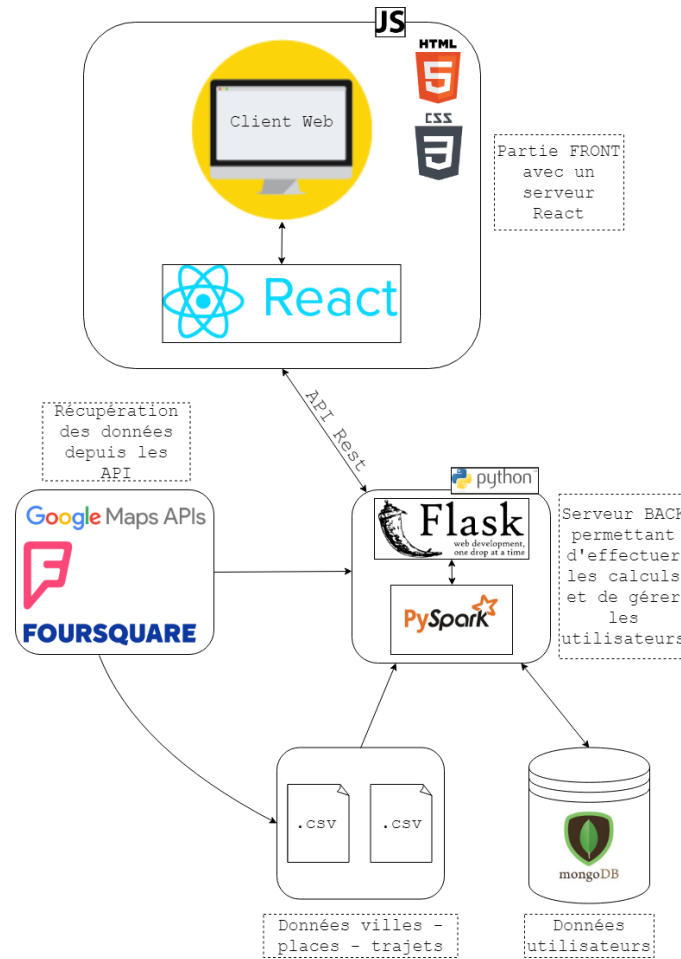
## 3. Déployer le serveur MongoDB

Une fois MongoDB installé dans votre **C:\Program Files\** Vous trouverez dans le dossier **MongoDB\Server\X.X\bin** un exécutable nommé **mongod**. Les services démarrent sur le port **27017** :



```
C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe
2018-08-07T09:44:19.555+0200 I CONTROL [initandlisten] ** WARNING: This server
is bound to localhost.
2018-08-07T09:44:19.556+0200 I CONTROL [initandlisten] ** Remote system
will be unable to connect to this server.
2018-08-07T09:44:19.556+0200 I CONTROL [initandlisten] ** Start the se
rver with --bind_ip <address> to specify which IP
2018-08-07T09:44:19.557+0200 I CONTROL [initandlisten] ** addresses it
should serve responses from, or with --bind_ip_all to
2018-08-07T09:44:19.557+0200 I CONTROL [initandlisten] ** bind to all
interfaces. If this behavior is desired, start the
2018-08-07T09:44:19.558+0200 I CONTROL [initandlisten] ** server with
--bind_ip 127.0.0.1 to disable this warning.
2018-08-07T09:44:19.558+0200 I CONTROL [initandlisten]
2018-08-07T09:44:19.559+0200 I CONTROL [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2018-08-07T09:44:19.559+0200 I CONTROL [initandlisten]
2018-08-07T09:44:19.732+0200 W FTDC [initandlisten] Failed to initialize Per
formance Counters for FIDC: WindowsPdhError: PdhExpandCounterPathW failed with
error 1060. The object specified n'a pas été trouvé sur l'ordinateur. for counter 'Memory\Avai
lable Bytes'
2018-08-07T09:44:19.732+0200 I FTDC [initandlisten] Initializing full-time d
iagnostic data capture with directory 'C:/data/db/diagnostic.data'
2018-08-07T09:44:19.736+0200 I NETWORK [initandlisten] waiting for connections
on port 27017
```

## 4. L'architecture



L'objectif de cette architecture était de scinder notre application Web en deux parties (le back et le front). Nous avons donc conçu dans un premier temps toutes les fonctions permettant de calculer nos trajets et nos recommandations puis les avons testés sur des templates Flask. Ensuite, nous avons débuté la séparation totale du back et du front en réalisant un serveur front basé sur React et un serveur back basé sur Flask API. Nous avons parsé à partir de scripts tous les JSON que nous avons récupérés à travers l'API Foursquare et l'API Google Maps avant que ceux-ci ne changent de politique de monétisation. Lors de chaque demande de trajet, l'application requête n fois l'API Google pour obtenir des informations sur les distances entre les lieux personnels de l'utilisateur et les villes que nous possédons en base. L'objectif est de calculer les temps et distances les plus courts.

## 5. Les API

Pour la récupération des données, nous avons utilisé l'API Google Maps et l'API Foursquare. N'essayez pas OpenStreetMap. Vous n'obtiendrez que très peu de réponses à vos requêtes (environ une sur trois).

Foursquare API nous a permis de récupérer des lieux d'intérêt dans chaque ville et de les stocker dans des fichiers CSV afin de les utiliser directement comme dataframes. Le nombre de requêtes est désormais très limité et nombreuses d'entre elles requièrent désormais un compte premium.

L'API Google Maps nous permet toujours d'effectuer des requêtes sans avoir activé la facturation. Cependant les requêtes sont très lentes à raison d'une réponse par seconde environ lorsque nous effectuons des tests à SOLUTEC contre moins de 0.4s à nos domiciles. Voici un exemple de requête nous permettant d'obtenir au format JSON nos données trajet :

```
link="https://maps.googleapis.com/maps/api/directions/json?origin="+str(lat_
json_data=requests.get(link)
```

Il vous faudra utiliser un TOKEN qui vous sera fourni par les services Google. Vous pouvez vous en procurer un à partir de votre compte Gmail par exemple.

## 6. Lancer Spark (Standalone & Cluster Nodes)

### STANDALONE (Windows)

Actuellement, nous avons passé nos script pyspark à pandas suite à des problèmes liés au serveur sur lequel étaient hébergées les VM. Durant les trois premiers mois du stage, nous avons codé notre algorithme de scoring /recommandation à partir de pyspark.

Dans un premier temps, il vous faut télécharger la version binaire de la distribution Spark 2.3.0 pour apache Hadoop 2.7 que vous trouverez à l'adresse ci-dessous :

<https://spark.apache.org/downloads.html>

Rien ne vous empêche de choisir l'une des dernières versions. Il vous faudra également définir les variables d'environnement suivantes :

```
PYSPARK_PYTHON C:\anaconda\python
SPARK_HOME C:\spark-2.3.0-bin-hadoop2.7
```

Mais aussi le **JAVA\_HOME** :

```
JAVA_HOME C:\Program Files\Java\jdk1.8.0_144
```

Déposez le dossier téléchargé à la racine **C:/**. Il vous faudra ouvrir un invité de commandes en mode administrateur et vous rendre dans le dossier **C:\spark-2.3.0-bin-hadoop2.7\bin** pour lancer la commande suivante :

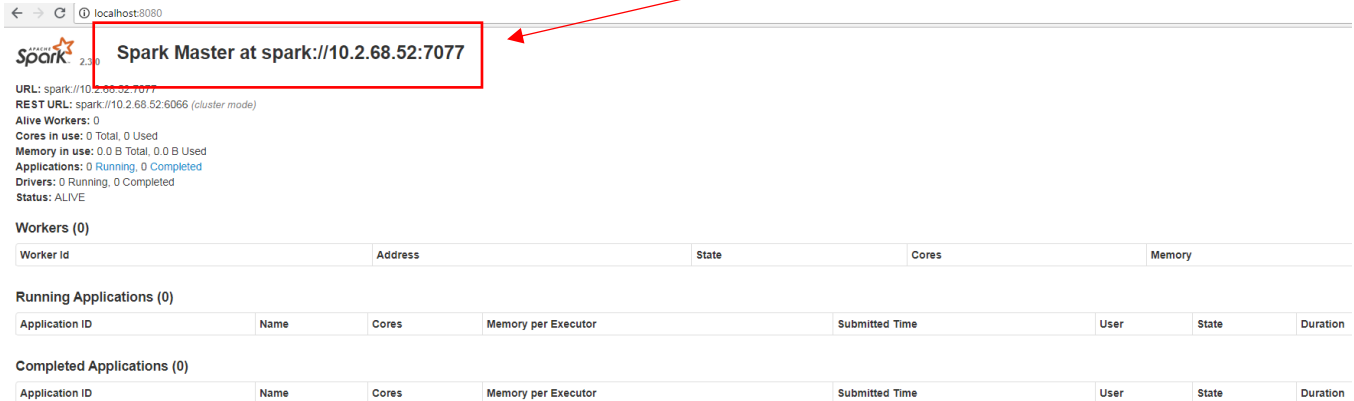
```
spark-class org.apache.spark.deploy.master.Master
```

En vous rendant sur votre navigateur et en saisissant l'adresse <http://localhost:8080> vous aurez accès à l'UI.

Pour ajouter un worker le principe est similaire mais avec la ligne de commande suivante:

```
spark-class org.apache.spark.deploy.worker.Worker spark://10.2.68.52:7077
```

Vous trouverez l'adresse de votre master sur l'UI :



The screenshot shows the Spark UI running in a web browser at localhost:8080. A red box highlights the text "Spark Master at spark://10.2.68.52:7077" in the top left corner. A red arrow points from the text "l'adresse de votre master sur l'UI" to this box. Below the header, the UI displays various status metrics: URL, REST URL, Alive Workers (0), Cores in use, Memory in use, Applications (0 Running, 0 Completed), Drivers (0 Running, 0 Completed), and Status (ALIVE). There are three main sections: "Workers (0)", "Running Applications (0)", and "Completed Applications (0)", each with a table header.

Worker Id	Address	State	Cores	Memory
-----------	---------	-------	-------	--------

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

Nous avons au début tenté d'installer une distribution Cloudera. Cependant avec les problèmes liés au proxy nous n'avons pas poursuivi nos démarches.

Avec cette méthode, votre worker est lié à votre Master. Vous pouvez désormais lancer un job.

## Cluster Nodes (Ubuntu)

Lors de notre première configuration (avant le crash de VM), nous avons suivi ce tutoriel très complet sur Ubuntu 16.04 :

<https://data-flair.training/blogs/install-apache-spark-multi-node-cluster/>

Nous vous recommandons d'en faire de même si des VM sont disponibles ou alors de réaliser une configuration sur la machine de calcul mise à disposition des stagiaires.

## 7. Reporting

A l'heure actuelle, nous parvenons à planifier des trajets respectant les contraintes utilisateur à partir de notre graphe. Nous envoyons le trajet planifié au format CSV à partir d'un GET. Les calculs de scoring sont réalisés dans le back à partir de l'appel de la fonction **get\_classement** dans le fichier **routes.py** (fonction form). Cette fonction fournira un classement des villes par affinité avec les tags rentrés par l'utilisateur.

Ensuite, le résultat de ce classement sera passé en argument de la fonction **get\_path**. Cette fonction vous fournira les villes à visiter avec leur id et leur nom ainsi que les heures de passage dans chacune d'entre elles.

Les résultats sont envoyés au front au format JSON une fois les calculs effectués. Les principes de fonctionnement sont expliqués dans mon rapport de stage. Je vous invite à le lire si vous souhaitez réutiliser nos algorithmes.

## 8. Problèmes rencontrés

Beaucoup de choses sont à déplorer notamment l'état des machines virtuelles qui ne nous permettent pas de travailler à 4 groupes sur un même serveur (problème de partage de ressources). Pour pallier à ce problème, vous disposerez d'une machine de calcul mise à disposition de tout le lab'Paris. Je vous invite donc à repenser l'architecture de façon à intégrer ce nouvel outil.

Les problèmes de proxy nous empêchent d'installer certains bundles de manière intempestive. C'est pourquoi je vous invite à configurer votre fichier **cntlm.ini** et parfois à redémarrer vos services à partir de la commande :

```
# net stop cntlm
```

```
# net start cntlm
```

Vous retrouverez un tutoriel pour configurer proprement cntlm en toute sécurité à l'adresse : **V:\Installation IDE – Gagner contre le proxy.**

Pour utiliser le gestionnaire de paquets npm, je vous conseille également de configurer les paramètres proxy de ce dernier une fois le projet React déployé. Rendez-vous à la racine de votre projet et saisissez la commande suivante :

```
# npm config set proxy http://user:password@10.2.4.2:8080
```

```
# npm config set https-proxy http://user:password@10.2.4.2:8080
```

Vos paquets seront désormais téléchargés en passant par le proxy de SOLUTEC.

Notre application est basée sur un nombre de requêtes important sur l'API Google. Cependant, les requêtes sont très longues. Un chargement de carte peut prendre jusqu'à 5 min à calculer contre 1 à notre domicile. Nous vous conseillons d'opter pour une autre API ou un autre type de service de planification.

## 9. Améliorations

A l'heure actuelle, l'application peut :

- Calculer des recommandations
- Planifier des trajets
- Calculer le temps de transport
- Gérer les utilisateurs (connexion, déconnexion, modification du profil)

Les améliorations possibles sont :

- Intégrer les trajets en transports en commun (API SNCF ou autres)
- Amélioration du front pour réaliser une application web responsive
- Rescripser le calcul de recommandation en pyspark