

# Rapport du projet de PWEB

David BenKhemis	207
Benjamin Pedron	205
Fantin Raimbault	205
Nicolas Tahon	206

## Sommaire

- Présentation générale
  - Principe de l'application
  - Type d'application et d'IHM
  - Quelle API
  - Design
- Technicité du projet
  - Front-end
  - Gestion de données JSON
  - Gestion de la carte
  - Animation de l'application, Animation sur la carte
  - Architecture projet
- Implantation du projet
  - Descriptions des fichiers et répertoires
  - Procédure d'installation et de lancement

# I Présentation générale :

## Principe de l'application

Notre application permet de visualiser toutes gares de France et les voies empruntées par les trains. Les gares comprises dans l'application sont les gares SNCF, cela inclut donc les transiliens ainsi que le RER C, D et E.

On peut choisir à l'aide d'un bouton d'afficher les gares voyageurs ou les gares de fret (marchandise).

Pour chaque gare on peut cliquer dessus afin d'afficher les prochains départ et on recense le nombre d'objets trouvés.

De plus l'application permet d'afficher la température à un endroit précis en fonction du clic de l'utilisateur.

## Type d'application

Nous avons choisi de faire une application à page unique afin de la rendre simple d'utilisation et ne pas perturber l'utilisateur ainsi que pour faciliter l'ergonomie. Une seule page est donc nécessaire pour afficher les différents services proposés par notre API.

## API

Les API utilisées pour ce projet ont été : deux APIs SNCF, OpenWeatherMap et l'API Node.JS que nous avons développé.

Nous avons choisi de stocker plus de types de données que celles qu'on utilise comme la commune ou le département pour chaque gare qui ne sont pas encore utilisés mais qui pourraient nous servir dans le futur pour diverses améliorations de l'application.

## Design

En terme de design, nous avons gardé un aspect épuré en utilisant une carte en dégradé de couleurs. Nous avons placé 3 boutons, un pour chacune des fonctionnalités de l'application qui sont représenté par une icône qui se veut explicite. Nous avons mis une carte à l'aspect vieilli en fond d'écran afin de rester dans la continuité du projet.

Chaque gare est représenté par un cercle rouge, les voies par une ligne rouge. On peut cliquer sur chaque gare et un pop-up affichera les informations sur les prochains départ de la gare ainsi que le nombre d'objets trouvés en 2020.

## II Technicité du projet :

### Front-end

Dans notre dossier front-end nous avons un dossier avec à l'intérieur un fichier JS permettant d'utiliser l'API leaflet, un fichier .html (single page application), un fichier CSS et enfin notre script JS qui permet de communiquer avec notre API nodeJS. Le fichier récupère les données via les routes implémentées dans notre back-end. Les données manipulées sont toutes sous format JSON.

### Animation de l'application

Nous avons mis une animation de chargement lorsque l'on effectue une requête à l'API afin de montrer à l'utilisateur le chargement. Pendant l'animation, on désactive également les clics sur la carte pour ne pas que le chargement soit trop long.

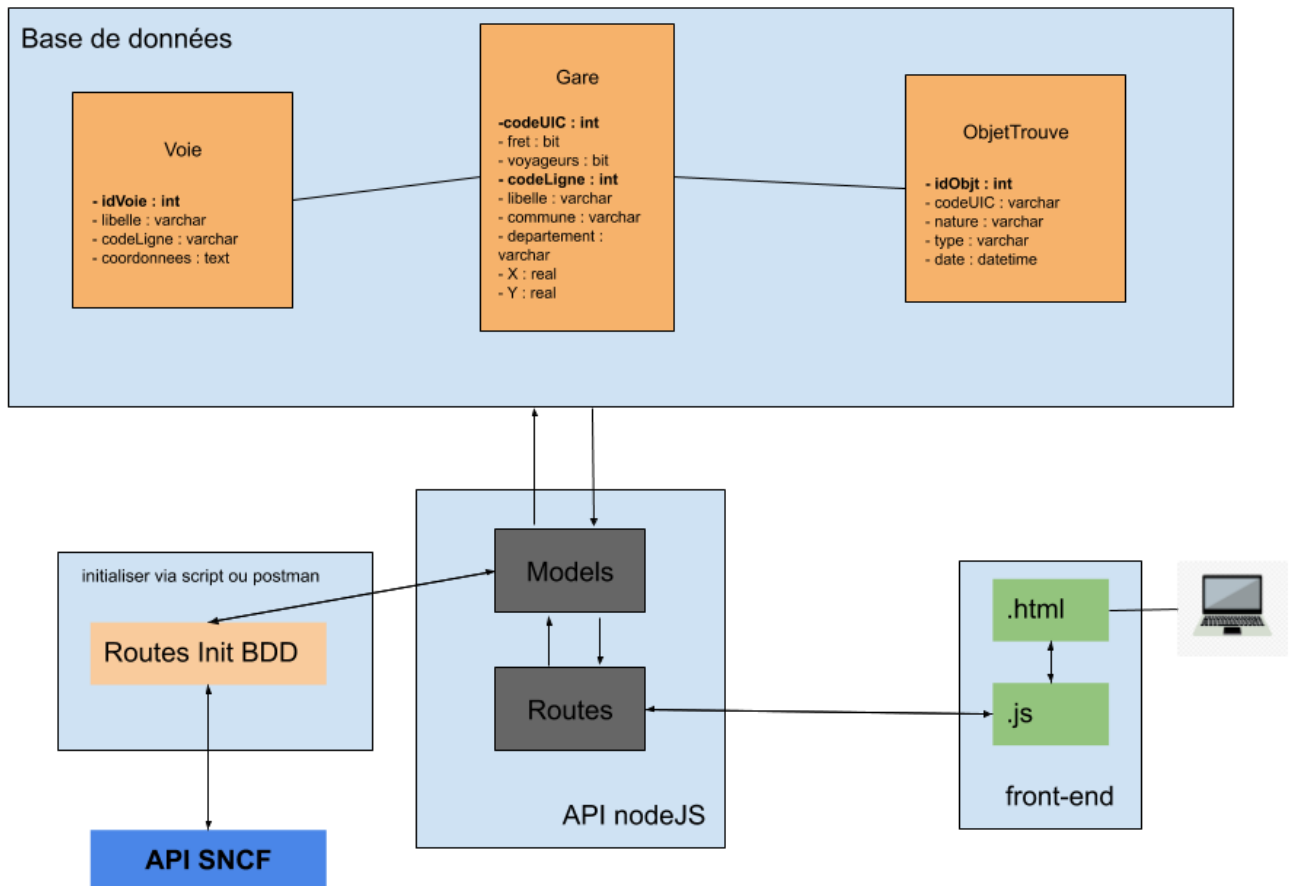
### Gestion de la carte

Nous ne chargeons la page qu'une seule fois, les actions de l'utilisateur déterminent l'affichage ou non de nos services sur cette carte. Chaque composant leaflet est stocké dans un tableau correspondant que nous pouvons manipuler à notre guise !

Nous avons utilisé Leaflet Provider (<https://github.com/leaflet-extras/leaflet-providers>) pour gérer les différents rendus de la carte (chargement, et normal). Voici par exemple, le chargement, la démo de cet outil est disponible ici : <https://leaflet-extras.github.io/leaflet-providers/preview/>



## Architecture projet



### III Implantation du projet :

Nous avons décidé de créer notre propre API pour ces raisons :

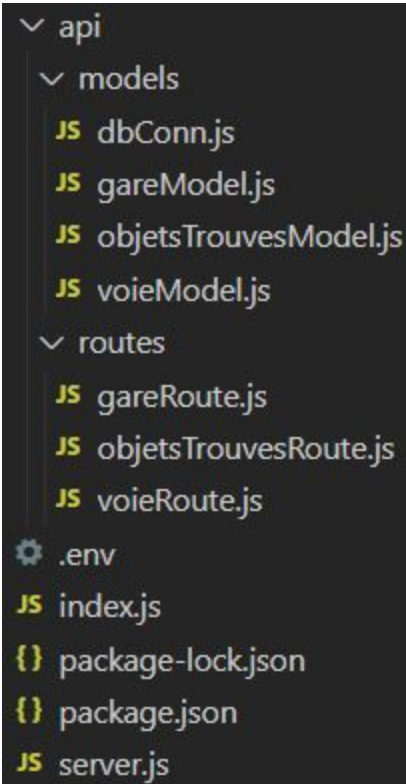
- Ne pas être dépendant à 100% des APIs externes
- Optimiser le temps de traitement pour certains services
- Réduire le nombre de requêtes envoyées aux APIs externes
- Uniformiser les données récupérées pour les utiliser

En effet, pour certains services développés, nous étions obligé de passer par cette gestion car des limites de requêtes sont imposées par les fournisseurs de services que nous utilisons (une des APIs SNCF par exemple limite le nombre de requêtes à 15.000 par jour , au vu de la taille des données que nous utilisons, ce chiffre ne nous est pas suffisant pour un affichage en temps réel).

Voici l'organisation de nos projets :

L'API Node.JS :



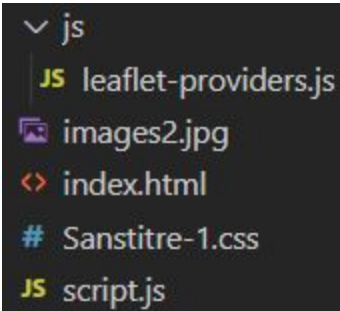


```

  api
  ├── models
  │   ├── dbConn.js
  │   ├── gareModel.js
  │   ├── objetsTrouvesModel.js
  │   └── voieModel.js
  ├── routes
  │   ├── gareRoute.js
  │   ├── objetsTrouvesRoute.js
  │   └── voieRoute.js
  ├── .env
  ├── index.js
  ├── package-lock.json
  ├── package.json
  └── server.js

```

Le front :



```

  js
  ├── leaflet-providers.js
  ├── images2.jpg
  ├── index.html
  ├── Sanstitre-1.css
  └── script.js

```

## Lancer le projet :

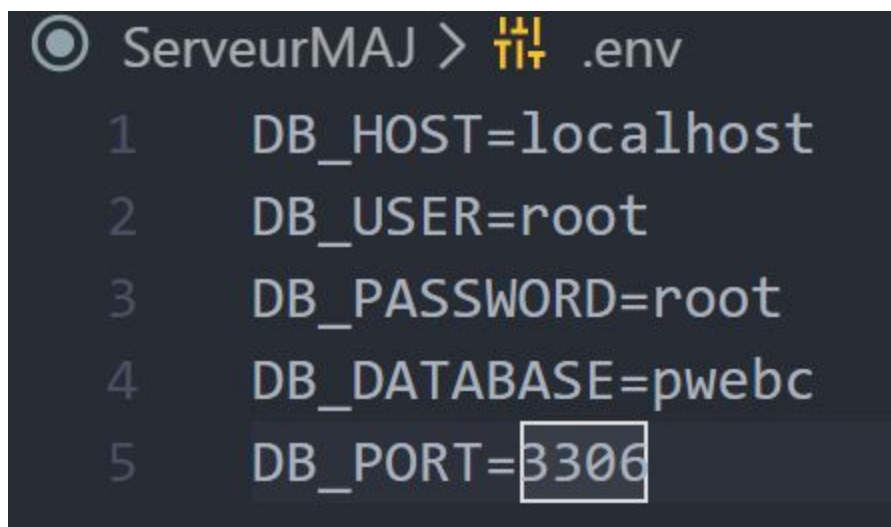
Nous avons utilisé mysql workbench pour la BDD.

<https://dev.mysql.com/downloads/windows/installer/8.0.html>

Le script de la BDD est bien évidemment fournis.

\*\*\*\*(descendez plus bas pour pallier à un problème durant l'installation de mysql workbench)\*\*\* (page8)

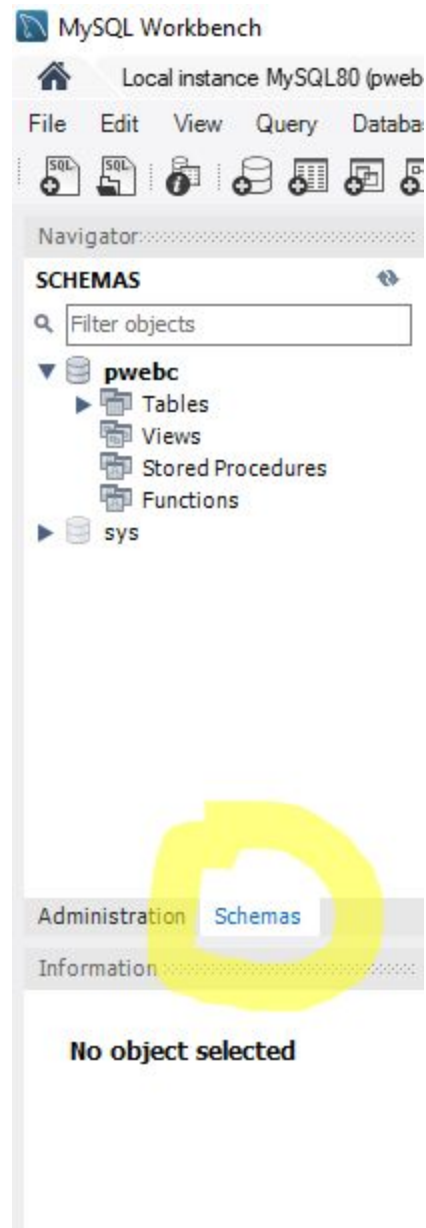
Notre .env ressemble à cela :



```
ServeurMAJ > .env
1  DB_HOST=localhost
2  DB_USER=root
3  DB_PASSWORD=root
4  DB_DATABASE=pwebc
5  DB_PORT=3306
```

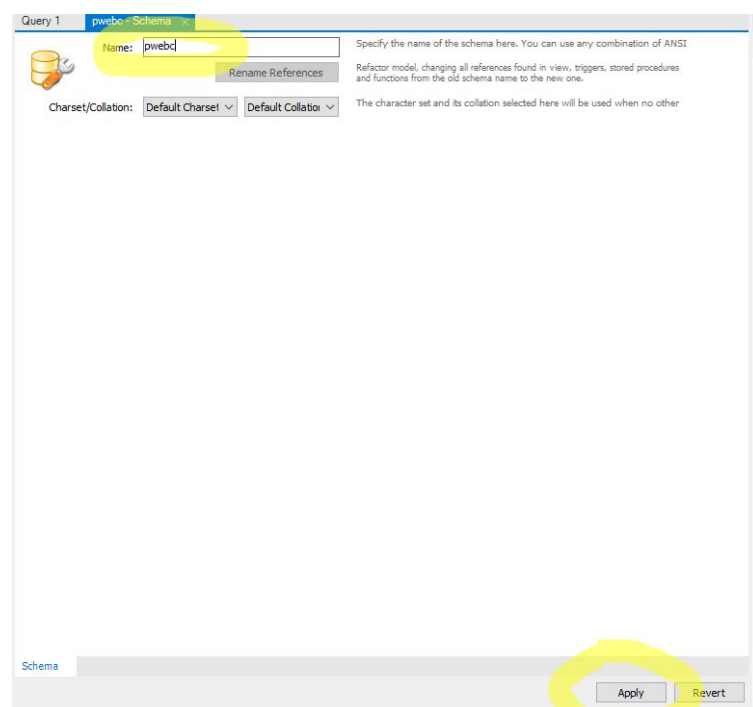
assurez vous de rentrer les bonnes informations pour pouvoir se connecter à la base.  
(fichier de config disponible dans le dossier de l'api, dans le fichier ".env")

Dans workbench (si vous l'utilisez), après avoir installé et s'être authentifié, cliquez sur **Schemas**



Clique droit + create Schema → appelez le 'pwebc' et cliquez sur apply en bas.





Ensuite allez dans 'query' copiez collez le script '**lefichier.sql**' et exécutez le.

Votre BDD est prête.

A présent, ouvrez le projet api (avec visual studio code par exemple), placez vous à la racine du projet et dans un terminal, tapez la commande '**npm install**' pour installer des dépendances du projet spécifiées dans le fichier '**package.json**' !

Tapez ensuite '**npm run dev**' dans ce même terminal pour lancer l'api.

Maintenant il faut initialiser les données dans la BDD via l'api SNCF et nos route dans notre API.

Nous avons utilisé POSTMAN vous pouvez ensuite envoyer les bonnes URL sur le navigateur par exemple.

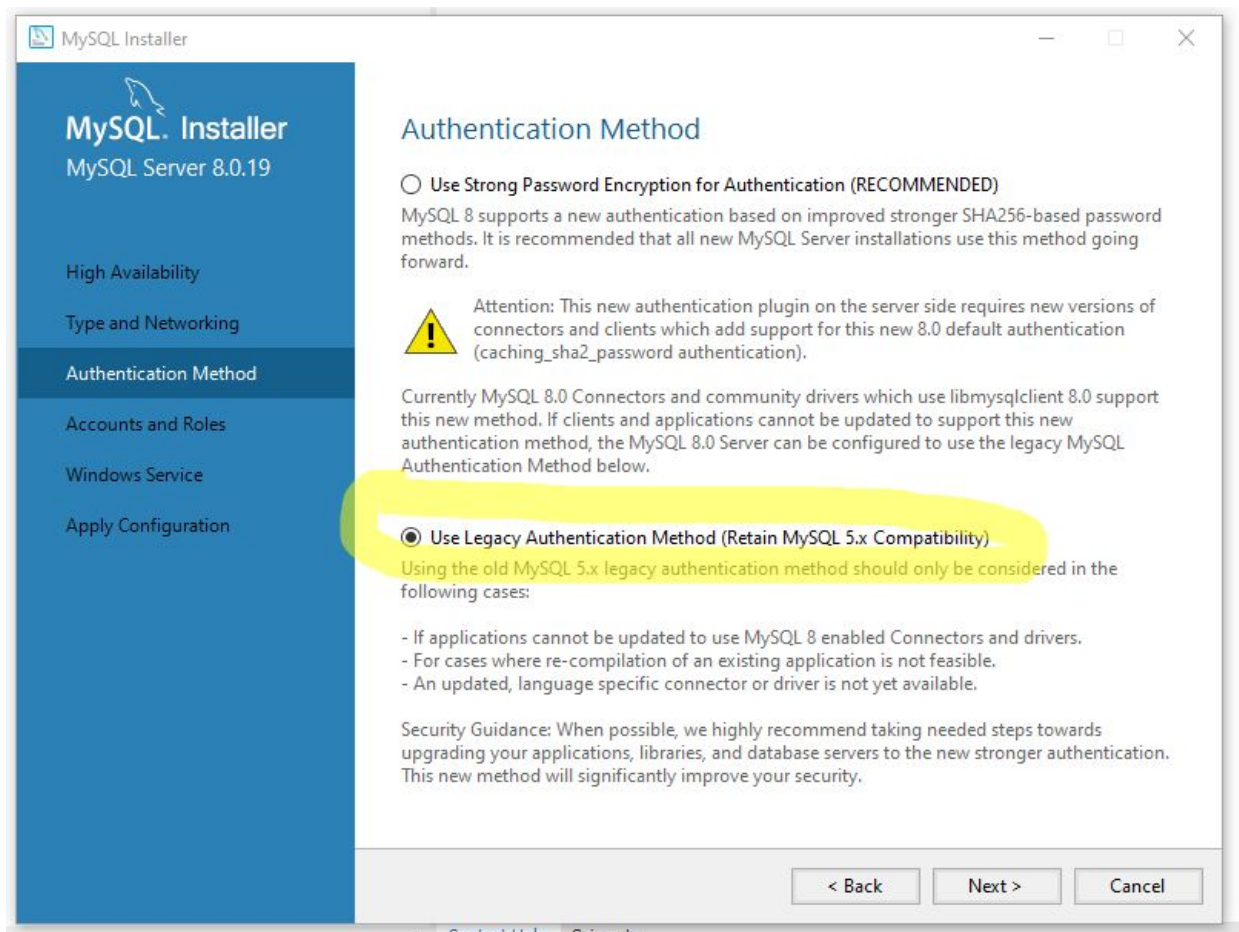
\*\*\*\*(éventuel problème)\*\*\*

Attention il est possible que vous ayez une erreur de type :

**"authentication protocol requested by server; consider upgrading MySQL client"** lors de l'initialisation des données avec postman

pour y remédier, lors de l'installation de mysql workbench, dans la partie Authentication Method, cochez : 'Use legacy Authentication ...'

le problème disparaîtra.

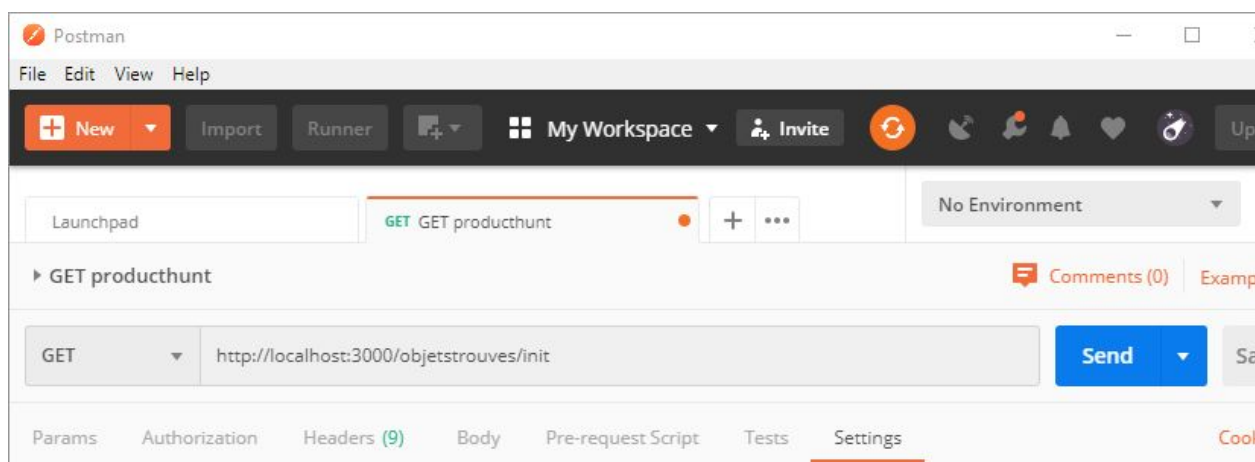


Avec l'api qui tourne, lancez les requêtes suivantes :

<http://localhost:3000/gare/init>

<http://localhost:3000/voie/init>

<http://localhost:3000/objetstrouves/init>



Dans le terminal où tourne l'API il devrait y avoir des informations de type (début initialisation ... fin initialisation)

```
[nodemon] starting `node server.js`
serveur créé, écoute sur le port 3000
Début de l'initialisation des voies
180
[=====]
Fin de l'initialisation des voies : 1806voies ajoutées sur 1806 stockées initialement sur SNCF
test: 2.794s
Début de l'initialisation des objets trouvés en 2k20
1943
[]
```

Une fois toutes les données chargés vous pouvez lancer le .html dans le dossier front-end.

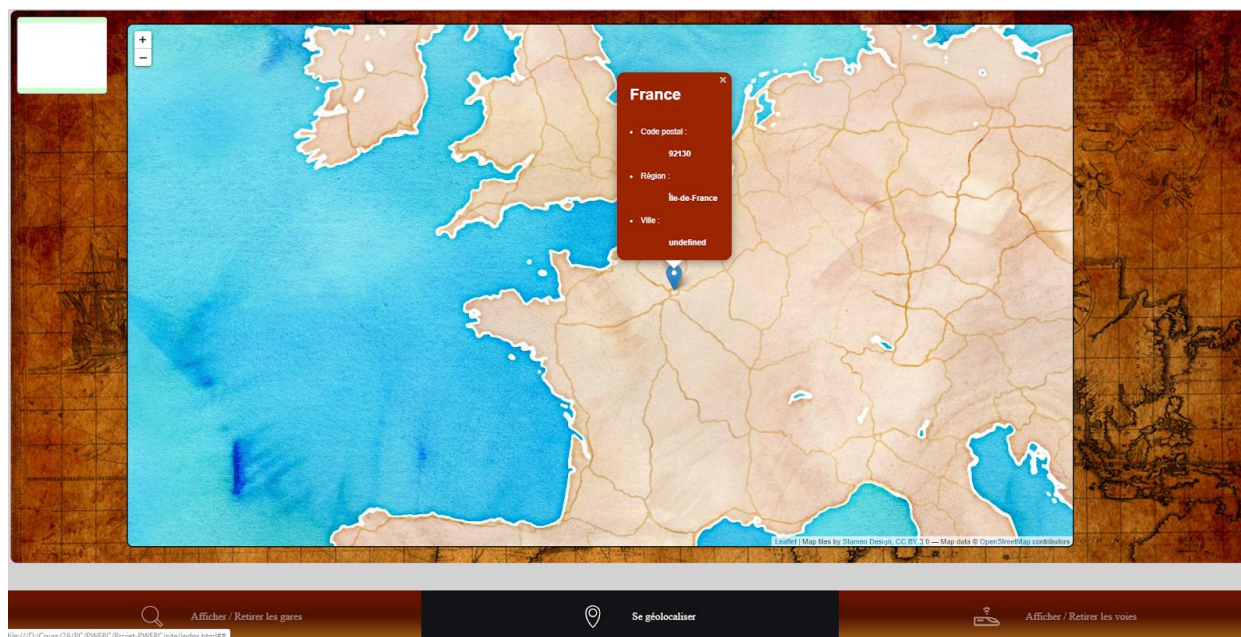
Vous pouvez chargez les gares, voir les voies exploités par les trains, cliquez sur la carte pour connaître la météo à différents endroits, etc Cliquez sur les gares pour connaître le prochain départ et le nombre d'objets retrouvés en 2020.

Maintenant que notre API est opérationnelle, il nous est facile d'ajouter des services concernant les données qu'elle stocke. Nous avons commencé à développer des vues SQL nous permettant par exemple d'afficher les gares d'une couleur différente suivant le nombre d'objets qui y a été trouvé.

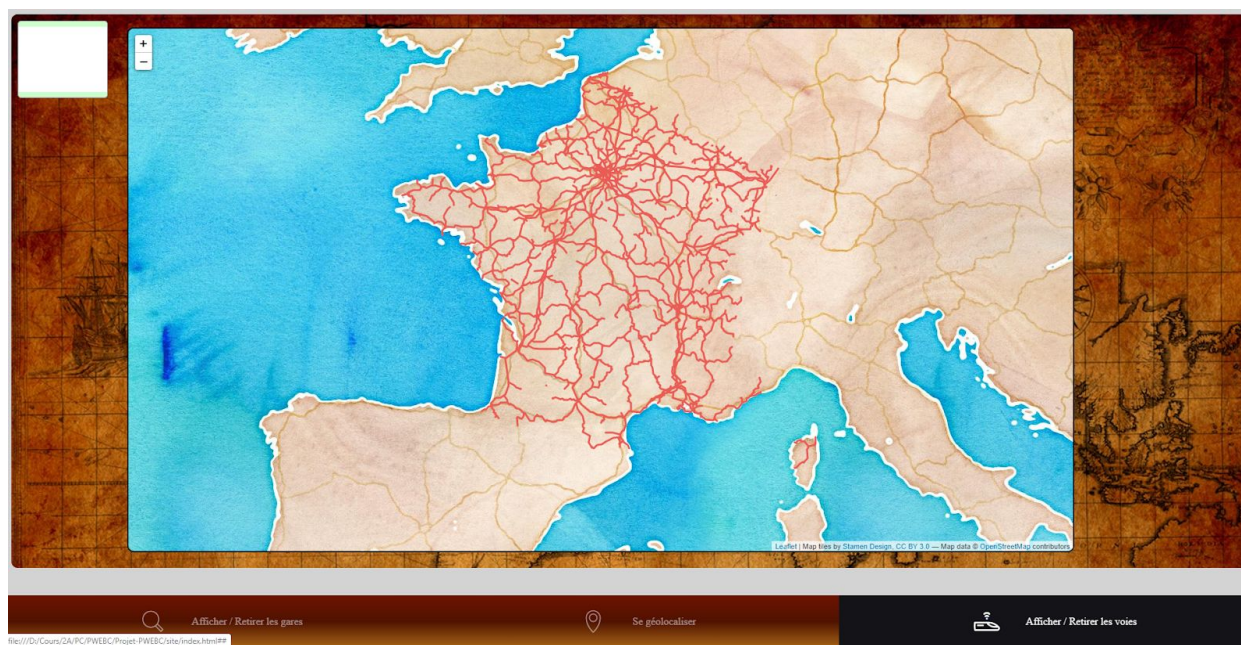


Voici des exemples :

L'affichage de notre localisation sur la carte

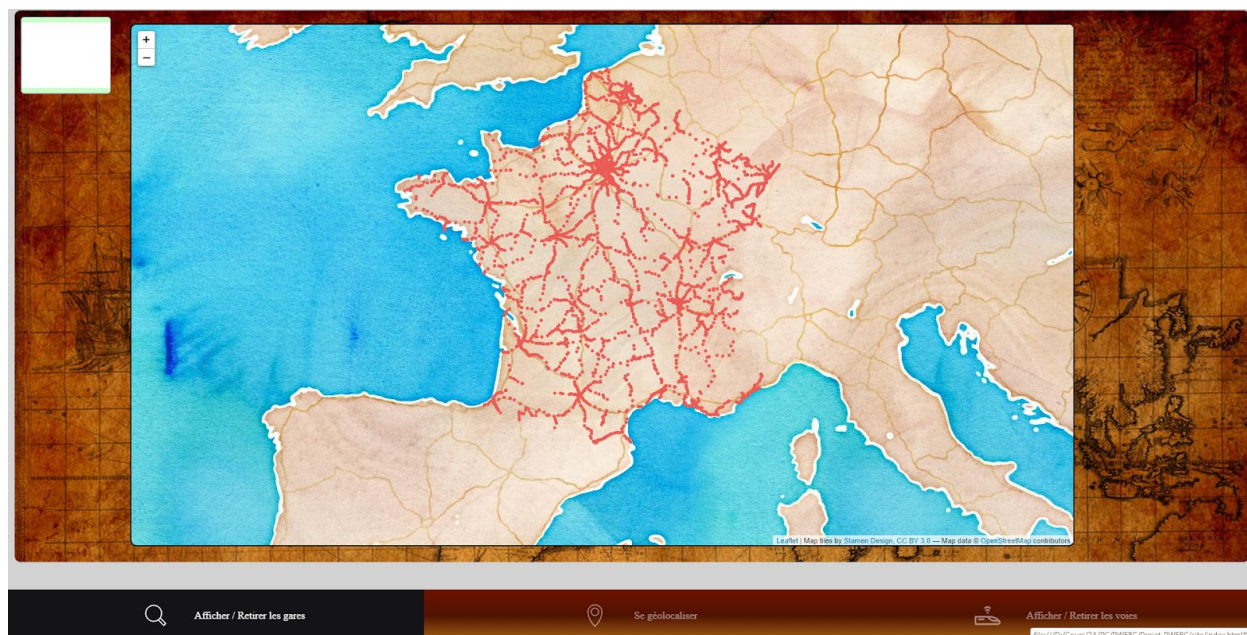


L'affichage des différents réseaux ferrés nationaux ouvert aux voyageurs et opérationnels

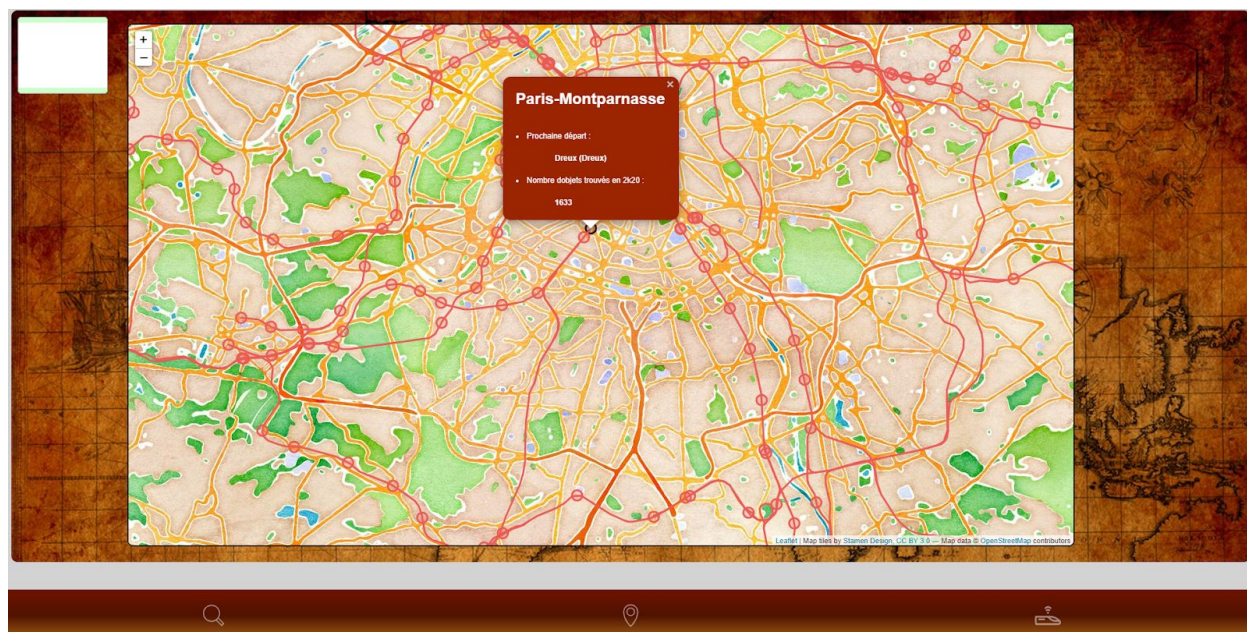




## L'affichage des gares accueillants des voyageurs



## Un exemple d'affichage lors du clique sur une gare :



Ces services peuvent être affichés individuellement ou non. Il nous est possible d'afficher les voies SNCF, les gares, la température, les informations des gares, le tout en même temps.