Rapport de projet Développement d'Applications Réticulaires : "Pokecard Gotta Collect 'Em All"

Berkane Karim, Bunel Richard, Chen Patrick, Diallo Mohamed

22 novembre 2017

Sous la direction de *Romain Demangeon*

Table des matières

1 Introduction

L'objectif du projet est de développer une application web utilisant une ou plusieurs API externes dynamiques. Elle sera dotée d'une architecture client-serveur en se servant d'appels simultanés avec un traitement côté serveur.

Dans le cadre de l'UE Développement d'Applications Réticulaires (DAR), nous avons réalisé l'application Web « Pokecard Gotta Collect 'Em All » permettant aux utilisateurs de rechercher des cartes Pokemon via une barre de recherche, les marquer comme étant possédées, voir leur prix, leur rareté.. Ils pourront se créer un compte et ainsi ajouter dans leur panier des cartes ainsi que les proposer à l'échange et organiser des rendez-vous.

Pour cela, nous utilisons donc une API externe, qui donne plusieurs informations sur les cartes, et qui est régulièrement mise à jour pour leux prix. Celle-ci se nomme PokéPrice et sa documentation est disponible à l'adresse suivante : `pokeprices.doeiqts.com/api/documentation`

Notre application possède également un aspect "réseau social", grâce à la possibilité de discuter via des messages privé une fois connecté.

2 Manuel d'utilisation

Nous allons présenter dans un premier temps comment se présente chaque écran. L'application Pokecard est en ligne grace aux services d'*HEROKU*.

2.1 Description de l'interface

Voici le premier écran, lorsque on accède au site web à l'adresse : `pokecard.herokuapp.com`

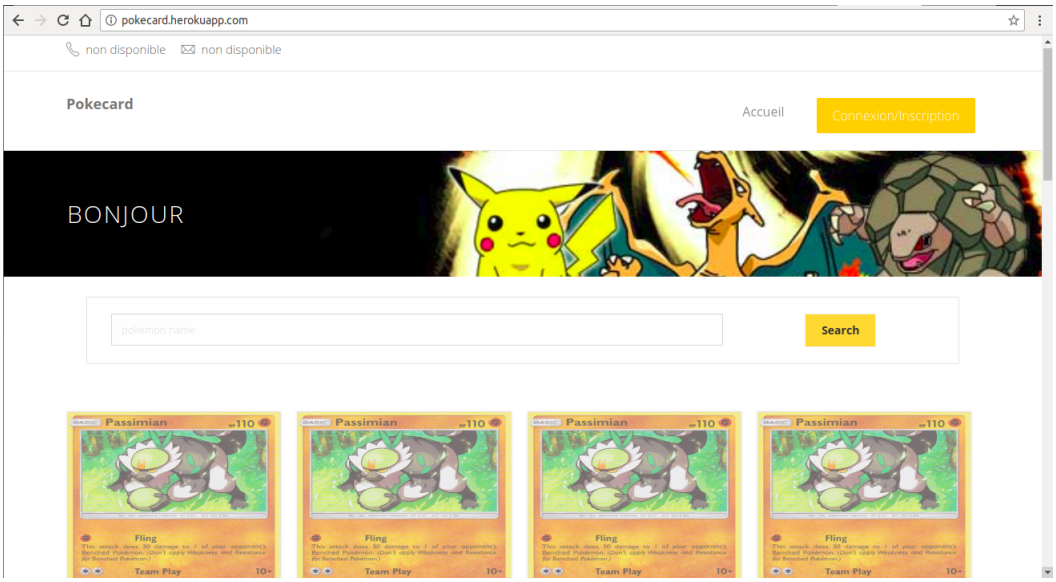

FIGURE 1 – Écran de base

Voici l'écran de base lorsqu'on accede au site web à l'adresse : `pokecard.herokuapp.com` Le design s'est fait avec Bootstrap, ce qui le rend élégant et responsive (mobile et web).

En cliquant sur connexion/inscription, nous avons la possibilité de nous inscrire à gauche ou bien de se connecter. Lorsqu'il y a une erreur (mot de passe ou autre), celle-ci est signalée.

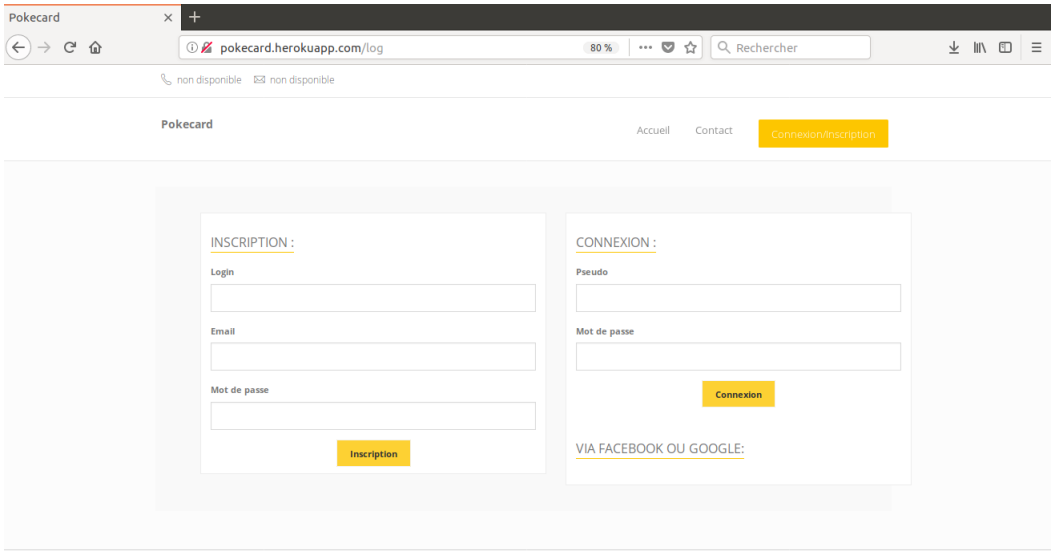

FIGURE 2 – inscription ou connexion

Une fois connecté, l'utilisateur peut donc voir dans son acceuil des cartes (marquées comme possédées par d'autres utlisateurs), et peut déterminer (en la cochant) si il la possede ou si il la cherche. Cocher "cherche" va créer une discussion avec son propriétaire.

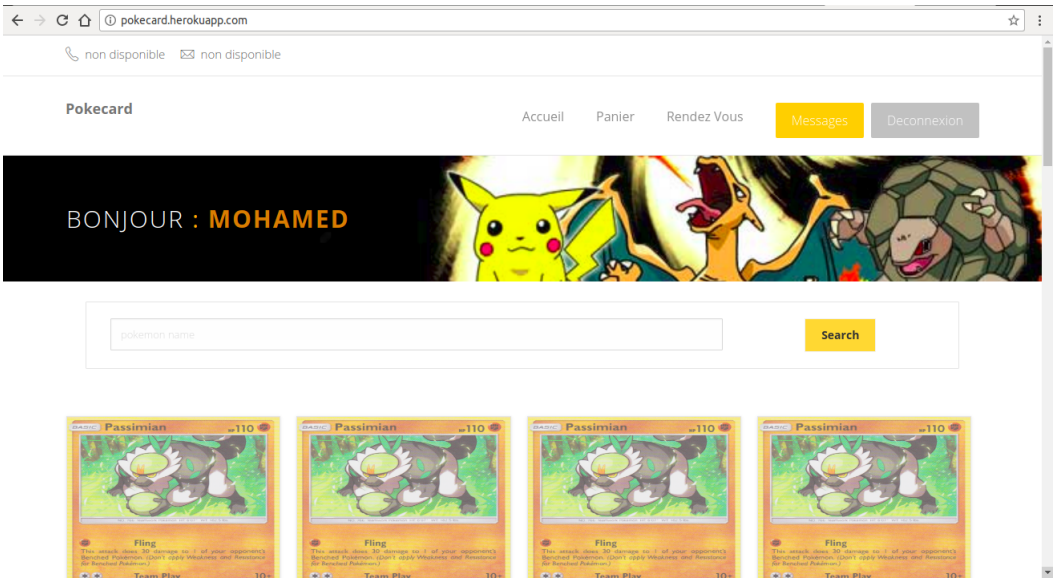

FIGURE 3 – Ecran de base une fois connecté

L'utilisateur peut aussi accéder à sa messagerie, dans la partie réseau social de l'application.

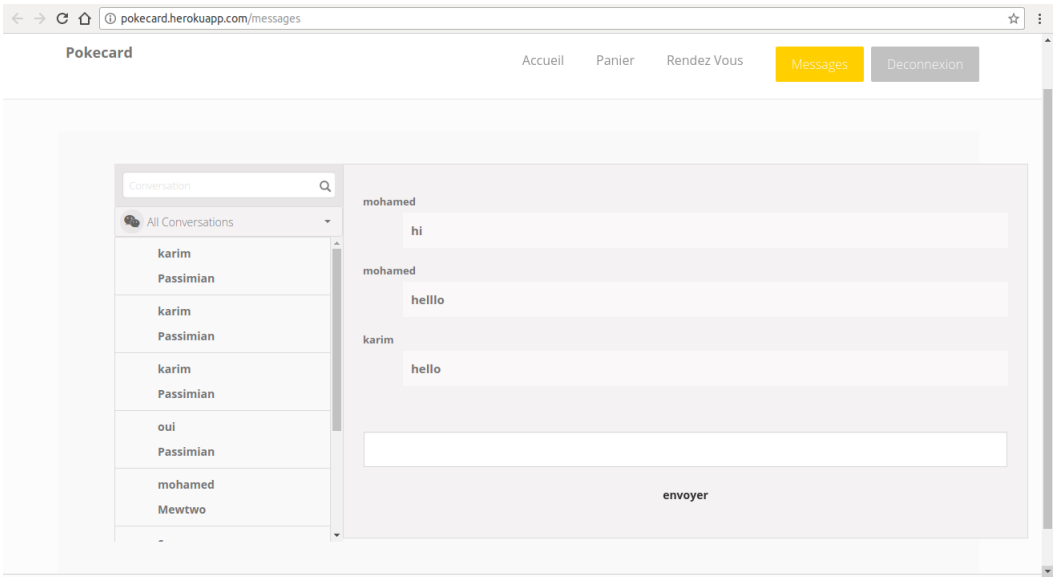

FIGURE 4 – Messagerie interne

Mais aussi avoir accès à son panier de cartes, c'est ici que l'on stocke les cartes que l'on a et que l'on cherche.

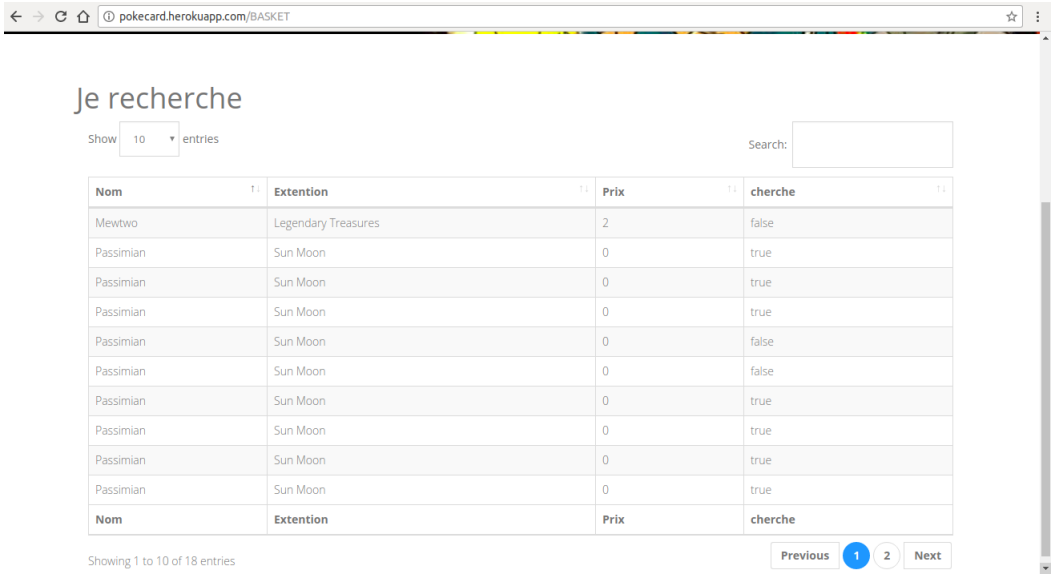

FIGURE 5 – Panier de cartes

Chaque utilisateur peut utiliser la barre de recherche qui permet de chercher depuis la base de données des cartes Pokémon déjà possédées et de les afficher avec leur prix et leurs caractéristiques. C'est ici que se fait l'appel à l'API PokePrice.

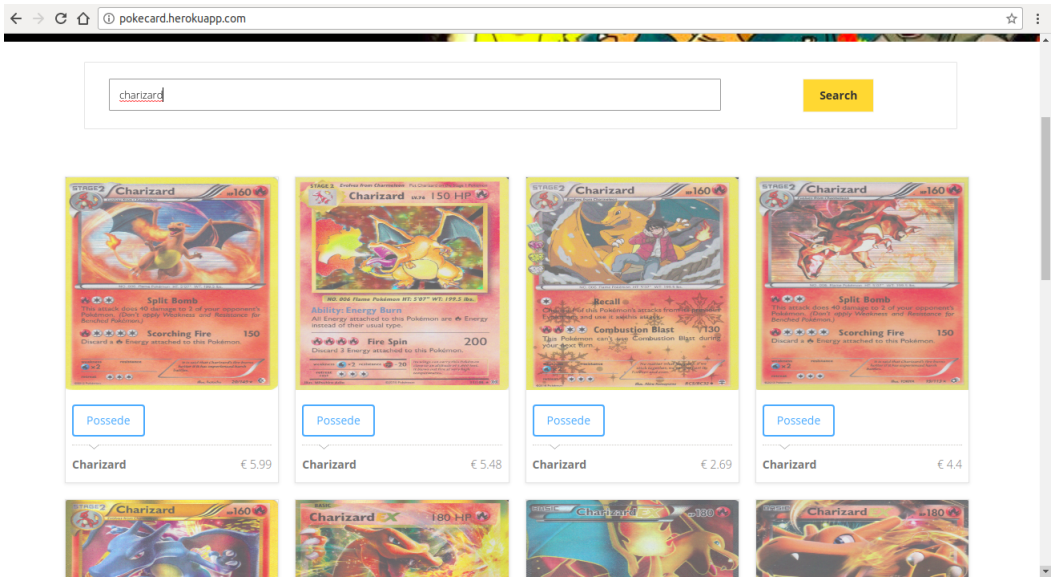

FIGURE 6 – Exemple de recherche de cartes de Charizard (Dracaufeu en VF)

Il est aussi possible de planifier des rendez-vous afin, par exemple, d'échanger une carte particulière :

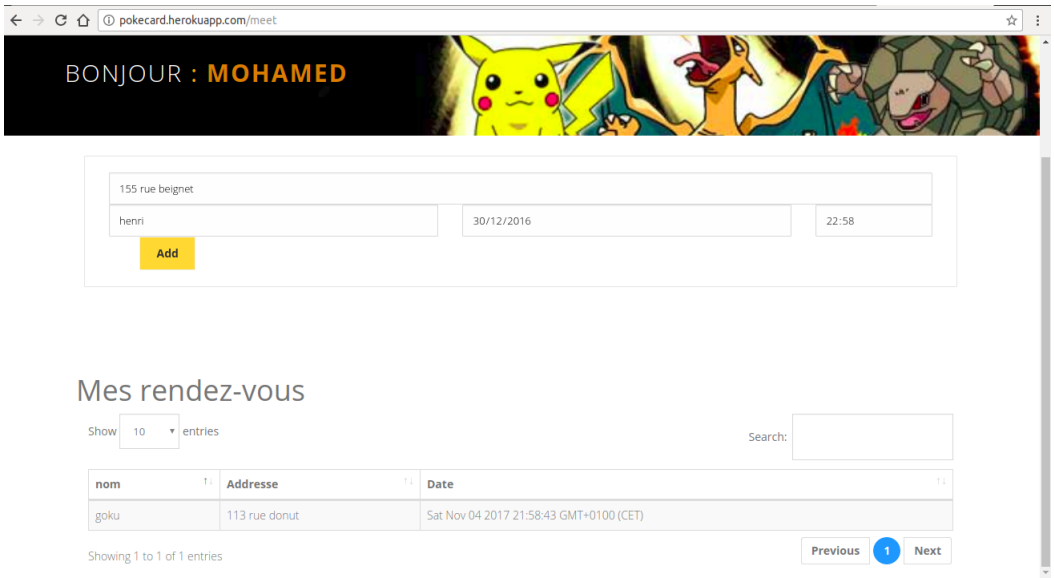

FIGURE 7 – Exemple de rendez-vous

2.2 Les Uses Cases de Pokecard

Voici les différents Uses Cases possible de notre application :

- (Creation de profil)
  L'utilisateur Homer consulte la page de Pokecard, et choisit de créer un profil, pour cela il clique sur inscription/connexion et sur la gauche il remplit les informations demandées par le formulaire.

- (Connexion)
  Homer choisit maintenant de se connecter, il clique alors encore sur inscription/connexion et cette fois ci remplit les deux champs sur la droite de l'écran. Il est maintenant connecté.

- (Faire une recherche)
  Homer decide de chercher la carte "Charizard" dont il écrit le nom dans la barre de recherche.

- Si on clique sur "cherche", une discussion entre le possesseur de la carte et l'utilisateur est automatiquement crée.

- (Messagerie)
  Mais Homer a reçut un message, il clique donc dans sa messagerie et y voit un message auquel il répond.

- (panier)
  C'était une question pour savoir si Homer avait une carte Pikachu. Homer clique donc dans l'icone panier, et regarde alors dans sa liste de carte si il la possède ou non. Dans le panier on stocke les cartes qu'on possède et qu'on cherche.

- (Rendez vous)
  Homer n'a pas cette carte, il decide donc d'organiser un rendez-vous avec son interlocuteur afin d'échanger et de potentiellement obtenir la carte en question. Il clique donc dans rendez-vous, et doit remplir les champs (la date, le lieu, le prénom..) et l'ajoute à ses rendez-vous prévus.

3 Architecture et technologies mises en œuvre

L'architecture de notre application se fonde sur le schéma monolithique. C'est un choix technique que nous pensons suffisant pour une application de l'envergure de Pokecard. En effet, le protocole HTTP est utilisé pour la communication entre la partie client et la partie serveur.
Ceci se faisant a l'aide d'un serveur Jetty. En ce qui concerne les services Web, dans un souci de rapidité nous avons utlisé le framework Jersey, tout en veillant à coder un Servlet par nous mêmes, afin de montrer notre maîtrise de ces derniers (cette pratique a été autorisée par M. Demangeon).

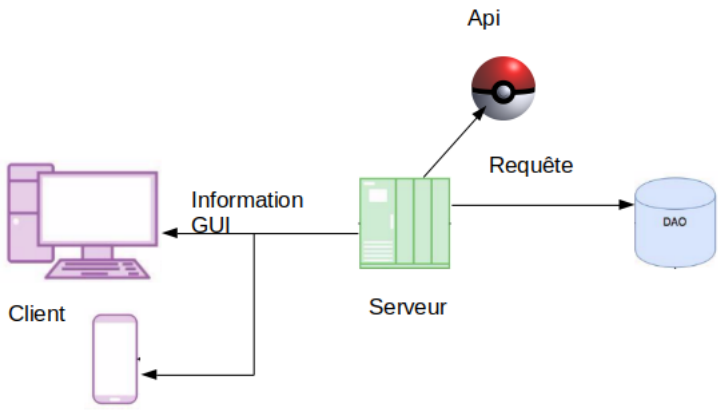

FIGURE 8 – Schéma global de l'architecture de l'application (monolithique)

3.1 Communication Client/server

Nous avons utilisé Ajax pour la communication entre le client et le serveur ce qui permet de rendre les interactions asynchrones. La serveur renvoie des données en JSON qui vont être traitées et affichées dans les vues.

Exemple de l'utilisation d'AJAX Les fonctions permettant les requêtes POST et GET :

```
function getServerData(url, success, failure) {
  // console.log('get server data');
  $.ajax({
    dataType : "json",
    url : url,
    async : false,
    type : "GET",
    success : success,
    error : function(jqXHR, textStatus, errorThrown) {
      console.log("textstatus : " + textStatus);
    }
  });
}

function postServerData(url, success, failure) {
  $.ajax({
    dataType : "json",
    url : url,
    type : "POST",
    success : success,
    error : failure
  });
}
```

FIGURE 9 – POST et GET avec AJAX

Voici par exemple la recherche d'une carte en particulier :

```
function searchCard() {
  // alert('mesd');
  console.log($("#searchPokemonName").val());
  getServerData("/card/search/" + $("#searchPokemonName").val(),
      printSearchCard, null);
}
```

FIGURE 10 – Recherche d'une carte (AJAX)

3.2 La base de données SQL

Nos entités étant liées, nous avons opté pour une base de données relationnelle. SQL permet de structurer les informations et de les réutiliser facilement. Nous utilisons donc des appels directs à la base de données Postgresql d'Heroku.

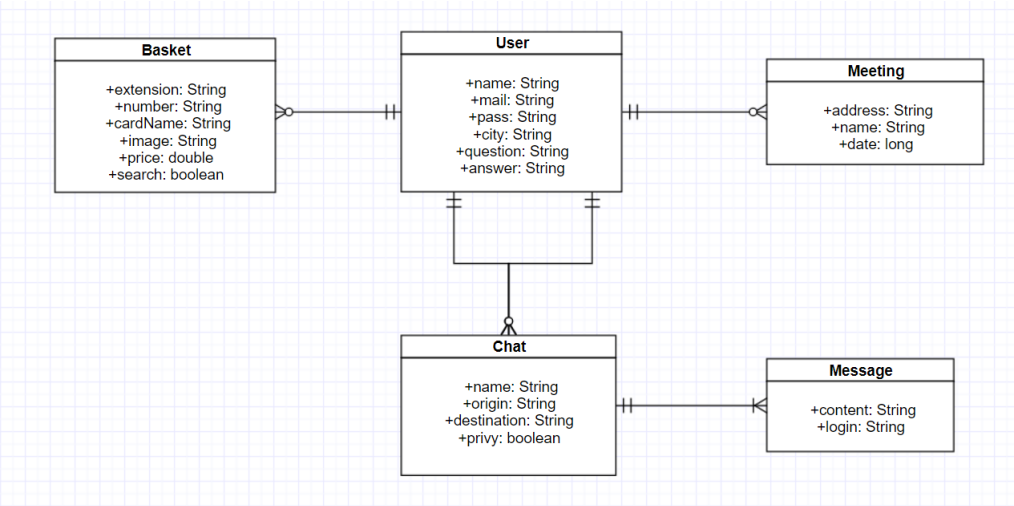

FIGURE 11 – Schéma relationnel de la base de données

Nous nous connectons directement dans la base de données via le code suivant :

```
public static Connection getConnection() throws URISyntaxException, SQLException {
      URI dbUri = new URI(System.getenv("DATABASE_URL"));

      String username = dbUri.getUserInfo().split(":")[0];
      String password = dbUri.getUserInfo().split(":")[1];
      String dbUrl = "jdbc:postgresql://" + dbUri.getHost() + dbUri.getPath();
      System.out.println("print of dbUrl " +dbUrl);

      return DriverManager.getConnection(dbUrl, username, password);
}
```

FIGURE 12 – Connexion à la BDD

3.3 API utilisée

L'API utilisé est PokéPrice : `http://pokeprices.doeiqts.com` ; cette API permet d'avoir de façon dynamique le prix des cartes Pokémon ainsi que plusieurs informations supplémentaires. Nous faisons appel à cette API coté serveur et redistribuons les résultats via les web-services.

4 Méthode de developpement utilisée

Une méthode de développement agile est un ensemble de principes et de pratiques qui aide les équipes de développement à livrer des produits à cycles courts, ce qui permet une rétroaction rapide, une amélioration continue et une adaptation rapide au changement. Nous avons développé notre projet en essayant de suivre au mieux cette méthodologie.
Les plateformes de discussion ont été utilisé pour échanger sur le projet.

Le projet a été travaillé dans le dépot git privé d'Heroku.

Le rapport a été redigé en Latex avec l'outil Overleaf.

Les schémas ont été réalisés avec draw.io et le site Gliphy.

5 Perspectives et résultats

5.1 Points fort

Parmi les points fort de notre application nous pouvons parler de :

- nous avons conçu une interface ergonomique avec Bootstrap afin de faciliter l'utilisation.
- la base de données relationnelle : choix de SQL pour assurer la persistance et l'intégrité des données.
- l'originalité de notre projet
- l'agilité : développer cette application en agile nous a donné une grande visibilité sur l'avancement du projet.

5.2 Points d'amélioration

Voici des aspects qui sont à ameliorer dans le projet :

- Amélioration de la sécurité : mettre en place des dispositif de cryptage des données et de sécurisation des transmissions (par exemple via HTTPS).
- Mise en place de tests unitaires avec par exemple l'utilisation de Jenkins.
- Si l'API évolue d'avantage, revoir l'architecture en la passant par micro-services.
- Mécanisme d'auto-complétion pour la recherche de cartes.

6 Conclusion

Ce projet nous a permis de mettre en pratique les connaissances acquises dans le cadre de l'UE Développement d'Applications Réticulaires. Pour certains d'entre nous, il nous a aussi servi à nous préparer pour les travaux que nous aurons à effectuer dans le monde de l'entreprise.

Ce projet nous a aussi permis de développer à approfondir les bonnes pratiques de gestion de projet, vues concurremment en cours de GPSTL.