Université de Sherbrooke

REAL ESTATE

Rapport

Demandeur : Frédéric BERGERON

Projet réalisé dans le cadre du cours IFT717

Remis le 10 décembre 2021

Réalisé par Anna MASANTE, Lilian MISSER, Léo MOLLARD et Julien WIEGANDT

Sommaire

Table des figures

I. Description du projet

Notre projet est une application d'annonces immobilières. Particuliers comme professionnels peuvent poster des annonces de leurs bien immobiliers. Ces biens sont visualisables sur une carte Google Map interactive, en plus de fournir une fiche pertinente d'informations sur le bien proposé. En tant qu'acheteur, je peux rechercher un logement autour d'une adresse ou autour de ma position. En cliquant sur une annonce dans la liste affichée ou sur le pointeur d'un bien sur la carte, j'accède aux détails de l'annonce concernée.

Lancer le projet :

Côté serveur :

- Créer un fichier .env contenant les données suivantes :

```
PORT=3000
DB_USER=lilianmisser
DB_HOST=postgresql-lilianmisser.alwaysdata.net
DB_NAME=lilianmisser_projectdb
DB_PASS=5318167990
JWT_KEY=2hEWUyyPv7n7BkiW4i9t
```

- Pour lancer le serveur écrire **npm run dev** dans le terminal

Côté application :

- Remplacer l'IP de la classe Application par l'IP de l'API :

```
var IP = "XXX:3000/api"
```

- Remplacer la clé de l'API maps dans le fichier local.properties :

```
MAPS_API_KEY=AIzaSyBnFQjE-XX4jzKUoHgIu9yRiRu7j0fjmEI
```

II. Architecture

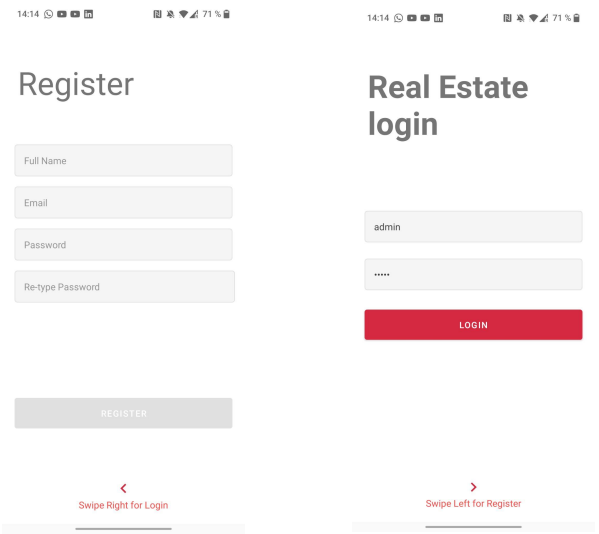
1. Client android

Notre application mobile à été effectuée en android natif. Nous avons fait ce choix pour pouvoir mettre en pratique les différentes notions abordées en cours et avoir l'opportunité de vraiment prendre en main le langage kotlin.

Notre application est composée de 2 activités. Chaque activité est elle-même composée de différents fragments qui viennent s'afficher à l'écran de l'utilisateur. Nous avons fait ce choix pour ne pas surcharger notre application et utiliser la modularité des fragments ce qui permet une navigation plus fluide dans une application.

a) MainActivity : L'authentification

L'authentification constitue une première activité. C'est la première activité que l'utilisateur verra lorsqu'il lancera l'application pour la première fois. Celle-ci va utiliser un *ViewPager* qui permettra d'interchanger 2 fragments en un simple geste. *ViewPager* est un pattern introduit par Android dès la version 1.6 via le package Android Support, afin d'assurer une navigation horizontale en permettant à l'utilisateur de faire défiler les différentes portions du contenu d'une application sur un seul et même écran. Dans notre cas, cela permet à l'utilisateur de faire défiler 2 fragments. Le premier fragment étant



l'authentification de l'utilisateur et le second, la création d'un compte comme nous pouvons le voir ci-dessous :

Figure 1: captures d'écran des 2 fragments d'authentification

b) AppActivity : Accès aux fonctionnalités de l'application

La seconde activité contient ensuite les fonctionnalités de l'application. Pour cela, nous avons utilisé une *BottomNavigationView* qui nous a permis de créer un menu et de le lier à différents fragments.

- Le premier fragment va proposer à l'utilisateur de voir les différentes annonces disponibles à travers une carte ou une liste. Pour gérer cet affichage, nous avons introduit un *supportFragmentManager* qui nous a permis de switcher entre le fragment contenant la carte et le fragment contenant un *recyclerView* pour afficher les annonces sous forme de liste.

- Le second fragment s'occupe d'aller chercher les annonces qui ont été créées par l'utilisateur.

- Enfin, le dernier fragment permet à l'utilisateur de gérer certains paramètres de son application comme son mot de passe ou encore s'il souhaite recevoir des notifications.

c) Les échanges avec l'API

Tous les échanges concernant la création et l'affichage d'annonce ainsi que l'authentification s'effectuent avec la librairie Volley. Cette librairie permet d'envoyer des requêtes HTTP au serveur.

d) Gestion des notification

Lorsqu'une nouvelle annonce est mise en ligne, le serveur va push des notifications aux différents utilisateurs. Pour cela, nous utilisons la librairie *SocketIO* qui nous permet de recevoir les notifications et de les faire apparaître sous forme de toast ou de Notification selon si le client est sur l'application ou non.

2. API Rest

Pour pouvoir interagir avec les données nous avons décidé de développer une API Rest en utilisant NodeJs avec Express.

Pour nous l'intérêt d'utiliser une API Rest est tout d'abord qu'elle est stateless. En effet, le serveur ne stocke pas d'informations sur le client qui demande ou crée des informations. Cela permet au serveur de traiter toutes les requêtes de manière indépendante car si le client a besoin d'autorisation pour différentes requêtes il devra toujours montrer un moyen d'identification. Cela permet au serveur de pouvoir évoluer plus facilement si il y a un plus

grand nombre de clients de plus cela rend l'API plus simple et plus prévisible. Finalement une interface REST nous permet d'avoir une interaction plus facile avec les applications clientes car on expose nos ressources sur la même URI et les applications clientes ont simplement besoin de changer les verbes HTTP en fonction de s' ils veulent créer, obtenir, modifier ou supprimer une ressource.

Nous avons choisi d'utiliser NodeJs pour créer notre serveur. En effet c'est une plateforme logicielle utilisant javascript et qui est orientée vers les applications événementielles et concurrentes. En effet Node permet de pouvoir monter en charge facilement pour des opérations rapides car elle utilise des événements non bloquants ce qui permet de paralléliser différentes requêtes ainsi pouvoir accueillir de nombreux clients simultanément. Son désavantage est qu'il utilise seulement un thread donc il est moins efficace pour des opérations lourdes or içi le serveur nous sert d'interface pour communiquer entre les clients et nos modèles. Nous avons donc choisi d'utiliser Node pour ses qualités de serveur événementiel.

De plus Node fournit npm (Node Package Manager) ce qui nous permet d'installer facilement des librairies qui sont très fournies car c'est une technologie très plébiscitée.

Ensuite nous avons choisi d'utiliser ExpressJS qui est un micro-framework nous permettant de pouvoir déclarer des contrôleurs de façon beaucoup plus claire et lisible que la libraire http intégré a Node. Grâce à son système de routeurs, il permet d'avoir un code beaucoup plus compréhensible et maintenable.

Finalement nous avons utilisé TypeScript comme surcouche à JavaScript afin de pouvoir avoir la force d'un typage fort en utilisant javascript. En effet TS nous permet d'avoir moins de vecteurs d'erreurs et un développement plus facile grâce à un code plus compréhensible. Celui-çi se transpile ensuite en javascript qui sera exécuté par Node ce qui ne nous donne pas de soucis de performances.

3. Base de données

Nous avons choisi d'utiliser PostgreSQL pour ce projet. Ce choix a été justifié car selon nous une base de données relationnelle nous permet d'avoir le plus de sécurité vis à vis de la cohérence de nos données. Une base de données non relationnelle serait envisageable si le nombre d'utilisateurs augmente mais nous nous sommes permis d'avoir le choix avec une architecture de code qui sépare les interactions à la base de données de la logique des routeurs.

III. Fonctionnalités

1. Authentification

L'authentification de notre système s'effectue grâce à des Json Web Tokens (jwt). Lors d'une demande de connexion, une requête HTTP de type POST est effectuée vers le serveur qui va vérifier les identifiants entrés par l'utilisateur en base de données. Si l'utilisateur existe bien et qu'il a fourni le bon mot de passe, un token va être généré et envoyé au client android. Ces tokens sont générés par l'API grâce au package npm *jwt*. Le token est ensuite stocké dans la mémoire du téléphone mais également dans la classe Application au lancement de l'application pour avoir ce token en variable globale. Il est ensuite utilisé pour authentifier l'utilisateur lorsqu'il effectue des requêtes nécessitant d'être sécurisé. Le token est ajouté aux headers des requêtes comme par exemple, une requête de création d'une annonce.
Ce token possède une durée de vie, et une fois cette durée écoulée, l'utilisateur est déconnecté et il doit à nouveau s'identifier pour obtenir un nouveau token.

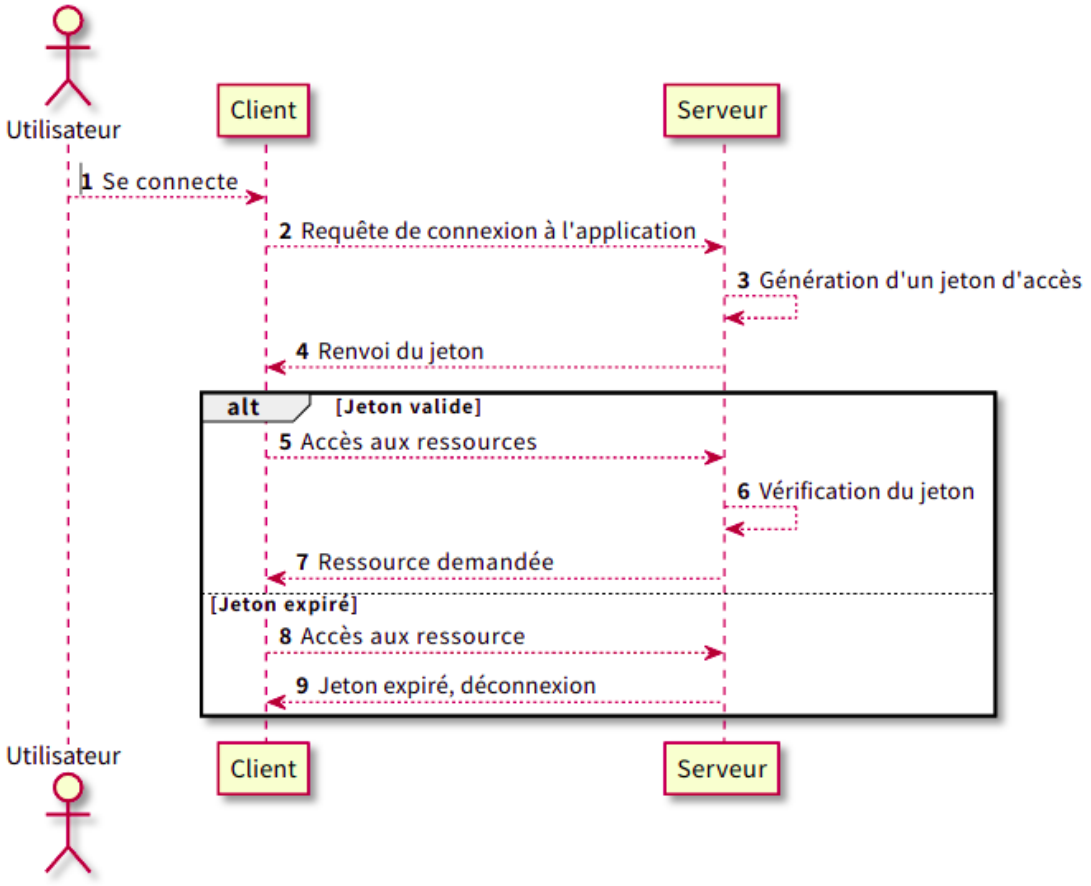


Figure 2 : Diagramme de séquence de l'utilisation des JWT

2. Création des annonces

Les utilisateurs peuvent créer des annonces via l'interface de l'application. Ils y rentrent les différentes informations ainsi que leurs contacts et doivent ajouter une photo. Nous hébergeons la photo via une API publique "imgbb" et nous stockons ensuite l'uri de la photo dans notre base de données.

3. Affichage des annonces

a. Par la page des annonces créées

La liste des annonces que nous avons créé est disponible sur le bouton du menu du milieu de l'application.

b. Par la carte

Au démarrage, par défaut quand nous arrivons sur cette partie de l'application nous pouvons visualiser les logements qui se trouvent autour de notre position. Si nous voulons voir d'autres logements, nous pouvons entrer une autre adresse dans le champ dédié en haut de l'application.
Lorsque nous cliquons sur le bouton de position, nous avons automatiquement un focus sur notre position et nous pouvons visualiser les logements autour de nous.
Pour récupérer la position de l'utilisateur nous demandons les permissions nécessaires et si nous ne les avons pas nous ne pouvons pas avoir la position. Une fois les permissions obtenues nous récupérons sa position grâce à la librairie **FusedLocationProviderClient**. Puis nous montrons sur la carte les logements qui sont autour de cette position, pour cela nous faisons une requête au serveur avec les coordonnées de l'utilisateur qui sélectionne tous les logements de notre base de données qui sont à 20km à la ronde.

4. API Google Maps 

Pour pouvoir afficher les différences annonces, nous avons utilisé l'API Google maps qui nous a permis de placer des marqueurs représentant une annonce sur une carte. Ces marqueurs sont placés grâce à des coordonnées sous forme de latitude et de longitude. Ces données sont stockées avec l'adresse de l'annonce en base de données et pour les obtenir, nous avons utilisé la dépendance *Géocoder*. Cette dépendance nous permet de transformer une adresse en coordonnées (latitude/longitude). Cela nous permet également de détecter si une adresse entrée est incorrecte.

Par soucis de scalabilité, il ne fallait pas devoir charger toutes les annonces existantes d'un coup sur la carte, nous avons donc mis un système de filtrage. L'affichage des logements sur la carte peut s'effectuer selon 2 filtres :

Figure 3 : Capture d'écran de la carte

Position de l'utilisateur

La position de l'utilisateur est nécessaire pour le localiser et obtenir ses coordonnées sous forme latitude/longitude. Une fois ces coordonnées acquises, un marqueur est placé sur la carte représentant l'utilisateur et une requête est faite à l'API pour obtenir les annonces dans un rayon de 20 kilomètres.

Nom de ville

L'utilisateur à également la possibilité de filtrer les annonces en indiquant la ville dans laquelle il souhaite voir les annonces. Pour cela, il lui suffit d'écrire le nom de la ville souhaité et une requête est ensuite faite à l'API pour récupérer les annonces correspondant à la ville.

IV. Difficultés rencontrées

1. Connexion à l'API Google

Au début, nous avons cherché un moyen de convertir une adresse "humaine" (ex : 1334 rue saint-Luc) en une adresse qui correspond à des coordonnées GPS. La solution que nous avons trouvée fut d'utiliser l'API Google : lorsque nous utilisons ce service pour la première fois nous avons 300$ dollars de crédit ce qui permet d'utiliser le service "gratuitement".
Nous avons eu des difficultés pour connecter l'API à l'application.

2. Récupérer une sélection de logements

Lorsque nous sommes sur la carte nous voulons pouvoir chercher les appartements autour de nous ou ceux autour d'une adresse. Nous avons donc dû coder une méthode qui recherche les logements qui sont dans une certaine zone autour de la position grâce aux coordonnées GPS.

3. Affichage de fragment avec map

Nous avons un bouton qui permet de switcher entre deux fragments, il permet de montrer l'un ou l'autre grâce au paramètre **VISIBLE.** Pour gérer cet affichage, comme vu plus haut nous avons introduit un *supportFragmentManager* qui nous permet de switcher entre le fragment contenant la carte et le fragment contenant un *recyclerView* pour afficher les annonces sous forme de liste. Cependant, nous avons eu du mal à implémenter cette fonctionnalité dans un premier temps car imbriquer des fragments dans d'autres fragments génère des effets de bords indésirables.

V. Conclusion

Pour réaliser ce projet nous avons donc travaillé à 4 personnes. Nous avons réalisé une application avec un système d'authentification, de gestion de différents logements mais également d'un système de notification push. Cette application est maintenant fonctionnelle. De plus ce projet nous a permis de découvrir les outils proposés par Google relatifs à la localisation, ce sont des compétences qui nous seront utiles. Finalement ces éléments nous ont permis de réfléchir à de potentielles améliorations de notre application comme la recherche par catégorie de logement, par prix etc.

Nous sommes heureux d'avoir pu réaliser ce projet ensemble, il nous a permis de nous améliorer techniquement et d'améliorer nos compétences