Réalisé par : Léa AFCHAIN, Amberine KHAMASSI, Florian KOSZUL, Roger PRIOU et Valentin SOUDY

Projet E3 2019

Sherloc

Projet réalisé entre le 6 mai et le 25 juin 2019.

Il participe à la journée des projets du 25 juin et concourt au challenge innovation du 11 juillet 2019.

ESIEE PARIS

Table des matières

Tout d'abord, nous souhaitons tous remercier très chaleureusement Monsieur Jean Larrat, notre tuteur, pour toute son aide et sa bienveillance durant tout le projet. Il a su nous aiguiller durant toute l'aventure et nous sommes très reconnaissant pour ses actions et sa bonne humeur.

I. L'idée Sherloc

Ne vous est-il jamais arrivé de devoir courir les magasins au dernier moment pour trouver une nécessité de dernière minute ? Ne sachant pas où trouver l'objet vous êtes obligé de visiter différents magasins pour enfin espérer tomber sur celui qui proposera dans ses stocks l'objet convoité ? Et c'est encore pire si vous ne connaissez pas les lieux.

Sherloc répond à ce besoin de savoir ou acheter immédiatement un objet, en indiquant les magasins proches proposant à la vente le produit convoité. Cela fait gagner du temps et évite le stress ou la déconvenue de se déplacer pour rien dans un magasin sans intérêt.

La caractéristique de Sherloc, c'est qu'il veut fonctionner partout, il utilise internet et les ressources disposés en libre-service sur internet qu'il parcourt à très grande vitesse avec une certaine intelligence pour obtenir les meilleurs résultats pour l'utilisateur.

Utilisation d'Android :

Lorsque l'on parle d'application, un choix s'impose : Android ou IOS d'Apple ?
Nous avons jugé qu'Apple présentait trop de désavantages par rapport à Android : Un langage spécifique, restreint à une utilisation sur machine également sous IOS. Or nous ne possédons pas de Mac.
La distribution via l'Apple Store coûte 100 dollars par ans alors qu'Android permet de publier son application pour 25 dollars une unique fois.

Coder l'application Android via AndroidStudio permet aussi de coder en java, langage dont nous sommes pour la plupart familier grâce aux cours en première année

II. Organisation de l'équipe

Etant tous débutants sur Android studio, nous avons débuté par une semaine de prise en main du logiciel en créant de petites applications et suivant pas à pas les tutoriels de Open Classroom par exemple. Cela nous a permis de reprendre la main sur Java que nous avons peu pratiqué en E3.

Dès les premiers jours, nous nous sommes réunis tous les 5 afin de planifier tout le projet et établir ensemble nos objectifs. Voici les notes que nous avons prises ce jour-là :

Le minimum de l'application :
->barre de recherche
-> regarder autour de soi où notre produit est disponible
-> l'afficher sur la mini-carte
-> tracer l'itinéraire

- Charte graphique

Les plus :
-> possibilité de trier (marque, prix, types généraux, enseignes, distance)
-> créer des comptes locaux
-> historique de recherche
-> compte utilisateur :
    -> magasins favoris
    -> préférences / favoris
    -> niveaux utilisateurs en fonction de son activité / de ses retours / commentaires
-> compte commerçant :
    (À voir pour les détails)
-> indice de fiabilité basée sur le retour sur expérience utilisateur + de précision sur nos recherches)
-> prédilections de recherches lorsque l'utilisateur tape dans la barre de recherche

---

Nous avons conclu que nous voulions créer une communauté et fidéliser nos utilisateurs en leur créant un espace client personnalisé mais également une expérience unique en 'récompensant' les plus actifs.

Nous voulons créer un système de fiabilité qui va permettre aux utilisateurs de nous aider pour avoir la meilleure estimation de chance d'avoir le produit recherché. Cela va également permettre aux utilisateurs de s'aider entre eux. Nous estimerons que ceux qui sont le plus actif et participent à ce système auront une voix et un statut plus important lors de ce vote (en vue de la motivation et la présence sur l'application, ces utilisateurs sont beaucoup plus « fiables » et impliqués, cela pourra être modéliser par de l'expérience et un niveau de fidélité.)

Avec ces objectifs, nous avons décomposer les différentes tâches sur des domaines équivalents en charge de travail et les plus indépendants possibles les uns des autres afin de pouvoir progresser en autonomie.

Le découpage est le suivant :

Valentin s'occupe de la partie graphique, du design de l'application et de l'expérience utilisateur. Cela comprend notamment la création de tous les différents fragments et activités. La partie Design comprend le choix qui définissent l'identité du projet : son nom, son logo et sa charte graphique globale.

Florian doit assembler les parties de chacun dans le processus de recherche, il gère la structure de la recherche et le multi-threading. Il est chargé de gérer l'activité de recherche, l'affichage de la carte, sa manipulation et les diverses requêtes à Google et la base de données.

Roger s'occupe de la partie Web qui a pour objectif de simuler la connexion à un site d'e-commerce dont l'adresse a été récupérée par Florian, et d'y effectuer la recherche pour en extraire les résultats. Il a également aidé Valentin dans la réalisation de l'interface graphique de l'application.

Léa se charge de la base de données...

Amberine, quant à elle, veillera à créer un indice de fiabilité…

Durant cette semaine, nous avons mis en place les différents outils de projet habituels afin de pouvoir travailler efficacement.

Pour l'organisation des différentes tâches, nous utilisons Trello. Cela nous permet d'inscrire l'avancement dans différentes tâches afin de pouvoir nous situer dans le temps et communiquer sur des parties qui pourraient nous poser des problèmes.

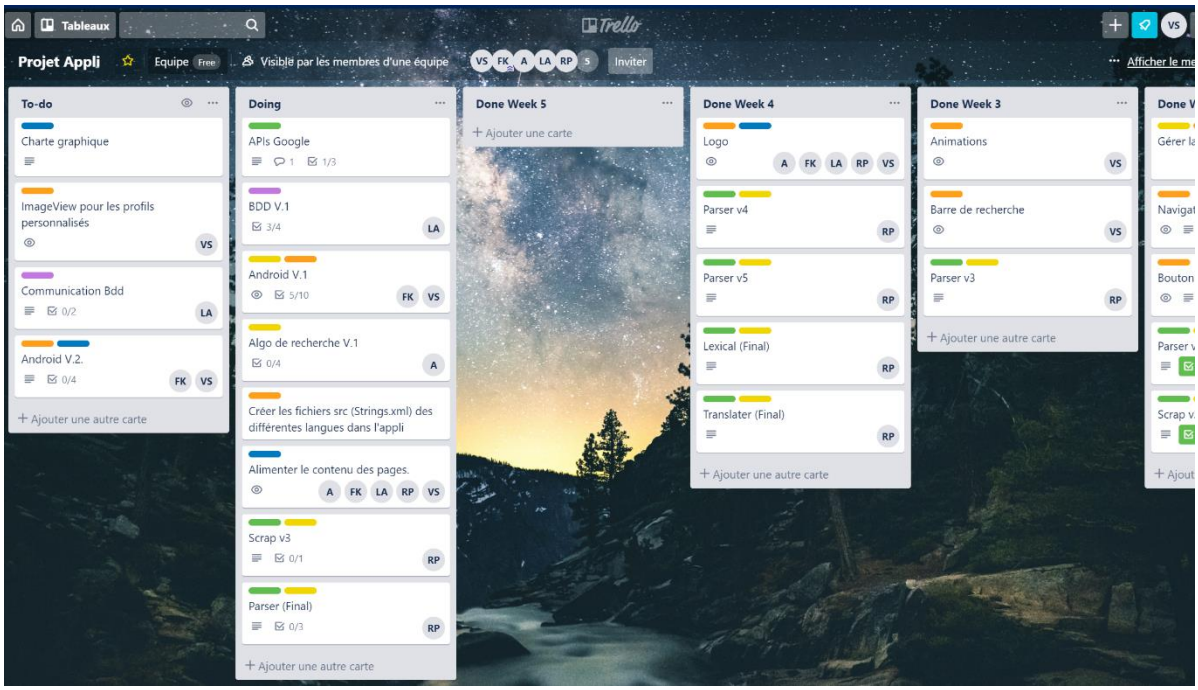
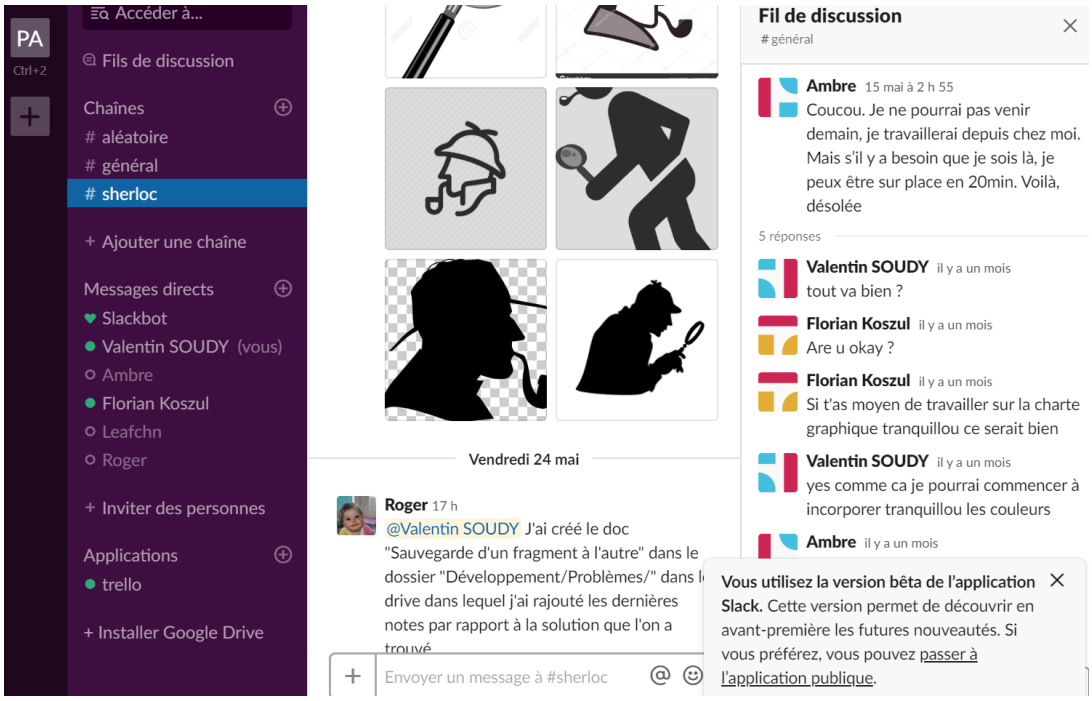


Figure 1 : Capture d'écran de notre Trello

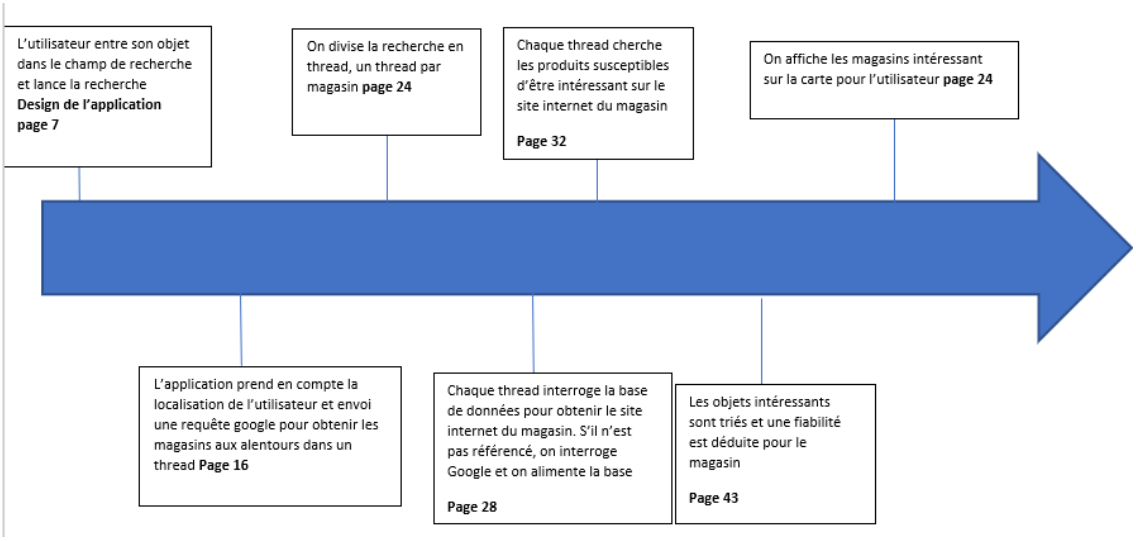
Nous avons mis en place un Github pour pouvoir partager notre code et l'héberger sur le cloud. Nous n'avons pas tellement utilisé cet outil car nous avions bien découpé les tâches entre nous afin d'être le plus indépendant de l'avancée des autres.

Nous n'avons réuni les parties de tout le monde qu'à la fin du projet, le raccordement était rapide car nous avions au préalable bien défini ce que nous attendions et ce que nous transmettions pour que la chaine de transmission de l'information soit bien respectée.



Pour la communication quotidienne, nous utilisons Slack. Cette application est un classique dans les communications de projet en entreprise et nous nous habituons à l'utiliser systématiquement. Nous avons ainsi pu communiquer comme le partage de documentations etc.

L'aboutissement d'un travail d'équipe



III. Design de l'application

Le design de l'application est important dans l'accroche que l'on va avoir auprès de l'utilisateur.

Il faut notamment que l'utilisateur retrouve les repères qu'il peut avoir dans les applications habituelles comme celles de Google. On étudie une attention particulière sur les applications proposant une carte comme Google Map.

Nous avons remarqué que la plupart des applications proposaient un squelette similaire.

1. Squelette de l'application

Nous avons choisi au départ de créer 2 activités au total.

La première que nous appelons « LoadingScreen » concerne l'ouverture de l'application. C'est l'écran de chargement à l'ouverture. Cette activité sera ensuite détruite d'où le fait que ce soit une activité qui ne serve qu'à cela.

La deuxième est MainActivity et elle regroupe tous les fragments qui constituent l'application. Le fait que tous les fragments de l'activité soient regroupés dans une même activité a facilité par la suite la communication entre les fragments.

Néanmoins, Nous nous sommes vite rendus compte qu'il était nécessaire de connaitre le cycle de vie d'une activité afin de pouvoir maitriser la création, la sauvegarde et la destruction de son contenu. Premièrement, nous avions opté pour la création systématique, d'un nouveau fragment quand on cliquait sur un item depuis la navigationView. Or quand on quittait le fragment, il était détruit et un nouveau est créé quand on veut retourner sur ce fragment :

```
    switch (item.getItemId()) {
        case R.id.nav_credits:
            getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full, R.anim.full_to_top)
                    .replace(R.id.fragment_container, new CreditsFragment()).commit();
            visibleMenuToolBar.setVisibility(View.GONE);
            visibleFragToolBar.setVisibility(View.VISIBLE);
            break;

        case R.id.nav_informations:
            getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full, R.anim.full_to_top)
                    .replace(R.id.fragment_container, new InformationsFragment()).commit();
            visibleMenuToolBar.setVisibility(View.GONE);
            visibleFragToolBar.setVisibility(View.VISIBLE);
            break;

        case R.id.nav_settings:
            getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full, R.anim.full_to_top)
                    .replace(R.id.fragment_container, new ParametresFragment()).commit();
            visibleMenuToolBar.setVisibility(View.GONE);
            visibleFragToolBar.setVisibility(View.VISIBLE);
            break;

        case R.id.nav_share:
            Toast.makeText( context: this,  text: "share", Toast.LENGTH_SHORT).show();
            break;

        case R.id.nav_history:
            getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full, R.anim.full_to_top)
                    .replace(R.id.fragment_container, new HistoryFragment()).commit();
            visibleMenuToolBar.setVisibility(View.GONE);
            visibleFragToolBar.setVisibility(View.VISIBLE);
            break;
    }
    drawer.closeDrawer(GravityCompat.START);//ferme le drawer quand on ouvre un fragment
    return true;
}//Renvoie le menuItem de l'item cliqué afin qu'il s'ouvre.
```

Comme on peut le voir ci-dessous (police jaune), nous créons à chaque fois un nouveau fragment quand on sélectionne l'item :

```
getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full,    R.anim.full_to_top).replace(R.id.fragment_container,    new InformationsFragment()).commit();
```

Cela pose notamment problème par la suite quand l'utilisateur rentre des informations puis quitte le fragment et y retourne. Cela engendre par exemple une réinitialisation de la recherche sur la carte quand l'utilisateur se rend sur un autre fragment et cette situation n'est pas souhaitable.

Voici le cycle de vie d'un fragment :

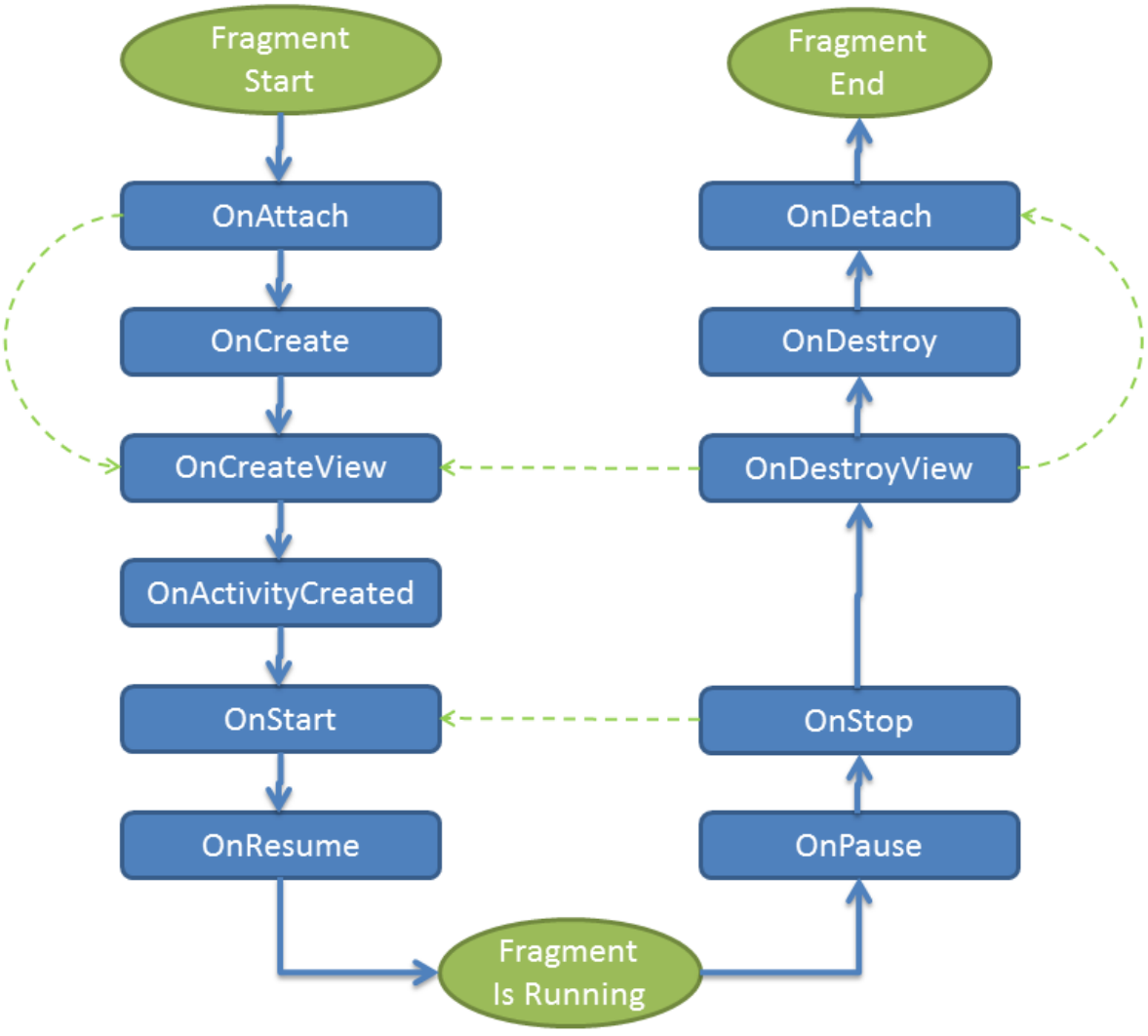


Figure 2 : Cycle de vie d'un fragment

Nous devons initialiser tous les attributs et aller chercher toutes les variables provenant des fichiers XML dans les fonctions onCreate et onCreateView afin de ne pas créer par exemple des null pointer exceptions. Dans notre problème, une fois le fragment quitté, la fonction onDestroy détruit le fragment.

Pour remédier au problème, nous procédons de la manière suivante :

```
switch (item.getItemId()) {
    case R.id.nav_credits:
        afficheFragment( tag: "credit",savedActivityInstance);
        visibleMenuToolBar.setVisibility(View.GONE);
        visibleFragToolBar.setVisibility(View.VISIBLE);
        break;

    case R.id.nav_informations:
```

```
public void afficheFragment(String tag, Bundle sauvegarde) {

    Fragment fragmentTMP = hashFrag.get(tag);

        if (fragmentTMP==null) {
            switch (tag) {
                case "param":
                    fragmentTMP = new ParametresFragment();
                    break;

                case "credit":
                    fragmentTMP = new CreditsFragment();
                    break;

                case "menu":
                    fragmentTMP = new Menu();
                    break;

                case "admin":
                    fragmentTMP = new AdminFragment();
                    break;

            }

            hashFrag.put(tag, fragmentTMP);
        }

    if (!fragmentTMP.isInLayout()) {
        getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.bottom_to_full,R.anim.full_to_top)
                .replace(R.id.fragment_container, fragmentTMP, tag).commit();
    }
}
```

Nous stockons dans une HashMap les différentes sauvegardes des fragments quand on les quitte et les restitue, s'ils sont créés. Lors de la prochaine sélection de l'item, on renvoie la sauvegarde du fragment concerné.

En entrant dans l'application, vous arrivez sur la page principale avec la carte.

Nous disposerons d'une barre de recherche pour taper le produit et d'une navigation view afin d'ouvrir un volet.

Drawer layout widget :

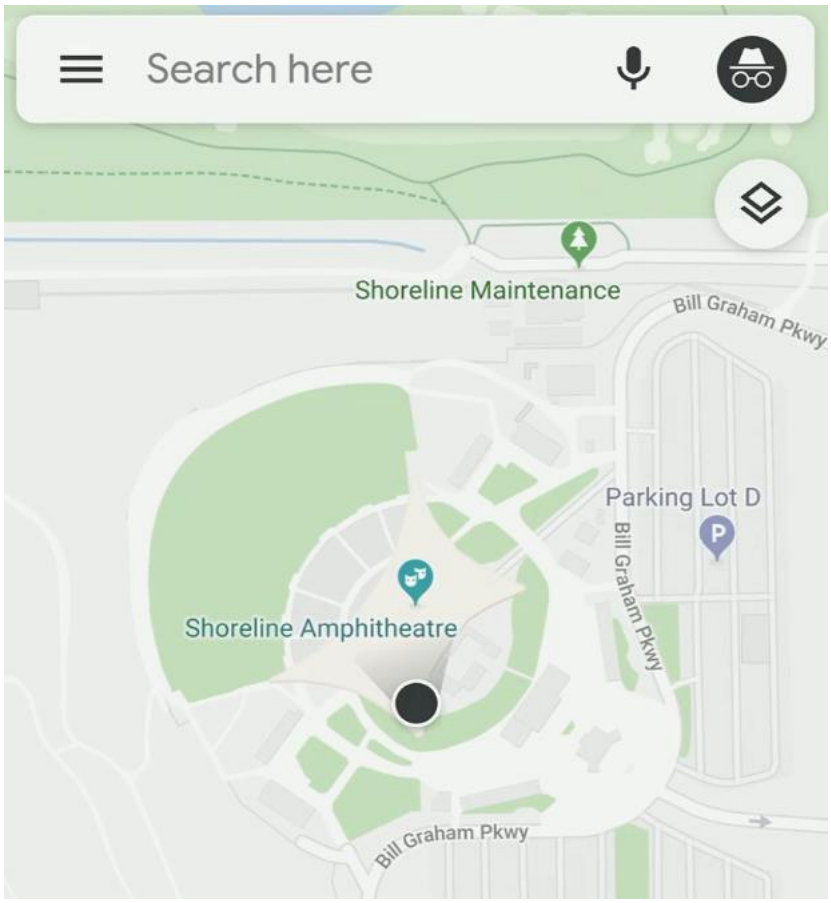
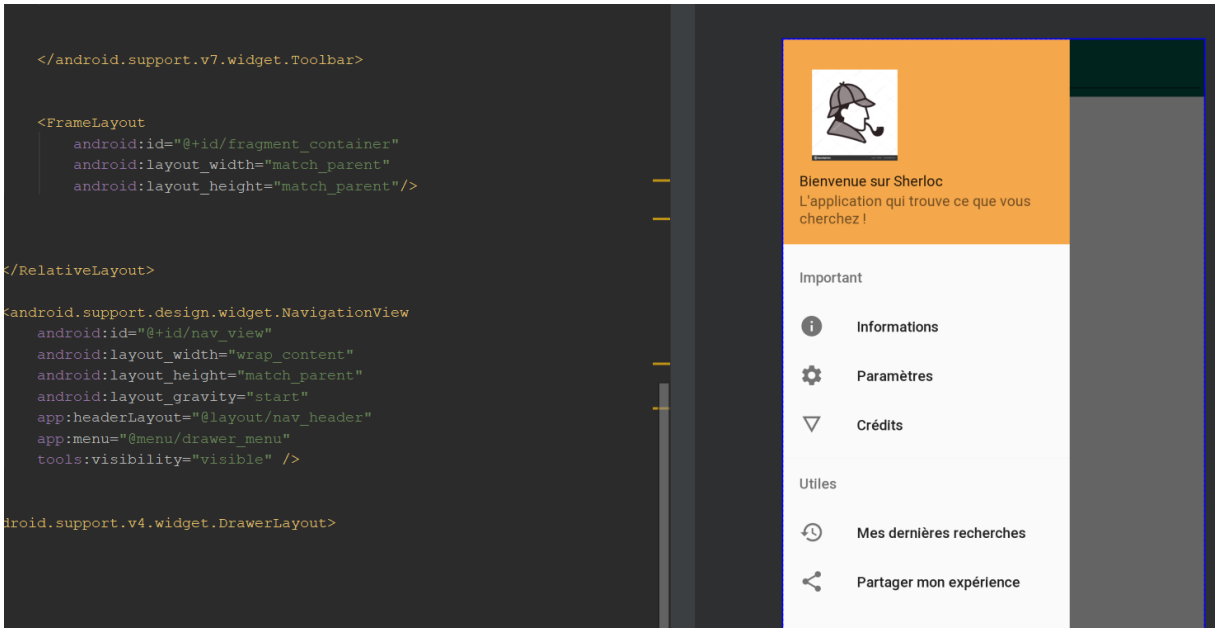


Figure 3 : Exemple de Google Map

En cliquant sur le bouton drawer, nous obtenons cette navigation View.

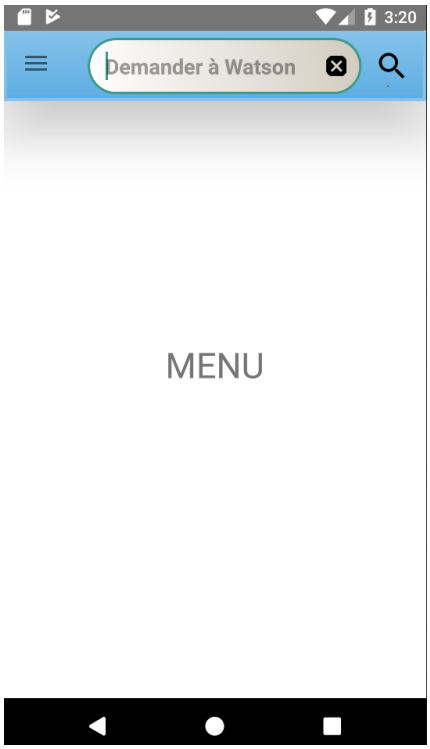
Voici un aperçu de la première Navigation View avec le code xml associé.



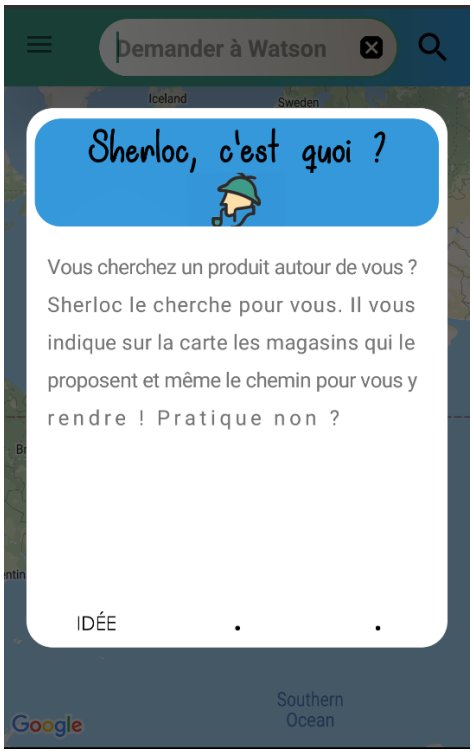
Ce volet a pour objectif de nous permettre de naviguer entre différents fragments que nous allons créer par la suite comme les fragments paramètres, crédits…

Par la suite, nous changerons et opterons pour la création de pop up sous forme de Dialog Fragment. En effet, cela fluidifie la navigation dans l'application et nous semble également plus esthétique pour l'utilisateur. Nous garderons l'aspect fragment que pour Crédits et le menu avec la carte.

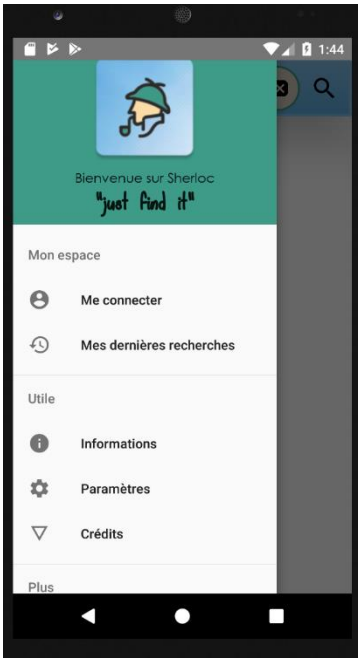
Avant :



Après :



En opérant des remplacements avec le nouveau logo et les nouveaux pop-ups, voici le nouveau volet.



En créant les différents EditText, nous nous sommes rendus compte que le clavier ne disparait pas automatiquement quand on clique ailleurs sur l'écran. Ce sont plusieurs problèmes que l'on n'imagine pas sans les avoir encore rencontrés. En tant qu'utilisateur, on n'imagine pas toutes les actions qui sont pour nous naturelles et qui sont gérées et anticipées par les développeurs.

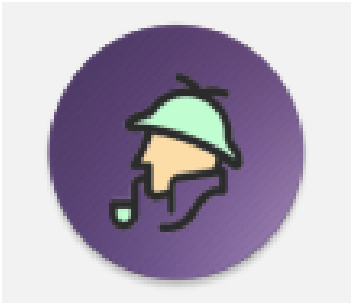
2. Création du logo

Nous avons créé différents prototypes pour le logo dont voici les illustrations.

Tout d'abord, le visage de Sherloc :

Nous avons opté pour le dernier avant de choisir le code couleur qui habillerait par la suite l'application via des rappels de couleur.

Voici les prototypes des différents combinaisons que nous avons faites :

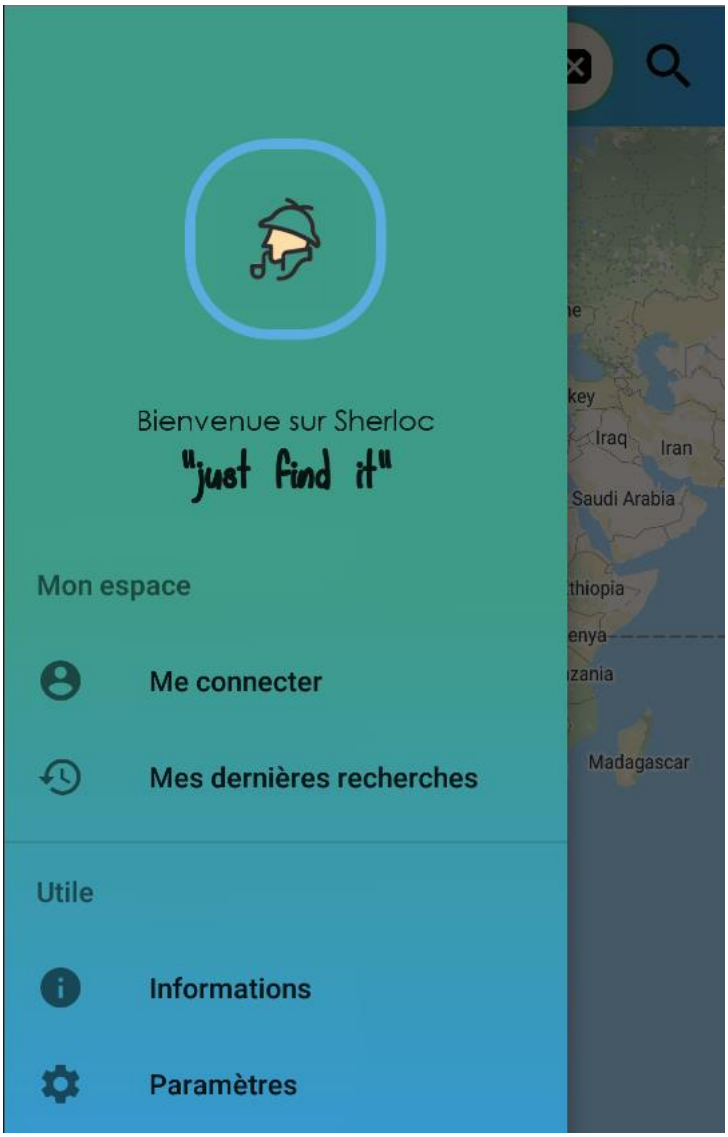


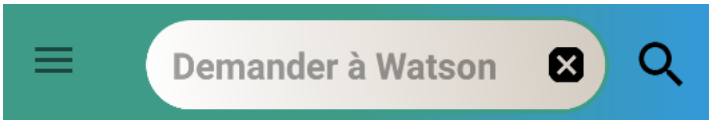
Avant d'aboutir sur notre version finale. A la fois, coloré, claire et joyeuse. Restant dans le thème Sherlock Holmes avec le chapeau vert :


3. Finition de l'application

Une fois le squelette de l'application terminé, nous nous sommes attaqués aux finitions de l'application afin de la perfectionner esthétiquement parlant.

Pour cela, nous avons créé des effets de style sur la plupart de notre interface :





Lors de la création d'un bouton pour revenir à la page du menu depuis un fragment, nous avons rencontré un problème en souhaitant centrer le bouton.

En effet quand on centre un bouton, on le centre par rapport à son parent… Voici ce que cela donne quand on centre le bouton « backHome ».

Le bouton est en effet centré sur son parent (cadre bleu) or nous le voulons centré sur l'écran entier. Le navigation View situé sur la gauche a le même niveau de hiérarchie que le parent du bouton. De plus nous aurions pu récupérer la taille que prend le nav button mais nous n'y sommes pas arrivés. Pour centrer le bouton « artificiellement » sur l'écran, nous avons effectué un décalage depuis la droite de l'écran d'un nombre de pixel classique chez la plupart des écrans à savoir 450px après des tests :

```
visibleFragToolBar.setPadding( left: 0,  top: 0,  right: 450,  bottom: 0);
```

IV. Repérer les magasins intéressants : déroulement d'une recherche

La recherche des magasins pouvant vendre l'objet par l'utilisateur est le centre d'intérêt de l'application.
La recherche suit une succession d'étapes et voit la signature de chaque membre de l'équipe dans son déroulement.

1. Intégration de la carte

La carte est un élément central de notre application. Le cahier des charges de notre application et les objectifs que nous nous sommes fixés inclus une localisation de l'utilisateur et un marqueur visible de sa position sur une carte tactile. L'ensemble des magasins proposants l'objet recherché seront affichés sur la carte. La carte est l'interface graphique reflétant le résultat de la recherche des produits.

Dans un premier temps l'objectif est d'intégrer la carte tactile dans l'application :

Il y a de nombreuses possibilités pour implémenter une carte sous Android studio. Nous avons décidé d'utiliser l'ensemble de ressources fournies par Google sous le nom de Google Map platforms.

Google Map Platform propose un ensemble de fonctions et de nouveaux objets java, dont l'objet google Map que nous pouvons manipuler dans un fragment d'activité.

Il y a des avantages certains à cette méthode : les packages fournis par Google pour implémenter la carte sont très bien optimisés pour Android Studio.

Ensuite la carte est manipulable à un très haut degré. Il y a un ensemble de de classes permettant de manipuler les zooms sur la carte, ajouter des markers à certaines localisation. Google est aussi une banque de donnée extrêmement conséquente et on profite de leur base de données en utilisant leur système de carte : Les rues et les magasins sont affichés sur la carte, bien que l'on n'ait pas de détails particuliers sur ceux-ci (il ne s'agit pas d'inclure Google-Map dans l'application, mais bien une carte jouant un rôle graphique de premier plan).

Nous n'avons pas eu de difficultés particulières à implémenter la carte.

Figure 4 : Carte de l'application sans géolocalisation

2. Implémentation de la géolocalisation

La géolocalisation permet de situer l'utilisateur et ainsi obtenir ses coordonnées géographiques : sa latitude et sa longitude. Il existe une procédure pour un objet map permettant d'afficher un marqueur bleu sur la localisation de l'utilisateur. Ce marker est régulièrement actualisé.

Cependant cette solution n'est pas suffisante car on a aucun moyen de disposer des coordonnées géographiques sous forme de longitude et de latitude.

Il existe plusieurs manières d'obtenir la géolocalisation d'un appareil. Par triangulation du réseau sur lequel le téléphone est connecté ou alors par GPS. Les différentes manières ont leurs avantages et leurs inconvénients, la triangulation du réseau demandera moins de batterie mais sera moins performante que le GPS, beaucoup plus gourmand.

Dans un cas général il est nécessaire de définir des critères de vitesses et de précision pour choisir le meilleur service de géolocalisation. Cette fonctionnalité fut la première implémentée mais elle était assez lourde en code et ne donnait pas de résultat très concluant : l'actualisation de la localisation était assez lente, peu précise alors que pourtant nous avions fixé tous cependant Google met à disposition un nouvel objet, FusedLocationClient, permettant de simplifier la mise en place de critère. Cet objet prend en paramètre un objet de type LocationRequest, un LocationCallback, et un thread qui agira en parallèle.

Le Locationrequest définit les paramètres de requêtes : on y règle l'intervalle de temps entre deux requêtes et la précision.

A l'aide de cet objet et de plusieurs fonctions, nous pouvons obtenir la latitude et longitude de l'appareil en temps réel et avec une très grande précision.

Il fut compliqué en revanche de gérer correctement les autorisations. Utiliser la géolocalisation d'un appareil nécessite forcément un accord de l'utilisateur. Pour cela on peut lui envoyer une fenêtre popup pour lui demander de l'activer.

Celui-ci peut accepter et refuser, il peut aussi décider de ne plus recevoir le pop-up, dans ce cas on affichera à l'écran une barre d'information précisant que le fonctionnement de l'application nécessite une activation de la géolocalisation et un bouton cliquable menant l'utilisateur dans les réglages.

Il est en effet important de penser à l'ergonomie de l'application qui est censé pouvoir être utilisée par des utilisateurs de smartphone novices.

Une fois la localisation activée, on peut obtenir en direct les coordonnées de l'utilisateur. Dans l'exemple sur la capture d'écran, on à placé un marqueur aux positions de l'utilisateur et on affiche sa longitude et sa latitude

3. Obtenir les informations des magasins

Dans le processus de recherche, l'application doit vérifier sur les sites internet de magasin si le produit est susceptible d'être trouvé.

Il est nécessaire de définir des critères : Quelles magasins doit-on étudier ? Comment récupérer leur site internet, leurs informations et y accéder ?

La première question est assez évidente : nous cherchons les magasins autour de l'utilisateur. Nous fixons pour le moment 1km de rayon autour de l'utilisateur mais il est prévu de pouvoir fixer cette distance dans une version prochaine.

Comment maintenant récupérer les magasins et leurs informations ?

Une solution assez évidente est d'utiliser une base de donnée disponible sur internet comme il en existent beaucoup : Google Map, Bing map, OpenStreetMap.. Les services référençant les magasins et leurs informations sur une carte ne manquent pas.

L'utilisation de la base de données de Bing Map est payante pour un nombre très élevé de recherches, ce qui laisse une certaine marge de manœuvre. Le service n'est en revanche pas optimisé pour Android Studio et il n'y a pas beaucoup de requêtes disponibles. Il y a également beaucoup moins d'informations que Google Map.

OpenStreetMap a l'avantage d'être totalement gratuit, la base de données étant notamment alimentée par les utilisateurs. Mais encore plus que Bing map, L'accès depuis l'application à la base de données est complexe et il y a peu d'informations sur les magasins, dont un certain nombre ne sont même pas référencés.

Google Map est le service proposant le plus d'information, c'est aussi le plus facile d'utilisation car il regroupe un grand nombre d'API permettant de simplement effectuer une requête http pour obtenir des informations de lieu.

Mais en contre-parti, c'est aussi celui le plus coûteux : le lot de 1000 requête est facturé depuis l'été 2018.

C'est la solution que nous avons choisie, les raisons étant expliquées ci-dessous :

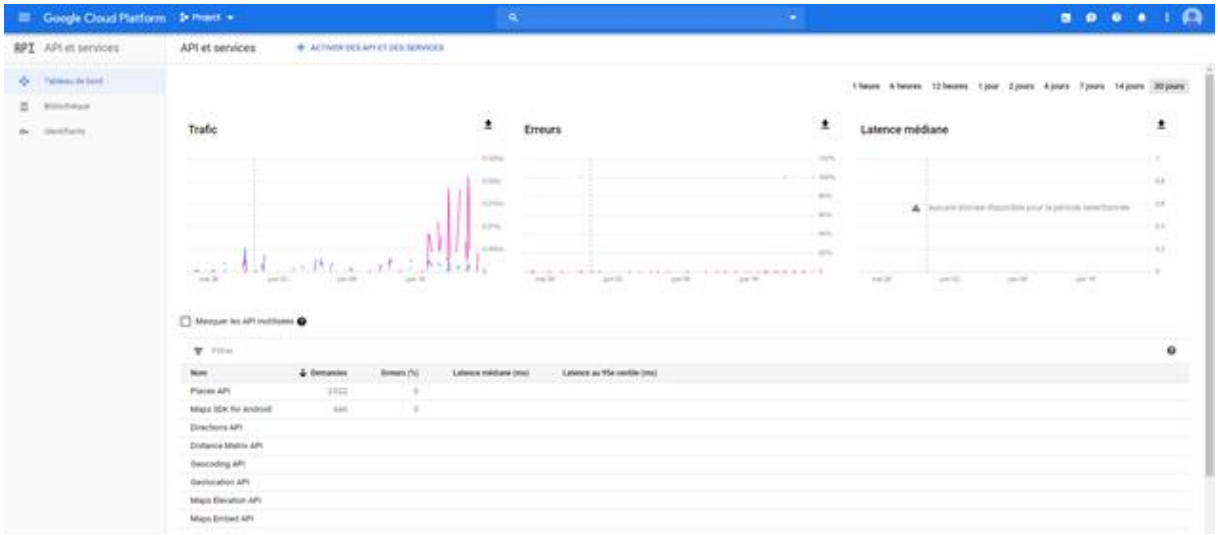
Google Cloud Platform

Google cloud platform est un service internet regroupant de nombreuses API. Permettant d'obtenir des informations de la base de données de Google map à l'aide de simple requête http.

Depuis 2018, un certain nombre d'API, notamment des API dont nous avons besoin pour le projet sont payante par lot de 1000 requêtes. Si cette contrainte de prix était un élément dissuasif au départ car demandant un certain investissement à perte dans le cadre du projet, nous avons trouvé une manière réaliste d'utiliser l'API sans dépenser un seul centime au moment où ce rapport est rédigé après la réalisation d'une première version de l'application

Dans le cadre du projet, nous avons besoin de l'Api map SDK, l'API Google Place et l'API Google Detail.

Pour utiliser le service, il faut créer un compte sur la platform et renseigner ensuite la clé fournie par Google dans notre projet android. Google laisse une ouverture pour les petits développeurs et la monétisation vise surtout les applis à grande échelle. Lors de la création du compte nous obtenons un crédit de 300$ gratuit de requêtes, amplement suffisant pour développer l'application car le coût de 1000 requête est de 2,5$ en moyenne. A la réalisation de la première version de l'application nous étions à 2000 requêtes.



Nous utilisons déjà l'L'API Google Map SDK, l'API fournissant la localisation de l'utilisateur et la Map. Celle-ci est appelé automatiquement et est gratuite

L'API Google Place permet de retourner un certain nombre de lieux avec un certain nombre de détail en fonction de la requête. Ces lieux doivent être enregistrés par Google sur Google map.

L'API Google Detail renseigne des détails sur un lieu en fonction de la requête effectuée.

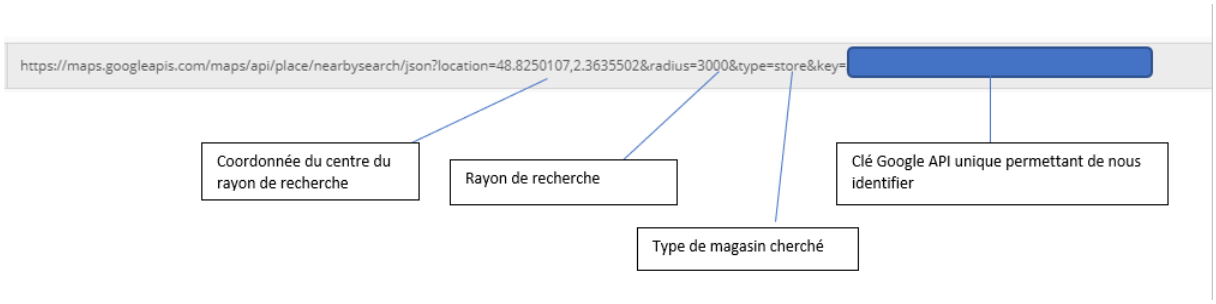
Pour obtenir les magasins autour de nous nous utilisons donc une requête Google Place.

Une requête pour obtenir les lieux autour de soi :

Pour obtenir le lieu autour d'un certain point, on utilise une requête de type « nearbysearch » à Google en renseignant certains paramètres :

- Les coordonnées du point central du cercle de recherche : Ces coordonnées sont la localisation de l'utilisateur.
- Un paramètre pour établir le rayon de recherche.
- Un paramètre pour établir le type de magasin recherché, dans notre cas on se contentera de « store », nous garantissant d'obtenir les magasins de tout type, bien que parfois certains magasins sans grand intérêt dans le cadre de l'application soient retournés (magasin de service événementiel par exemple).

On précise aussi notre clé fournie par Google pour nous identifier. Cette clé doit être sécurisé.




```
{
    "html_attributions": [],
    "next_page_token": "CqQCHAEAAMTizkM-pMr3k5uMy3ky-ANugQZHe9K6RW8fV4edDd1coX1Emcyme3YeRfjYsgUCnOt7I8BUgnQn9QYxhbZXedukCYnFEveCiPD1D3Hvr9-cQ1
    "results": [
        {
            "geometry": {
                "location": {
                    "lat": 48.8238934,
                    "lng": 2.3874909
                },
                "viewport": {
                    "northeast": {
                        "lat": 48.82530953029149,
                        "lng": 2.388817280291502
                    },
                    "southwest": {
                        "lat": 48.8226115697085,
                        "lng": 2.386119319708498
                    }
                }
            },
            "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/shopping-71.png",
            "id": "7152d19d6dee48e844668690c6af52abf3939822",
            "name": "Leroy Merlin Ivry-Sur-Seine",
            "opening_hours": {
                "open_now": true
            },
            "photos": [
```


Partie d'une réponse JSON fournissant les magasins alentours

Google nous renvoie une page en format Json qui est enregistré dans une variable.

L'avantage du format Json est qu'il est facile à parser, analyser à l'aide de classe dédiées à cette tâche déjà existante en java.

La réponse d'une requête pour obtenir les magasins autour de soi est constitué d'un champ « next_page_token » indiquant un lien servant de requête pour obtenir davantage de résultat.

La réponse est ensuite constituée d'une liste d'élément pour chaque magasin dont certains sont très utiles à exploiter pour nous : Le nom, les coordonnées, l'adresse, les horaires d'ouverture et le Google Id

Le Google id est la clé primaire de chaque magasin : il s'agit d'un identifiant unique pour tous les lieux référencés dans google Map.

Une requête n'est pas nécessaire car elle renvoie les informations de seulement 20 magasins au maximum. Il faut pouvoir exploiter le champs « next_page_token » pour effectuer une nouvelle requête pour en obtenir 20 de plus. Ainsi on peut effectuer jusqu'à 3 requêtes pour un total de 60 magasins. C'est un nombre satisfaisant mais qui impose tout de même une certaine limite.

Nous avons donc codé un algorithme envoyant la requête pour obtenir les magasins aux alentour et analyser la réponse en Json : Chaque magasin est enregistré dans une liste de Hashmap. Une Hashmap représente un magasin et contient son adresse, son état d'ouverture, son nom, son identifiant, sa longitude et sa latitude.

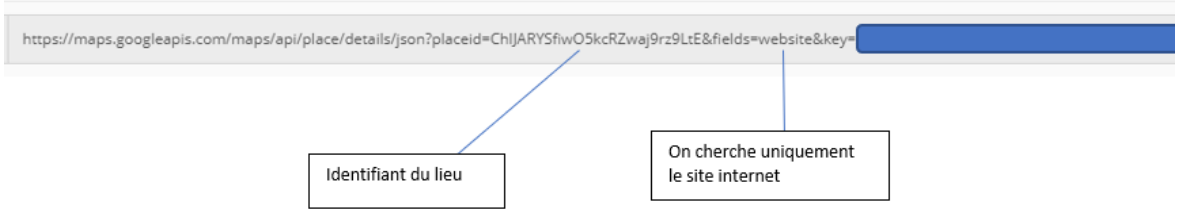
L'algorithme repère aussi s'il y a un champ de type « next_page_token » et effectue donc une nouvelle requête pour obtenir des magasins supplémentaires. Il continue ainsi de remplir la liste des magasins jusqu'à ne plus pouvoir obtenir de pages de résultat, donc 60 magasins maximum.

Une complication que nous avons rencontrée est que google impose un intervalle de temps entre chaque requête de ce type, et pour accéder à la requête pour avoir une nouvelle page de magasin, il a fallu imposer une attente d'environ deux seconde au processus pour ne pas recevoir un résultat nul.

Cette contrainte ajoute du temps à la recherche.

Une fois que nous avons la liste des magasins et les propriétés de nom, d'adresse, d'ouverture, de coordonnées et d'identifiant, nous devons obtenir le site internet pour pouvoir chercher dessus.

En effet, la requête que nous avons utilisé ne fourni pas le site internet dans le détail des magasins et nous devons utiliser une autre recherche de type « Detail Contact « pour chaque magasin afin d'obtenir le site internet à partir de son identifiant récupéré précédemment.



Une requête de type détail nécessite l'identifiant Google du magasin et notre clé personnelle Google.

Dans le cadre de notre projet nous avons seulement besoin du site internet, nous le précisons donc dans un paramètre

La réponse est en Json et ne contient que le site internet, elle est donc facile à Parser

4. Problématique du coût des requêtes

Effectuer trois requêtes pour obtenir les magasins aux alentour par recherche n'est pas très couteux mais si on doit à chaque fois effectuer jusqu'à 60 requêtes : une requête par magasins pour obtenir le site web, l'addition monterai à 63 requêtes par recherche. Alors que le modèle économique de profiter du nombre de requête gratuites proposé par Google suffisait, avec autant de requêtes par recherche, cela pourrait être trop coûteux sur le long terme et de devoir finalement payer, ce que nous voulons éviter absolument.

Nous avons donc établi une solution à ce problème : Nous créons une base de données sur un serveur distant pour stocker les informations des magasins et éviter d'effectuer la requête de détail du magasin pour les commerces déjà stockés dans la base de données.

Ainsi : lorsque la recherche est lancée, à l'aide des requêtes primaires on collecte les magasins à proximité. Dans un deuxième temps on interroge la base de données à l'aide de l'identifiant unique Google-id du magasin pour voir si les détails du magasin sont dans la base de données. Si oui on récupère le site internet et nous n'avons pas à faire de requête Google supplémentaire. Si non on effectue une requête Google pour obtenir le site internet et nous alimentons la base de données avec tous les détails du magasin pour ne plus avoir à faire de requête Google la prochaine fois que l'on a à traiter le magasin.

Sur le long terme cette solution économisera un très grand nombre de requête Google Detail.

5. Le multi-threading

Dans une application android, la connexion au réseau doit être fait dans un Thread différent du thread principal, nous appelons donc une Asynctask pour se connecter à l'API de Google et obtenir les magasins aux alentour.

Une fois que tous les magasins sont listés, il faut vérifier si chacun d'eux est répertorié dans la base de données pour obtenir le site web. Si ce n'est pas le cas, il faut se connecter à L'API Google pour récupérer le site internet et alimenter la base pour ne plus à avoir à faire de requête Google la prochaine fois qu'une recherche aura à traiter ce magasin.

Une fois le site récupérer, il faut le parser et chercher dessus la présence de l'objet recherché. Cela renvoie un certain nombre de produit qui sont ensuite traités par la l'algorithme d'évaluation qui juge de la pertinence du ou des produits retournés.

Effectuer chacune de ces étapes pour chaque magasin les uns après les autres s'avère très gourmand en temps et l'utilisateur n'aura sûrement pas envie de patienter plusieurs minutes devant son téléphone.

Nous avons donc décidé d'utiliser du multithreading : un thread permet l'exécution d'une tâche en parallèle du programme principal. Si on utilise plusieurs threads on peut donc traiter plusieurs magasins en même temps et gagner beaucoup de temps.

Les asynctask permettent une utilisation de thread pour de courtes tâches et ont une implémentation assez légère sur android studio : Une classe asynctask est composé de plusieurs fonction dont deux que nous allons utiliser : la fonction doInBackground qui exécute le code dans un thread à part. La fonction onPostexecute qui exécute le code dans le thread principal et prend en paramètre les résultats du travail du thread annexe.

Il y a certaines précautions à prendre avec les threads : ceux-ci sont dédiés à des tâches propres, si on veut manipuler des objets utiles à l'application dans le futur après que le thread soit exécuté, il faut mieux le faire dans le thread principal.

Notre approche est donc la suivante : le thread principal appelle un thread annexe pour obtenir tous les magasins aux alentour en interrogeant l'API de Google. La liste est des résultats est retourné au thread principal. Le thread principal se divise alors en une multitude de thread : un par magasin, pour effectuer la tâche de recherche. C'est-à-dire que chaque thread annexe interroge la base de données pour savoir si le magasin est déjà stocké dedans et si on peut récupérer son site internet.

Si oui le site internet est retourné, si non alors on interroge l'API de Google avec une requête de Google Detail pour obtenir le site internet et on prend le soin d'alimenter la base de données avec les informations du magasin pour éviter de devoir refaire une requête à Google la prochaine fois qu'un utilisateur sera confronté à ce magasin dans sa recherche.

Le thread va alors parser le site du magasin et retourner une liste des produits trouvés en fonction du mot recherché par l'utilisateur, cette liste est exploitée par un algorithme qui en déduit une fiabilité. Cette fiabilité est la probabilité théorique de trouver le produit dans le magasin.

Le thread principal est de nouveau appelé avec les résultats et la fiabilité est ajouté comme caractéristique du magasin dans la liste globale des magasins.

Pour lancer tous les threads on utilise une boucle qui se répète pour chaque magasin à étudier.

Cela peut représenter jusqu'à 60 threads et nous sommes alors limité par la technologie :

Un téléphone ne peut pas exécuter autant de threads en parallèle, le nombre de thread exécuté en parallèle dépend du nombre de cœur qu'il possède. Après des recherches, il s'avère qu'un téléphone de 8 cœurs peut par exemple exécuter 8 threads en parallèle. Dans notre cas nous nous sommes aperçut qu'uniquement 4 threads étaient exécutés en parallèle, nous en avons conclu que les autres threads étaient exécutés en fond pour d'autres application ou fonctionnalités du téléphone (par exemple le thread principal)

Les threads qui ne sont pas exécutés immédiatement sont placés en liste d'attente le temps qu'un thread ai terminé sa tâche et que la place se libère.

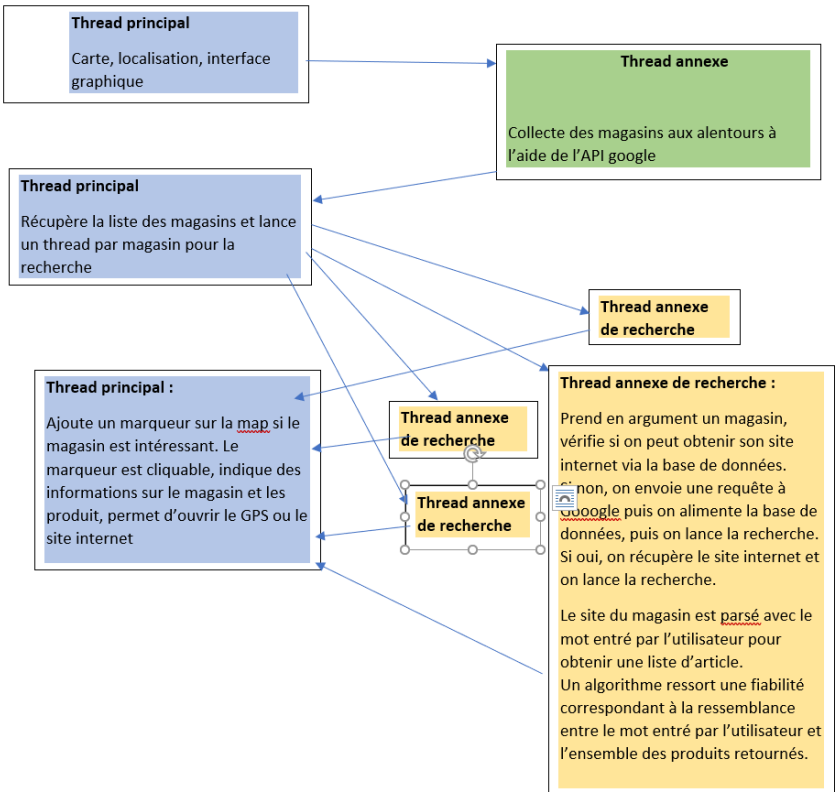
Cela induit le fait que tous les magasins ne sont pas traités en même temps mais quatre par quatre, ce qui reste assez rapide.

Ce principe de file d'attente nous a causé des problèmes pour instaurer des timeouts, cela veut dire instaurer un temps limite de fonctionnement au thread au cas où il se bloquerai. En effet, on peut fixer un handler au thread, cela veut dire une action qui se déclenche au bout d'un certain temps par exemple.

Le temps est décompté à partir du moment où on exécute le thread, or lorsque le thread est exécuté, il est placé en fil d'attente s'il n'y a pas de place pour exécuter le code. Le temps sera décompté même si le thread est en fil d'attente et la limite de temps interviendra alors que l'étude du magasin n'a pas encore commencé et que le thread est encore en fil d'attente.

Une fois qu'un thread a terminé l'étude du magasin, il ajouté les informations du magasin dans la liste et le thread principal affiche ces informations à l'écran si la fiabilité de trouver l'objet dans le magasin est satisfaisante.

Selon ce principe les résultats apparaissent au fur et à mesure à l'écran. Une recherche peut durer jusqu'à une minute. L'état de la recherche est caractérisé par une petite loupe qui tourne lorsque les threads étudient les magasins, ce qui permet à l'utilisateur de savoir que la recherche est en cours et que l'application travail.



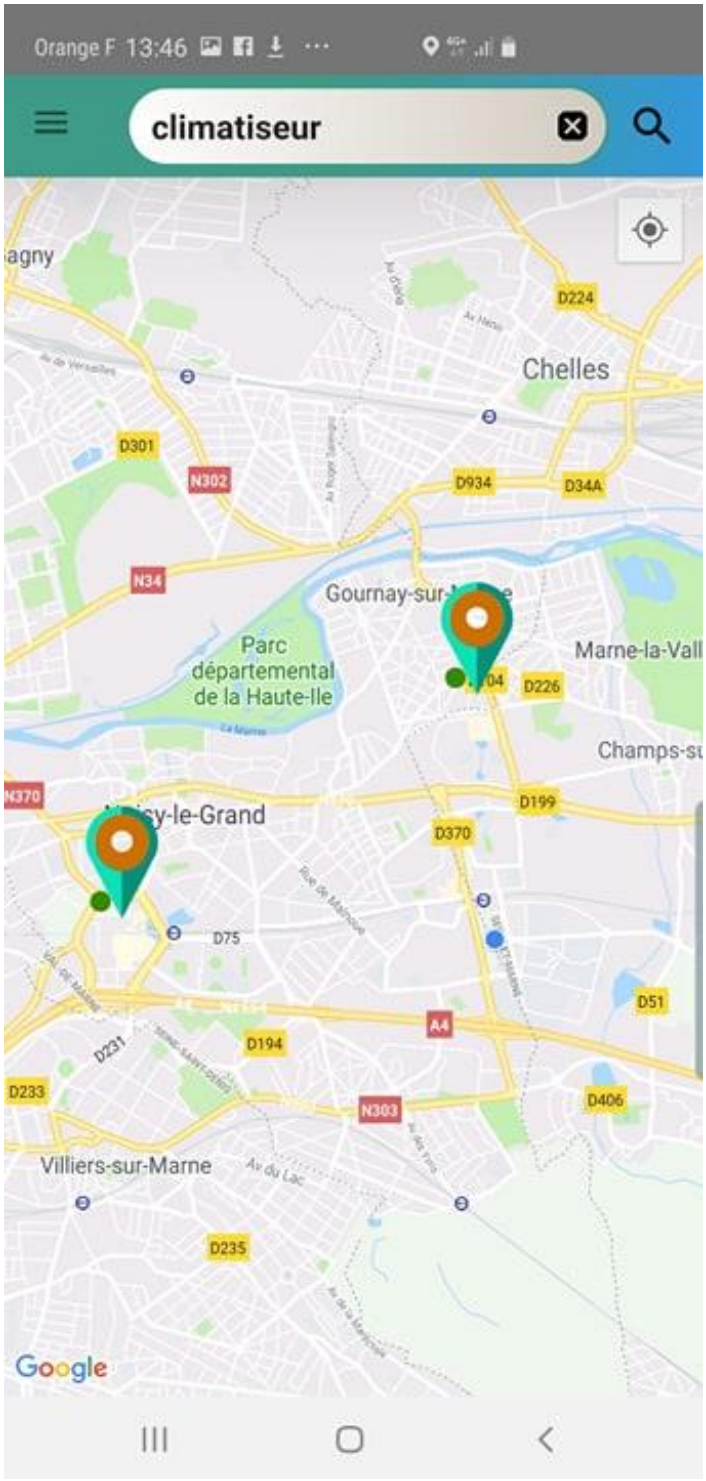


Figure 5 : Résultat de magasins proposant des climatiseurs

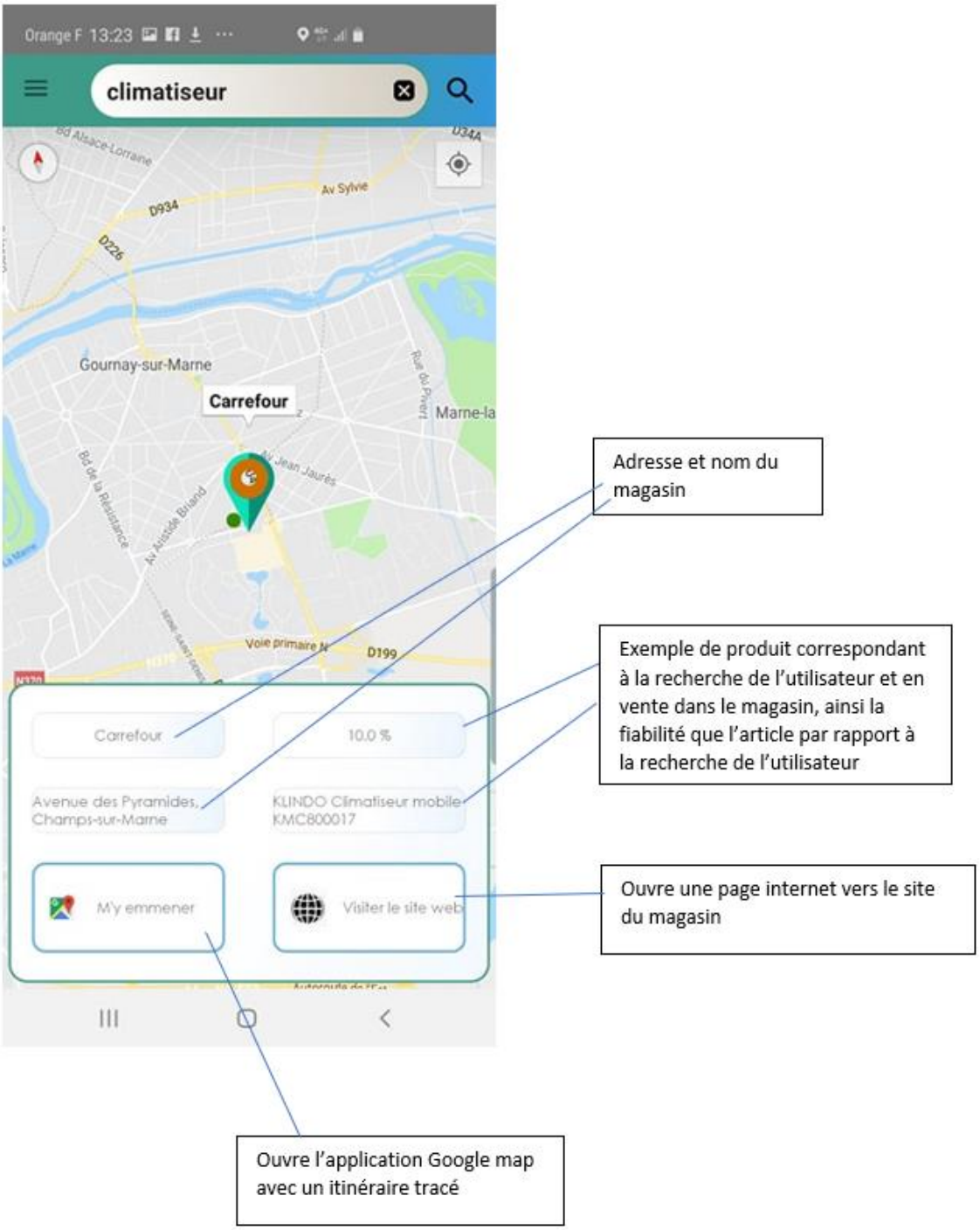
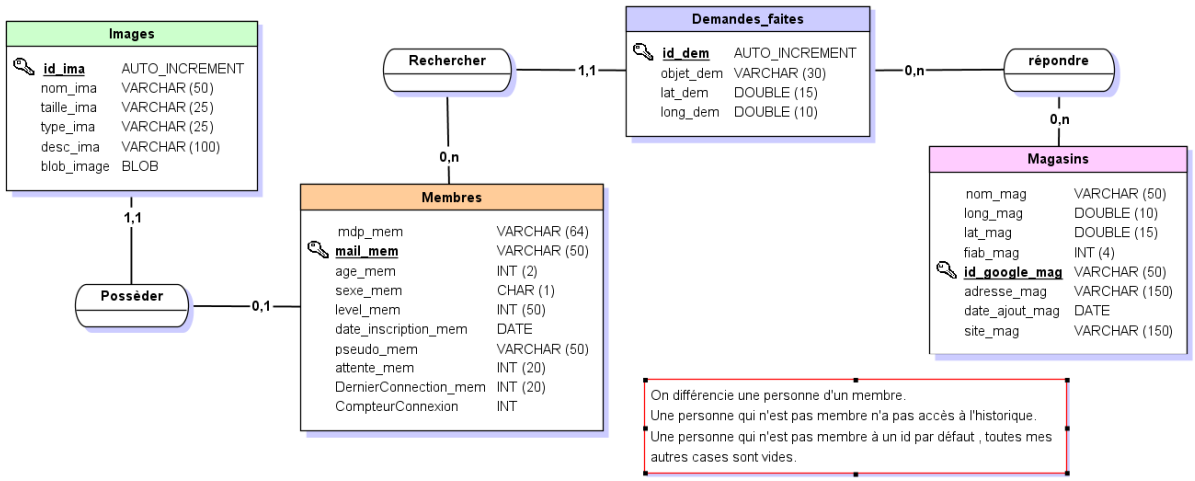


Figure 6 : Détails sur les informations de ces magasins

V. La Base de données

1. La création de la base de données

Pour notre appli, nous avons dû créer une base de données. Monsieur Larrat nous a généreusement réservé une partie de la mémoire de son serveur pour notre projet. Nous avons dû designer notre base donnée, concevoir un MCD et en ressortir un MLD. Cette phase de conception a été plutôt longue et le mcd a été changé à mainte reprise car l'application a beaucoup évolué même au fil des semaines. Nous avons commencé avec ce MCD :



MCD de la base de données

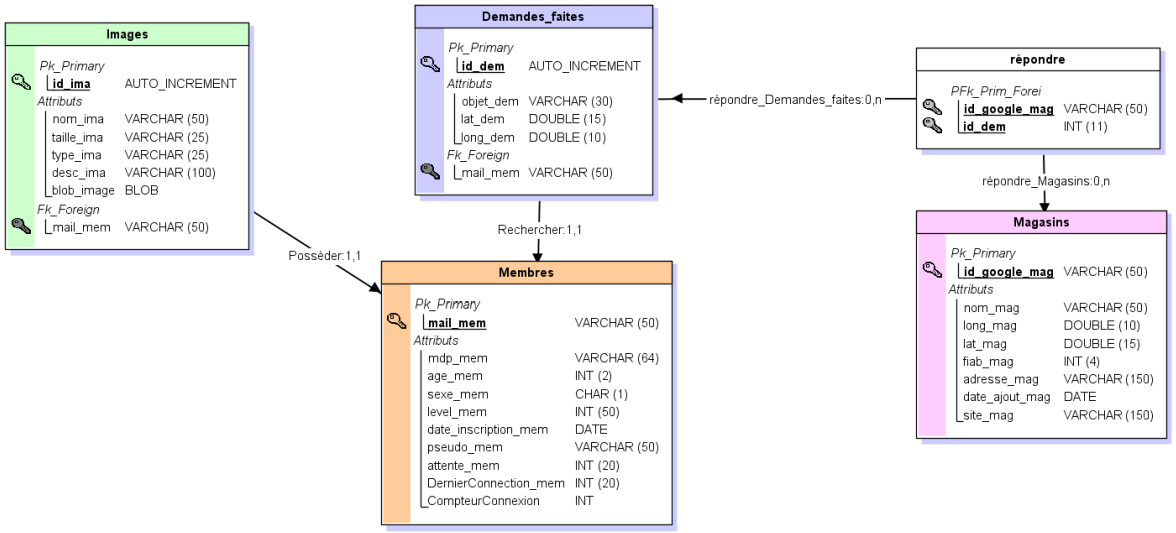


Figure 7 : MLD de la base de données

Comme nous pouvons le voir sur le schéma, nous avons un espace client personnalisable avec une photo de profil, et il est possible d'avoir un historique pour les utilisateurs inscrits. Pour la première version, nous voulions stocker la localisation et l'objet rechercher pour pouvoir faire des requêtes au

même endroit afin de créer un historique. Mais cela est trop puisant en énergie et demande de réeffectuer des requêtes payantes que nous avions faites auparavant. C'est ainsi que nous avons décidé de créer une table Magasins qui va permettre de stocker toutes les requêtes sur magasins que nous faisons.

2. L'implantation de la base de données

Après avoir designer et utiliser des noms de tuples sous un format adéquat, c'est-à-dire avec le nom de l'objet suivis d'un Under score et les trois premières lettres de la table. Ainsi il sera plus facile pour tout le monde de s'y retrouver. Ensuite il a fallu se connecter à Workbench MySQL, le serveur distant que nous avons décidé d'utiliser est sous maria DB mais compatible MySQL. Nous entrons le script qui permet de créer les tables et les dépendances entre elles : les tables ont table leur(s) clé primaire et leurs clefs étrangères associées. Nous avons rencontré plusieurs problèmes, car au début nous voulions créer et de rendre l'identifiant unique dans la table 'membres' puis nous avons décidé que ce sera l'e-mail qui servira pour s'authentifier. Voici un exemple de ce qu'on peut voir dans Workbench avec la table Magasins :

id_google_mag	nom_mag	long_mag	lat_mag	flab_mag	adresse_mag	date_ajout_mag	site_mag
ChIJ-T4Kfj9y5kcRvavrwrNCoXo	l-_appartement_d-animalis_pa...	48.833426	2.3875993	NULL	centre-commercial-bercy-village,-47-r...	2019-06-19	https://magasin.animalis.com/25165...
ChIJ-_Kp2E1y5kcRwrHWHo_jFEc	auto_feu_vert_ivry_sur_seine...	48.81999649999999	2.3957182	NULL	centre-commercial-bords-de-seine,-2...	2019-06-20	https://www.feuvert.fr/centres-auto/i...
ChIJ04lToX4N5kcR6MclxZ3ekwI	monceau_fleurs_-_fleuriste_p...	48.8495601	2.4983228	NULL	44-boulevard-d-alsace-lorraine,-le-per...	2019-06-20	http://www.monceaufleurs.com/
ChIJ08ON0d4N5kcRCsyUeQisiaI	pharmacie_des_tilleuls	48.827414	2.5413632	NULL	47-rue-du-general-de-gaulle,-villiers-s...	2019-06-18	http://www.pharmatilleuls.byethost7....
ChIJ0RBVH54S5kcRAvbtJvlNDLk	traiteur_auchan_fontenay-so...	48.853441	2.4832174999999999	NULL	avenue-du-marechal-joffre,-fontenay...	2019-06-20	https://traiteur.auchan.fr/magasin/au...
ChIJ0RBVH54S5kcRSScT1ADadfY	auchan_fontenay_sous_bois	48.8534117	2.4835492	NULL	avenue-du-marechal-joffre,-fontenay...	2019-06-20	https://www.auchan.fr/magasins/fon...
ChIJ0ZwrNlhy5kcReeXK-huC_-8	tang_freres	48.8236783	2.3654755	NULL	48-avenue-d-ivry,-paris	2019-06-17	http://www.tang-freres.fr/
ChIJ15b0DDQO5kcR89090cGSqGo	s3s	48.83556950000001	2.5861265	NULL	18-rue-albert-einstein,-champs-sur-m...	2019-06-17	https://www.groupe-s3s.fr/contact
ChIJ1epw2pgN5kcRrlj7z1CkPQA	saga_nogent_-_citroen	48.83489419999999	2.5065258	NULL	127-129-avenue-pierre-brossolette,-l...	2019-06-20	https://www.gueudet.fr/concession/c...
ChIJ1W-cwwsN5kcR_adysqH1JFA	on_off_project	48.8342208	2.5751106	NULL	20-rue-du-ballon,-noisy-le-grand	2019-06-17	http://www.onofflighting.fr/
ChIJ1WhbjX1x5kcRx-XWdZ3enpQ	e.leclerc	48.809073299999999	2.36219	NULL	106-avenue-de-fontainebleau,-le-kre...	2019-06-19	http://www.e-leclerc.com/le-kremlin
ChIJ1_6cQo9x5kcRYkl0IJkD7xo	maisons_du_monde	48.8265336	2.3574794999999999	NULL	57-avenue-d-italie,-paris	2019-06-17	https://www.maisonsdumonde.com/...
ChIJ2VseprRz5kcRVeM7m6hMPmw	leroy_merlin_ivry-sur-seine	48.8238934	2.3874909	NULL	2---12-rue-françois-mitterrand,-ivry-s...	2019-06-19	https://www.leroymerlin.fr/ivry-sur-s...
ChIJ2_U7Slhy5kcR2Sbhkthlp7Q	paris_store	48.8234067	2.365931	NULL	44-avenue-d-ivry,-paris	2019-06-17	http://www.paris-store.com/
ChIJ2_U7Slhy5kcRZtLvR96Hr0Y	boutique_musica	48.8237201	2.3661375	NULL	44-avenue-d-ivry,-paris	2019-06-17	http://www.asiaworldmusic.fr/
ChIJ2_U7Slhy5kcR_TceygPn6UI	tai_you	48.8235254	2.3657049	NULL	44-avenue-d-ivry,-paris	2019-06-17	https://www.taiyou.fr/
ChIJ3cEAXY5x5kcRKhJquA8bqPw	go_sport	48.8292678	2.3552186	NULL	30-avenue-d-italie	2019-06-17	https://stores.go-sport.com/fr-fr/ma...
ChIJ3cEAXY5x5kcRs-Dk4QPD1-c	fnac	48.8292678	2.3552186	NULL	30-avenue-d-italie-centre-commercial...	2019-06-17	https://www.fnac.com/Paris-Italie-2/F...
ChIJ3cEAXY5x5kcRS8OYo1IpIFM	bouygues_telecom	48.8292678	2.3552186	NULL	30-avenue-d-italie-ccial-italie-2,-paris	2019-06-17	https://boutiques.bouyguestelecom.f...
ChIJ3cEAXY5x5kcR_imLMHhl8Lc	foot_locker	48.8292678	2.3552186	NULL	cc.-italie-ii,-30-avenue-d-italie,-paris	2019-06-17	https://www.footlocker.fr/
ChIJ3e3_OzMO5kcRtaQkPpZTwPc	mb_monetique	48.837969	2.5904775	NULL	23-rue-alfred-nobel,-champs-sur-mar...	2019-06-18	-NA-
ChIJ3QkDP-cP5kcRqfqJY7Jh7vY	market_du_luzard	48.8443048	2.6154962	NULL	75-cours-des-roches,-noiseil	2019-06-20	-na-
ChIJ3TNHbjxy5kcRL_7nWVo8DpU	manga_cafe_v2	48.8308478	2.379798	NULL	9-rue-primo-levi,-paris	2019-06-19	http://www.mangacafe.fr/

Table Magasins vu sur Workbench

3. Création et mise en place des services web

Maintenant que la base de données est créée, il faut faire communiquer l'application avec celle-ci. Nous ne connaissions pas grand-chose en gestion de bases de données, uniquement les cours que nous avons suivis à l'ESIEE : savoir comment la créer et en soutirer des informations intéressantes. Nous avons fait beaucoup de recherches sur internet qui n'ont pas porté leurs fruits. Nous avons essayé avec Symfony, un logiciel utilisé dans le monde professionnel, composer, git ou le logiciel Firebase. Firebase était notre dernier choix car il y a beaucoup de points négatifs à l'utiliser comme le fait que google ait accès aux informations ou encore que le serveur puisse de temps en temps se fermer. Mais nous avons également directement essayé avec le driver JDBC d'Android. Mais ce dernier était une très mauvaise idée car les identifiant de connexion et le mot de passe de la base de données été envoyé en clair. Cela est très dangereux pour notre application et la base de données aurait été vulnérable et propice à l'attaque de « hackers ». Nous avons donc décidé avec l'aide de Monsieur Larrat d'utiliser des services web. Nous avons dû apprendre à coder en PHP, un langage qui nous est tout à fait inconnu. Après s'être instruit à ce sujet nous avons créé notre premier script :

Connection.php, ce script permet de créer une connexion entre la base de données du serveur. En créant ce script il suffit ensuite de le mettre en 'require' et le lien est créé pour tous les scripts.

Avec getmagasin.php nous pouvons retrouver le magasin stocké dans la base de données. Il est appelé dans la fonction historique pour retourner les magasins où nous avons trouvé l'objet précédemment cherché. De plus nous pouvons regarder si le magasin trouvé existe déjà dans notre base de données ou s'il faut refaire appelle à une requête google. Nous retournons un Json.

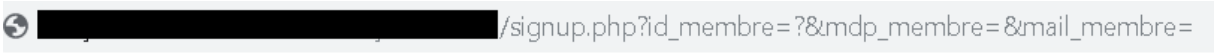
insertmagasin.php qui permet à l'aide d'une requête SQL d'insérer les données du magasin dans notre base de données. Cet algorithme est surtout utilisé lorsque nous faisons des requêtes avec l'API google afin d'éviter de refaire des requêtes déjà effectuée. Voici la requête SQL ainsi que son url :

Getmail.php permet de retourner le mail de membre inscrit si nous l'entrons en paramètre. Cela permet à l'utilisateur de s'identifier dans notre base de données, se connecter mais aussi personnaliser sa session. Le fichier de sorti est un json

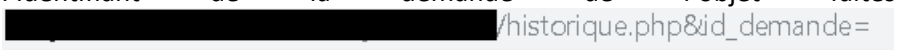
Updateconnexion.php : Avec ce script PHP nous pouvons mettre à jour la table membre afin de savoir combien de fois l'utilisateur à essayer de s'identifier, après un certain nombre de tentative de connexion, la personne doit attendre un lapse de temps.

Supprmdp.php permet de modifier le mot de passe de l'utilisateur et supprcompte.php permet de supprimer le compte utilisateur

Signup.php est un script qui enregistre toutes les données du client dans la base de données membres lors de son inscription



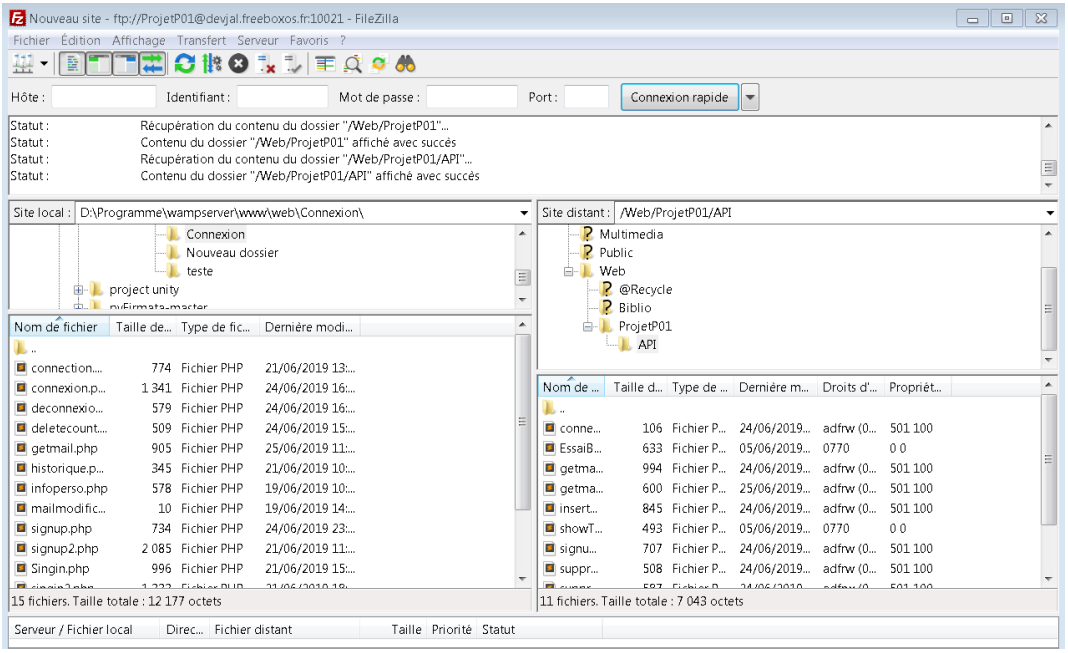
Historique.php est une fonction qui marche comme telle : nous alimentons la table demande_faites avec le mail_membre, la date et l'objet. Il faut remplir la table répondre avec l'identifiant de la demande précédente et faire une boucle pour ajouter chaque magasin trouvé et ensuite faire un inner join avec les demandes_faites, repondre et les magasins. Où l'identifiant de la demande est égal à l'identifiant de la demande de l'objet faites par l'utilisateur



Ce qui a été présenté ci-dessus sont les scripts de bases, il y avait un script pour chaque fonction. Mais j'ai créé deux nouveaux scripts : get.php et update.php. Grâce aux fonctions empty et isset j'ai pu regrouper les scripts précédents dans un seul script php. En fonction de ce qui est entrée en paramètres, la requête qui est associé s'effectue. Ainsi Get.php et uptadate.php s'occupe respectivement d'afficher en json l'élément rechercher et de modifier ou insérer dans les bases de données.

Tous les services avec le préfixe get n'affichent pas le Json s'il y a un caractère spécial tel que le les accents ou le c cédille. Pour remédier à cela, nous retournons et entrons toutes les données en minuscule et sans accents

Lors de la création de ces fonctions, nous avons utilisé WampServer et phpMyAdmin pour tester nos service web localement avant de les implanter via Filezilla sur notre serveur.



Connexion serveur avec Filezilla.

La sécurité des bases de données.

Une chose très importante lorsque nous créons les comptes utilisateurs : nous avons la responsabilité des données de nos utilisateurs et nous devons au mieux les protéger. Nous avons utilisé la fonction prepare() et execute car elle n'exécute pas en simultané la requête SQL. Pour les paramètres entrés en url, nous avons également les fonctions suivantes :

```
$id_google_mag = addslashes(htmlspecialchars(htmlentities(trim($_GET['id_google_mag']))));
```

Addslashes enlève tous les guillemets doubles et simple, ainsi que les antislash et l'élément NULL. Htmlspecielchars et Htmlentitites convertit les caractères spéciaux en entités html. Et Trim supprime les espaces en début et fin de chaîne

Toutes ces fonctions sont issues de la bibliothèques PHP. Ainsi nous sommes en sécurité vis-à-vis de l'injection SQL. De plus nous avons ajouté un système de d'attente lorsque l'utilisateur se trompe trop de fois en essayer de se connecter. Ainsi nous pouvons éviter le Brut force.

VI. Partie web

La partie web constitue la plus grande source de toutes les données qui seront affichées et utilisées par l'application, effectivement à partir de la liste de magasin récupéré précédemment on va « scraper » sur ces sites toutes les informations que l'on veut : ici on va rechercher le produit souhaité par l'utilisateur. Avant de continuer dans les détails de mise en œuvre de cette partie, mettons au clair les données que nous voulons récupérer ; ce sont pour chaque produit sont nom de vente ainsi que son prix.

Scraper, c'est quoi ?

Le « scraping » est un terme anglais qui se traduit par « grattage » très utilisé dans l'informatique puisque dans le web il décrit l'action de récupérer des informations d'un site web, on peut y voir notamment des utilisations dans la veille pour surveiller les évolutions des tarifs ou autres valeurs marchandes.

C'est dans ce procédé que nous trouvons la première étape de cette partie Web ; qui a pour but de se connecter au site dont nous avons l'adresse et de pouvoir y exploiter son contenu comme le ferait un utilisateur navigant sur internet.

Dans une première version comme nous avions déjà utilisé ce procédé pour récupérer des données sur un site web avec le langage Python, nous avons utilisé une librairie similaire sous Java qui se nomme Jsoup[1]. Cette librairie permet en plus de pouvoir se connecter à un site internet, de pouvoir y effectuer des manipulations sur le code source de la page écrit en HTML+CSS. Ces manipulations ont pour but de récupérer des éléments de la page internet à l'aide de recherche par balise ou encore nom de classe que nous verrons plus en détail par la suite.

Cependant un premier problème est survenu est l'erreur de requête HTTP 403[2] ; pour résumé cette erreur survient pour la majorité des cas lorsque l'accès à une adresse est refusé. En effet, pour la plupart des sites web auxquels le programme essayait des se connecter, cette erreur survenait. La raison de se blocage réside dans le fait que le serveur vers lequel la requête GET (demande du code de la page) est envoyée détecte une instance Java au lieu d'un utilisateur derrière son ordinateur voulant se connecter à un site. De ce fait, pour prévenir du Botnetting, utilisation de robot pour naviguer et récupérer des informations sur ces sites, on modifie donc le Header de la demande GET du protocole HTTP. Le Header correspond comme son nom l'indique à l'en-tête de la demande et il contient une quantité d'informations qui vont permettre au récepteur d'agir en conséquence dans son éventuelle réponse.

[1] https://jsoup.org/
[2] https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203551-erreur-403-definition/

La première solution a été de modifier cet Header avec des informations qui étaient récoltées à l'aide d'un plug-in nommé « HTTP Header Live [3]» qui va permettre de voir les informations échangées lors du chargement d'un site web. En les récupérant on peut effectuer cette demande mais que pendant un certain temps. En effet, il faut ensuite récupérer manuellement la valeur de l'attribut cookie qui semble être la source du problème.

La première solution n'étant pas pleinement satisfaisante, il nous fallait trouver une autre solution ; après quelques recherches nous avons découvert l'API[4] HtmlUnit[5] permettant de se faire passer pour un navigateur web et également de pouvoir interagir avec la page en question. Grâce à cet API, nous avons été en mesure de nous connecter à une page web ; cette connexion se fait en deux étapes en termes de programmation. Dans un premier temps nous créons l'objet Java : navigateur web (que nous appellerons le navigateur par la suite) en utilisant les outils fournis puis nous nous connectons à la page en question à l'aide de son URL[6] et d'une méthode de notre navigateur.

```
this.navigateur = new WebClient(BrowserVersion.BEST_SUPPORTED);
this.navigateur.getOptions().setJavaScriptEnabled(false);
this.navigateur.getOptions().setThrowExceptionOnFailingStatusCode(true);
this.navigateur.getOptions().setThrowExceptionOnScriptError(true);
this.navigateur.getOptions().setCssEnabled(false);
```

On peut voir sur la capture ci-dessus comment le navigateur est défini en faisant appel à la classe *WebClient* et en créant une instance de cette dernière. Puis nous utilisons les différentes méthodes pour configurer ce navigateur comme nous le souhaitons, dans notre cas nous avons désactivé le chargement du CSS et du JavaScript qui sont tous les deux des outils pour améliorer l'aspect visuel d'une page internet ainsi que l'interaction, car nous n'en avons pas l'utilité. De plus nous avons activé le *throw* des différentes exceptions possibles pour que lors de la connexion à une page ces exceptions ne fassent pas quitter le programme.

```
this.page = this.navigateur.getPage(this.url);
```

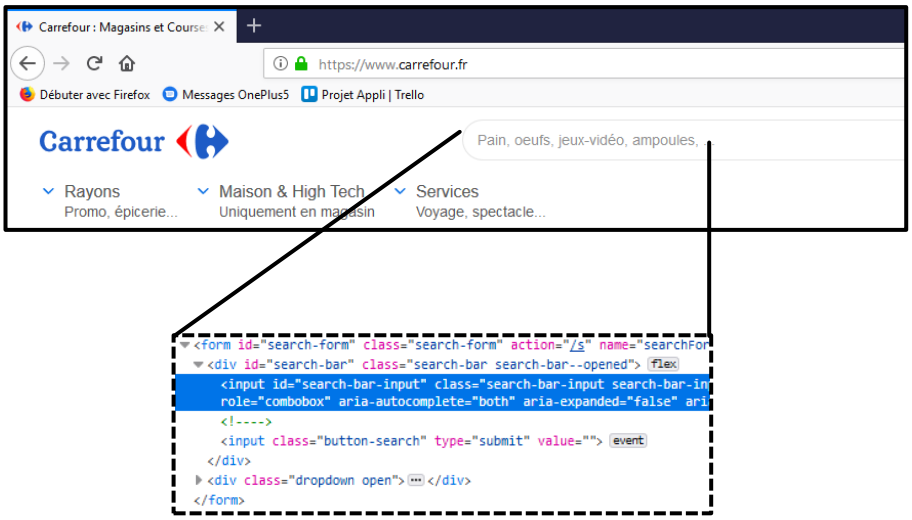
Ensuite, pour nous connecter à la page internet nous utilisons la méthode présente sur la capture ci-dessus en lui passant en paramètre l'adresse de cette page internet.

Une fois connecté comment rechercher un produit ?

Voici la deuxième problématique du *Scrap*, effectivement une fois que nous sommes connectés à la page nous arrivons à la page d'accueil du site commerciale puisque c'est cette adresse qui est référencée par l'API de Google et donc celle qui nous est fournie.

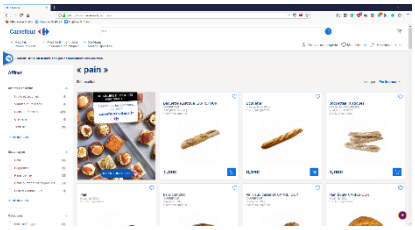
[3] https://github.com/Nitrama/HTTP-Header-Live
[4] API : Application Programming Interface ou Interface de Programmation, constitue un ensemble de normalisé d'outils informatique.
[5] http://htmlunit.sourceforge.net/
[6] Uniform Ressource Locator

Or comme dit précédemment l'API HtmlUnit nous permettant la connexion à la page nous propose également un service d'interaction avec les différents éléments d'une page internet tels que les formulaires à remplir ou encore les boutons.0



Avant d'expliquer la suite du programme, il nous faut détailler un peu la structure de la page à laquelle nous nous sommes au préalable connecté. Ci-dessous une capture du site de Carrefour [7] qui servira d'exemple ; nous avons zoomé sur la partie nous intéressant c'est-à-dire la barre de recherche. En effectuant un clic droit sur cette barre de recherche et ensuite « Examiner l'élément » ; tout ça depuis un navigateur tel que Firefox nous pouvons visualiser le code HTML correspondant à cette partie de la page (photo encadrée en pointillée ci-dessus). On constate que le champ dans lequel la recherche est tapée possède la balise HTML « input » qui est elle-même contenue dans une balise « form ». Après des recherches nous nous sommes rendus compte qu'l était convention de faire de la sorte pour les barres de recherche mais nous verrons quelques exceptions par la suite.

Ayant identifié la balise concernée nous pouvions récupérer l'élément à l'aide des méthodes fournies dans l'API HtmlUnit. Néanmoins il nous fallait trouver un moyen d'émettre la requête après avoir rempli le champ de recherche avec le produit voulu par l'utilisateur, mais également d'obtenir la nouvelle page internet issue de cette requête. Pour ce faire nous nous sommes basés sur les travaux d'un internaute [8]qui créer un bouton pour émettre la requête et qu'il l'ajoute par programmation au formulaire récupéré avant-coureur, pour ensuite cliquer toujours par le code sur celui-ci. De ce fait une requête va être émise et nous sommes en mesure de nous procurer la nouvelle page. Si nous poursuivons notre précédent exemple et que nous recherchons du pain nous allons avoir d'afficher sur notre navigateur la page suivante :



[7] https://www.carrefour.fr/
[8] https://stackoverflow.com/questions/7573558/htmlunit-how-to-post-form-without-clicking-submit-button

À la fin de cela nous étions en mesure, par utilisation d'un programme Java, de nous connecter à une page connue et d'y émettre une recherche en récupérant la réponse, la nouvelle page web dans notre cas, de cette recherche.

Néanmoins ce programme était loin d'être universel puisque pour trouver le formulaire sur une page nous l'identifions par le nom de ses attributs, seulement ces noms changent d'une page à l'autre ; par conséquent pour chaque site web nous devions connaître le nom de l'attribut en question pour pouvoir faire fonctionner le reste du programme. Il en est évidemment de même pour le bloc « input ».

Version 2, plus universelle

Avant d'émettre une solution nous avons effectué un travail d'identification du formulaire pour des dizaines de sites marchands. Grâce à cela nous avons pu extraire une racine commune au nom de l'attribut que nous voulions. C'est ici qu'intervient une première utilisation des expressions régulière[9] qui seront par la suite très utiles pour la recherche d'un produit. Les expressions régulières ou encore RegEx permettent de décrire plusieurs expressions à l'aide d'une seule et même syntaxe, ce qui, dans notre cas, permet de cibler tous les noms possibles d'attribut à l'aide d'une seule et même expression.

```
"(.)*?(S|s)+?(E|e)+?(A|a)+?(R|r)+?(C|c)+?(H|h)+?(.)*?";
"(.)*?(C|c)+?(H|h)+?(E|e)+?(R|r)+?(C|c)+?(H|h)+?(E|e)+?(.)*?";
```

Les expressions ci-dessus sont celles utilisées dans notre programme, il en existe deux, une anglo-saxonne et une française faisant référence au mot « chercher » car après analyse comme dit précédemment nous en sommes venus à la conclusion que l'occurrence de ce mot aussi bien en français qu'en anglais faisait dans la majorité des cas référence au formulaire de la barre de recherche. Sans rentrer dans les détails les deux expressions vont cibler tous les noms d'attribut contenant les racines « search » ou « cherche », majuscule et minuscule confondues.

Ensuite la recherche de la balise « input » se fait suivant le même procédé, néanmoins nous avons fait face au manque de référencement dans certains sites qui ne donnait pas de noms aux attributs et/ou des noms ne matchant pas avec nos expressions régulières. Pour pallier grossièrement nous avons pris par défaut la première balise « input » dans le formulaire trouvé quand aucune n'était trouvée à l'aide des expressions régulières.

Finalement nous avions obtenu à la fin de cette étape une version fonctionnelle et nettement plus universelle que la précédente, ne dépendant que de la structure des sites web. Effectivement certains n'utilisait pas la structure en « form / input », problème nous n'avons pas su trouver une solution viable et également universelle.

Pour aller plus loin ?

[9] https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re

Dans l'optique d'universalité du programme, la prochaine étape est le Deep Learning et l'utilisation d'une pseudo-intelligence qui analysera la page pour trouver les éléments voulus de manière quasi systématique. Cependant n'ayant pas les connaissances requises ni le temps pour découvrir le nécessaire pour notre programme, nous n'avons étudié en profondeur cette piste la jugeant trop risquée en termes de temps que cela nous aurait pris pour un résultat dont nous n'étions pas certains. Néanmoins dans l'optique d'une amélioration de l'application et pour des versions ultérieures la recherche dans le Deep Learning semble inévitable qualitativement parlant.

Nous allons maintenant nous intéresser à la deuxième partie qui est le « parsing » de la page web que l'on récupérer précédemment, cette partie a pour objectif d'extraire la liste d'un nombre fini, de préférence, de produits avec pour chacun son nom et son prix au minimum.

Parser, c'est quoi ?

En anglais le terme « to parse » signifie analyser, informatiquement parlant il vient de pair avec la partie « Scrap » dans la majorité des cas. Il a pour but d'extraire l'information souhaitée d'une page internet en utilisant la structure de codage de ce page qui est le HTML, un langage hiérarchisé ce qui facilite la mise en œuvre de cette opération. Nous allons voir l'évolution au fil des versions de la manière dont nous avons été chercher ces données car dans une optique d'optimisation du temps il nous fallait réduire au maximum le temps de recherche sur la page web.

Version 1 : Récupération statique de bloc de code

Cette première version nous a permis de découvrir les bases d'utilisation de la librairie Jsoup vu plus haut, en effet elle permet l'extraction d'objet *Element* qui possède plusieurs attributs ainsi que des méthodes de mise en exergue du texte présent dans cet objet, autrement du texte que l'on voit sur la page internet. Bien évidemment ces *Element* peuvent être recherché en fonction de la valeur des attributs ou encore des attributs eux-mêmes. Dans cette version nous avons pris le site de Carrefour et nous nous sommes rendu compte que l'Element contenant le prix du produit comportait comme nom de l'attribut « class » : « product-card__footer ». Le premier point problématique de cette version a été le bloc récupéré qui se décalait dans certaines conditions que nous allons voir, de plus cette version ne récupérait que le prix séparément du nom du produit.

Sur l'affichage ci-dessus on voit le bloc récupéré, on voit la balise entourée en vert qui n'est pas dans un format de balise ouvrante / balise fermante mais simplement une simple balise. De ce fait le compteur de balise dans le bloc que l'on veut récupérer n'est pas décrémenté. Ce qui a pour effet de décaler les balises prises en compte dans le bloc de 1 dans notre cas.

Version 2 : changement de la méthode d'extraction

Dans cette version nous avons effectué la recherche des blocs contenant le nom du produit ainsi que son prix découpé en plusieurs parties :

1. Récupération de la liste des matchs de nom de class pour le nom du produit et son prix.
2. Récupération des valeurs de class du HTML et vérification de leur appartenance à la liste. Si elle est dans la liste alors elle est ajoutée à une deuxième liste. Chaque valeur comporte un compteur puisque plusieurs valeurs de class peuvent être trouvé et l'on choisit celle qui a la plus grande occurrence.
3. On configure les expressions régulières en fonction des valeurs pour le prix et pour le nom du produit que l'on a trouvé précédemment. Le regex a pour objectif de choisir un conteneur quelconque ayant pour valeur de l'attribut class sans avoir de sensibilité de case mais n'ayant pas d'enfant avec les mêmes caractéristiques. Ceci dans le but de prendre le dernier élément et non pas un bloc trop important contenant des informations inutiles.

4. On effectue un select qui est une méthode de la librairie Jsoup qui permet de sélectionner des Element en effectuant une requête CSS sur le HTML suivant les expressions régulières précédemment configurées.

Le principal problème de cette version est de trouver un point commun entre toutes les valeurs de l'attribut class pour tous les sites web. En effet, certains sites web utilisent une typologie anglo-saxonne, d'autres française, ou encore des abréviations… Ainsi les possibilités sont immenses et il est difficile de satisfaire tous les sites en même temps sans modifier le code entre chaque recherche.

Version 3 : Changement de la méthode d'extraction dans un but d'universalité

Dans cette version le système de recherche a été repensé dans sa globalité en utilisant le principe d'un arbre et remontant depuis les extrémités vers le centre. Exemple ci-dessous avec un code HTML :

Dans l'exemple ci-dessus on repère l'élément contenant le nom du produit, ceci grâce à la comparaison avec une expression régulière que l'on va mettre dans un select. Cet élément (en orange) est la feuille de l'arbre, pour aller vers le tronc (encadré en vert foncé) on prend d'abord la petite branche qui correspond à l'élément parent le plus proche hiérarchiquement (ici en jaune) puis on fait de même jusqu'à arriver à l'endroit voulu. La condition d'arrêt se situe dans le fait qu'à chaque remontée dans la hiérarchie on va vérifier s'il y a la présence d'un prix (toujours à l'aide des expressions régulières), si c'est le cas alors on sait que dans cet élément on aura le prix ainsi que le nom du produit puisque que ce dernier est contenu dans un élément hiérarchiquement inférieur à celui contenant le prix. De ce fait on garde cet élément duquel on pourra extraire le texte et qui correspondra à un produit. Dans notre cas l'arrêt se situe dans l'encadré vert clair puisque le prix encadré en rouge est situé dans un de ses descendants (autre que celui d'où l'on vient).

On a également ajouté une condition qui évite les blocs de texte de description et autres textes de trop grande taille puisqu'une taille maximale de caractère est définie.

Dans cette version la difficulté principale a été de trouver les expressions régulières correspondant à ce que l'on voulait. De plus dans la classe deux utilisations des regex dans deux langages différents (CSS et Java) sont utilisés ce qui mène parfois à la confusion. C'est pourquoi l'utilisation de testeur de regex en ligne a été très utile permettant de tester une expression sans se soucier du fonctionnement du reste du programme.

Version 4.1 : Changement de la méthode d'extraction dans un but d'optimisation

Grâce à cette version la sélection des éléments contenant le prix ne se fait qu'en une étape utilisant la méthode *select* sur un objet *Document* grâce à la librairie Jsoup, cette méthode prend en paramètre une chaîne de caractère qui correspond à la requête en *CSS* qui sera exécutée sur le *Document*. Le principe de la requête qui est faites en utilisant les expressions régulières est d'isoler un élément qui contient le mot de la recherche ainsi que le signe "€" mais qui ne contient pas un enfant avec les mêmes caractéristiques. Cette dernière partie est très important car elle permet d'isoler le dernier élément, hiérarchiquement parlant, contenant les informations concernant un produit, et non pas cet élément ainsi que tous ses parents.

Le point d'amélioration de cette version quasi universelle est l'optimisation, effectivement cette requête via la méthode *select* est très coûteuse en mémoire et en temps.

```
Petits pains | 3.60 € / Pièce
Pain campagnard emmental/noix CARREFOUR | 7.00 € / Kilogramme
Pain Campagnard FILIERE QUALITE CARREFOUR | 3.50 € / Kilogramme
Pain du Maghreb | 4.00 € / Kilogramme
Pain de campagne CARREFOUR | 3.50 € / Kilogramme
Pain complet CARREFOUR BIO | 5.33 € / Kilogramme
Pain boule CARREFOUR | 2.50 € / Kilogramme
Pain campagnard s/gluten | 10.56 € / Kilogramme
Pain boule tranchée s/gluten | 11.25 € / Kilogramme

======================
Temps de calcul (en millisecondes) : 39233.0
Nombre de résultats trouvés : 38
```

Comme le montre cette capture d'écran le temps nécessaire pour une recherche est d'environ 40 secondes. Nous avons voulu tester avec la meilleure connexion qu'il soit et nous gagnons environ 5 secondes comme le montre la capture ci-dessous :

```
Pain campagnard s/gluten | 10.56 € / Kilogramme
Pain boule tranchée s/gluten | 11.25 € / Kilogramme

======================
Temps de calcul (en millisecondes) : 35716.0
Nombre de résultats trouvés : 38
```

En isolant le temps de calcul de la méthode en question on remarque qu'en effet, elle prend le plus grand temps de calcul :

```
======================
Temps de calcul (en millisecondes) : 29386.0
======================
```

Pour prouver cette hypothèse j'ai effectué cinq simulations consécutivement, toutes identiques, et j'en ai extrait pour chacune le temps de calcul de la méthode *select*, le temps de calcul de la fonction globale (*Scrap*) ainsi que le pourcentage du temps de calcul de la méthode *select* par rapport au temps global. On obtient la moyenne pour chacune des informations suivantes ci-dessous :

```
Temps moyen de calcul du select : 30921.4
Temps moyen de calcul du scrap : 35306.0
Pourcentage moyen du calcul du select par rapport au scrap : 89.04089
```

Il faut également noter que cette méthode ne permet pas d'identifier un nombre d'élément défini au préalable (pour éviter la recherche de trop d'élément par exemple) mais sélectionne tous les éléments correspondant à la recherche via la méthode *select*.

Version 4.2 : Amélioration des sélecteurs pour l'optimisation

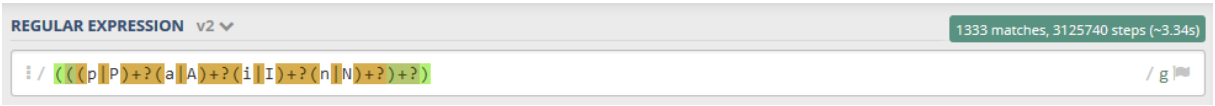
La structure de recherche de cette version reste la même que celle de la version précédente mais pour résoudre les différents points d'amélioration nous avons modifié les expressions régulières utilisées pour les sélecteurs. En effet dans les expressions régulières le sélecteur universel a été minimisé au maximum dans son utilisation. Effectivement son utilisation rallonge énormément le temps de calcul, ce qui paraît logique puisque lors du test de l'expression avec l'expression régulière demandée le nombre de possibilités est augmenté de manière considérable.

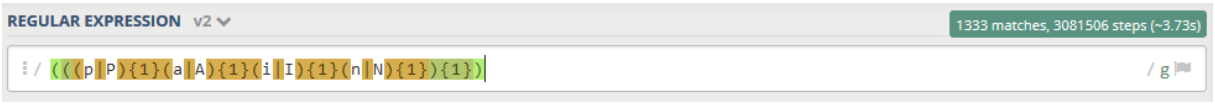
Par exemple prenons l'une des dernières versions d'Unicode qui à l'heure actuelle contient près de 110000 caractères[10] de ce fait lorsque l'on met le caractère universel par lequel le programme comprend qu'un caractère parmi les 110000 peut prendre place, le nombre de possibilités augmentent du même nombre. Un autre paramètre rentre en compte dans les expressions régulières, le quantificateur il permet dans une expression de spécifier le nombre d'occurrence pour un groupement ou un caractère. Il en existe plusieurs :

- + : présent 1 fois ou plus;
- * : présent 0 fois ou plus;
- {n} : présent n fois;
- etc.

[10] http://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html

En utilisant des quantificateurs adéquats comme dans l'exemple ci-dessous on peut empêcher que le programme exécute des étapes supplémentaires et donc de gagner du temps. Ainsi on constate qu'en utilisant le quantificateur "{1}" au lieu de "+" près de 50000 étapes ont été occultées. Sachant que les recherches ont été effectuées dans un code HTML correspondant au code d'une page à laquelle le programme final pourra faire face.





De plus comme l'utilisation des expressions régulières faisait partie intégrante de sélectionneur CSS complexe il a fallu revoir la recherche de ces derniers. Effectivement au lieu de rechercher dans tout le fichier HTML, nous avons au préalable sélectionné l'intérieur du tag "body" du code de la page récupérée. Ce choix n'a aucune incidence sur les résultats trouvés puisque le cœur de la page web, autrement dit toutes les informations que l'utilisateur va visualiser sur son moniteur sont contenues dans cette partie. Ainsi comme le sélecteur CSS vérifie pour chaque tag matchant ses blocs enfants, cela évite de prendre en compte les plus gros blocs du code HTML.

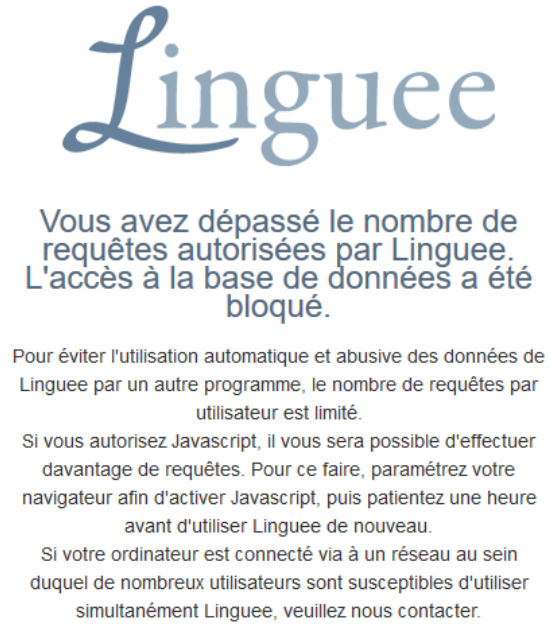
```
<html>
    <head>
        Info about page goes here…
    </head>
    <body>
        Web page goes here…
    </body>
</html>
```

Problèmes rencontrés

Lors de plusieurs, alors que la connexion et la simulation de navigateur ne posait plus de problème, l'erreur 503 issue du protocole HTTP est apparue, elle signifie que le service web distant n'est pas disponible. Après plusieurs tentatives infructueuses avec le programme, nous nous sommes rendus manuellement sur le site défectueux Linguee [11]pour vérifier que le site n'était en maintenance ou autre. En effectuant notre requête, ici une demande de traduction d'un mot, le site a affiché le message suivant (photo ci-dessous). Ce message pose un nouveau problème du nombre de requête que l'on va effectuer mais après réflexion, l'ordinateur duquel les requête étaient effectuées utilisait la connexion partagée de L'ESIEE, il se peut donc que d'autre requêtes aient été effectuées par d'autres élèves, surchargeant ainsi la demande. C'est pour cela que nous avons essayé avec une connexion personnelle et le message n'est pas réapparu depuis, ce problème n'est donc pas primordial pour le moment sauf si celui-ci

[11] https://www.linguee.fr/

devient récurrent et/ou ne suffit pas aux demandes que nous pourrons effectuer dans des versions finales.



Il s'est avéré par la suite que les requêtes vers ce site prenaient beaucoup de temps, nous avons donc décidé de mettre entre parenthèse la traduction du mot recherché.

Pour aller plus loin ?

Comme pour la partie précédente le principal point d'amélioration est l'optimisation du temps de recherche et la pertinence des résultats trouvés ; or grâce aux expressions régulières nous avons poussé au maximum l'utilisation de la librairie Jsoup et l'étape suivante constitue la recherche intelligente sur la page. Effectivement, être capable de développer une intelligence artificielle capable de distinguer les produits affichés sur le site ainsi que le prix de ce produit apparaît comme étant l'évolution nécessaire à un déploiement de l'application plus important. Cela permettrait également d'améliorer la pertinence des éléments extraits puisque la structure des sites internet est très variable et quelques fois la description du produit va être extraite en plus du nom, alors que nous ne la voulions pas.

Pour conclure sur la partie centrée web, nous pouvons dire que nous avons poussé au maximum la programmation classique pour la recherche dans les sites d'e-commerce et qu'avec le développement de l'application une programmation intelligente et apprenante serait primordiale pour obtenir des résultats encore meilleurs.

VII. Algorithme de recherche

L'application se différencie des autres du même genre notamment par son indice de fiabilité. Cet indice a pour but de renseigner l'utilisateur des chances qu'il a de trouvé le produit qu'il recherche autour de lui. Au même titre que la base de données ou l'affichage sur la carte, l'algorithme de ressemblance constitue une partie de l'application indépendante mais nécessaire.

Dès le début de la réalisation de ce projet, nous nous sommes mis d'accord qu'il convenait mieux de procéder par étapes, d'aller d'une application simpliste jusqu'au produit final que nous désirions. C'est dans cet esprit que l'algorithme a été élaboré.
Le principe de base de l'algorithme est de comparer deux chaines de caractères et de retourner le degré de ressemblance entre ces deux chaines de caractères. La première chaine de caractères est la recherche de l'utilisateur, la seconde est le nom du produit trouvé.

La première version de cet algorithme consistait en une approche naïve du problème. Si les deux chaines de caractères sont identiques, l'indice de fiabilité est élevé. Autrement, cet indice est faible.
Cette approche ne convient pas aux recherches de plus d'un mot, il suffit alors qu'au moins un des mots composant l'une des chaines de caractères soit différent d'un de l'autre chaine pour que nous obtenions un indice de fiabilité très mauvais. Pour pallier ce problème, il a fallu différencier les cas où l'expression recherchée contenait un ou plusieurs mots.
Voici l'algorithme en pseudo-code :

1. Algorithme 1 : Fiabilité

Données : listeM, Produit ; avec listeM un tableau 2D contenant les noms des produits par magasin et Produit la recherche de l'utilisateur
Résultat : listeF

```
P = Produit.split ;
si P.length ≤ 1alors
      listeF = fiable1(listeM, P)
sinon listeF = fiable(listeM, P)

Def fiable

n = listeM.length, m = listeM[0].length, Résultat

pour i allant de  0 à n-1 faire
      indice = 0
      liste = listeM[i][m-1]
      split = liste.length
      j = 0
      tant que j < split faire
            si P[j] = liste[j] alors
                  indice++, j++
      si indice = 0 alors fiabilité[i][m-1] = « mauvais »
      si indice = 1 alors fiabilité[i][m-1] = « bon »
      si indice > 1 alors fiabilité[i][m-1] = « excellent »
```

Le principe de la fonction fiable1 qui est appelée lorsque la recherche n'est composée que d'un seul mot, est similaire à celui de la fonction fiable, excepté que l'indice ne peut être que « mauvais » si aucun mot du nom du produit trouvé en magasin ne correspond pas au mot recherché par l'utilisateur. Autrement, si au moins un mot est identique au mot recherché par l'utilisateur, la fiabilité devient maximum.
Cette méthode n'est pas la plus précise car elle ne prend pas en compte la présence de caractères spéciaux au milieu du nom du produit, ou encore la présence d'un ou plusieurs mots mal orthographiés.

Nous nous sommes rendu compte que l'algorithme devait finalement répondre à une problématique liée à la distance entre deux chaînes de caractères. Il était alors préférable de se référer à des algorithmes de calcul de distance déjà existant. La seconde version de l'algorithme s'est alors reposée sur la distance de Levenshtein.

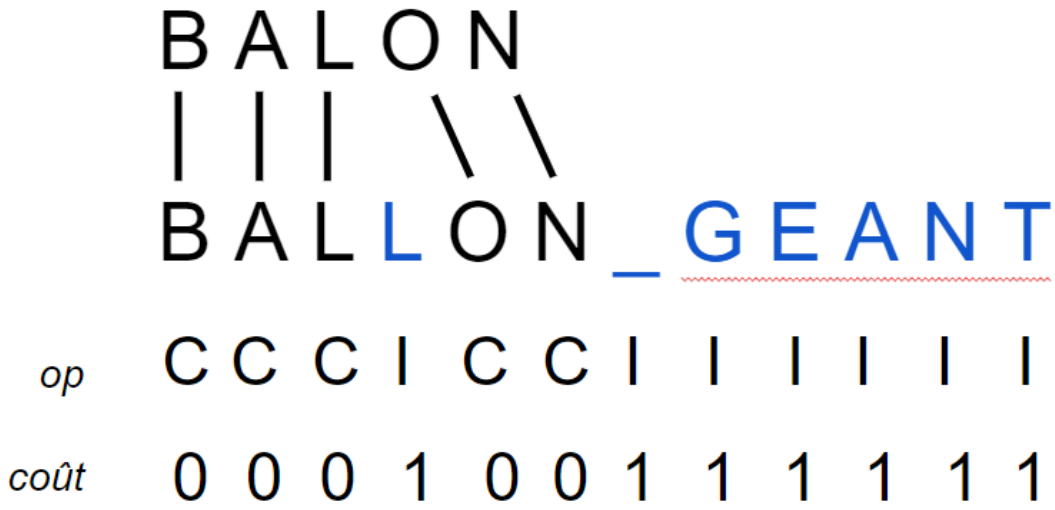
La distance de Levenshtein est une distance, au sens mathématique du terme. Cette distance traduit la différence entre deux chaînes de caractères et calcule le coût minimum selon le nombre de caractères qu'il faut supprimer, remplacer ou insérer pour passer de la première chaîne à la seconde. Plus les deux chaînes de caractères sont différentes, plus cette distance sera élevée. En général, le coût entre les opérations de suppression, insertion ou substitution est de 1. Dans notre cas, nous n'avons pas souhaité modifier ces coûts et avons conservé les valeurs par défaut.

Mathématiquement, la distance de Levenshtein s'exprime de la façon suivante :

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

En informatique, travailler avec des fonctions récursives peut se révéler inefficace et une perte de temps dans certains cas. Appliquer strictement la définition de la distance de Levenshtein nous ferait perdre énormément de temps ici, car notre principale préoccupation est la rapidité et la satisfaction de l'utilisateur. Or, la définition recalcule plusieurs fois la distance de la même sous-chaîne plusieurs fois, ce qui est une perte de temps. Nous avons donc retravaillé l'algorithme afin de ne pas utiliser la récursivité.

Concrètement, si l'utilisateur souhaite acheter un ballon mais que celui-ci fait une faute dans l'orthographe, l'algorithme compare la requête de l'utilisateur avec le nom d'un des produits correspondants trouvés.



Chaque lettre va être comparée à une autre lettre au même rang qu'elle, et il va être déterminé s'il faut exécuter une copie de la lettre, une insertion ou une substitution afin d'arriver à l'expression finale.

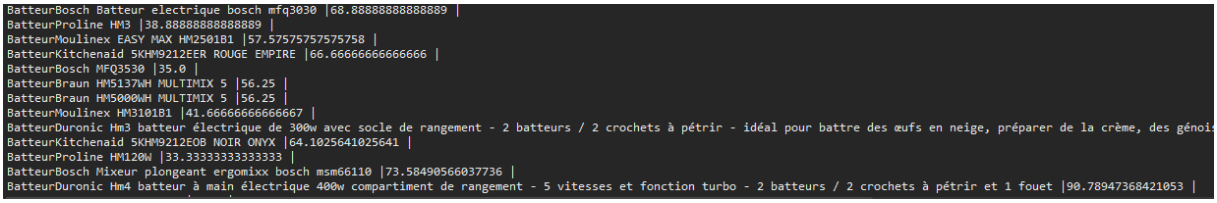
2. Algorithme 2 : Distance de Levenshtein

Données : listeM, Produit
Résultat : Pourcentage ; qui est le degré de ressemblance en pourcentage entre le mot cherché et le nom des produits trouvés

```
Long = Produit, court = listeM

si Produit.lengh < listeM.length alors
      long = listeM
      court = Produit.length
longDistance = long.length
si longDistance =  0 alors
      retourner 1

retourner Pourcentage = longDistance - Distance(long, court) / longDistance

Def Distance

Résultat : Coût

Coût, n = long.length

Si n < 21 alors
      Pour i allant de 0 à n
            Dernier = i
            Pour j allant de 0 à court.length
                  Si i = 0 alors
                        Coût[j] = j
                  Sinon
                        Si j > 0 alors
                              Nouveau = Coût[j-1]
```

```
                              Si long[i-1] ≠ court[j-1] alors
                                    Nouveau = min(Nouveau, Dernier, Coût[j])
+ 1
                              Coût[j-1] = Dernier
                              Dernier = Nouveau
            Si i > 0 alors Coût[court.length – 1 ] = Dernier
Sinon
      Pour i allant de 0 à 20
            Dernier = i
            Pour j allant de 0 à court.length
                  Si i = 0 alors Coût[j] = j
                  Sinon
                        Si j > 0 alors
                              Nouveau = Coût[j-1]
                              Si long[i-1] ≠ court[j-1] alors
                                    Nouveau = min(Nouveau, Dernier, Coût[j])
+ 1
                              Coût[j-1] = Dernier
                              Dernier = Nouveau
            Si i > 0 alors Coût[court.length – 1 ] = Dernier
```

L'avantage de cette nouvelle approche du problème est la meilleure prise en charge des fautes d'orthographe qui peuvent se trouver dans l'une ou l'autre chaîne de caractères.
Au niveau du temps d'exécution, nous n'observons pas de grande différence. Cet algorithme est plus efficace quant aux résultats, car ceux-ci sont sous la forme de pourcentages.



Sur l'image ci-dessus, les noms des produits ont été récupérés par le parser web et comparés avec la recherche de l'utilisateur : « bateur ». Etant donné que l'algorithme compare caractère par caractère afin de calculer la distance, les erreurs d'orthographe ne posent plus problème. Le coût est minime pour ajouter la lettre manquante pour passer de « bateur » à « batteur », le pourcentage calculé reste raisonnable.

Lors de la conception de l'application, nous avons limité la recherche de l'utilisateur à 20 caractères. Afin d'harmoniser les pourcentages calculés, les noms de produits qui excèdent 20 caractères, espaces et tirets compris, sont tronqués après le 20e caractère. Ceci permet de ne jamais comparer une recherche qui ne peut pas dépasser 20 caractères avec un nom de produit qui en fera 100. Il s'agit là d'un gain de temps non négligeable.

Un nouveau problème se pose avec l'utilisation de la distance de Levenshtein. La distance est calculée de la même manière, entre deux lettres entre elles, une lettre et un tiret, et une lettre et un espace. Le coût augmente, par conséquent le pourcentage diminue, car l'algorithme ne peut pas faire la différence entre une lettre et un caractère spécial.
L'algorithme de Smith-Waterman est un algorithme utilisé en bioinformatique pour donner le meilleur score possible de correspondance entre deux séquences. Cet algorithme optimal se base sur l'utilisation de plusieurs matrices de substitution pour déterminer la distance la plus courte entre deux séquences.

Outre le fait d'être plus rapide car il est en O(mn) contre O((m+1)(n+1)) pour Levenshtein, il est plus aussi plus optimal que l'algorithme de Levenshtein. L'algorithme de Smith-Waterman prend en compte la présence des tirets et des espaces dans les expressions.
Mathématiquement, l'algorithme de Smith-Waterman se traduit comme suit :

$$M(i,j) = \max \begin{cases} 0 \\ M(i-1,j-1) + D(A_i, B_j) \\ M(i-1,j) + \Delta \\ M(i,j-1) + \Delta \end{cases}$$

Avec A, B les deux séquences comparées, Δ le coût associé à la présence d'un espace, D(A,B) le score d'alignement entre les deux séquences. L'algorithme va remplir une matrice par récurrence et rechercher les maximums locaux des alignements locaux. Une fois le calcul terminé, il va être cherché le coefficient le plus élevé présent dans la matrice. Ce coefficient correspond alors à la distance optimale d'une séquence à l'autre.

Faute de temps lors de la réalisation de ce projet, cet algorithme n'a pas pu être implémenté à temps dans l'application présentée. Cet algorithme ne constitue pas une nécessité pour la recherche des produits entourant l'utilisateur. L'algorithme de la distance de Levenshtein fournit déjà un résultat tout à fait satisfaisant.

VIII. Création du poster

Dans notre poster, nous souhaitons exprimer le besoin auquel nous répondons et que les passants lors de la journée des projets puissent assez vite comprendre le projet et son utilité. Nous avons misé sur un design attrayant et peu de texte afin que tout le monde puisse balayer l'affiche des yeux en moins de 5 secondes. On le sait, le temps que nous passons à découvrir une affiche est assez court et nous souhaitons ne pas trop surcharger l'affiche.

Voici les différents prototypes que nous avons créé :

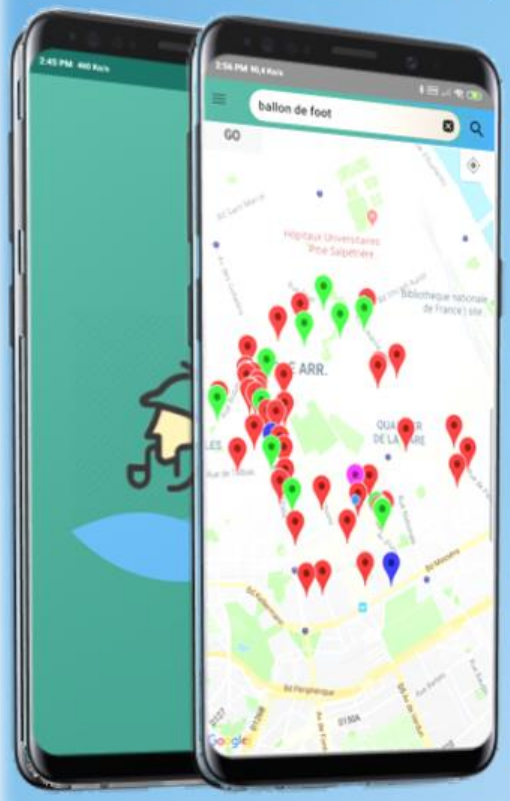
Le Jour des Projets
Révélateur de nouveaux talents

ESIEE PARIS

Sherloc

Application mobile android sous java

Vous voulez un produit autour de vous ?



Cherchez

Trouvez

Let's go !

Avec Sherloc, Just find it !

DISPONIBLE SUR Google Play

Léa AFCHAIN
Amberine KHAMASSI
Florian KOSZUL
Roger PRIOU
Valentin SOUDY

Département informatique
Projet coordonné par M. Jean LARRAT

E3

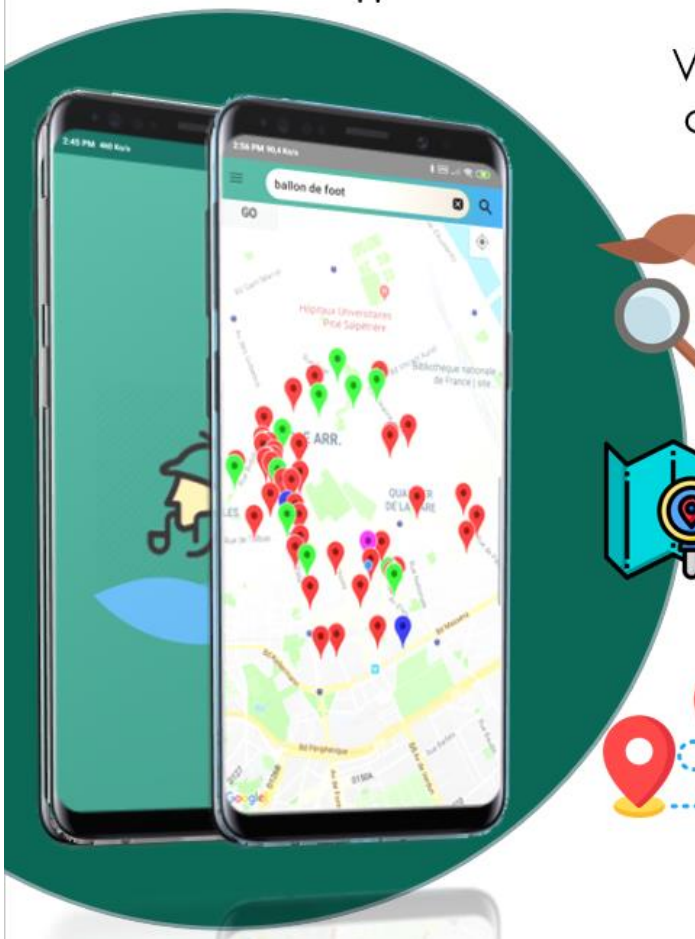
une école de la CCI PARIS ILE-DE-FRANCE

Le Jour des Projets
Révélateur de nouveaux talents

ESIEE PARIS

Sherloc

Application mobile android sous java



Vous voulez un produit autour de vous ?

Cherchez

Trouvez

Let's go !

Avec Sherloc, Just find it !

DISPONIBLE SUR Google Play

Léa AFCHAIN
Amberine KHAMASSI
Florian KOSZUL
Roger PRIOU
Valentin SOUDY

Département informatique
Projet coordonné par M. Jean LARRAT

E3

une école de la CCI PARIS ILE-DE-FRANCE

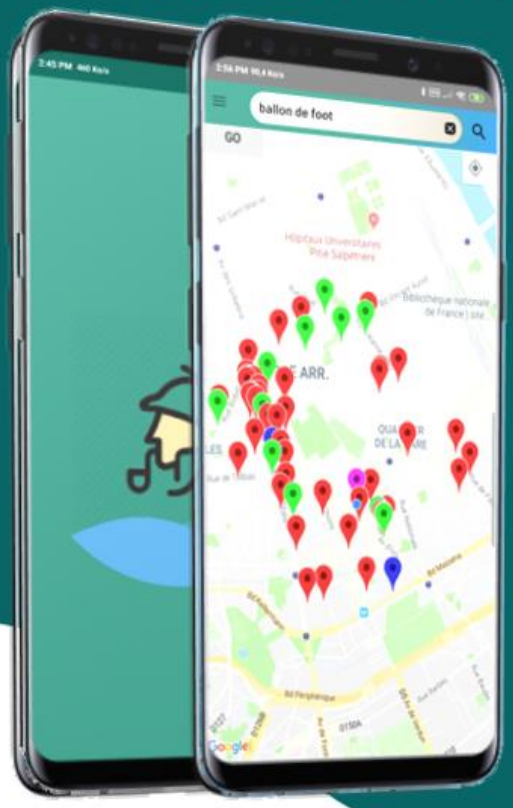
Le Jour des Projets
Révélateur de nouveaux talents

ESIEE PARIS

Sherloc

Application mobile android sous java

Vous voulez un produit autour de vous ?



Cherchez

Trouvez

Let's go !

Avec Sherloc, Just find it !

DISPONIBLE SUR Google Play

Léa AFCHAIN
Amberine KHAMASSI
Florian KOSZUL
Roger PRIOU
Valentin SOUDY

Département informatique
Projet coordonné par M. Jean LARRAT

E3

une école de la CCI PARIS ILE-DE-FRANCE

Avant d'aboutir sur la version finale que nous afficherons lors du jour des projets :

IX. Quelle est la suite de Sherloc ?

Dans le début du projet, nous avions fixé certains objectifs comme supplémentaires si nous avions du temps. Nous aimerions étendre le spectre de l'application non plus seulement à ceux qui cherchent mais aussi ceux qui possèdent des articles. Nous pourrions par exemple créer un profil professionnel dans lequel l'utilisateur pourrait entrer lui-même des données pour son magasin s'il ne possède pas de site internet et n'est donc pas encore pris en compte par notre application.

Nous aimerions également être le plus indépendant possible de Google à stockant un maximum de données sur les recherches des utilisateurs pour pouvoir les réutiliser.

Si notre application obtient un grand succès, les magasins pourront nous donner accès à certaines bases de leurs données, voudront peut-être être mieux référencer ou encore afficher des réductions sur certain produit. Certains magasins qui n'ont pas de sites web pourrons nous donner la liste des objets qu'ils ont en magasins. Chaque magasin pourrait également de lui-même alimenter la base de données pour indiquer ce qu'il possède. Il serait beaucoup plus efficace de faire tourner nos scripts sur un serveur à grosse puissance de calcul, cela permettrait d'obtenir les résultats en un fraction de seconde.

Nous pouvons élargir les recherches en ajoutant des filtres tel que le prix le moins élevé ou le magasin le plus proche. Des restaurants seront aussi ajoutés à nos recherches, il sera ainsi possible de rechercher des plats spécifiques selon les envies du jour.

Pour obtenir les meilleurs résultats nous pourrons nous aider d'un réseau neuronal qui s'entrainerais sur des sites différents pour en sortir une structure générale pour trouver les objets à rechercher sur ces sites.

Il serait également possible, si nous le souhaitons de créer, de gagner de l'argent grâce aux pubs pour pouvoir télécharger l'application et l'utiliser gratuitement, ou nous pourrions également poser une question aux utilisateurs et l'information sera revendu à des entreprises de recherche.

Créer une application en java serait un bon moyen de voir les statistiques de l'application, cela va nous permettre de voir ce que font les clients et améliorer notre service.

Sources:

Merci à toute la communauté Stack OverFlow pour les forums qui nous ont bien aidés à régler nos problèmes durant tout le projet.

Splash Screen : https://www.youtube.com/watch?v=gt1WYT0Qpfk

Tuto pour la première application : https://openclassrooms.com/fr/courses/4517166-developpez-votre-premiere-application-android

Cycle de vie d'une activité : https://openclassrooms.com/fr/courses/4517166-developpez-votre-premiere-application-android/4586901-comprenez-le-cycle-de-vie-dune-activite

Distance de Levenshtein : https://en.wikipedia.org/wiki/Levenshtein_distance

https://commons.apache.org/sandbox/commons-text/jacoco/org.apache.commons.text.similarity/LevenshteinDistance.java.html

Algorithme de Smith-Waterman : https://stackoverflow.com/questions/9453731/how-to-calculate-distance-similarity-measure-of-given-2-strings/38235657#38235657

https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

---