Serado Annonces

David Alvarez Restrepo <devdav@pm.me>

21 novembre 2019

Chapitre 1

Analyse

1.1 Définition des besoins

Deux acteurs utiliserons le système :
- Directeur de Serado
- Possesseurs de l'application mobile « Serado Annonce »

Le directeur de Serado pourra insérer des annonces, afin que ces dernières soient consultable à partir de l'application mobile. Il pourra également insérer d'autres informations (informations générales) avec des liens (par exemple. référence vers un article), qui n'ont pas de rapport avec les annonces.

Les utilisateurs de l'application mobile pourront consulter les annonces ainsi que consulter les informations générales. Les missions les plus proches apparaîtront en premier dans la liste. Ils pourront aussi trier les annonces selon leur ancienneté. Il n'y aura qu'une seule liste, les annonces temporaires et fixes étant mélangées. Sur la liste qui affichera les annonces, il n'y aura peu d'informations. Lorsque l'utilisateur clique sur une annonce, il aura une vue détaillée de cette dernière.

L'application devra aussi contenir une partie qui référencera les sites suivants :
- employee-de-maison.ch
- Pôle-Formation
- Hotellerie-Restauration

Enfin, l'application devra être disponible sur iPhone et sur Android.

1.2 Charte graphique

À part le logo qui pourra être réutilisé pour les applications mobiles, il n'y a aucune contraintes.

1.3 Analyse du système

Cette section décrit plus en détail et schématiquement ce qui a été dit dans la section précédente.

Le diagramme de cas d'utilisations permet d'avoir un aperçu des acteurs et de leurs actions au sein du système.

Il y a deux acteurs : le **directeur** et les **employés**. Le **directeur** peut déjà **insérer/modifier/supprimer**

une annonce à partir de *WordPress* (Ce qui est déjà existant avant l'analyse est entouré d'un rectangle gris). En ce qui concerne le cas d'utilisation **insérer/modifier/supprimer une info générale**, Le mieux sera de l'intégrer à *WordPress*. Si nous n'y parvenons pas, soit un autre système sera créé, soit cette fonctionnalité sera mise de côté.

L'**employé** pourra consulter les **annonces** (ainsi que les détails de ces dernières) et consulter les **informations générales** et leurs détails.

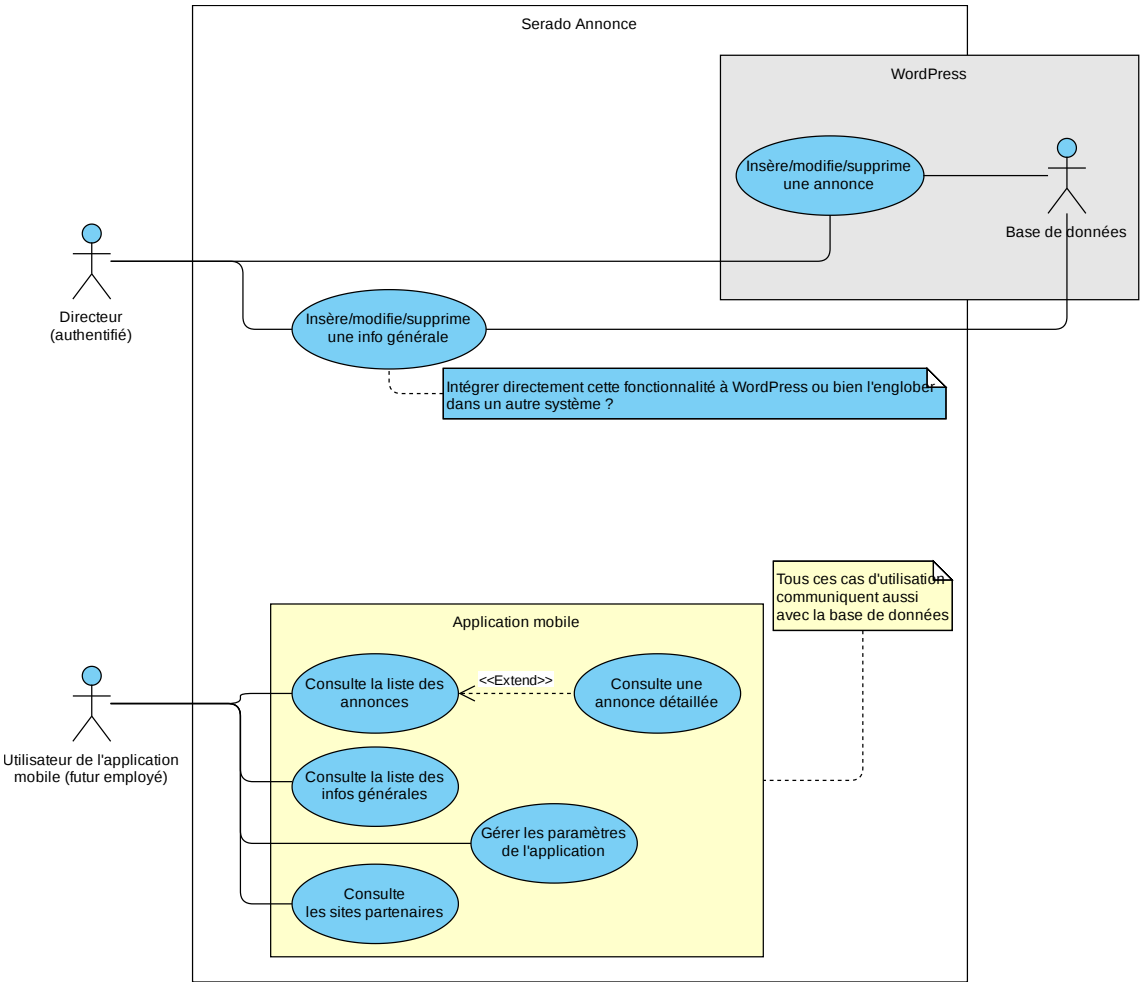

FIGURE 1.1 – Diagramme de cas d'utilisation

1.3.1 Analyse de l'existant

Le cas d'utilisation **Insérer/modifier/supprimer** existe déjà et il est effectué à partir de *WordPress*. Vu qu'il existe, inutile de créer un autre système qui effectuera la même chose. L'idéal est d'utiliser ce qui existe déjà. Il va donc falloir trouver un moyen de récupérer ces informations afin de les afficher dans l'application mobile.

Remarque : ces informations peuvent être visualisées à la page `https://serado.ch/offres-demploi/`

Il y a deux moyens pour récupérer ces informations :

1. Étudier *WordPress* afin de voir comment mettre en place une API [1] qui servent les entités

1. Application Programming Interface

offres d'emploi au format JSON.

2. Analyser le contenu de l'URL `https://serado.ch/offres-demploi/` afin de créer un objet JSON à partir du contenu de ce lien.

~~Puisque la personne qui gère *WordPress* est absente cette semaine (11.11.2019), nous partirons plutôt sur la méthode n°2.~~ C'est la méthode n°1 qui a été finalement faite.

Ensuite, puisque on utilise *WordPress* pour insérer les offres d'emploi, l'idéal serait de aussi l'utiliser pour insérer les informations générales. Autrement il faudrait créer un autre sytème entier uniquement pour cela, ce qui n'est pas rentable. Cette fonctionnalité pourra être effectuée à partir de la deuxième semaine (18.11.2019) du projet, lorsque la personne qui gère *WordPress* sera disponible.

Pour résumer, il faudrait utiliser *WordPress* pour le cas d'utilisation **Insérer/modifier/supprimer une info générale** et voir s'il est possible de créer une API qui renvoie uniquement les données demandées (et non pas toute la page HTML).

1.4 Choix des technologies

1.4.1 Langage de programmation / Framework

Le but est de livrer une application mobile qui soit compatible avec *iOS* et *Android*. On va donc utiliser un langage qui permet la programmation d'application *cross-platform*. J'en connais deux :

— *React Native*
— *Ionic*

Je choisis ***Ionic*** pour les raisons suivantes :

1. Pas besoin d'avoir une application native (60 FPS [2]), une *WebView* à 30 FPS est suffisant.
2. *Ionic* est basé sur *Angular*, que je trouve plus « propre » que *React Native*.
3. *Ionic* offre aussi une application pour *Windows Phone* (même si cela représente moins de 1% du marché).
4. Le projet *Ionic* pourra être adapté très facilement afin de créer un site web mobile, et ainsi toucher même les personnes n'ayant pas l'application installée.

En ce qui concerne l'**interface administrateur**, nous continuerons d'utiliser *WordPress*.

1.4.2 API tierces

L'application devra avoir accès au services externes ci-dessous.

Google Maps API

Afin d'afficher une *map* (carte), l'API de google sera utilisée : `https://developers.google.com/maps/documentation/javascript/tutorial`. Afin de pouvoir l'utiliser, il faudra rentrer un numéro de carte de crédit. L'utilisation de l'API est gratuite : `https://cloud.google.com/maps-platform/pricing/sheet/`.

ATTENTION : pendant la première année un crédit de 300$/mois est offert pour l'utilisation des services, ce qui devrait être suffisant pour l'entreprise, en partant du principe qu'il n'y aura pas

2. Frame Per Second

énormément d'utilisateur. Il me semble qu'après cette année, certains services sont restreints, voir : `https://cloud.google.com/free/docs/gcp-free-tier`.

Deux API sont utilisées :

Geocoding API Trouve les coordonnées GPS d'un lieu à partir de son adresse.

Maps JavaScript API Affiche la carte, ainsi que plusieurs marqueurs.

La **Geocoding API** est normalement peu utilisé, elle récupère les coordonnées des annonces, puis les stocke dans la mémoire interne. Elle sera donc appelée une fois par annonce par utilisateur. La **Maps JavaScript API** quant à elle est beaucoup utilisée. Elle est appelée à chaque fois qu'un utilisateur clique sur une annonce.

Si il s'avère que l'utilisation gratuite est dépassée à cause de cela, je propose deux solutions :

1. Payer pour l'API et regarder les frais sur un mois, et, continuer ainsi selon les frais.
2. Utiliser **Maps Embed API** (son utilisation est normalement complètement gratuite). Voir `https://developers.google.com/maps/documentation/embed/guide`

La dernière option permet seulement d'afficher une map avec un seul marqueur. On perdrait alors la fonctionnalité qui permettait à l'utilisateur de voir à la fois sa position et celle de l'annonce sur la carte.

Voir ce lien pour les autre soptimisations possibles : `https://developers.google.com/maps/optimization-guide`

WP Job Manager

Nous aurons aussi besoin de récupérer la liste des offres d'emplois. Pour cela, nous utiliserons l'API du plugin *WP Job Manager* :

Liste des jobs `https://www.serado.ch/wp-json/wp/v2/job-listings`

Détail d'un job `https://www.serado.ch/wp-json/wp/v2/job-listings/{id}`

Les liens ci-dessus ont été trouvés à l'adresse suivante : `https://github.com/Automattic/WP-Job-Manager/pull/1628`

1.4.3 Déploiement

Le framework *Ionic* permettra de créer les exécutables (`.apk` et `.app`). Il faudra ensuite les mettre dans leur store respectif *Play Store* et *App Store* afin que les futurs utilisateurs puissent la télécharger. Pour ce faire, voici la procédure :

Play Store Créer un compte *Android Developer* (25$, à vie), puis poster l'exécutable (`.apk`) et attendre que *Google* valide l'application. Procédure : `https://ionicframework.com/docs/publishing/play-store`.

App Store Créer un compte *Apple Developer Program* (100$/année), puis suivre la procédure : `https://ionicframework.com/docs/publishing/app-store`.

1.5 Permissions

L'application demandera à l'utilisateur de le localiser afin de :

1. Trier les annonces en fonction de la distance
2. Afficher l'utilisateur sur une carte. La carte sera affichée dans chaque annonce détaillée. Ceci permettra de voir la distance entre l'utilisateur et le lieu de l'emploi en un clin d'œil

Chapitre 2

Conception

Dans ce chapitre, les cas d'utilisation seront décrit plus en détail (fiches descriptives). Il comprendra aussi les maquettes UI [1].

2.1 Fiches descriptives

Les fiches descriptives sont dans le dossier `no_code/fd/`. Il y en a une par cas d'utilisation. Elles sont au format *Markdown* (à ouvrir directement dans *GitHub* ou bien dans le programme *Typora*). Chaque fiche descriptive a une référence vers un ou plusieurs écrans présents dans les maquettes.

2.2 Navigation

La navigation se fera à l'aide d'un *drawer*. Ceci permet de ne pas surcharger toutes les pages avec un *bottom navigation* par exemple.

Les pages suivantes seront accessibles à partir du *drawer* :
- Liste des annonces
- Liste des information générales
- List des partenaires

L'application affichera l'écran "Liste des annonces" par défaut.

1. User Interface

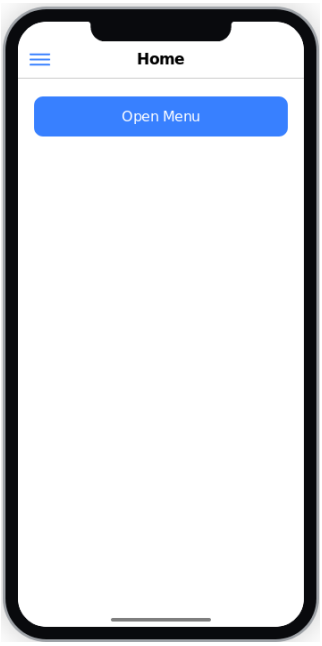

FIGURE 2.1 – Drawer fermé

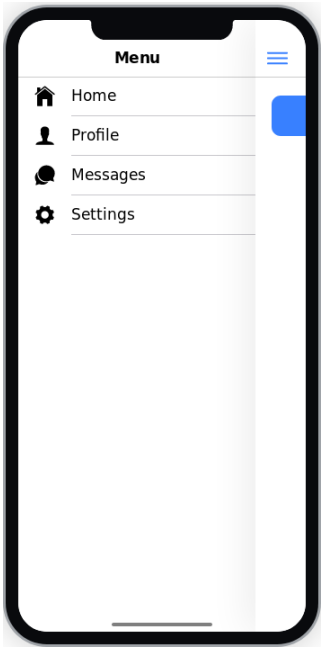

FIGURE 2.2 – Drawer ouvert

2.3 Maquettes

Les maquettes sont des fichiers au format `pdf` dans le dossier `no_code/mocks/`.

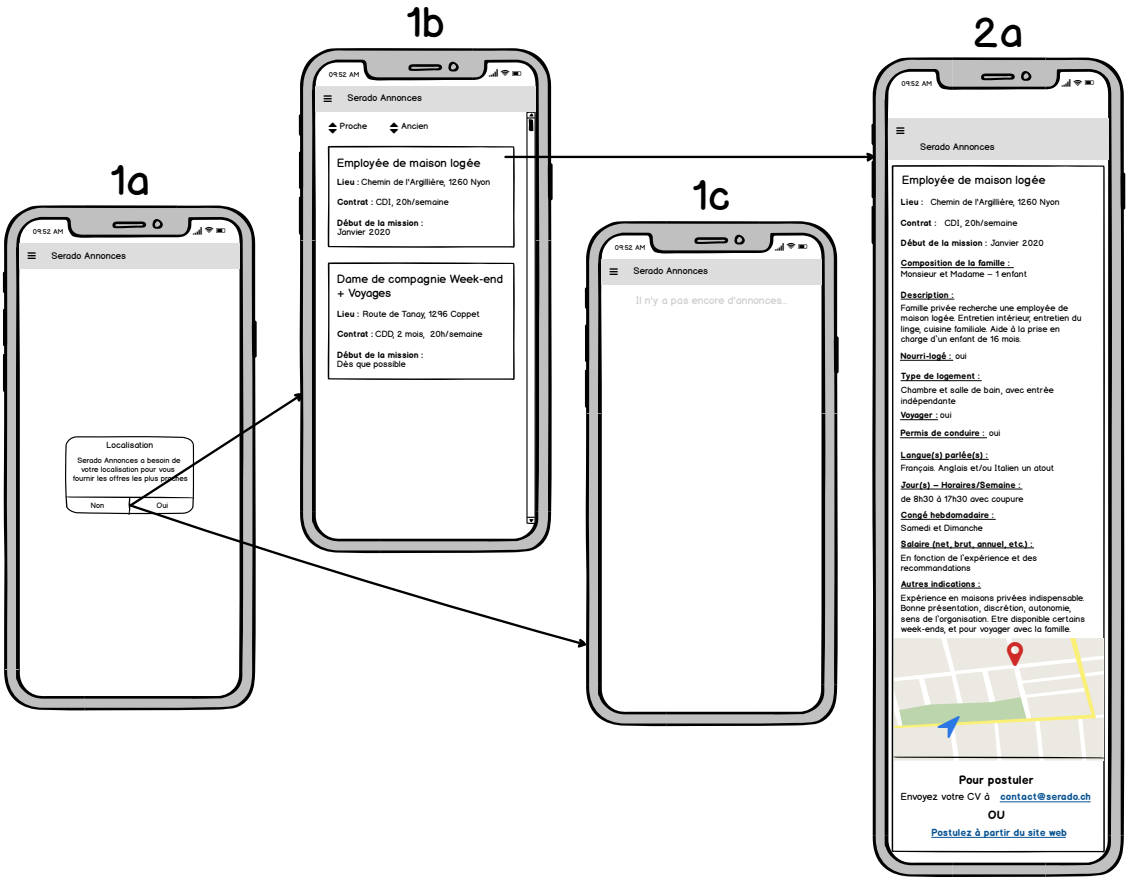

FIGURE 2.3 – Maquettes - Annonces

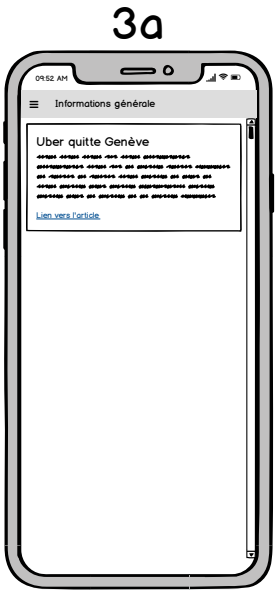

FIGURE 2.4 – Maquettes - Informations générales

FIGURE 2.5 – Maquettes - Partenaires

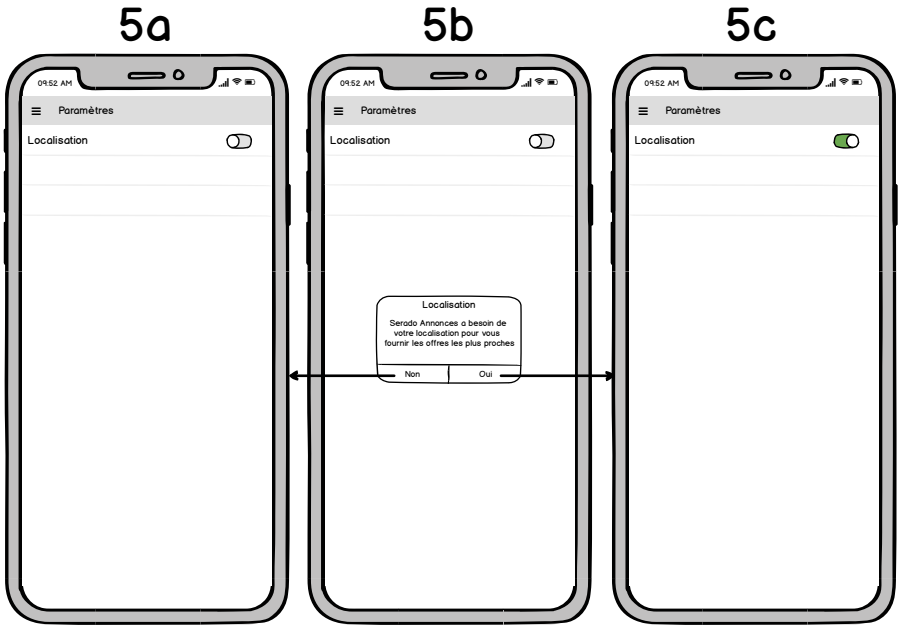

FIGURE 2.6 – Maquettes - Paramètres

Il est préférable d'ouvrir directement le fichier `pdf` afin de mieux voir chacun des écrans.

2.4 Traitement des listes

Vu que le nombre maximal d'objet dans une liste est d'environ 50 (50 annonces). Nous pouvons récupérer directement tous les éléments de la liste à partir de la base de données. Pas besoin de s'embêter avec un système de pagination.

Les listes (**annonces** & **infos générales**) seront rafraîchie dans les cas de figure suivants :
- *Pull to refresh*[2].
- Lorsque l'écran se charge pour la première fois.
- Lorsque une page s'affiche, alors que cela fait plus de 5 minutes qu'elle avait charger ses données. seulement si cela fait plus

2.5 Traitement de la position

La position de l'utilisateur sera rafraîchie à chaque fois que la liste **annonces** sera rafraîchie, si l'application a l'autorisation de localiser l'utilisateur.

2.6 État global de l'application

Nous utiliserons *@ngrx/store* (`https://ngrx.io/guide/store`) afin de gérer l'état global de l'application.

Les éléments suivants devront être stockés :
- **Liste des annonces**
- Heure du dernier chargement de la **liste des annonces**
- **Liste des informations générales**
- Heure du dernier chargement de la **liste des infos générales**
- La position GPS de l'utilisateur

Le diagramme ci-dessous décrit le comportement principal de l'application, c'est-à-dire, dans l'ordre :

1. Charger (trouver) la position actuelle de l'utilisateur.
2. Charger les annonces.
3. Trouver et ajouter les coordonnées GPS de chacune des annonces (avec Google Maps API).
4. Calculer et ajouter les distances entre la position actuelle de l'utilisateur et la position de chacune des annonces.

2. *Swipe* vers le bas lorsque l'on se trouve en haut d'une liste

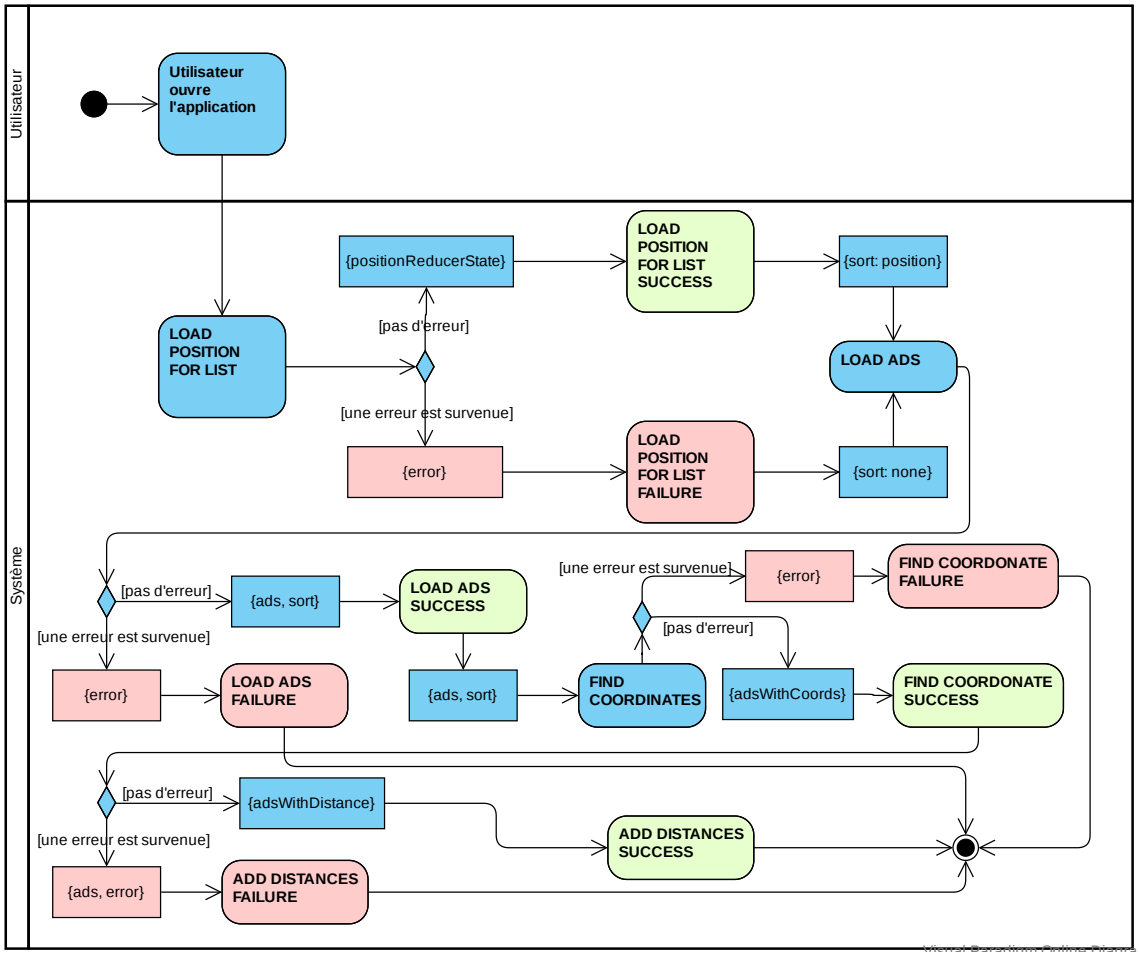

FIGURE 2.7 – Diagramme d'activité

Ces actions correspondes à des actions *NgRx* et peuvent être trouvée dans le dossier `ngx-store/actions`.

2.6.1 Actions

Afin de modifier l'état global de l'application, nous allons devoir *dispatcher* des actions. Les voici (les parenthèses à côté de l'action indique le *payload*) :

- `LOAD_ADS`
- `LOAD_ADS_SUCCESS`
- `LOAD_ADS_FAILURE`
- `SORT_ADS (sort)`
- `LOAD_INFOS`
- `LOAD_INFOS_SUCCESS`
- `LOAD_INFOS_FAILURE`
- `SET_POSITION (position)`

2.7 Paramètres de l'application

Il n'y aura qu'un seul paramètre :

— Activer/désactiver la localisation

Ceci permettra à l'utilisateur qui avec refuser l'accès à sa localisation de l'autoriser plus tard s'il le souhaite.

2.8 Requêtes HTTP

Puisque l'application sera disponible à partir d'applications mobiles et du web, il va falloir traiter les requêtes HTTP différemment :

— Utiliser la classe *HttpClient* pour les requêtes web

— Utiliser la classe *HTTP* pour les requêtes mobiles

On créera une *Factory* qui retournera le bon objet en fonction de la plateforme où nous nous trouvons.

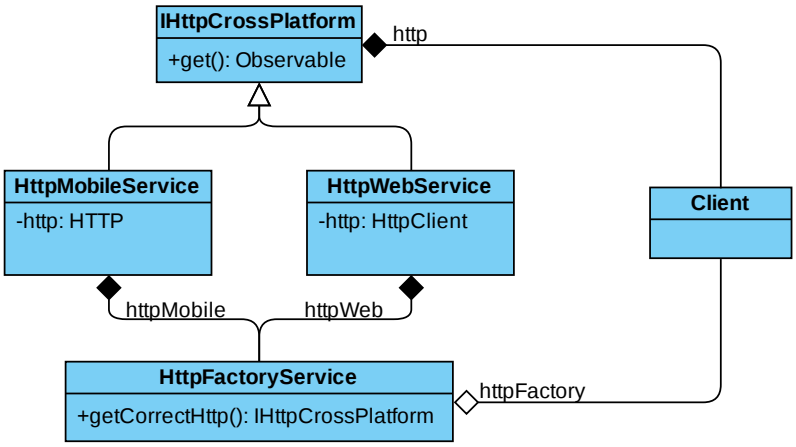

FIGURE 2.8 – Diagramme de classe de la HttpFactory

Un *HttpInterceptor* sera aussi utilisé pour simuler les requêtes HTTP, en attendant de maîtriser la génération (ou quelque chose à voir) d'API depuis *WordPress*.