

Adjji Sambe

Mehdi Nait-Sidous

David Levayer

Kai Guo



RICM4

Projet Android

Choix de conception



I. DESCRIPTION DU SUJET (OBJECTIFS)

Le but du projet est de réaliser une application Android permettant la visite guidée de la ville de Grenoble. Cette application doit, entre autre, recenser une liste de points d'intérêt et offrir à l'utilisateur des informations sur ces mêmes points.

Pour localiser l'utilisateur, l'application utilise principalement le GPS de l'appareil mobile, même si l'activation du WIFI peut par exemple permettre une localisation plus fine. Lorsqu'il affiche la carte de la ville de Grenoble, l'utilisateur peut consulter les différents points d'intérêt enregistrés pour la carte. Il voit notamment une photo et un court texte présentant le lieu. Il a également un accès direct à la page Wikipédia. Pour les personnes ayant une déficience visuelle, il est possible de lire le texte à haute voix.

Si la carte est active, une page Internet s'ouvre dès que l'utilisateur s'approche de l'un des points d'intérêt. Si la carte n'est pas au premier plan, une notification apparaît à la place, invitant l'utilisateur à remplacer la carte au premier plan pour bénéficier d'informations complémentaires.

Le principe est facilement généralisable à plusieurs villes. L'utilisateur peut également ajouter simplement des points d'intérêt personnels.



II. STRUCTURATION GENERALE DE L'APPLICATION

L'application est composée des classes/activités suivantes :

MAINACTIVITY

Cette activité est l'activité principale. Elle se contente juste de lancer MainActivity, mais reste néanmoins défini comme la première activité de l'application (celle lancé par le système).

MAINMENUACTIVITY

Cette activité représente le menu principal de l'application. Elle propose à l'utilisateur une liste de boutons permettant d'utiliser les différentes fonctionnalités implémentées. On peut notamment :

- Afficher la carte
- Accéder aux paramètres de réglage du GPS
- Configurer le service (activer ou désactiver les notifications)
- Quitter l'application

PARSER

Contient la fonction qui « parse » le fichier XML contenant les différents points d'intérêt. Les méthodes contenues dans cette classe s'appuient notamment sur la bibliothèque SAX (parseur JAVA).

TESTPARSING

Application des méthodes de la classe Parser pour notre application. Cette classe permet entre autre de récupérer les différents points d'intérêt sous forme d'objets Java exploitables (de type Poi).

MAPSERVICE

MapService est une classe implémentant BroadCast Receiver. C'est ce receiver qui affiche les notifications dans la barre haute de l'appareil mobile lorsque l'utilisateur s'approche d'un point d'intérêt.

MYITEMIZEDOVERLAY

MyItemizedOverlay redéfinit la classe ItemizedOverlay. De cette façon, il nous est possible de redéfinir les actions à effectuer lors de l'appui sur un Pol.

MYMAPACTIVITY

Cette classe hérite de MapActivity et implémente LocationListener. Elle est au cœur du projet. En effet, c'est elle qui initialise la carte, configure le GPS, charge les points d'intérêt et active les différents triggers.

MYTTS

Cette classe implémente les méthodes liées au text to speech. De cette manière, l'utilisateur peut lire les descriptions des lieux.

POINT

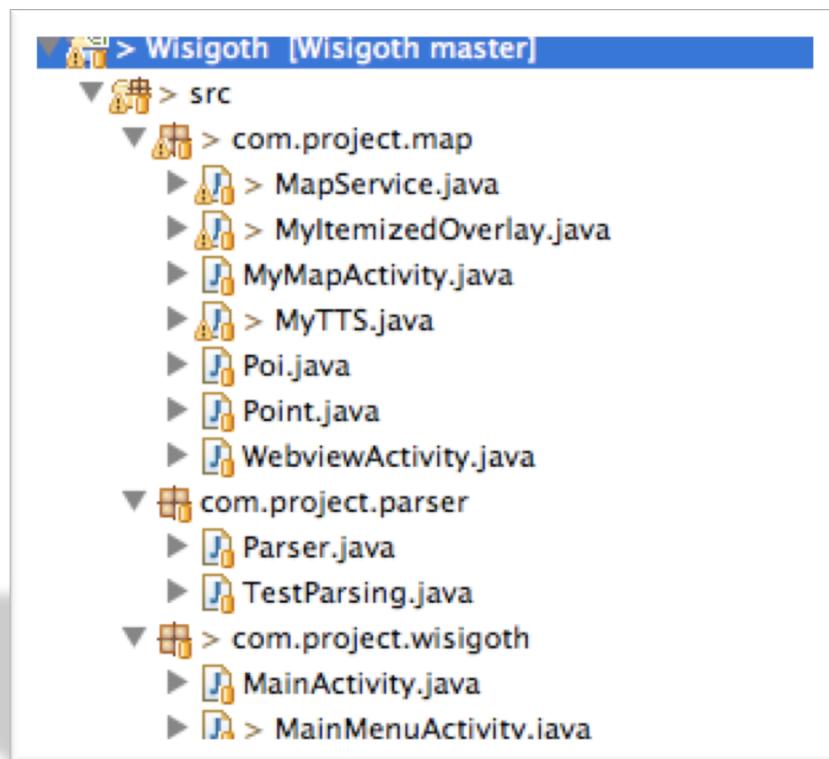
Classe basique qui définit la notion de point (pour permettre les différents calculs de position et la manipulation de coordonnées GPS. Elle hérite de OverlayItem.

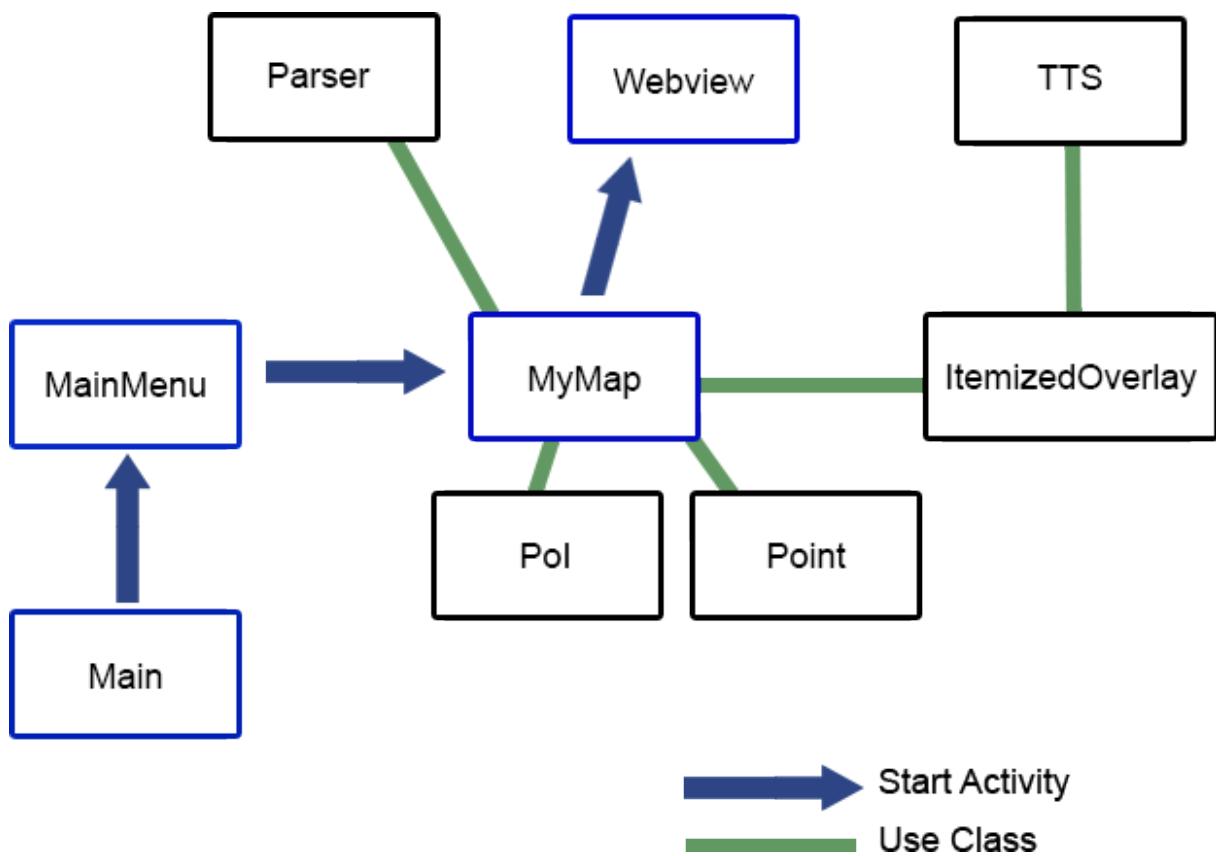
WEBVIEWACTIVITY

Cette activité contient la définition d'une Webview. C'est elle qui est appelée lorsque l'on a besoin d'afficher une webview à l'écran. Cet appel est paramétré par la nature de l'information à afficher.

POI

Cette classe hérite de Point et représente les marqueurs affichés sur la carte.





III. DETAILS D'IMPLEMENTATION

MYMAPACTIVITY

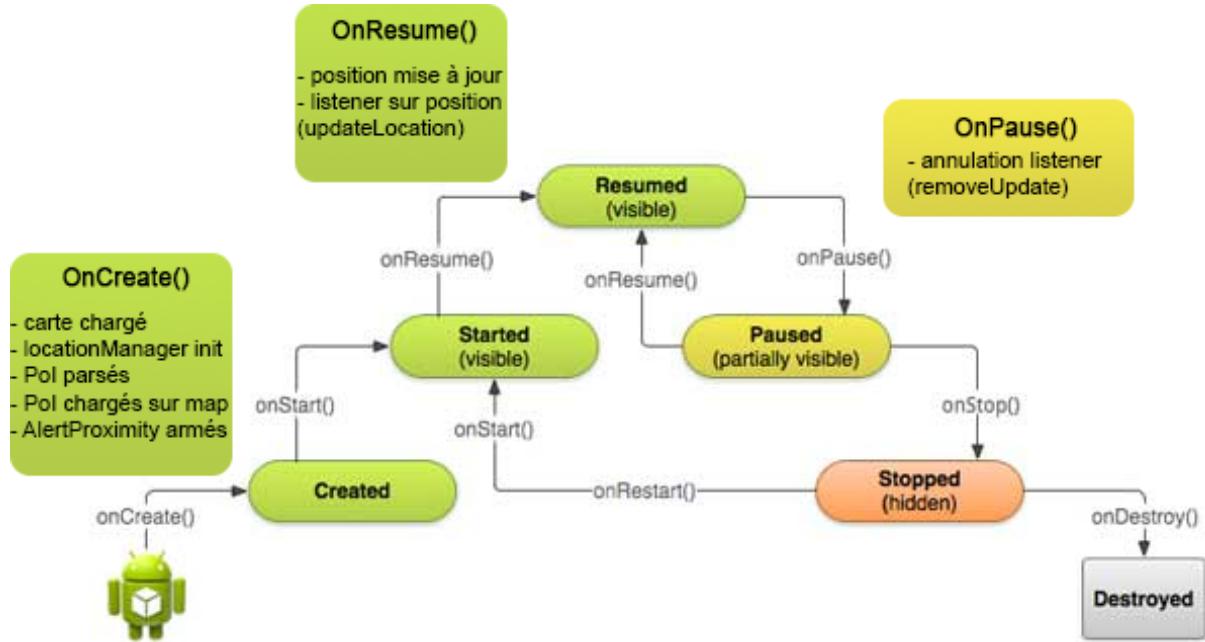
Dans la méthode `onCreate()`, beaucoup d'initialisation ont lieu. Tout d'abord, on crée la map et on parse la liste des points d'intérêt. Les propriétés de la map sont fixés (`setClickable`, `setZoom`).

On initialise ensuite le `locationManager`. C'est ce composant qui permet d'effectuer toutes les opérations liées à la géolocalisation. On utilise pour cela le service `LOCATION_SERVICE` fourni par Android. Vient ensuite toute une série de tests qui visent à contrôler l'état de l'appareil mobile. Le `locationManager` est configuré pour tirer parti du mieux possible des moyens de localisation disponibles (GPS, WIFI).

L'étape suivante consiste à rajouter les points d'intérêt sur la map. On parcourt la liste des `Pol` (point of interest) générée par le parseur. Pour chaque `Pol`, on ajoute un `Overlay` (méthode `addOverlay`). Les `Pol` sont désormais représentés sur la map par des marqueurs (ou pins).

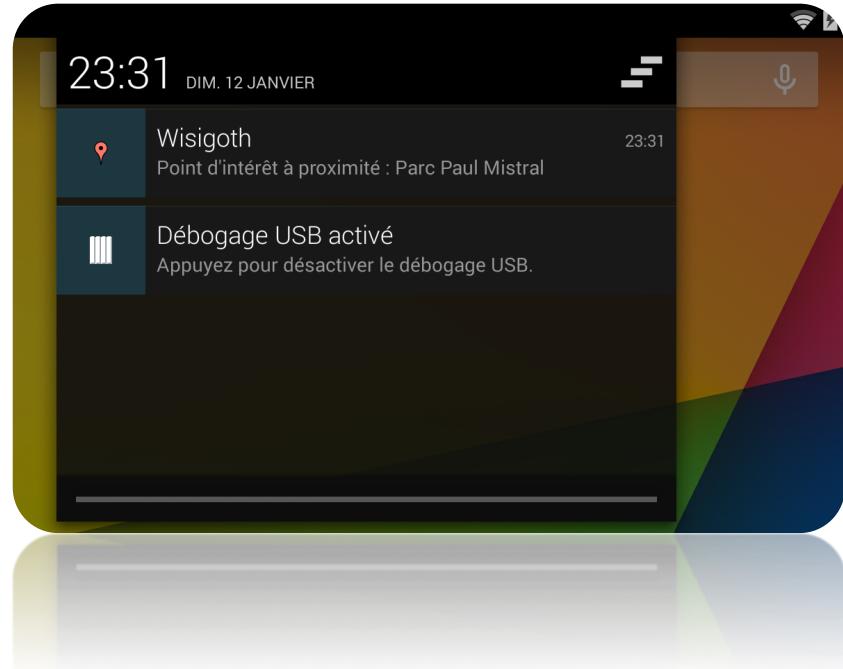
Enfin, pour chaque Pol, on ajoute également une alerte de proximité. Cette fonctionnalité est proposée par Android via la méthode addProximityAlert. Les paramètres sont les suivants :

- latitude
- longitude
- radius (distance en mètre autour du point)
- durée (-1 pour une alerte permanente)
- pending intent



Ce dernier composant est le plus intéressant. Le pendingIntent permet de définir le comportement à adopter lorsque le système va détecter que l'on approche de l'un des points d'intérêt. En effet, il est très probable que notre application ne soit plus au premier plan (elle sera dans l'état `onPause()`). Le pendingIntent permet au système d'accéder à un morceau de code prédéfini lors du déclenchement de l'événement. Ainsi, on ne sait pas quelle sera l'application active lors du déclenchement de l'alerte. En revanche, le système sera capable de donner suite à cette alerte en exécutant le code lié au pendingIntent.

Dans notre cas, le pendingIntent est lié à un broadcast receiver. Ce dernier attend que l'une des proximityAlert soit activé. Le pendingIntent appelle alors la fonction auquelle il est lié ; il s'agit de la méthode `onReceive()` de notre receiver. Cette dernière crée alors une notification personnalisée. En effet, les informations relatives à l'identité de l'alerte (et notamment le nom du point d'intérêt à proximité) sont liées au pendingIntent via un bundle. Le receiver n'a plus qu'à extraire les extras du bundle pour récupérer le nom du Pol. Il affiche ensuite la notification.



Par ailleurs, la méthode `onLocationChanged()` nous permet de lancer une webview dès lors que l'utilisateur approche d'un Pol ET que l'application est active. Pour cela, nous comparons la position de l'utilisateur avec la liste des positions des Pol. Si l'utilisateur est proche (dans le rayon de déclenchement défini par le fichier XML), alors la webview associée au Pol se lance automatiquement. Pour plus de confort, on notera que la même webview ne peut pas se lancer deux fois de suite.



Enfin, on peut noter que le comportement du `locationManager` évolue via les méthodes `onPause()` et `onResume()`. En effet, il convient de mettre à jour les informations relatives à la position de l'utilisateur à chaque fois que l'application revient au premier plan.

MYITEMIZEDOVERLAY

Chaque marqueur est représenté par un objet de type ItemizedOverlay. On implémente donc notre propre version afin de redéfinir la méthode onTap(). Si l'utilisateur appuie sur un marqueur, on affiche une alertDialog à l'écran.

Depuis cette boite de dialogue, l'utilisateur a accès aux différentes informations. Il peut décider de lancer la webview pour plus de précision, lire le texte ou revenir à la carte. Sur le plan technique, la première étape consiste à identifier le marqueur qui a été actionné. Une fois l'index récupéré, il est facile de générer une alertDialog personnalisée.

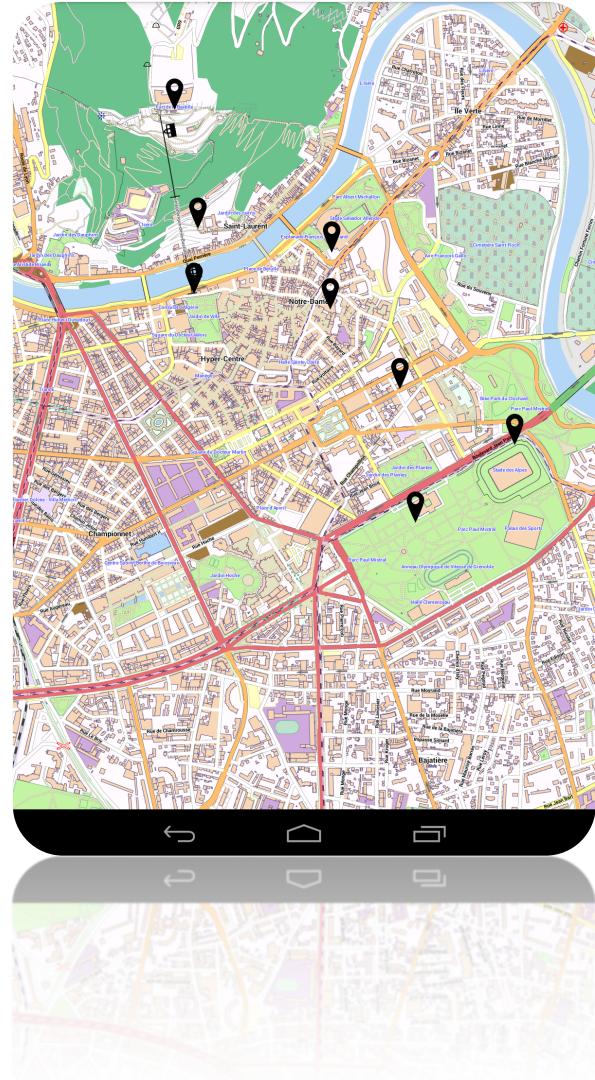
MAPSERVICE

Comme précisé précédemment, MapService implémente l'interface BroadCast Receiver. Cette interface définit le comportement basique pour recevoir des intents transmis via sendBroadcast(). C'est notre broadCast qui crée les notifications. Lorsque la méthode onReceive() est appelée, le receiver n'a pas besoin de connaître la source. Il se contente de désencapsuler les extras (c'est à dire le nom du point d'intérêt) contenus dans l'intent et de générer la notification associée.

WEBVIEWACTIVITY

La classe Webview crée et lance les webviews appropriées. On peut noter que nous avons décidé de bloquer l'affichage des webviews dans le browser. Les conséquences de ce choix sont les suivantes :

- lancement de la webview moins lourd et interne à notre application
- pas de dépendance vis-à-vis du browser
- pas de problème de compatibilité
- pas de contrôle d'erreur : ces mécanismes, qui sont intégrés dans les browsers, ne sont pas gérés dans notre application.



IV. CONCLUSION

Ce qui marche :

- l'application compile et s'exécute normalement
- on peut afficher sur une carte les différents points d'intérêt ainsi que la position de l'utilisateur.
- au lancement de l'application, un menu apparaît pour guider l'utilisateur et lui permettre d'accéder au contenu.
- afficher des informations sur le point d'intérêt sélectionné. On peut charger une page locale ou la page Wikipédia. La description succincte peut, au besoin, être lue à haute voix.
- Si on approche de l'un des points d'intérêt, la webview se déclenche automatiquement (à condition que l'application soit active).
- Si on approche de l'un des points d'intérêt, une notification apparaît pour informer l'utilisateur et l'inviter à relancer l'application.

Ce qu'il reste à faire (et idées d'amélioration) :

- permettre à l'utilisateur d'ajouter ces propres points d'intérêt. Les points sont ajoutés à partir de la position actuelle de l'utilisateur OU d'une position désignée sur la carte. L'utilisateur peut ajouter une description et prendre une photo (ou la choisir dans sa galerie) afin de la lier au PoI. Ce principe est facilement réalisable car nous l'avons anticipé en plaçant les points d'intérêt dans un fichier XML.
- Activation et désactivation rapide des notifications.
- Activation de la rotation de l'écran.
- Orientation de l'utilisateur.
- Itinéraire piéton vers le point d'intérêt le plus proche (avec la distance).
- Podomètre pour mesurer la distance parcourue.
- Rajouter un Splash screen.
- Reverse Geocoding : l'utilisateur ne devrait jamais voir ou manipuler des coordonnées géographiques (latitude-longitude).
- Ajout d'un son lorsque l'on approche d'un PoI (ou utilisation du vibrreur pour les téléphones).
- Ajouter de nouvelle ville et permettre à l'utilisateur de choisir sa ville dans le menu principal. De part la structure de l'application, cet ajout est largement réalisable. Il suffit de restart l'activité MyMapActivity avec les nouveaux paramètres.
- Permettre à l'utilisateur de suggérer un point d'intérêt. De cette manière, les développeurs obtiennent des retours et peuvent actualiser l'application.
- Permettre à l'utilisateur de publier sa position et les informations relatives au PoI sur les réseaux sociaux.