Google Maps

Clément Dumazy (FA)
09/11/2017

Résumé

Nous allons voir dans ce TP comment utiliser l'API Google Maps dans une application Android. L'API contenant un grand nombre de fonctionnalités, nous n'allons pas toutes les étudier. Bien sur, je vous conseille de regarder les liens en fin de TP pour découvrir par vous même l'étendue de celle ci.

Le but de ce TP est de vous montrer les fonctionnalités suivantes :

- Afficher une map
- Se localiser sur la map
- Créer un itinéraire entre deux lieux et l'afficher sur la map
- Utiliser différents thèmes pour la map

Pré-requis

- Savoir exploiter des données JSON
- Savoir générer une requête HTTP
- Avoir un compte Google pour récupérer les clés API
- Savoir gérer les permissions

Code source

Code source **initial** disponible à https://github.com/dumazyc/GoogleMapsApp/tree/master

Code source **final** disponible à https://github.com/dumazyc/GoogleMapsApp/tree/final

Explications du TP

Étape 0 : Utiliser Google Maps dans une WebView

Voyons tout d'abord l'intérêt d'utiliser l'API Google Maps par rapport à une WebView classique.

Dans le projet initial, vous trouverez une simple WebView qui appelle l'URL suivante :

https://www.google.fr/maps/dir/Villeneuve-d'Ascq/Moscou,+Russie

Cet URL nous trace l'itinéraire entre Villeneuve d'Ascq et Moscou. Plusieurs désavantage se font sentir. Le premier est l'impossibilité d'être localisé dans la WebView. En effet, la localisation n'est possible que sur l'application Google Maps native. Le deuxième problème est l'impossibilité de personnaliser la carte. Concrètement, utiliser l'API Google Maps va nous permettre d'enlever les limites précédentes.

Étape 1 : Créer la clé API nécessaire à l'utilisation de Google Maps

Pour utiliser l'API, il vous faudra créer une clé API. Pour cela rendez vous sur https://developers.google.com/maps/android/?hl=Fr et cliquez sur le bouton **Obtenir une clé**.

Pour des raisons de simplicité, vous pourrez durant ce tp utiliser ma clé API :

AIzaSyCoIHoW34aRpWJW6dUOKpNaI25DV3cFJCg

Étape 2 : Créer une activité Google Maps

Comme Android Studio est bien fait, vous pouvez créer une Google Maps Activity. Si vous lancez l'application une fois l'activité créée, vous allez avoir un message d'erreur. En effet, il nous faut d'abord renseigner la clé API précédemment créée dans le fichier *google_maps_api.xml* qui se trouve dans */res/values*. Si ce n'est pas fait, ajoutez cette dépendance au build.gradle de votre application : `compile 'com.google.android.gms:play-services-maps:11.0.4'`

Une fois la clé API renseignée, si vous relancez votre application, vous devriez voir apparaître une map avec un marqueur sur Sidney comme dans la capture d'écran ci dessus.

Étape 3 : Ajout de la localisation

L'ajout de la localisation se fait par l'appel de la méthode setMyLocationEnabled(boolean). Cette fonction demande la mise en place d'une demande de permission pour la localisation GPS.

Cette fonction ajoute un bouton qui permet un zoom sur l'endroit où l'appareil a été localisé. Ce bouton peut être enlevé via l'appel de la méthode UiSettings.setMyLocationButtonEnabled(boolean).

Étape 4 : Ajout de la recherche d'itinéraire et du traçage simple

Du côté du fichier layout, on va juste ajouter deux champs pour rentrer l'adresse de départ et d'arrivée ainsi qu'un bouton pour lancer la recherche.

Dans le build.gradle, nous allons rajouter deux dépendances :

```
compile 'com.android.volley:volley:1.0.0'

compile 'com.google.maps.android:android-maps-utils:0.4+'
```

Dans l'activity, on récupère d'abord les valeurs inscrites dans les champs. Ensuite, on effectue une requête HTTP à cette adresse https://maps.googleapis.com/maps/api/directions/ en lui appliquant comme paramètre les deux adresses ainsi que la clé API. Nous utilisons la classe Volley qui permet de faire des requêtes très facilement.

Vous trouverez un exemple de résultat de la requête Direction ici : https://developers.google.com/maps/documentation/directions/intro?hl=fr#DirectionsResponses

Une fois le résultat JSON récupéré, on va parcourir le fichier JSON pour trouver une *Polyline* encodée appelée *overview_polyline*. L'objet *Polyline* est ce qui permet d'afficher une ligne entre 2 points sur la carte. Les *Polyline* encodées sont des chaînes de caractères représentant un à plusieurs objets *Polyline*.

Nous allons utiliser la classe PolyUtil présente dans android-maps-utils pour décoder la *Polyline* encodée *overview_polyline* et afficher toutes les *Polyline*. Cette chaîne de caractère contient toutes les étapes pour aller d'un point A à un point B.

Relancez l'application et afficher un itinéraire. Vous pouvez constater que votre itinéraire est tracé sur la carte, mais que les différentes étapes sont tracées à vol d'oiseau et ne suivent pas la courbure des routes.

Étape 5 : Traçage de l'itinéraire complexe

Pour permettre un affichage des étapes en suivant la route, nous allons récupérer dans le résultat de la requête les *Polyline* encodées de chaque étape. Pour cela on va descendre dans le JSON par *routes* → *legs* → *steps*. Pour chaque étape (*steps*), nous allons récupérer l'attribut *polyline* → *points* et le décoder avec PolyUtil.

Étape 6 : Ajouter des marqueurs entre chaque étape

Entre chaque étape, on peut rajouter des marqueurs *Marker* pour renseigner la distance qu'il y a jusqu'à la prochaine étape (*distance*), la durée (*duration*), ainsi que les instructions GPS au format HTML (*html_instructions*). Toutes ces données sont disponibles dans le résultat de la requête que vous avez faite.

Étape 7 : Override du onClick des *Marker*

Grâce à la fonction *setOnMarkerClickListener(MarkerListener)*, nous pouvons redéfinir l'action qui sera faite lors d'un clic sur l'un des marqueurs. Dans ce tp, nous allons juste lancer une nouvelle activity qui va afficher le title (qui est le texte au format html de la direction à prendre) bien formaté ainsi que le snippet (Distance et durée) du *Marker*.

Nous aurions pu imaginer de changer le listener onclick de la bulle du *Marker*. Le principe reste le même.

Étape 8 : Zoom et déplacement automatique

Cette étape répond à la problématique comment afficher l'itinéraire de la meilleur façon qu'il soit. Pour faire cela, nous allons passé un objet *CameraUpdate* à la fonction *animateCamera(CameraUpdate)*. Ce *CameraUpdate* va être construit grâce aux extrémités Nord-Est (*routes* → *bounds* → *northeast*) et Sud-Ouest (*routes* → *bounds* → *southwest*) de notre itinéraire, extrémités qui sont fournies dans le résultat de la requête. Grâce à ces extrémités, l'objet *CameraUpdate* permet de zoomer et déplacer la caméra pour qu'on puisse voir l'intégralité de l'itinéraire. En plus des extrémités, on peut renseigner un padding (en pixel) à l'objet pour avoir une vue un peu plus grande que l'itinéraire sur l'écran.

Étape 9 : Changer de thème

La dernière fonctionnalité que nous allons voir durant le TP, c'est le changement du thème de la map.

Vous pouvez changer le type de map en appelant la méthode *setMapType(MAP_TYPE_SATELLITE)* pour avoir la vue satellite. Vous pouvez aussi pour les maps de type normal (comme on a utilisé durant tout ce tp), changer les couleurs des éléments de la map comme la couleur des routes, de l'eau en appliquant un fichier JSON à votre map.

Informations complémentaires

- https://developers.google.com/maps/android/?hl=Fr : point d'entrée à l'API Google Maps d'Android
- https://developers.google.com/maps/documentation/directions/?hl=Fr: L'API de Direction qui est cross platform.