Rapport de Développement mobile

Frédérick Fabre Ferber , M1 Informatique

21 novembre 2019

Résumé

Ce rapport va présenter l'élaboration d'une application mobile dans le cadre du cours de **Développement Mobile** (sous Android uniquement malheureusement), ici un gestionnaire de concerts. Nous verrons ici plusieurs points concernant cette présentation :

— La description générale de l'application
— Son architecture globale
— Les points à améliorer et les problèmes rencontrés

1 Description de l'application

Nous avons ici une application de gestion de concerts.
Un concert dispose de plusieurs informations :

— Son titre
— La date de l'événement
— L'heure de l'évènement
— La durée de l'évènement
— Sa position (en latitude et longitude)
— Et une photo

Elle permet notamment plusieurs interactions avec l'utilisateur :

— **Afficher des concerts sur la carte :** Lorsque l'on démarre l'application l'on accède à une carte où tous les concerts enregistrés sont marqués via des marqueurs. Si l'on clique sur le marqueur on obtient les informations (voir ci dessus). Nous avons une certaine position sur cette carte et lorsque l'on se situe à moins de 4 kilomètres d'un des concerts présent une boite de dialogue s'ouvre et nous propose se diriger vers la position du concert.
— **Rajouter un concert** : Si l'on appuie sur le bouton *"Rajouter un concert"* l'application ouvre une autre fenêtre et nous montre un formulaire où l'on va renseigner toutes les informations décrites précédemment via des champs (les champs latitude et longitude étant remplis à l'avance avec notre position). On a également un bouton **"Ajouter une photo"** qui va définir l'image du concert en prenant une photo via l'appareil photo d'android. Et un bouton **Ajouter une photo via la galerie** qui va cette fois sélectionner à partir de la galerie.
— **Liste des concerts :** Si l'on appuie sur le bouton *"Liste des concerts"* l'application ouvre une autre fenêtre et affiche via une liste tous les concerts enregistrés, chaque case de la liste va renseigner toutes les informations sur un concert. Il contient un bouton *"GO"* qui va nous diriger vers la position de celui ci et un bouton pour supprimer le concert de la liste.

Nous avons décrit globalement comment se comportait l'application nous allons maintenant voir plus en détails certaines de ces fonctionnalités.

2 Architecture globale de l'application

2.1 Représentation d'un concert

L'application va devoir manipuler des données représentant un concert, nous avons donc un objet **ConcertWindowData** fait pour ça. Il possède comme attributs les mêmes champs décrit dans la section précédente (Voir 1). Avec des attributs de type **double** pour la longitude et la latitude du concert, un objet **SerializableBitmap** pour la photo du concert (nous y reviendrons juste après) et pour tout le reste des attributs de type **String**.

Description de la classe ConcertWindowData :

```
public class ConcertWindowData implements Serializable {

    private String nom;
    private SerializableBitmap image;
    private String date;
    private String duree;
    private String heure;

    private double lat;
    private double lng;

    //.... getter et setter
```

2.1.1 Cas particulier de la classe SerializableBitmap

Nous le verrons plus tard dans ce rapport mais notre application a besoin de transférer des données entres nos différentes activités et nottament des objets **ConcertWindowData**. Tout ceci se fait via des Intent qui par défaut ne peuvent envoyer que des types simples et non des objets. On envoie alors les données avec un **Bundle** par la methode **putSerializable(Object o)** qui va pouvoir envoyer un objet **sérialisé** à une activité (on utilise l'interface **Serializable** pour faire cela). Or la classe Bitmap (qui est la classe pour représenter une image) n'est pas sérialisable ce qui empêche son envoie via un Intent. On utilise alors la classe **SerializableBitmap** qui va transformer notre image **Bitmap** en tableau d'entiers (un tableau de pixels) et va reconstituer l'image Bitmap via la methode **getBitmap()**.

```
public class SerializableBitmap implements Serializable {
    private final int[] pixels;
    private final int width, height;

    //transforme notre image Bitmap en tableau de pixels
    public SerializableBitmap(Bitmap bitmap) {
        width = bitmap.getWidth();
        height = bitmap.getHeight();
        pixels = new int[width * height];
        bitmap.getPixels(pixels, 0, width, 0, 0, width, height);
    }

    //Reconstruit l'image Bitmap
    public Bitmap getBitmap() {
        Bitmap b =  Bitmap.createBitmap(pixels, width, height, Bitmap.Config.ARGB_8888);
        return b;
}

}
```

*C'est aussi pour la même raison que l'on utilise deux variables de type **double** au lieu d'utiliser un objet **LatLng** car il n'est pas sérialisable non plus.*)

2.2 L'activité MapActivity : Afficher nos concerts sur une carte