Hannachi Thamer l3dsi EAD

Atelier : les services

1. Introduction

Ce rapport décrit le fonctionnement d'une application Android qui affiche des informations sur les planètes et utilise des services pour gérer la lecture de musique et afficher des notifications. L'application comprend deux services principaux : un service de musique (MusicService) et un service de notification (NotificationService), qui interagissent pour offrir une expérience utilisateur enrichie.

2. Architecture de l'application

L'application est structurée autour d'une activité principale (PlanetDetailActivity) qui récupère des objets représentant des planètes. Chaque planète a des attributs tels que le nom, la masse, la période de révolution, etc. Lorsque l'utilisateur sélectionne une planète, l'application lance PlanetDetailActivity, qui déclenche les services pour gérer la musique et les notifications.

3. Détails des services (MusicService)

```
public class MusicService extends Service {
    private MediaPlayer mediaPlayer;  20 usages

    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override  4 usages
    public int onStartCommand(Intent intent, int flags, int startId) {
        if (intent != null) {
            String action = intent.getAction();

            if ("PLAY_MUSIC".equals(action)) {
                int musicResId = intent.getIntExtra( name: "musicResId",
                playMusic(musicResId);
                // Show notification when music starts
                sendNotification("Lecture en cours");
            } else if ("PAUSE_MUSIC".equals(action)) {
                pauseMusic();
                sendNotification("Musique en pause");
            } else if ("RESUME_MUSIC".equals(action)) {
                resumeMusic();
                sendNotification("Lecture reprise");
            } else if ("STOP_MUSIC".equals(action)) {
                stopMusic();
                stopSelf();
                return START_NOT_STICKY; // Stops the service without re
            }
        }
        return START_STICKY;
    }

    private void playMusic(int musicResId) {  1 usage
        // Stop the previous music if necessary
        if (mediaPlayer != null) {
```

Le MusicService est un service standard qui gère la lecture de la musique en arrière-plan. Voici un aperçu des méthodes clés :

- **onStartCommand(Intent intent, int flags, int startId)**

Cette méthode est appelée chaque fois qu'un client démarre le service avec un appel à startService(Intent). Elle reçoit un Intent contenant des informations sur l'action à exécuter.

Paramètres

- Intent intent : L'intention contenant l'action demandée (ex. "PLAY_MUSIC", "PAUSE_MUSIC").
- int flags : Des indicateurs sur la façon dont le service doit être démarré.
- int startId : Un identifiant unique pour cette demande de démarrage.

Fonctionnement

- Vérification de l'intention : La méthode commence par vérifier si l'intention (intent) n'est pas nulle.
- Action déterminée : En fonction de l'action spécifiée dans l'intention, elle exécute l'une des méthodes suivantes :
  - PLAY_MUSIC : Joue la musique en appelant playMusic(musicResId).
  - PAUSE_MUSIC : Met en pause la musique en appelant pauseMusic().
  - RESUME_MUSIC : Reprend la musique en appelant resumeMusic().
  - STOP_MUSIC : Arrête la musique et termine le service en appelant stopMusic() et stopSelf().
- Retourne un statut : Renvoie START_STICKY pour indiquer que le service doit être redémarré automatiquement si le système termine le service, sauf si un autre service a été démarré explicitement.

- **playMusic(int musicResId)**

Joue un morceau de musique spécifique désigné par son ID de ressource. Cette méthode s'assure qu'aucune musique n'est jouée simultanément.

```
private void playMusic(int musicResId) {  1 usage
    // Stop the previous music if necessary
    if (mediaPlayer != null) {
        mediaPlayer.stop();
        mediaPlayer.release();
    }
    if (musicResId != 0) {
        mediaPlayer = MediaPlayer.create( context: this, musicResId);
        mediaPlayer.setLooping(true); // Loop playback
        mediaPlayer.start();
    }
}
```

Paramètres

- int musicResId : L'ID de ressource de la musique à jouer (par exemple, R.raw.venus).

Fonctionnement

- Arrêt de la musique précédente : Si une instance de MediaPlayer est déjà active, elle est arrêtée et libérée pour libérer les ressources.
  - mediaPlayer.stop()
  - mediaPlayer.release()
- Création du nouveau MediaPlayer : Un nouveau MediaPlayer est créé avec l'ID de ressource fourni et configuré pour boucler.
  - mediaPlayer = MediaPlayer.create(this, musicResId)
- Démarrage de la lecture : La musique commence à jouer avec mediaPlayer.start().

- **pauseMusic()**

```
private void pauseMusic() {  1 usage
    if (mediaPlayer != null && mediaPlayer.isPlaying()) {
        mediaPlayer.pause();
    }
}
```

Met en pause la musique en cours de lecture. Cette méthode est appelée lorsque l'utilisateur demande une pause.

Fonctionnement

- Vérification de l'état : Avant de mettre en pause, la méthode vérifie si mediaPlayer n'est pas nul et s'il est actuellement en train de jouer :
  - if (mediaPlayer != null && mediaPlayer.isPlaying())
- Mise en pause : Si les conditions sont remplies, la lecture est mise en pause avec mediaPlayer.pause(). Cela conserve l'état de la musique pour pouvoir la reprendre ultérieurement.

- **resumeMusic()**

```
private void resumeMusic() {  1 usage
    if (mediaPlayer != null && !mediaPlayer.isPlaying()) {
        mediaPlayer.start();
    }
}
```

Reprend la musique qui a été mise en pause. Cette méthode est appelée lorsque l'utilisateur souhaite reprendre la lecture.

Fonctionnement

- Vérification de l'état : Avant de reprendre la musique, la méthode vérifie que mediaPlayer n'est pas nul et qu'il n'est pas déjà en train de jouer :

- if (mediaPlayer != null && !mediaPlayer.isPlaying())

- Reprise de la lecture : Si les conditions sont remplies, la musique est reprise avec mediaPlayer.start().

- **stopMusic()**

```
private void stopMusic() {  1 usage
    if (mediaPlayer != null) {
        mediaPlayer.stop();
        mediaPlayer.release();
        mediaPlayer = null;
    }
}
```

Arrête la musique en cours et libère les ressources associées au MediaPlayer. Cette méthode est appelée lorsque l'utilisateur souhaite arrêter complètement la musique.

Fonctionnement

- Vérification de l'état : La méthode commence par vérifier si mediaPlayer n'est pas nul :
  - if (mediaPlayer != null)
- Arrêt de la musique : Si mediaPlayer est actif, il est arrêté avec mediaPlayer.stop() et ensuite libéré avec mediaPlayer.release(). La référence mediaPlayer est mise à null pour éviter les fuites de mémoire et les appels ultérieurs non désirés.

- **sendNotification(String message)**

```
private void sendNotification(String message) {  3 usages
    Intent notificationIntent = new Intent( packageContext: this, NotificationService.class);
    notificationIntent.putExtra( name: "message", message);
    startService(notificationIntent);
}
```

Envoie un message au NotificationService pour afficher une notification indiquant l'état actuel de la musique (lecture, pause, arrêt).

Paramètres

- String message : Le message à afficher dans la notification, informant l'utilisateur de l'état actuel de la musique.

Fonctionnement

- Création d'un Intent : Un nouvel Intent est créé pour démarrer NotificationService.
- Ajout du message : Le message est ajouté à l'intention pour qu'il soit utilisé par le service de notification.
- Démarrage du service de notification : Le service est démarré avec startService(notificationIntent), ce qui déclenche l'affichage d'une notification avec le message fourni.

3.2 Détails des services (NotificationService)

Ce service gère l'affichage des notifications et fournit des contrôles de musique via des actions intégrées dans la notification.

1. createNotificationChannel()

```
private void createNotificationChannel() {  1 usage
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                CHANNEL_ID,
                name: "Channel Name",
                NotificationManager.IMPORTANCE_DEFAULT
        );
        channel.setDescription("Channel Description");
        NotificationManager manager = getSystemService(NotificationManager.class);
        if (manager != null) {
            manager.createNotificationChannel(channel);
        }
    }
}
```

Cette méthode crée un canal de notification, ce qui est nécessaire pour afficher des notifications sur les appareils exécutant Android 8.0 (API niveau 26) et supérieur. Les canaux de notification permettent aux utilisateurs de gérer les paramètres de notification de manière plus granulaire.

Fonctionnement

- **Création du canal** : Si la condition est remplie, un nouvel objet NotificationChannel est créé avec les attributs suivants :
  - **CHANNEL_ID** : Identifiant unique du canal.
  - **Channel Name** : Nom visible par l'utilisateur.
  - **Importance** : Niveau d'importance du canal (NotificationManager.IMPORTANCE_DEFAULT).
- **Description du canal** : Une description est ajoutée pour informer l'utilisateur de l'usage du canal :
- **Enregistrement du canal** : Le canal est enregistré avec le NotificationManager, ce qui permet aux notifications de ce canal d'être affichées :

2. **showNotification(String message)**

```
55    public void showNotification(String message) {  1 usage
56        Intent pauseIntent = new Intent( packageContext: this, MusicService.class);
57        pauseIntent.setAction("PAUSE_MUSIC");
58        PendingIntent pausePendingIntent = PendingIntent.getService(
59                context: this,  requestCode: 0, pauseIntent,  flags: PendingIntent.FLAG_UPDATE_CURRENT | PendingInte
60        );
61
62        Intent stopIntent = new Intent( packageContext: this, MusicService.class);
63        stopIntent.setAction("STOP_MUSIC");
64        PendingIntent stopPendingIntent = PendingIntent.getService(
65                context: this,  requestCode: 0, stopIntent,  flags: PendingIntent.FLAG_UPDATE_CURRENT | PendingInten
66        );
67
68        Intent resumeIntent = new Intent( packageContext: this, MusicService.class);
69        resumeIntent.setAction("RESUME_MUSIC");
70        PendingIntent resumePendingIntent = PendingIntent.getService(
71                context: this,  requestCode: 0, resumeIntent,  flags: PendingIntent.FLAG_UPDATE_CURRENT | PendingInt
72        );
73
74        NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder( context: this, CHANNEL_
75                .setContentTitle("Solar System Music")
76                .setContentText(message)
77                .setSmallIcon(R.drawable.ic_music_note)
78                .addAction(R.drawable.ic_pause,  title: "Pause", pausePendingIntent)
79                .addAction(R.drawable.ic_stop,  title: "Stop", stopPendingIntent)
80                .addAction(R.drawable.ic_play,  title: "Resume", resumePendingIntent)
81                .setPriority(NotificationCompat.PRIORITY_DEFAULT)
82                .setOngoing(true);
83
84        startForeground( id: 1, notificationBuilder.build());
85    }
```

Cette méthode crée et affiche une notification avec des actions intégrées pour contrôler la musique, permettant aux utilisateurs de mettre en pause, reprendre ou arrêter la musique directement depuis la notification.

Paramètres

- **String message** : Message à afficher dans la notification, qui informe l'utilisateur de l'état de la musique (par exemple, "Lecture en cours").

Fonctionnement

- **Intent pour les actions** : Pour chaque action (pause, arrêt, reprise), un nouvel Intent est créé et associé à l'action spécifique :
    - **Pause** : Un Intent est créé pour appeler MusicService avec l'action "PAUSE_MUSIC".
    - **Stop** : Un Intent est créé pour arrêter la musique avec l'action "STOP_MUSIC".
    - **Resume** : Un Intent est créé pour reprendre la musique avec l'action "RESUME_MUSIC".

- **PendingIntent** : Chaque Intent est encapsulé dans un PendingIntent qui sera déclenché lorsque l'utilisateur cliquera sur l'action de la notification :

- **Construction de la notification** : Un NotificationCompat.Builder est utilisé pour créer la notification avec les éléments suivants :
  - **Titre** : Titre de la notification (ex. "Solar System Music").
  - **Texte** : Texte de notification affichant le message reçu.
  - **Icône** : Icône affichée dans la notification (setSmallIcon(R.drawable.ic_music_note)).
  - **Actions** : Ajout des actions pour contrôler la musique (addAction).
  - **Ongoing** : Définir la notification comme en cours pour éviter que l'utilisateur puisse la balayer (setOngoing(true)).
- **Affichage de la notification** : La notification est ensuite affichée avec startForeground, ce qui la place dans la zone des notifications du système et l'affiche en tant que service en premier plan :

3. onStartCommand(Intent intent, int flags, int startId)

```
@Override  4 usages
public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d( tag: "NotificationService",  msg: "NotificationService started");

    if (intent != null) {
        String message = intent.getStringExtra( name: "message");
        if (message != null) {
            Log.d( tag: "NotificationService",  msg: "Showing notification with message: " + message);
            showNotification(message);
        } else {
            Log.d( tag: "NotificationService",  msg: "No message received to show notification.");
        }
    } else {
        Log.d( tag: "NotificationService",  msg: "Intent is null");
    }

    return START_STICKY;
}
```

Cette méthode est appelée chaque fois qu'un client démarre le service avec startService(Intent). Elle gère les commandes de notification et appelle showNotification pour afficher la notification.

Paramètres

- **Intent intent** : L'intention contenant les données de notification.
- **int flags** : Indicateurs sur la manière dont le service doit être démarré.
- **int startId** : Identifiant unique pour cette demande de démarrage.

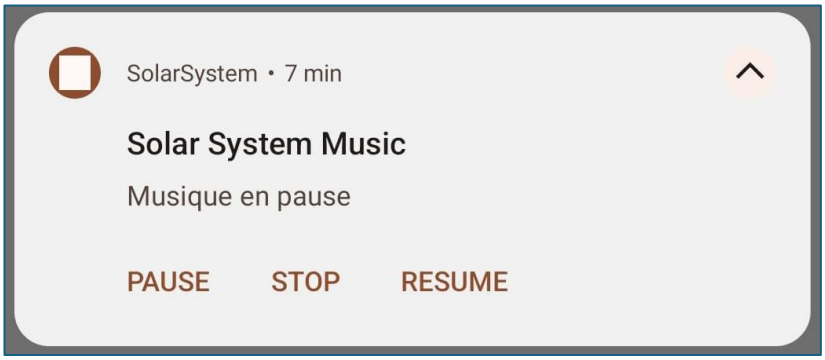
Fonctionnement

- **Vérification de l'intention** : La méthode vérifie si l'intention (intent) n'est pas nulle.
- **Extraction du message** : Si l'intention n'est pas nulle, elle extrait le message de la notification à partir de l'intention :
- **Affichage de la notification** : Si un message est présent, la méthode appelle showNotification(message) pour afficher la notification. Sinon, elle enregistre une entrée dans le journal pour indiquer qu'aucun message n'a été reçu.

4. Gestion des notifications

Le service de notification utilise NotificationCompat.Builder pour créer et afficher des notifications. Chaque notification comprend :

- Un titre et un message et des actions permettant de contrôler la musique (pause, arrêter, reprendre).



Ces actions utilisent PendingIntent pour déclencher des commandes dans le MusicService.

5. Permissions nécessaires

Pour que les services fonctionnent correctement, l'application doit déclarer certaines permissions dans le fichier AndroidManifest.xml, notamment :

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PLAYBACK" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
```

`<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />`
Autorise l'application à exécuter des services en premier plan, comme les notifications ou la lecture de musique.

`<uses-permission android:name="android.permission.FOREGROUND_SERVICE_MEDIA_PLAYBACK" />`
Spécifiquement destiné aux services de lecture multimédia, cette permission permet une gestion optimisée des services liés à la lecture de contenu audio ou vidéo en premier plan.

`<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />` Nécessaire pour afficher des notifications à l'utilisateur, cette permission est obligatoire à partir d'Android 13 (API niveau 33) pour garantir que les utilisateurs approuvent explicitement les notifications de l'application.

6. Logique d'appel des services

a. NotificationService

Dans la méthode onCreate, un Intent est créé pour démarrer le NotificationService :

```
Intent notificationIntent = new Intent( packageContext: this, NotificationService.class);
startService(notificationIntent);
```

- **Objectif** : Ce service gère l'affichage des notifications pour l'application, permettant à l'utilisateur de contrôler la lecture de la musique même lorsque l'application est en arrière-plan.
- **Démarrage du service** : La méthode startService est utilisée pour lancer le service. Cela signifie que le service s'exécute en arrière-plan, et l'application peut continuer à fonctionner sans interruption.

b. MusicService

Pour jouer la musique associée à la planète sélectionnée, un autre Intent est créé pour démarrer le MusicService :

```
// lancer music
Intent playIntent = new Intent( packageContext: this, MusicService.class);
playIntent.setAction("PLAY_MUSIC");
playIntent.putExtra( name: "musicResId", musicResId);
startService(playIntent);
```

- **Action** : L'action "PLAY_MUSIC" est définie dans l'intent pour indiquer au service de jouer la musique.
- **Passage des données** : Le fichier audio est identifié par son ID de ressource (musicResId), qui est passé comme extra dans l'intent.
- **Démarrage du service** : Encore une fois, startService est utilisé pour lancer la musique, permettant au service de gérer la lecture audio en arrière-plan.

2. Gestion de la Musique

a. Méthodes onPause(), onResume() et onDestroy()

Ces méthodes sont responsables de la gestion de l'état de la musique lorsque l'utilisateur interagit avec l'application :

- **onPause()** : Lorsque l'activité passe en arrière-plan, la musique est mise en pause :
- **onResume()** : Lorsque l'activité revient au premier plan, la musique est reprise :

- **onDestroy()** : Lorsque l'activité est détruite, la musique est arrêtée :

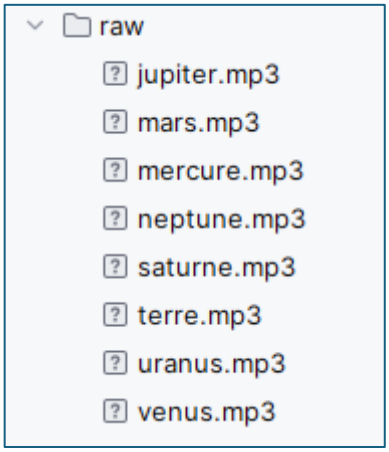
```
@Override
protected void onPause() {
    super.onPause();
    // Mettre en pause la musique lorsque l'application est en arrière-plan
    Intent pauseIntent = new Intent( packageContext: this, MusicService.class);
    pauseIntent.setAction("PAUSE_MUSIC");
    startService(pauseIntent);
}

@Override
protected void onResume() {
    super.onResume();
    // Reprendre la musique lorsque l'application revient au premier plan
    Intent resumeIntent = new Intent( packageContext: this, MusicService.class);
    resumeIntent.setAction("RESUME_MUSIC");
    startService(resumeIntent);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // Arrêter la musique lorsqu'on quitte complètement l'activité
    Intent stopIntent = new Intent( packageContext: this, MusicService.class);
    stopIntent.setAction("STOP_MUSIC");
    startService(stopIntent);
}
```

Ces méthodes garantissent que la musique est contrôlée de manière appropriée en fonction de l'état de l'activité, offrant une meilleure expérience utilisateur.

3. Liens entre les fichiers audio et les ressources

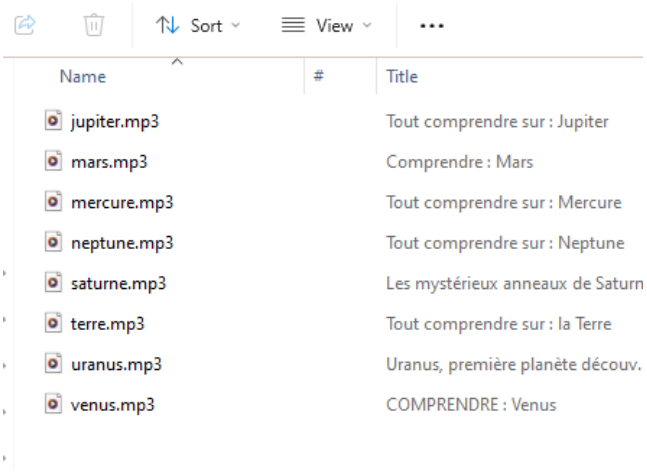
Les fichiers audio sont liés à la classe PlanetDetailActivity par le biais de la méthode getMusicResIdForPlanet(String planetName) :

```java
private int getMusicResIdForPlanet(String planetName) {
    switch (planetName) {
        case "Mercure":
            return R.raw.mercure;
        case "Vénus":
            return R.raw.venus;
        case "Terre":
            return R.raw.terre;
        case "Mars":
            return R.raw.mars;
        case "Jupiter":
            return R.raw.jupiter;
        case "Saturne":
            return R.raw.saturne;
        case "Uranus":
            return R.raw.uranus;
        case "Neptune":
            return R.raw.neptune;
        default:
            return 0;
    }
}
```



- **Méthode** : Cette méthode prend le nom d'une planète comme paramètre et retourne l'ID de ressource correspondant pour le fichier audio stocké dans le dossier res/raw/.
- **Utilisation des ressources** : Les fichiers audio (par exemple, mercure.mp3, venus.mp3, etc.) sont placés dans le répertoire res/raw/ de l'application, ce qui permet à l'application de les accéder facilement via leur ID de ressource.

3. Téléchargement des pistes audio



5. Déclaration des services dans le Manifest

Pour que les services soient reconnus parl' applicationon doit les déclarer dans le fichier AndroidManifest.xml. Voici comment procéder :

```
<service android:name=".MusicService"
    android:enabled="true"
    android:exported="false" />

<service android:name=".NotificationService"
    android:foregroundServiceType="mediaPlayback"
    android:enabled="true"
    android:exported="false" />
```

- **Services** : Ces lignes indiquent à Android que NotificationService et MusicService sont des services que l' application utilise.
- **Position** : Ces déclarations doivent être ajoutées à l'intérieur de la balise <application> de le fichier AndroidManifest.xml.

Hannachi Thamer l3dsi EAD