Développement Android Wear

Jean-François COUCHOT
couchot[arobase]femto-st[point]fr

8 février 2019

Table des matières

Chapitre 1

Analyse d'une interface classique

Ce premier chapitre se focalise sur les principaux composants d'une interface utilisateur : les deux menus `WearableNavigationDrawerView`, `WearableActionDrawerView` et les fragments au travers d'un exemple. Récupérer le projet android-WearDrawers (git clone https://github.com/googlesamples/android-WearDrawers.git), l'ouvrir dans studio et mettre à jour le projet en synchronisant le gradle. L'exécuter dans un émulateur android Wear carré ou rond adapté.

1.1 Une interface à base de `WearableDrawerLayout`

Exercice 1.1. *Eléments de l'interface. Jouer avec l'application proposée.*

1. *Constater la présence d'un menu supérieur à 8 éléments organisés horizontalement.*
2. *Constater la présence d'un menu inférieur permettant d'effectuer 4 actions sur la vue courante.*
3. *Constater que lorsqu'on active ces menus, l'image principale peut être décalée.*

Une partie de cette interface se comprend à l'aide de l'analyse de `activity_main.xml` dont une vue synthétique serait comme suit.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.wear.widget.drawer.WearableDrawerLayout ...>
<android.support.v4.widget.NestedScrollView>
 <FrameLayout
   android:id="@+id/content_frame" ... />
</android.support.v4.widget.NestedScrollView>
<android.support.wear.widget.drawer.WearableNavigationDrawerView
android:id="@+id/top_navigation_drawer" ... />

<android.support.wear.widget.drawer.WearableActionDrawerView
android:id="@+id/bottom_action_drawer" ... />

</android.support.wear.widget.drawer.WearableDrawerLayout>
```

Exercice 1.2. *Organisation de l'UI.*

1. *Que va contenir* `FrameLayout` *?*
2. *Que va contenir* `WearableNavigationDrawerView` *?*
3. *Que va contenir* `WearableActionDrawerView` *?*

Un squelette de l'activité `MainActivity` est donné ci-dessous.

```
public class MainActivity extends Activity implements
  WearableNavigationDrawerView.OnItemSelectedListener,
  MenuItem.OnMenuItemClickListener,
  PlanetFragment.OnFragmentInteractionListener{

 private WearableNavigationDrawerView mWearableNavigationDrawer;
 private WearableActionDrawerView mWearableActionDrawer;

 private ArrayList<Planet> mSolarSystem;
```

```
 private int mSelectedPlanet;
 private PlanetFragment mPlanetFragment;

 @Override
 protected void onCreate(Bundle savedInstanceState) { ...}

 @Override
 public void onItemSelected(int position) {...}

 @Override
 public boolean onMenuItemClick(MenuItem menuItem) {...}

 private ArrayList<Planet> initializeSolarSystem() {...}

 @Override
 public void onFragmentInteraction(Uri uri) {
}
```

Exercice 1.3. *Organisation de la classe* `MainActivity`. *Sans lire le code plus en détail :*

1. *Analyser les attributs de cette classe.*
2. *La méthode* `onItemSelected` *est liée à une classe abstraite. Laquelle ?*
3. *La méthode* `onMenuItemClick` *est liée à une classe abstraite. Laquelle ?*

1.2 L'initialisation

1.2.1 Initialisation du modèle

Cette application travaille avec des objets de la classe `Planet`. Une lecture simple de cette classe (la trouver, la comprendre) montre qu'elles contient :

— 6 attributs, tous de type `String`, même pour les données de type numériques ou les images (`navigationIcon`, `image`);
— un constructeur classique avec ces 6 attributs ;
— un accesseur par attribut.

Le tableau de planètes `mSolarSystem` est initialisé grâce à la méthode `initializeSolarSystem()` facile à comprendre. Lire cette méthode et l'analyser.

1.2.2 Initialisation du fragment `mPlanetFragment`

Le fragment `mPlanetFragment` est chargé lors des instructions

```
Bundle args = new Bundle();
int imageId = getResources().getIdentifier(mSolarSystem.get(mSelectedPlanet).getImage(),
                                           "drawable", getPackageName());
args.putInt(PlanetFragment.ARG_PLANET_IMAGE_ID, imageId);

mPlanetFragment = new PlanetFragment();
mPlanetFragment.setArguments(args);
FragmentManager fragmentManager = getFragmentManager();
fragmentManager.beginTransaction().replace(R.id.content_frame, mPlanetFragment).commit();
```

Exercice 1.4. *Compréhension de la mise en place du fragment.*

1. *Quel est l'objectif des trois premières lignes du code précédent ?*
2. *Où cet argument* `args` *est il utilisé ensuite ?*
3. *A quoi sert la dernière ligne du code précédent ?*

1.3 L'interactivité inférieure : `WearableActionDrawerView`

Exercice 1.5.
1. *Repérer les trois lignes de la méthode* `onCreate` *où est initialisé le menu inférieur* `mWearableActionDrawer`.
2. *Quelle méthode va être invoquée lorsqu'un item du menu inférieur va être cliqué ?*
3. *Analyser et comprendre cette méthode.*

1.4 L'interactivité supérieure : `WearableNavigationDrawerView`

Exercice 1.6.
1. *Repérer les quatre grandes lignes de la méthode* `onCreate` *où est initialisé le menu supérieur* `mWearableNavigationDrawer`.
2. *Quelle méthode va être invoquée lorsqu'un item du menu supérieur va être sélectionné ?*
3. *Analyser et comprendre cette méthode, notamment la mise à jour du fragment.*
4. *Analyser et comprendre ce qui est passé en paramètre de la méthode* `setAdapter(...)` *de* `mWearableNavigationDrawer`.

1.5 Le mode « Ambiant »

Certaines activité nécessitent de rester en premier plan (application de sport, de guidage...). Pour économiser la batterie, il apparaît comme judicieux de prévoir un mode d'affichage économe en énergie, appelé mode ambiant, lorsqu'il n'y a pas d'interaction avec l'utilisateur. Cette section montre comment intégrer cette possibilité.

Exercice 1.7.
1. *Dans le Manifest ajouter la permission* `android.permission.WAKE_LOCK`.
2. *Dans la méthode* `MainActivity.onCreate` *ajouter l'instruction* `setAmbientEnabled();`.
3. *Spécifier que* `MainActivity` *implémente la classe* `AmbientCallbackProvider` *et demander à studio d'implanter la méthode manquante* `getAmbientCallback()`.
4. *Remplacer la méthode proposée par celle qui suit après l'avoir comprise.*

```
@Override
public AmbientMode.AmbientCallback getAmbientCallback() {
  return new AmbientMode.AmbientCallback() {

   @Override
   public void onEnterAmbient(Bundle ambientDetails) {
    super.onEnterAmbient(ambientDetails);
    Log.d(TAG, "onEnterAmbient() " + ambientDetails);

    mPlanetFragment.onEnterAmbientInFragment(ambientDetails);
    mWearableNavigationDrawer.getController().closeDrawer();
    mWearableActionDrawer.getController().closeDrawer();
   }

   @Override
   public void onExitAmbient() {
    super.onExitAmbient();
    Log.d(TAG, "onExitAmbient()");

    mPlanetFragment.onExitAmbientInFragment();
    mWearableActionDrawer.getController().peekDrawer();
   }
  };
 }
```

5. *Ajouter les deux méthode suivantes au bon endroit et les comprendre.*

```
public void onEnterAmbientInFragment(Bundle ambientDetails) {
  Log.d(TAG, "PlanetFragment.onEnterAmbient() " + ambientDetails);
  // Convert image to grayscale for ambient mode.
  ColorMatrix matrix = new ColorMatrix();
  matrix.setSaturation(0);

  ColorMatrixColorFilter filter = new ColorMatrixColorFilter(matrix);
  mImageView.setColorFilter(filter);
 }

 // Restores the UI to active (non-ambient) mode.
 public void onExitAmbientInFragment() {
  Log.d(TAG, "PlanetFragment.onExitAmbient()");
  mImageView.setColorFilter(mImageViewColorFilter);
}
```

6. *Constater le bon fonctionnement du mode Ambiant.*

Chapitre 2

Les capteurs sensoriels

Ce chapitre se base sur le code du projet HealthyBody. Récupérer le projet `HealthyBody.zip`, le décompresser et l'ouvrir. Exécuter l'application pour wear dans l'émulateur.

Pour réaliser ce TP, il faudrait idéalement une montre connectée configurée en mode débugage comme ce qui est détaillé à l'url https://developer.android.com/training/wearables/apps/debugging.html.

2.1 Reprise de code existant

2.1.1 Une application pour montre et mobile

On commencera par repérer le fait que le projet a pour but de fournir une application pour montres intelligente et une pour mobile/tablette. Chacune des deux applications est développée séparément avec ses propres outils de compilation et d'autorisation.

Exercice 2.1.
1. *Repérer les fichiers* `build.gradle` *de chaque partie.*
2. *Repérer les fichiers* `AndroidManifest.xml` *de chaque partie.*
3. *Repérer l'activité* `MainActivity` *de la partie mobile. Quelles fonctionnalités y sont développées.*
4. *Dans la partie* `wear`*, repérer les deux paquetages* `fragments` *et* `utils` *ainsi que l'activité* `MainActivity`.
   (a) *Quelles fonctionnalités sont développées dans* `MainActivity` *?*
   (b) *Quels fragments ont été développés ? Quelles fonctionnalité vont-il apporter à terme.*

2.1.2 Analyse du fragment `HeartRateFragment`

Le fragment `HeartRateFragment` va être utilisé pour afficher le rythme cardiaque.

Exercice 2.2. *Analyse du gabarit puis de* `HeartRateFragment`.
1. *Quelle vue va afficher une valeur numérique de la fréquence cardiaque ?*
2. *Quel objet va afficher une animation ? Repérer le chargement de l'image du cœur.*
3. *Repérer l'initialisation des vues dans* `HeartRateFragment`.

2.1.3 Analyse du fragment `StepCounterFragment`

Le fragment `StepCounterFragment` va être utilisé pour afficher le nombre de pas effectués après le lancement de l'application et uniquement le jour même.

Exercice 2.3. *Analyse du gabarit puis de* `StepCounterFragment`.
1. *Repérer les deux TextViews dans le gabarit.*
2. *Repérer l'initialisation de la vue comptabilisant le nombre de pas dans* `StepCounterFragment`.

2.1.4 Chargement des fragments par `WearableNavigationDrawerView`

Dans la classe `MainActivity`, les deux fragments présentés précédemment sont utilisés par la la vue `WearableNavigationDrawerView` pour remplacer à la volée le framelayout identifié dans le gabarit xml par `content_frame`, comme au chapitre 1.

Exercice 2.4. *Méthode* `OnCreate` *de* `MainActivity`.

1. *Repérer la ligne qui contient l'appel à la méthode* `initializeScreenSystem()`. *Comprendre cette méthode.*
2. *Comprendre les lignes relatives à l'utilisation du* `fragmentManager`.
3. *Comprendre toutes les lignes qui concernent* `mWearableNavigationDrawer`.

2.2 Afficher le cumul du nombre de pas

2.2.1 Un comptage de pas journalier : `DailyStep`

A partir de la version 2.0 l'API d'Android wear propose un capteur de pas (step counter) qui est initialisé à chaque démarrage de l'OS.

Exercice 2.5.

1. *Créer la classe* `DailyStep` *dans le paquetage* `utils` *dont l'objectif est de retourner le nombre de pas effectués après le lancement de l'application et uniquement le jour même.*
2. *Dans cette classe, créer l'attribut* `stepDeadline` *statique de type* `Calendar` *qui va mémoriser la date limite de validité du comptage de pas.*
3. *Dans cette classe, créer l'attribut statique de type entier* `steps` *qui va mémoriser le nombre de pas journaliers.*
4. *Dans cette classe, créer la méthode* `updateSteps(int stepsTaken)` *qui prend en paramètre un nombre de pas* `stepsTaken` *et qui*
   (a) *initialise au lendemain à 0h00 la variable* `stepDeadline` *si elle est nulle ou si la date courante est après cette date limite.*
   (b) *met à jour le compteur* `step` *soit en ajoutant le nombre de pas aux pas déjà effectués, soit en le réinitialisant avec ce nombre (si on est dans le cas précédent).*
5. *Ajouter la méthode accesseur* `getSteps`.

2.2.2 Le service `WearStepService` qui capture les pas

On crée un service `WearStepService` dont l'objectif est de récupérer lorsqu'il change le nombre de pas réalisés et de les retourner au fragment `StepCounterFragment` gérant l'affichage de ce nombre de pas. Pour pouvoir activer un service, celui-ci doit être déclaré dans le Manifest.

Exercice 2.6. *Mise en place du service.*

1. *Déclarer le service* `WearStepService` *dans le Manifest de la partie wear de l'application.*
2. *Créer la classe* `WearStepService` *qui étend* `Service` *et qui implante* `SensorEventListener`. *Demander à Studio d'implanter les méthodes manquantes.*
3. *Gérer l'initialisation du service en ajoutant la méthode suivante après l'avoir comprise :*

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  Log.d(TAG, "onStart");
  sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
  sensorManager.registerListener(this, countSensor, SensorManager.SENSOR_DELAY_NORMAL);
  return super.onStartCommand(intent, flags, startId);
 }
```

4. *Lorsqu'un événement comptant les pas survient, la méthode* `onSensorChanged` *est invoquée. remplacer les méthodes* `onSensorChanged` *et* `onAccuracyChanged` *engendrée par défaut par ce qui suit et ajouter les attributs manquants.*

```
@Override
public void onSensorChanged(SensorEvent event) {
 Log.d(TAG, "onSensorChanged");
 if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
  DailySteps.updateSteps((int)event.values[0]);
  sendStepCountUpdate();
 }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
 Log.d(TAG, "onAccuracyChanged");
 sendStepCountUpdate();
}

private void sendStepCountUpdate() {
 Intent intent = new Intent();
 intent.setAction(STEP_COUNT_MESSAGE);
 intent.putExtra(STEP_COUNT_VALUE,DailySteps.getSteps());
 sendBroadcast(intent);
}
```

5. *Que réalise l'instruction* `DailySteps.updateSteps((int)event.values[0]);` *? Que réalise la méthode* `sendStepCountUpdate()` *?*

2.2.3 Interagir avec le service `WearStepService`

Comme dans n'importe quel développement Android, démarrer et arrêter un service se fait dans un fragment avec les instructions `startService(nomDuService)` et `stopService(nomDuService)` respectivement. Ici le paramètre `nomDuService` est `stepServiceIntent` de type `Intent` et est initialisé dans la méthode `onCreateView` du fragment comme suit :

```
stepServiceIntent = new Intent(getActivity(), WearStepService.class);
```

Exercice 2.7.

1. *Compléter la méthode* `onCreateView` *pour initialiser l'Intent.*
2. *Ajouter les méthodes* `startService` *et* `stopService` *dans les méthodes* `onResume` *et* `onPause` *du fragment respectivement.*
3. *Modifier les trois méthodes* `onCreateView`, `onResume` *et* `onPause` *pour pouvoir capturer les messages envoyés par* `WearStepService.sendStepCountUpdate` *et mettre à jour l'affichage du nombre de pas réalisés.*

2.2.4 Un émulateur du capteur de nombre de pas

On exploitera les deux classes suivantes que l'on comprendra :

```
public class WearStepEmulatorService extends WearStepService {

    public static final String TAG = "WearStepEmulatorService";
    private static Timer mTimer =null ;

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d(TAG, "onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
       if (mTimer ==null) {
           mTimer = new Timer();
           long delay =  1000*5;
           long period = 1000*20;
           mTimer.scheduleAtFixedRate(new RandomStepCounterTimerTask(getApplicationContext()), delay, period);
       }
       return super.onStartCommand(intent, flags, startId);
    }
```

```
    @Override
    public void onDestroy() {
        mTimer.cancel();
        mTimer = null;
        super.onDestroy();
    }

}
```

et

```
public class RandomStepCounterTimerTask extends TimerTask {
    public static final String TAG="RandomStepCounterTimerTask";
    private Context ctxt =null;
    private static int stepCount ;

    public RandomStepCounterTimerTask(Context ctxt){
        this.ctxt = ctxt.getApplicationContext();
        stepCount = (int) (Math.random()*200);
        DailySteps.updateSteps(stepCount);
    }
    @Override
    public void run() {
        stepCount += (int) (Math.random()*15);
        DailySteps.updateSteps(stepCount);
        Intent intent = new Intent();
        intent.setAction(STEP_COUNT_MESSAGE);
        intent.putExtra(STEP_COUNT_VALUE,DailySteps.getSteps());
        ctxt.sendBroadcast(intent);
    }
}
```

2.3 Afficher la pulsation cardiaque

Sur le même principe qu'à la section précédente, développer un service qui va retourner régulièrement la fréquence cardiaque de l'utilisateur et l'interface qui va afficher cette fréquence.

On ferra attention à

— ne pas oublier de déclarer le service dans le Manifest ;

— arrêter le service lorsqu'on en a plus besoin ;

— bien gérer les permissions d'utilisation du capteur de fréquence cardiaque. On peut s'inspirer de l'exemple https://github.com/googlesamples/android-RuntimePermissionsWear.

On pourra aussi développer un capteur virtuel de pulsation cardiaque comme dans la section précédente pour pouvoir tester le code dans un émulateur.

Chapitre 3

Des données synchronisées (montre et mobile)

L'objectif de ce chapitre est de synchroniser des données entre le mobile et la montre connectée. Les données synchronisées (en temps réel) seront le nombre de pas journaliers et la fréquence cardiaque instantanée. Si le développement ne se fait qu'avec des émulateurs (c'est-à-dire sans dispositif physique), on effectuera l'exercice de la section 3.1. Sous windows cela peut ne pas fonctionner.

3.1 Emulateurs pour un développement autonome

Dans cette section, on construit un émulateur de téléphone et un émulateur de montre pouvant être jumelés entre eux. Si l'émulateur de montre a déjà été jumelé avec un dispositif, en créer un second.

Exercice 3.1. *Émulateur de téléphone avec Android Wear - Smartwatch. Il est nécessaire d'avoir un émulateur de téléphone avec Play Store déjà installé pour pouvoir ajouter ensuite Android Wear - Smartwatch permettant le jumelage avec la montre.*

1. *Construire un nouvel émulateur de téléphone à base de Google Nexus 5 (qui contient Google Play Store).*
2. *Fermer tous les émulateurs android et lancer cet émulateur de téléphone. Ajouter votre compte Google à cet émulateur si cela vous l'est demandé.*
3. *Télécharger la dernière version de de l'apk Android Wear - Smartwatch.*
4. *Installer cette application dans l'émulateur du téléphone :*
   `adb install com.google.android.wearable.....apk`.
5. *Lancer l'émulateur de montre. Repérer dans la barre de statut le numéro de port de cet émulateur de montre (il devrait être égal à 5554 ou 5556).*
6. *Exécuter cette application dans l'émulateur de téléphone en se restreignant à accepter les conditions d'utilisation. Ne PAS essayer de jumeler un dispositif.*
7. *Repérer dans la barre de statut le numéro de port de cet émulateur de téléphone (il devrait être égal à 5554 ou 5556).*
8. *Exécuter* `telnet localhost XXXX` *où* `XXXX` *est le numéro de port de l'émulateur du téléphone. Vous devriez avoir une réponse semblable à*

   ```
   Android Console: you can find your <auth_token> in
   '/home/couchot/.emulator_console_auth_token'
   ```

9. *Copier la clef KKKKKKKKKKKKK enregistrée dans le fichier défini par* `auth_token`*, puis dans la commande telnet exécuter* `auth KKKKKKKKKKKKK` *pour être authentifié.*
10. *Une fois authentifié, exécuter* `redir add tcp:5601:5601`.
11. *Réouvrir Android Wear - Smartwatch et demander le jumelage avec l'émulateur. Cela prend un peu de temps.*

A la fin de cette section, vous devriez savoir comment jumeler deux émulateurs android (une montre et un mobile).

3.2 Jumeler un mobile avec un émulateur de montre

Exercice 3.2. *1. Une fois le tépléphone connecté en USB (et à chaque fois que vous reconnnectez en USB le téléphone) rediriger les ports de communication vers ceux du smartphone*

```
adb -d forward tcp:5601 tcp:5601
```

2. *Sur le téléphone, dans l'application « Android Wear », lancez le processus d'appairage standard. Par exemple, sur l'écran Bienvenue, tapez sur le bouton « Configurer ». Sinon, si une montre existante est déjà jumelée, dans le menu déroulant en haut à gauche, appuyez sur « Ajouter une nouvelle montre ».*
3. *Sur le téléphone, dans l'application « Android Wear », choisir « Pair with Emulator ».*
4. *Dans le menu de « Paramètres », chosir « Emulateur ».*

3.3 Un projet pour montre et mobile

On va travailler à la fois sur la partie téléphone et sur la partie montre. Il suffit pour cela d'avoir un projet pour les deux supports simultanément.

Exercice 3.3. *Un projet pour montre et téléphone.*

1. *Construire un nouveau projet (Healthy-3) pour téléphone, tablettes ET montres.*
2. *Exécuter la partie téléphone et la partie montre dans les dispositifs jumelés.*
3. *Intégrer tout le code du projet Healthy-2.*

Dans ce genre de projets multi-cibles, il est pratique de partager des variables entre la partie wear et la partie mobile, particulièrement les constantes statiques.

Exercice 3.4. *Un module supplémentaire commun de constantes.*

1. *Créer un nouveau module de type « bibliothèque Android », le nommer « common » et finaliser la procédure de création.*
2. *Dans ce module créer une classe nommée « Constants » contenant pour l'instant l'unique attribut* `STEP_COUNT_VALUE` *défini comme suit :*

```
public static final String
   STEP_COUNT_VALUE="com.iutbm.example.iutbm.couchot.healthybody.STEP_COUNT_VALUE";
```

3. *A votre avis, à quoi va servir cette constante ?*
4. *Dans le* `build.gradle` *de chaque autre module (mobile, wear), ajouter la dépendance*

```
compile project(":common")
```

5. *Remplacer dans le code toutes les références à* `WearStepService.STEP_COUNT_VALUE` *par* `Constants.STEP_COUNT_VALUE`.
6. *Généraliser en déplaçant toutes les constantes statiques du projets dans la classe* `Constants`.
7. *Dans la classe* `Constants` *créer l'attribut suivant.*

```
public static final String STEP_COUNT_PATH="/count";
```

A la fin de cette section vous devriez savoir comment organiser un projet pour montre et téléphone avec en plus un module commun aux deux.

3.4 Envoyer les données en asynchrone

L'envoi des données d'un dispositif à un autre pour la synchronisation de celles-ci se fait au travers d'une tâche asynchrone.

Exercice 3.5. *Une Tâche asynchrone.*

1. *Copier le code suivant et demander à Studio de compléter l'implantation des méthodes.*

```
public class SynchronizeAsyncTask extends AsyncTask<Integer, Integer, Integer> {
    private DataClient mDataClient ;

    public SynchronizeAsyncTask(Activity  mContext) {
        super();
        mDataClient = Wearable.getDataClient(mContext);
    }

    @Override
    protected Integer doInBackground(Integer... params) {
        int stepCount = params[0];
        PutDataMapRequest putDataMapReq = PutDataMapRequest.create(Constants.STEP_COUNT_PATH);
        putDataMapReq.getDataMap().putInt(Constants.STEP_COUNT_VALUE, stepCount);
        PutDataRequest putDataReq = putDataMapReq.asPutDataRequest();
        putDataReq.setUrgent();
        Task<DataItem> putDataTask = mDataClient.putDataItem(putDataReq);
        return -1;
    }
}
```

2. *Comprendre le code précédent.*
3. *Dans la méthode* `onReceive` *du broadcastReceiver déclaré dans la classe* `StepCounterFragment` *ajouter les deux lignes suivantes, après les avoir comprises.*

```
mSynchronizeAsyncTask  = new SynchronizeAsyncTask(getActivity());
mSynchronizeAsyncTask.execute(new Integer(nbstep));
```

4. *Quel est l'objectif de la deuxième ligne du code ci-dessus ? Quelle méthodes sont invoquées lorsque l'instruction* `execute` *est effectuée ?*

A la fin de cette section vous devriez avoir compris comment envoyer des données à un autre dispositif en utilisant une tâche asynchrone

3.5 Récupération des données envoyées

Les données sont récupérées depuis une classe implantant `DataClient.OnDataChangedListener`. Elles vont être affichées dans un TextView de l'activité `MainActivity` de la partie Mobile

Exercice 3.6. *Récupération des données modifiées.*

1. *L'activité* `MainActivity` *du module mobile contient un TextView affichant le text "Hello World". Remplacer le texte par "No Datta" et identifier ce textview par* `tv_stc`.
2. *Dans cette même activité, créer l'attribut* `tv_stc` *de type TextView et le lier au TextView de l'interface identifié par le même nom.* `onCreate`.
3. *Préciser que cette activité implante* `DataClient.OnDataChangedListener`. *Ajouter la méthode suivante que l'on comprendra.*

```
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
 Log.d(TAG, "onDataChanged" + dataEvents);

 for (DataEvent event : dataEvents) {
  if (event.getType() == DataEvent.TYPE_CHANGED) {
   // DataItem changed
   DataItem item = event.getDataItem();
   if (item.getUri().getPath().compareTo(Constants.STEP_COUNT_PATH) == 0) {
    DataMap dataMap = DataMapItem.fromDataItem(item).getDataMap();
    int st = dataMap.getInt(Constants.STEP_COUNT_VALUE, 0);
    tv_stc.setText(String.valueOf(st) + " pas effectué(s) !");
   }
```

```
  }
 }
}
@Override
protected void onResume() {
 super.onResume();
 Wearable.getDataClient(this).addListener(this);
}

@Override
protected void onPause() {
 super.onPause();
 Wearable.getDataClient(this).removeListener(this);
}
```

4. *Améliorer le code précédent, particulièrement l'utilisation de la chaîne de caractères " pas effectué(s)!".*
5. *Exécuter les deux applications et constater que les données sont bien envoyées de la montre vers le mobile.*

A la fin de cette section vous devriez avoir compris comment récupéer des données à l'aide de `DataClient.OnDataChangedListener`.

3.6 Affichage des pulsations cardiaques moyennes sur le mobile

L'objectif de cette section est d'afficher sur son mobile la moyenne des pulsations cardiaques de l'utilisateur sur

— les 10 dernières secondes ;
— les 20 dernières secondes ;
— les 30 dernières secondes.

Lorsqu'il n'y a pas assez de données, l'utilisateur doit être informé que celles-ci ne sont pas suffisantes pour afficher de telles moyennes.

Chapitre 4

L'API Firebase

Firebase est un ensemble de services fournis par Google tels que de l'hébergement de bases de données NoSQL, de l'authentification simple à mettre en place avec son compte de réseau social (Google, Facebook, . . .),. . . On se concentre ici sur l'authentification et les bases de données NoSQL.

4.1 Authentification avec Firebase

Cette section est inspiré du livre [Mor17].

La plupart des applications ont besoin d'identifier les utilisateurs. Cela permet à ce dernier de définir ses préférences, stocker ses données. . . sur tous les appareils connectés de l'utilisateur. une application digne de ce nom doit permettre d'inscrire des nouveaux utilisateurs, de connecter les utilisateurs existants, de gérer la déconnexion. . .

Pour des raisons d'efficacité de développement et de sécurité des données stockées, il est préférable de s'appuyer sur des bibliothèques développées et maintenues par une tierce partie (Firebase) qui elle-même demandera éventuellement à des tierces-parties (Facebook, Gmail, par exemple) d'identifier l'utilisateur.

Le projet suivant se réalise sur un mobile classique, mais il pourrait tout à fait être exécuté sur une montre.

4.1.1 Les éléments du projet

Le projet contiendra une activité d'accueil (nommée `ConnectionActivity`) et toute l'architecture issue de projet à base NavigationDrawer, contenant notamment `MainActivity`. C'est enfin la bibliothèque `FirebaseUI` qui sera utilisée pour créer toutes les interfaces (formulaires de connexion et autres) nécessaires à la connexion.

Exercice 4.1. *`ConnectionActivity` et `MainActivity`.*

1. *Créez un nouveau projet pour mobile nommé ApplicationTestFirebase à base de* `NavigationDrawer`*, vérifier que le numéro de version minimale est le 16 (nécessaire pour Firebase).*
2. *Demander à Studio de créer l'activité nommée* `ConnectionActivity`*.*
3. *Modifier le Manifest pour que l'activité lancée par l'application soit* `ConnectionActivity` *et pas* `MainActivity`*.*
4. *Toute l'authentification va utiliser Internet. Ajouter la permission suivante dans le Manifest.*

```
<uses-permission android:name="android.permission.INTERNET" />
```

Nous allons d'abord mettre en place les briques de base indispensables à la compilation d'un projet qui utilise l'authentification par Firebase.

Exercice 4.2. *Mise en place de Firebase Authentication.*

1. *Dans* `Tools > Firebase` *, choisir « Email and password Authentication ».*

2. *Demander ensuite la connexion de l'application à Firebase (point 1.) et choisir de créer un nouveau projet nommé* `ApplicationTestFirebase`. *Studio crée automatiquement dans le projet les éléments nécessaires à Firebase/Authentication et (sur le serveur de Firebase) le projet* `ApplicationTestFirebase`.
3. *Demander à Studio d'ajouter Firebase Authentication à votre application (point 2.). Cela modifie les* `build.gradle` *de votre projet Android.*
4. *Repérer "To use an authentication provider, you need to enable it in the Firebase console ...." (point 2.) et suivre le lien pour aller sur la page web de la console Firebase.*
5. *Sur la page web de la console Firebase, choisir dans le menu gauche* `Authentication`, *puis l'onglet « Mode de connexion ». Activer enfin la méthode* `Adresse e-mail/Mot de passe` .

Il reste à ajouter les bibliothèques open source FirebaseUI et une dépendance à `fabric.io` utilisées dans le projet Android.

Exercice 4.3. *Mise en place de FirebaseUI.*

1. *Dans le* `build.gradle` *du projet, dans la section* `allprojects` *ajouter après* `jcenter()` *la ligne* `maven { url 'https://maven.fabric.io/public'}`.
2. *Vérifier que vous n'avez pas ajouté cette ligne dans la section* `buildscript` *qui contient aussi* `jcenter()`.
3. *Dans le* `build.gradle` *de l'application ajouter*

```
implementation 'com.google.firebase:firebase-core:16.0.5'
implementation 'com.firebaseui:firebase-ui-auth:4.3.1'
```

4. *Dans le* `build.gradle` *du projet, changer le numéro de version* `4.1.0` *de la bibliothèque* `com.google.gms:google-services` *pour* `4.0.1`. *La 4.1.0 semble contenir un bug.*

4.1.2 Identification par mail

L'activité `ConnectionActivity` (vide) va vérifier l'authentification au lancement de l'application et redirigera l'utilisateur vers `MainActivity` en cas de succès

Exercice 4.4. *Vérification d'authentification dans* `ConnectionActivity`.

1. *Dans le gabarit* `activity_connection.xml`, *ajouter un bouton identifié par* `signInButton` *et contenant le texte « Connection ».*
2. *Ajouter l'attribut privé* `FirebaseAuth mAuth` *et un bouton nommé* `signInButton`
3. *Dans la méthode* `onCreate` *instancier* `mAuth` *et rediriger vers l'activité* `MainActivity` *si un utilisateur est connecté comme suit :*

```
FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
if(currentUser == null){
  startActivity(new Intent(this,ConnectionActivity.class));
  finish();
  return;
}
```

4. *Dans cette même méthode, instancier le bouton* `signInButton` *en le liant au bouton identifié par* `signInButton` *en XML et ajouter un écouteur de clic à ce bouton.*
5. *Dans la méthode* `onClick` *de cet écouteur, ajouter le code suivant :*

```
 startActivityForResult(
                       AuthUI.getInstance().createSignInIntentBuilder()
                               .build(),
                       RC_SIGN_IN);
```

et définir la constante `RC_SIGN_IN` *avec une valeur entière arbitraire (901 par exemple). Cela lance une activité de connexion, construite par*

```
AuthUI.getInstance().createSignInIntentBuilder().build()
```

en passant en paramètre `RC_SIGN_IN` *et attend un retour.*

Une fois cette activité de connexion terminée, l'activité `ConnectionActivity` *devrait recevoir un résultat. Si celui-ci est de type* `RC_SIGN_IN`*, un traitement spécifique peut être effectué.*

6. *Ajouter les 2 méthodes suivantes dans la classe* `ConnectionActivity` *et les comprendre.*

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
  super.onActivityResult(requestCode, resultCode, data);
  if(requestCode == RC_SIGN_IN){
    handleSignInResponse(resultCode, data);
  }
}

private void handleSignInResponse(int resultCode, Intent data) {
  IdpResponse response = IdpResponse.fromResultIntent(data);
  Toast toast;
  // Successfully signed in
  if (resultCode == RESULT_OK) {
    Intent intent = new Intent(this, MainActivity.class);
    startActivity(intent);
    finish();
    return;
  } else {
    // Sign in failed
    if (response == null) {
      // User pressed back button
      toast = Toast.makeText(this, "Sign in was cancelled!", Toast.LENGTH_LONG);
      toast.show();
      return;
    }
    if (response.getError().getErrorCode() == ErrorCodes.NO_NETWORK) {
      toast = Toast.makeText(this, "You have no internet connection", Toast.LENGTH_LONG);
      toast.show();
      return;
    }
    if (response.getError().getErrorCode() == ErrorCodes.UNKNOWN_ERROR) {
      toast = Toast.makeText(this, "Unknown Error!", Toast.LENGTH_LONG);
      toast.show();
      return;
    }
  }
  toast = Toast.makeText(this, "Unknown Error!", Toast.LENGTH_LONG);
  toast.show();
}
```

7. *Constater le bon fonctionnement de l'application. Elle permet à un utilisateur de s'authentifier avec un email/mdp. Cet utilisateur est ensuite enregistré dans la console de Firebase (le vérifier).*

Exercice 4.5. *La partie identifiée.*

1. *Créer un fragment nommé* `DeconnectionFragment` *(le code java et le gabarit xml).*
2. *Dans le fichier* `content_main.xml` *identifier le* `ConstraintLayout` *par* `content_main`*.*
3. *Dans le menu gauche, remplacer le libellé « Tools » par « Deconnection » et remplacer son identifiant par* `nav_connection`*.*
4. *Faire en sorte que cliquer sur le texte intitulé « Deconnection » dans le menu charge le fragment* `DeconnectionFragment` *dans le ConstraintLayout identifié par* `content_main`*.*
5. *Ajouter deux* `TextView`*s identifiés par* `user_email` *et* `user_display_name` *dans le fragment* `fragment_connection.xml` *et faire en sorte que ces* `TextView`*s affichent l'adresse mail et le nom de l'utilisateur connecté, comme à la figure 4.1. Ces informations devraient être extraites de l'objet* `currentUser` *dans la classe* `MainActivity` *et passées en paramètres à la création du fragment.*

Il reste maintenant à prévoir la déconnexion.

Exercice 4.6. *Déconnecter l'utilisateur.*

1. *Ajouter un bouton identifié par* `signOutButton`*, contenant le texte "Sign Out" dans le fragment* `DeconnectionFragment`*.*

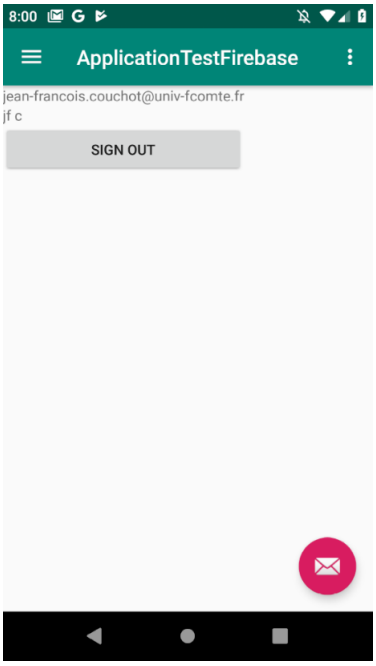

FIGURE 4.1 – Informations de connexion

2. *Ajouter un écouter de clic à ce bouton qui invoquera la méthode* `signOut` *ci-dessous :*

```
public void signOut(){
    AuthUI.getInstance().signOut(this)
         .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
              if(task.isSuccessful()){
                 startActivity(MainActivity.createIntent(SignedInActivity.this));
                 finish();
              } else {
                 // Signout failed
              }
            }
         });
 }
```

4.1.3 Identification par Gmail/Facebook

Firebase Authentication propose des modes de connexion avec les identifiants des réseaux sociaux. Cela se passe principalement dans la console « Firebase > Authentication > Mode de connexion ».

Exercice 4.7. *Modification des paramètres du projet Firebase*

1. *Activer le mode de connexion par Google, ajouter une adresse mail.*
2. *Suivre le lien proposé dans le texte « dans vos applications Android, ajoutez l'empreinte SHA1 à chaque application ».*
3. *Télécharger le fichier* `google-services.json` *et remplacer celui qui est enregistré dans le dossier* `app` *par celui-là.*
4. *Dans l'écouteur de clic du bouton de connexion, remplacer le code*

```
startActivityForResult(AuthUI.getInstance().createSignInIntentBuilder()
                              .build(),
                       RC_SIGN_IN);
```

par

```
List<AuthUI.IdpConfig> selectedProviders = new ArrayList<>();
selectedProviders.add(new AuthUI.IdpConfig.EmailBuilder().build());
selectedProviders.add(new AuthUI.IdpConfig.GoogleBuilder().build());

startActivityForResult(AuthUI.getInstance().createSignInIntentBuilder()
```

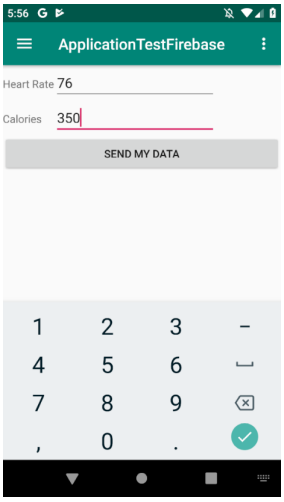

FIGURE 4.2 – Fragment pour envoyer des données personnelles

```
                .setIsSmartLockEnabled(true)
                .setAvailableProviders(selectedProviders)
                .build(),
        RC_SIGN_IN);
```

qui intègre les comptes Gmail. Constater le bon fonctionnement de cette nouvelle méthode de connexion.

4.2 Bases de données NoSQL : Realtime Firebase

La base de données Firebase Realtime Database est une base de données NoSQL dans le Cloud qui synchronise les données de tous les clients en temps réel et offre des fonctionnalités hors ligne. Les données sont stockées dans la base de données en temps réel en tant que fichiers JSON. Tous les clients connectés partagent la même instance de cette base, reçoivent automatiquement les mises à jour avec les données les plus récentes.

D'un point de vue programmation, l'utilisation d'une base de données Realtime Firebase est très différente de celle avec les bases de données relationnelles comme MySQL ou SQLite. En premier lieu, les données doivent être structurées complètement différemment :sous la forme d'un fichier JSON au lieu des tables et jointures traditionnelles.

Commencez par récupérer l'archive `ApplicationTestFirebaseAuthentificationFinie.zip` et reconstruisez le projet Android.

Dans un premier temps, n'importe qui pourra lire et écrire des données dans la base. Ensuite, seules les personnes authentifiées dans l'application grâce à la démarche présentée au chapitre précédent pourront effectuer ces actions.

4.2.1 Création de la base et gestion des droits

Exercice 4.8. *Le fragment* `SendPersonnalDataFragment` *des données à envoyer.*

1. *Demander à Studio de créer le nouveau fragement vide* `SendPersonnalDataFragment`.
2. *Remplacer le libellé "Send" du menu par "Send P. Data" et faire en sorte que le fragment* `SendPersonnalDataFragment` *soit chargé lorsqu'on clique sur cet élément.*
3. *Modifier le gabarit* `fragment_send_personnal_data`*xml+ pour qu'il contienne tous les éléments identifiés à la figure 4.2. Particulièrement :*
   — *l'identifiant de l'*`EditText` *est pour la fréquence cardiaques* `edit_text_hr` *;*
   — *celui pour les calories est* `edit_text_calories` *;*

— *l'identifiant du bouton est* `send_my_data_button`.

4. *Le nom de l'utilisateur connecté ainsi que son identifiant (UId) doit être passé en paramètre lors de la création de ce fragment pour pouvoir être utilisé par la suite.*
5. *Prévoir un écouteur de clic (pour l'instant vide) pour le bouton* `send_my_data_button`.

Exercice 4.9. *Ajouter Realtime Database au projet et gérer les droits.*

1. *En utilisant l'assistant* `Tools>Firebase`*, trouver la section* `Realtime Database` *et choisir (naturellement)* `Save and retrieve data`.
2. *Le projet est déjà connecté à Firebase en raison de l'authentification. Cliquer sur* `Connect to Firebase` *et demander la synchronisation.*
3. *Ajouter* `Realtime Database` *à l'application va modifier les deux fichier gradles. Dans le* `build.gradle` *de l'application, mettre les numéros de version les plus élevés pour les bibliothèques Firebase (qui peuvent être différents) en regardant la page des versions.*
4. *Dans la console web de Firebase, aller dans le projet ApplicationTestFirebase déjà construit à la section précédente, puis à) l'onglet "Database".*
5. *Demander l'ajout d'une base de données de type Realtime et choisir le mode "verrouille" pour la sécurité de la base : personne ne peut lire ou ajouter de données dans celle-ci.*
6. *On modifie les règles pour n'autoriser que les personnes authentifiées à lire et ajouter des données. Cela se fait avec la syntaxe suivante :*

```
{
  "rules": {
    ".read": "auth != null" ,
    ".write": "auth != null"
  }
}
```

7. *On sélectionne une "référence" particulière dans cette base à l'aide de*

```
myRef = database.getReference("Sessions");
```

Si la référence n'existe pas, elle est créée. Ajouter l'attribut `myRef` *de type* `DatabaseReference` *au fragment* `SendPersonnalDataFragment`*. Dans sa méthode* `onCreate`*, ajouter le code :*

```
myRef = database.getReference("Sessions");
myRef.setValue("Un Exemple");
```

Constater dans la console web que la référence "Sessions" a bien été ajoutée dans la base de données avec la valeur "Un exemple". Supprimer ces données dans la console web.

On va maintenant envoyer des données de type à la base dans le Cloud

Exercice 4.10. *Insérer des objets dans la base*

1. *Une session de sport concernera plusieurs sportifs actifs. Commencer par créer les deux classes* `SportSession` *et* `ActiveAthlete` *comme suit et ajouter les getters et setters :*

```
public class ActiveAthlete {
  private String name;
  private int hr;
  private int calories;

  public ActiveAthlete(String firstName, String lastName, int hr, int calories) {
    this.name = name;
    this.hr = hr;
    this.calories = calories;
  }
}

public class SportSession {
  ArrayList<ActiveAthlete> activeAthletes;

  public SportSession(ArrayList<ActiveAthlete> activeAthletes) {
    this.activeAthletes = activeAthletes;
  }
}
```

2. *Créons des sessions de sport comme suit :*
   (a) *Créer une* `ArrayList` *de session sportives* `sss` *et une* `ArrayList` *d'athlètes nommée* `athletes`.
   (b) *Créer deux athlètes actifs et les ajouter à* `athletes`. .
   (c) *Créer une session de sport avec ces* `athletes` *et l'ajouter à la liste des sessions sportives créée en 2a.*
3. *Remplacer* `myRef.setValue("Un Exemple");` *par* `myRef.setValue(sss);` *et constater qu'une arborescence a été créée dans la console web de Firebase.*
4. *Modifier le code précédent pour que les données envoyées soient celles de l'utilisateur connecté (assez difficile).*

Bibliographie

[Mor17] L. Moroney. *The Definitive Guide to Firebase : Build Android Apps on Google's Mobile Platform*. Apress, 2017.