Programmation de mobiles
Interfaces graphiques et domotique sous Android

2SN - Parcours R et T

Quentin Bailleul
Benoît Perroux
Katia Jaffrès-Runser

2023-2024

Brève description Ce sujet vous permettra de mettre en pratique les concepts abordés lors du cours de développement mobile. Il vous permettra d'appréhender les concepts liés au cycle de vie d'une application Android, de manipuler des Activités, des Intents, des Threads ainsi que des Handlers. L'objectif de ce TP est de concevoir une application de contrôle d'une maison connectée.

Déroulement Vous disposez de 2 séances de TP encadrées pour réaliser ce sujet individuellement. Ces séances seront suivies d'un Projet qui réutilise le code développé ici. Nous vous prêtons pendant toute la durée du TP un smartphone Android et un câble USB par personne. Merci d'en prendre soin.

Ressources Vous disposez d'un cours avec toutes les informations (création des Activity, Intent, Threads, Handler, ...) nécessaires pour mener à bien ce sujet. De même, le site principal pour les développeurs d'applications Android est : `https://developer.android.com`. Il vous sera aussi d'une très grande aide.

Test Attention, pour tester votre application, il faudra la déployer sur le smartphone physique fourni (pas d'émulateur). Pour que vous puissiez le faire, il faut que le système Android soit paramétré en mode *debug USB*. Il se peut que votre smartphone le soit déjà, mais si ce n'est pas le cas, il faut réaliser la procédure suivante : Allez dans les paramètres, puis paramètre du système et appuyez 7 fois sur le numéro de build. Il s'affichera ensuite un message pour vous signaler que vous êtes maintenant développeur Android ;-) Ensuite, revenez en arrière, et vous aurez un nouveau menu : Options développeurs. Dans ce dernier, activez le mode Débogage USB.

1 Première activité

Nous allons dans un premier temps créer une première Activité affichant un bouton. L'objectif est d'obtenir une interface proche de celle proposée sur la figure 1. Vous pouvez bien entendu laisser libre cours à votre imagination...

Afin d'atteindre cet objectif, il est nécessaire de créer une nouvelle application Android. Pour cela nous allons utiliser l'environnement de développement officiel pour Android qui est Android Studio. La première étape est la création d'une application Android. *Allez chercher l'assistant de création d'application dans le menu* `Fichier > Nouveau > Nouveau Projet.`

Donner un nom au projet et fournir les différentes informations demandées. Il faut par la suite choisir le SDK minimum supporté par notre application. Souvenez-vous que ce choix est crucial : plus le SDK est récent, plus vous bénéficierez des fonctionnalités avancées des

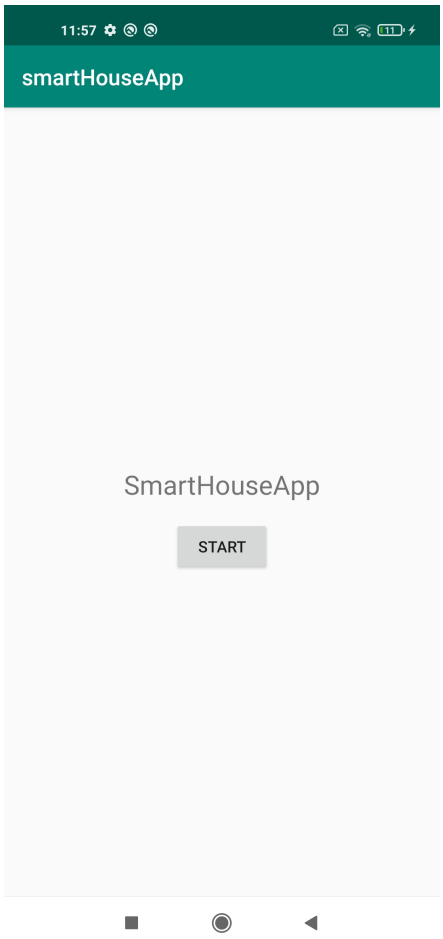

Figure 1: Une première activité

terminaux récents. Par contre, votre application ne sera utilisable que sur des smartphones récents. À l'inverse, si vous choisissez un SDK minimum plus ancien, vous importerez un nombre plus important de terminaux. Il est souvent intéressant de couvrir au moins les deux tiers des terminaux. Nous allons donc choisir l'**API 29** et **JAVA** comme langage de développement.

L'étape finale de la création d'une application est la création d'une activité. Commencer ici avec une activité vide (*Empty Views Activity*) en indiquant ses caractéristiques (nom, ...).

Vous avez maintenant fini la création de votre premier projet et la fenêtre principale d'Android Studio doit ressembler celle proposée dans la figure 2

Nous vous conseillons maintenant de cliquer sur l'onglet `1:Projet` situé dans la colonne gauche de Android Studio. Vous pourrez ainsi voir la structure interne de votre projet. Parcourir la structure de votre projet et identifier les différents éléments en fonction de ce que vous avez vu lors du cours.

À ce stade, vous pouvez connecter le téléphone en USB. Une fois le téléphone détecté par Android Studio, lancez un build et une installation de votre application sur le téléphone à l'aide de la flèche verte (ou du symbole "play" vert), dans le but de vérifier le bon fonction-

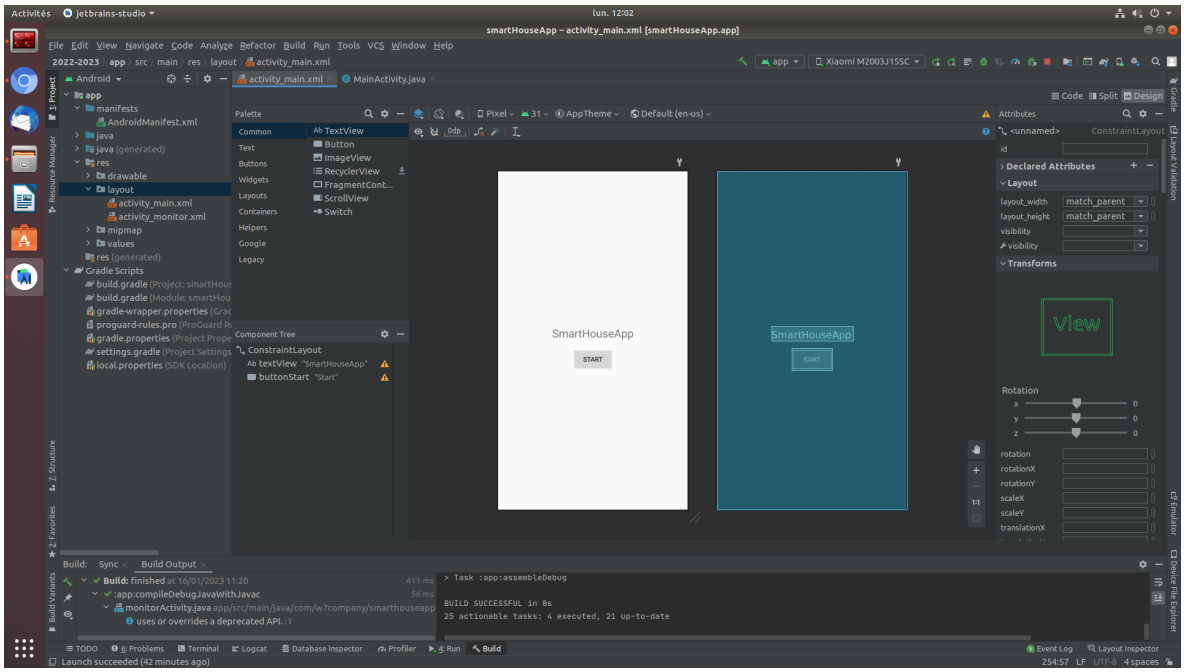

Figure 2: Création de la première activité avec l'outil de design d'Android Studio

nement de la chaîne d'outils.

2 Bouton, création d'Intent et lancement d'une autre Activité

2.1 Manipulation d'interfaces graphiques

La création d'une interface graphique pour une activité passe par la création d'un `Layout`. Un `Layout` est un fichier .xml situé dans le sous dossier `layout` du dossier `res` (ressources) de votre projet. Lorsque vous ouvrez un fichier layout, une interface de composition de l'interface graphique est affichée comme dans la partie droite de la figure 2.

Vous disposez dans cette interface d'une vue graphique du layout, d'une palette proposant les différents widgets et layouts permettant de créer une interface, d'un arbre de composants correspondant à l'instance du layout édité et finalement d'un espace décrivant les différents propriétés de l'objet graphique sélectionné (dans l'arbre ou dans la vue graphique).

Par défaut, un fichier layout contient un objet de type Layout (Contraint, Linéaire, Table ou Relatif) qui est un conteneur à Layout ou à Widgets. Un Layout est donc un bon candidat pour être l'élément racine d'un arbre structurel d'interface graphique.

Ajouter un bouton au milieu de l'activité. Ce bouton servira à lancer une autre activité affichant les différents appareils connectés de la maison. Lors de l'ajout d'un élément dans l'interface (widget ou layout), il est très important (voir vital) d'aller modifier ses propriétés et plus particulièrement la propriété `id`. Cette propriété est utilisée par la suite pour récupérer les éléments graphiques de l'interface dans le code et les manipuler. Dans le cas de la création d'un bouton, une autre propriété est importante : la propriété `text` contenant la chaîne de

caractère affichée par le bouton. Il est possible de fournir la valeur de cette chaîne de caractère directement dans les paramètres mais on préfère utiliser les ressources à cette fin. En effet, dans les ressources, vous trouverez un sous-dossier `values` contenant la définition pour des chaînes de caractères, des couleurs, des tailles, ... Ces éléments sont configurés ici et utilisés dans le reste de l'application afin de simplifier leur modification par la suite sans avoir à modifier des éléments de code un peu partout. *Ajouter une nouvelle chaîne de caractère pour le texte du bouton dans les* `values` *et assigner cette chaîne de caractère à la valeur du champ* `text` *du bouton.*

2.2 Création d'Intent et lancement d'une autre activité

Notre objectif ici est de lancer une nouvelle activité lors du clic de l'utilisateur sur le bouton. Il nous faut donc réaliser les activités suivantes:

- *Création de la nouvelle activité à lancer*
- *Capture du clic sur le bouton*
- *Assignation d'un comportement lors du clic sur le bouton*

Création de la nouvelle activité à lancer Pour créer une nouvelle activité, faire un clic droit sur l'arborescence du projet et sélectionner le sous-menu `Nouveau`. Choisir ensuite le type d'activité que vous souhaitez créer (nous préférerons ici des activités simples telles que l'activité vide). Renseigner les différentes informations demandées.

Capture du clic sur le bouton Afin d'atteindre cet objectif, il nous faut récupérer le bouton pour lequel nous souhaitons capturer les clics. Votre cours vous permettra de trouver comment faire cela. Une fois le bouton récupéré, nous allons capturer un clic sur le bouton par la création d'un listener.

Assignation d'un comportement lors du clic sur le bouton Une fois le listener assigné au bouton, il nous faut y ajouter le code permettant de logger le clic et de lancer la nouvelle activité. Une fois encore, votre cours contient les informations nécessaires.

Déploiement de l'application Déployer votre application sur le téléphone Android. *Observer les traces de l'exécution qui apparaissent dans la fenêtre Logcat d'Android Studio (cf. Figure 3) relatives à l'application que vous avez créée.*

3 Affichage des appareils connectés de la maison

Nous pouvons désormais passer à la modification de notre seconde activité. Nous allons commencer par créer les différents éléments graphiques et par la suite nous y ajouterons des éléments de comportement. Nous proposons d'obtenir un affichage des appareils connectés de la maison ayant la forme proposée dans la figure 4.

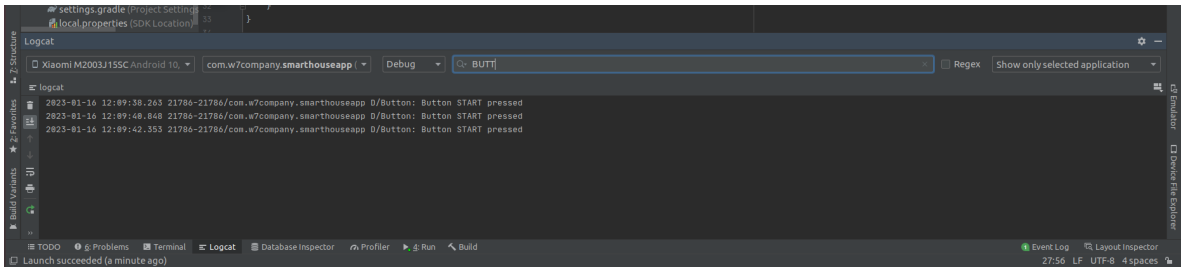

Figure 3: Logs de l'exécution de l'application

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_monitoring);
}
```

Listing 1: Méthode principale d'une activité

3.1 Préparation de l'interface graphique

L'objectif de cette sous-section est de créer la brique de base (Fig. 5) de notre affichage.

Nous proposons ici d'ajouter au layout principal un widget de type `ScrollView`. Ce widget est un conteneur avec une barre de défilement. Afin de pouvoir peupler cette ScrollView avec la liste des appareils connectés, ajouter un LinearLayout dans la ScrollView. Pour chacun de ces éléments, penser à définir la propriété `id`. Ce dernier LinearLayout devra être peuplé avec un nombre d'éléments dépendant du nombre d'appareils connectés de notre maison, il n'est donc pas possible de créer les widgets statiquement. Nous arrêterons donc ici l'ajout d'éléments d'interface à la main et ajouterons maintenant les différents widgets par programmation.

Dans la méthode principale de notre nouvelle activité, nous avons normalement quelques lignes de code générées automatiquement (cf. listing 1). Nous allons ici récupérer notre layout que nous pourrons par la suite peupler avec les informations des appareils connectés. Créer une variable privée globale à la classe et lui assigner la valeur du LinearLayout crée précédemment. On utilisera pour cela la méthode `findViewById`.

Créer par la suite une méthode java retournant un élément de type `View` et prenant en paramètre deux chaînes de caractères (Le nom/modèle de l'appareil et ses informations) et un booléen (qui reflète l'état On ou Off de l'appareil). Nous pourrons par exemple appeler cette méthode `createDeviceView`. Un appel à cette méthode va donc créer un élément graphique contenant les informations de l'appareil connecté. Nous proposons que cet élément View ait la forme très simpliste proposée dans la figure 5.

Il nous faut donc créer une vue contenant deux éléments textes (`TextView`) et un bouton. Les éléments textes sont alignées à gauche et le bouton est aligné à droite. Une bonne solution pour afficher des widgets et les placer spécifiquement à un endroit prédéterminé est de créer un Layout de type `RelativeLayout` et d'y insérer nos différents widgets. Nous vous proposons un exemple de code dans le listing 2 permettant de créer, insérer et positionner des vues dans un layout de type relatif. À vous de l'adapter pour composer votre affichage pour les

```
RelativeLayout layout = new RelativeLayout(this);

RelativeLayout.LayoutParams paramsTopLeft =
  new RelativeLayout.LayoutParams(
    RelativeLayout.LayoutParams.WRAP_CONTENT,
    RelativeLayout.LayoutParams.WRAP_CONTENT);
paramsTopLeft.addRule(RelativeLayout.ALIGN_PARENT_LEFT,
                      RelativeLayout.TRUE);
paramsTopLeft.addRule(RelativeLayout.ALIGN_PARENT_TOP,
                      RelativeLayout.TRUE);

TextView someTextView = new TextView(this);
someTextView.setText("Un certain texte");
layout.addView(someTextView, paramsTopLeft);

return layout;
```

Listing 2: Création de View par programmation

```
<application
    android:usesCleartextTraffic="true"
```

Listing 3: Création de View par programmation

informations des appareils connectés.

Modifier le code de la méthode `createDeviceView` *pour afficher les informations des appareils connectés selon le layout demandé.*

3.2 Récupération des informations relatives aux appareils connectés

Une fois l'affichage mis en place, il nous faut maintenant peupler les widgets avec les informations relatives aux appareils connectés. Nous devons donc récupérer le nom, le modèle, l'état, les données et les autres informations de nos appareils connectés. N'ayant pas de maison connectée à mettre à disposition, nous allons utiliser un simulateur de maisons connectées. Ce simulateur est contrôlable à l'aide d'une API REST (comme certains objets connectés). Une API REST est une interface de programmation d'application utilisant le protocole HTTP pour effectuer les échanges. Cet API est accessible avec l'URL suivante : `http://happyresto.enseeiht.fr/smartHouse/api/v1/`.

ATTENTION : Il faut ajouter le paramètre `android:usesCleartextTraffic="true"` à la balise application dans votre Manifest pour interroger un serveur non HTTPS (cf Listing 3) :

Si vous n'êtes pas sur le réseau WiFi de l'N7 (wifinp), il faut se connecter à un VPN.

La première fonctionnalité que nous allons utiliser est les requêtes GET pour récupérer l'ensemble des appareils d'une maison à l'aide de l'url suivante $\langle url \rangle / devices / \langle idmaison \rangle$ et les informations d'un appareil précis à l'aide de l'url suivante $\langle url \rangle / devices / \langle idmaison \rangle / \langle idappareil \rangle$. L'identifiant de la maison est unique par étudiant et vous sera fourni par votre encadrant.

```
dependencies {
    ...
    implementation 'com.android.volley:volley:1.2.1'
    ...
}
```

Listing 4: Dépendance dans le fichier build.graddle

Pour envoyer nos commandes et recevoir nos réponses, nous allons utiliser la librairie Android Volley (`https://google.github.io/volley/`). Pour débugger, vous pouvez aussi effectuer les requêtes GET à l'aide d'un navigateur web et les requêtes POST à l'aide de la commande CURL.

Commencer par ajouter la librairie aux dépendances de votre application (listing 4) et la permission `<uses-permission android:name="android.permission.INTERNET" />` au Manifest de votre application. Pour envoyer une simple requête, Volley utilise un objet `RequestQueue` qui est initialisé avec la méthode Volley.newRequestQueue(Context). Un objet `Request` peut être passé à cette `RequestQueue`. En cas de réponse (ou en cas d'erreur), un listener est déclenché pour traiter l'évènement. Un exemple simple est décrit dans le listing 5. Cet exemple utilise une `Request` de type `JsonObjectRequest` car notre API retourne des objets JSON. Volley supporte d'autre type de requête comme les `JsonArrayRequest`, les `StringRequest` et les requêtes custom.

Comprendre et expérimenter avec cet exemple simple. Puis implémenter une requête `jsonArrayRequest` pour récupérer l'ensemble des devices de la maison connectée et peupler la `ScrollView`.

Remarque Une requête web peut prendre du temps avant d'obtenir une réponse. Dans ce genre de cas, il faudrait utiliser un Thread pour ne pas bloquer l'exécution du thread principal et freezer notre application. Dans notre cas, nous n'avons pas besoin d'instancier un Thread car la librairie Android Volley s'en occupe.

4 Envoi de commandes

L'API nous permet aussi d'allumer ou éteindre un appareil connecté à l'aide d'une requête POST avec les paramètres suivant : $deviceId = \langle idAppareil \rangle$, $houseId = \langle idMaison \rangle$ et $action = turnOnOff$ et à l'url suivant : $\langle url \rangle /devices/$. Le listing 6 décrit comment réaliser une requête POST avec Android Volley.

Ajouter un listener sur le bouton de chaque appareil connecté qui permet d'envoyer une requête POST pour l'allumer ou l'éteindre. Après réception de la réponse, actualisez la `View` de cet appareil.

5 Monitoring régulier

En plus de l'utilisation vue en cours, le `Handler` permet aussi de retarder l'exécution d'un `Runnable` à l'aide de la méthode postDelayed(runnable, delayMs). Il est donc possible de

```
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_monitor);

    RequestQueue queue = Volley.newRequestQueue(this);
    JSONObjectRequest jsonObjectRequest = new JsonObjectRequest
        (Request.Method.GET, url, null,
        requestSuccessListener(),requestErrorListener());
    queue.add(jsonObjectRequest);
}

private Response.Listener<JSONObject > requestSuccessListener() {
    return new Response.Listener<JSONObject >() {
        @Override
        public void onResponse(JSONObject  response) {
           Log.d(TAG,response.toString());
        }
    };
}

private Response.ErrorListener volleyErrorListener() {
    return new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e(TAG,error.getMessage());
        }
    };
}
```

Listing 5: Exemple simple de mise en place d'Android Volley pour des JsonObjectRequest - Note : il faut importer `com.android.volley.*`

```
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_monitor);

    RequestQueue queue = Volley.newRequestQueue(this);

    StringRequest sr = new StringRequest(Request.Method.POST, url,
        requestSuccessListener(), volleyErrorListener()){
        @Override
        protected Map<String,String> getParams(){
            Map<String,String> params = new HashMap<String, String>();
            params.put("deviceId", String.valueOf(deviceId));

            return params;
        }

        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            Map<String,String> params = new HashMap<String, String>();
            params.put("Content-Type",
                "application/x-www-form-urlencoded");
            return params;
        }
    };
    queue.add(sr);
}
```

Listing 6: Exemple de requête POST avec StringRequest

```
Handler handler = new Handler();
Runnable runnableCode = new Runnable() {
    @Override
    public void run() {
      Log.d(TAG, "Appel periodique");
      handler.postDelayed(this, 10000);
    }
};
// Lance la premiere execution
handler.post(runnableCode);

// Arrete les executions periodiques du Runnable
handler.removeCallbacks(runnableCode);
```

Listing 7: Exemple d'exécution périodique avec un `Handler`

l'utiliser pour exécuter périodiquement une tâche comme illustré dans le listing 7.

En s'inspirant de ce listing, implémentez la mise à jour périodique des informations des appareils connectés.

6 Gestion des interruptions de l'application et du cycle de vie

Le système Android est le seul maître des ressources du système. Il est donc possible que notre activité se fasse interrompre à tout moment. C'est notamment le cas quand elle n'est plus visible.

Il serait donc bien de s'assurer que lorsque notre application est interrompue, tous les threads et tâches périodiques en cours d'exécution sont proprement arrêtés.
Modifier le code de l'application en conséquence.

Finalement, nous souhaiterions que lorsque l'activité affichant les informations des appareils connectés est résumé (revient au premier plan) la liste des applications soit mise à jour. *Modifier le code de l'application en conséquence.*

Figure 4: Une possibilité pour l'affichage des informations relatives aux appareils connectés de la maison

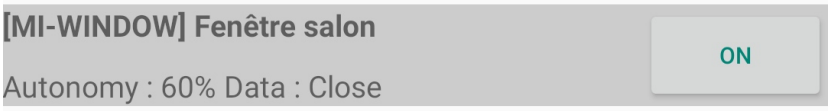

Figure 5: La vue affichant les informations pour un appareil connecté