Développement d'applications Android
Laboratoire n°1

Installation et prise en main de l'IDE

Introduction

Ce premier laboratoire est constitué de plusieurs manipulations destinées à réaliser une application mobile élémentaire sur un émulateur et/ou sur un smartphone *Android*, dans le but de vous familiariser avec l'environnement (IDE et SDK) de développement *Android*.

Installation de l'environnement de développement

Avant de commencer l'installation et la réalisation de ce laboratoire, nous vous demandons de prendre connaissance des règles et des informations générales concernant les laboratoires. Vous trouverez le document les regroupant sur la page Cyberlearn du cours.

Manipulations

1. Installation de l'IDE et création d'un nouveau projet

Téléchargez *Android Studio* et suivez les instructions d'installation sur la page du site *Android* https://developer.android.com/studio/index.html. Il est également possible, et même recommandé si vous l'avez déjà sur votre ordinateur, de passer par la *JetBrains Toolbox* pour installer *Android Studio*.

Une fois l'installation effectuée, veuillez télécharger le projet vide de base mis à disposition sur Cyberlearn dans la section « Laboratoires ». Vous pourrez partir depuis cette base pour les prochains laboratoires également si vous le souhaitez.

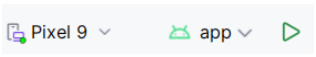
Ce projet est en réalité un projet de type **Empty Views Activity** auquel nous avons tout simplement ajusté et documenté quelques éléments mineurs afin de simplifier l'apprentissage à ce stade.

L'IDE va à présent prendre quelques instants pour configurer le projet, télécharger les dépendances et indexer son contenu, veuillez attendre que cette étape se termine.

Un émulateur est déjà proposé par défaut. Si vous souhaitez cependant ajouter un nouvel émulateur pour des besoins spécifiques, veuillez suivre le paragraphe suivant, pour ce laboratoire vous pouvez l'ignorer.

La dernière étape de cette manipulation est l'installation d'un émulateur, veuillez cliquer sur  et créer un nouvel émulateur, par exemple un Pixel 9 (avec le Play Store) avec une image du système en

version 36 (l'actuelle). Veuillez ensuite démarrer l'émulateur, le premier démarrage peut prendre quelques (longues) minutes.

Pour vérifier le bon déroulement des étapes précédentes, nous allons *builder* l'application nouvellement créée et la lancer sur l'émulateur. Pour cela cliquez sur la flèche verte : 

L'application devrait apparaitre sur l'émulateur, telle que sur la Fig. 1.

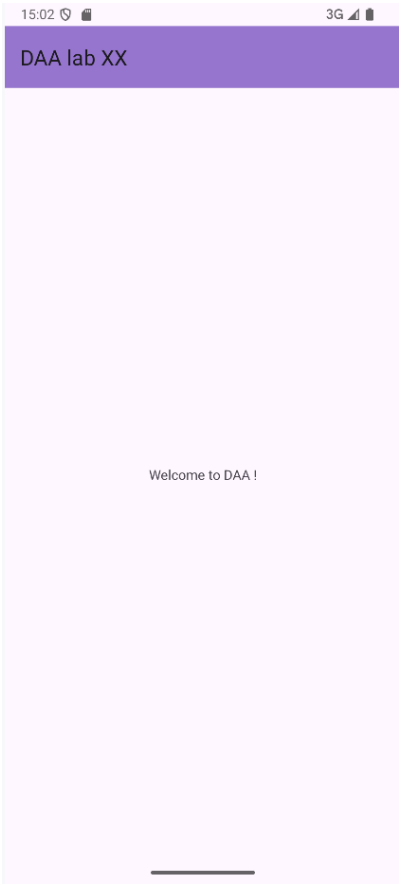


Figure 1 L'application par défaut démarrée sur l'émulateur

2. Fichiers gradle de build

Le projet contient deux scripts *gradle* pour le *build* de l'application : un pour le « project » et un autre pour le « module app ». Après avoir buildé/lancé une fois votre code, si vous ouvrez le fichier *gradle* du *module app*, nous constatons en bas de celui-ci dans la section *dependancies*, qui contient la liste des librairies utilisées dans notre projet, il s'agit de références vers les librairies définies dans le catalogue (fichier *libs.versions.toml*). En ouvrant le catalogue, il est possible que vous constatiez que plusieurs lignes sont surlignées en jaune. Ces *warnings* sont générés automatiquement par *lint*, le préprocesseur intégré à l'IDE qui permet de vérifier le code et la mise en œuvre des bonnes pratiques directement dans l'IDE. Ici, il nous indique que nous n'utilisons pas les dernières versions de certaines librairies, veuillez appliquer les corrections proposées. Une fois les changements de versions appliqués, vous constaterez un bandeau qui apparait en haut du fichier vous proposant de synchroniser *gradle* et ainsi d'appliquer les modifications. Vous avez la possibilité en tout temps de forcer la synchronisation de *gradle* à l'aide du bouton  .

3. Modification du *layout*

Dans le projet nous trouvons plusieurs éléments, nous allons en particulier nous intéresser à l'activité principale gérée par le fichier *MainActivity.kt* situé dans le dossier *java* (malgré ce nom un peu trompeur, elle est bien écrite en *Kotlin)*.

Dans ce fichier nous trouvons en particulier la ligne `setContentView(R.layout.activity_main)` qui spécifie quel fichier *layout* ressource doit être chargé comme interface utilisateur, ici c'est le fichier *activity_main.xml* situé dans le dossier *res/layout*.

Si nous ouvrons ce *layout*, nous constatons qu'il est composé d'un *ConstraintLayout* occupant tout l'espace à disposition. A l'intérieur de celui-ci, nous retrouvons un *AppBarLayout* contenant une *ToolBar* chargée d'afficher la barre d'action en haut. Un second *ConstraintLayout* est présent juste en-dessous et contient l'ensemble des vues (widgets) à afficher. En l'occurrence, il n'y a qu'une seule vue, centrée, de type *TextView* avec le texte « Welcome to DAA ! ».

Dans la suite de cette manipulation nous allons travailler avec les *layouts*. Pour cela il vous est conseillé de créer de nouveaux fichiers *layouts* dans le dossier *res/layout* puis de changer la référence utilisée dans le code de l'activité existante (il s'agit du fichier *Kotlin* cité plus haut). Ceci afin de pouvoir conserver le *layout* d'origine comme référence et travailler directement avec l'activité fournie.

Pour rappel, afin de créer un *layout* facilement il vous suffit d'effectuer un clic droit sur le dossier layout, sélectionner l'option *New >> Layout Resource File*, puis de choisir l'élément racine adapté (*LinearLayout*, *RelativeLayout*, *ConstraintLayout*, etc…).

Vous pouvez par exemple créer le fichier *activity_main_linear.xml* avec le contenu ci-dessous, changer la référence au niveau du code de l'activité vers ce nouveau *layout*, puis relancer l'application.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/main"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!-- Barre d'action -->
    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appbarlayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent">
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary" />
    </com.google.android.material.appbar.AppBarLayout>

    <!-- Contenu -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

L'éditeur de *layout* disponible dans l'IDE permet de prévisualiser celui-ci avec les options en haut à droite :  , mais celui-ci n'est pas toujours fidèle, l'émulateur permet de s'assurer du rendu correct du *layout*.

Barre d'action : attention, n'oubliez pas pour chaque nouveau *layout* d'y insérer, comme dans l'exemple ci-dessus, un *AppBarLayout* comme tout premier widget afin de retrouver la barre d'action. Afin que cette fonctionnalité se comporte correctement, veillez également à ne pas omettre l'identifiant au niveau du *layout* comme indiqué sur l'exemple fourni.

4. Le LinearLayout

Nous souhaitons dans cette manipulation adapter ce nouveau *layout* afin de retrouver la structure représentée dans la Fig. 2.

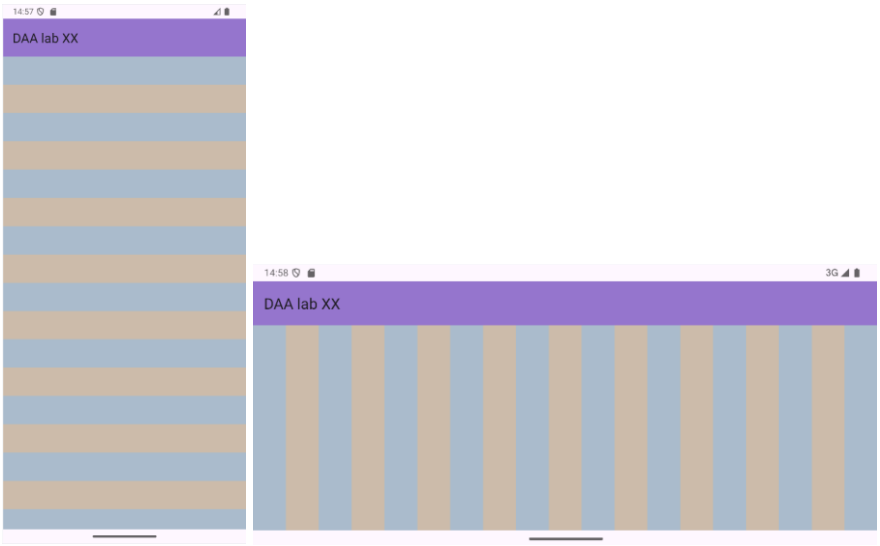


Figure 2 - Etat que l'on souhaite obtenir à l'aide d'un LinearLayout

Pour cela veuillez définir deux nouvelles ressources de type couleur : `mycolor1` et `mycolor2`. Veuillez également définir une ressource de type dimension nommée `finger_size` ayant pour valeur `48dp`. Les bonnes pratiques demandent à ce que les éléments « touchables » par un utilisateur aient une taille minimum de 48dp.

Dans le *LinearLayout* de la manipulation précédente, veuillez insérer une vingtaine (afin de remplir entièrement l'écran) de vues enfants en suivant l'exemple ci-dessous et en alternant entre les deux couleurs :

```
<View
    android:layout_width="match_parent"
    android:layout_height="@dimen/finger_size"
    android:background="@color/mycolor1" />
```

Veuillez discuter des points suivants :

1. Que se passe-t-il en cas de rotation de l'écran vers le mode paysage ? Observez-vous le résultat attendu ?
2. Que se passe-t-il au bord inférieur de l'écran ? Constatez-vous un comportement inattendu ?

Après avoir constaté les comportements étranges cités ci-dessus, nous vous proposons d'effectuer les modifications suivantes, qui devraient corriger vos problèmes :

1. Créer un *layout* spécialisé pour le mode *paysage*. (Faites également attention à gérer la barre d'action). Quelles sont les adaptations à effectuer et les fichiers à ajouter ?
2. Mettre en place une *ScrollView*. Ceci devrait permettre de gérer les vues dépassant les limites de l'écran. Attention à bien gérer la variante horizontale, ceci devrait nécessiter une *view* particulière.

5. Le RelativeLayout

Nous allons créer un nouveau fichier de *layout*: *activity_main_relative.xml* qui possède cette fois ci un *RelativeLayout* comme élément racine (attention de nouveau à la barre d'action). Nous souhaitons réaliser une fenêtre de login, composée d'un titre, de deux champs de saisie avec leur label et d'un bouton. Tous les éléments doivent être centrés. La Fig. 3 présente le résultat attendu, sur cette capture d'écran nous avons mis des *backgrounds* colorés aux *TextViews* pour les distinguer. A vous de faire parler vos talents artistiques et choisir la combinaison de couleurs qui vous plaise le mieux.

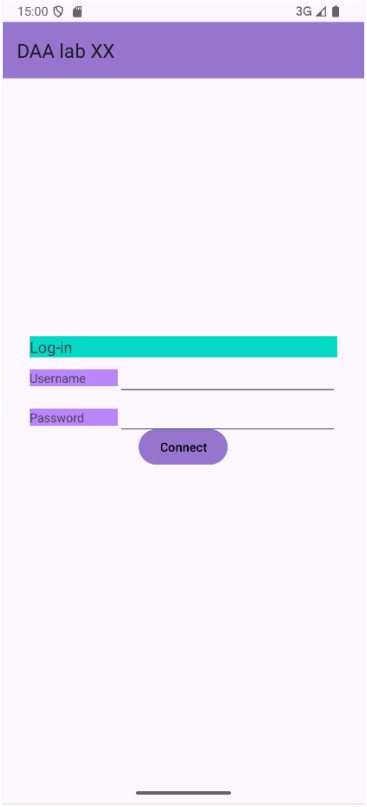


Figure 3 - Interface de login à réaliser

Veuillez discuter des problèmes suivants et adapter par la suite votre code :

1. Comment aligner au mieux une *TextView* et un *EditText* qui n'ont pas la même hauteur ? (Hint : `android:layout_alignBaseline`)
2. Comment est-ce que votre layout se comporte en mode *paysage* ? A l'affichage ? Lors de la saisie ? Est-ce que des adaptations sont nécessaires ?

6. Le Nine-Patch

Nous souhaitons mettre en évidence la zone de login de la manipulation précédente avec l'image *bg.9.png* comme fond, cf. Fig. 4. Veuillez définir les zones d'extension et de contenu dans l'image fournie, selon l'approche nine-patch, vous ferez attention aux parties ombragées.

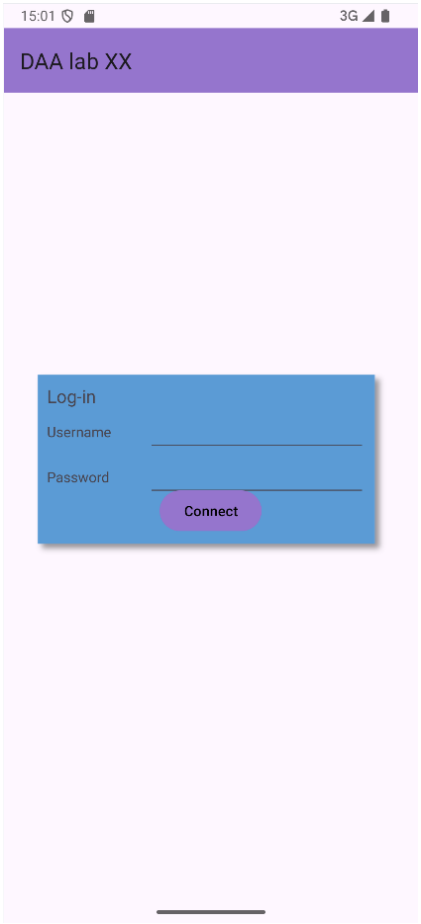


Figure 4 - Mise en évidence de la zone de login

7. Gestion d'un événement utilisateur

Comment pouvons-nous enregistrer une méthode callback qui sera appelée lorsque l'utilisateur appuie sur le bouton *Connect* ? Essayez d'afficher un Toast en réponse à cet événement.

Développement sur un smartphone physique

Nous pouvons vous prêter, durant les périodes de laboratoire, un smartphone *Android* qui vous permettra de tester votre application sur une cible physique.

Il est également possible d'utiliser votre propre smartphone *Android*, pour cela vous devrez activer le mode développeur. Dans les paramètres du smartphone, il faudra cliquer plusieurs fois rapidement sur le numéro de *build* de votre appareil, un pop-up indiquera ensuite que le mode développeur est activé. Un nouveau menu « Options pour les développeurs » apparait dans les paramètres, et dans celui-ci vous devrez activer le « Debogage USB ». Il est conseillé de désactiver le « Debogage USB » lorsque vous ne vous en servez pas.

Si vous ne disposez pas de smartphone *Android* et que votre ordinateur a de la peine à faire tourner l'émulateur, veuillez nous informer, il sera certainement possible de vous prêter un smartphone durant la durée du semestre.

Durée / Evaluation

- 4 périodes
- Ce laboratoire n'est pas à rendre et n'est pas noté, un corrigé vous sera fourni.