Introduction

Dans ce cours nous allons apprendre des développer des applications Android d'une façon moderne en utilisant Android JetPack.

Android JetPack

Android JetPack est un ensemble de librairies modernes mise à disposition par Google en plus du framework Android qui permettent d'augmenter le confort de développement et la qualité des applications pour les besoins usuels.

Installer Android Studio

Avec JetBrains Toolbox

Jetbrains Toolbox est un petit utilitaire très pratique qui permet d'installer et mettre à jours les logiciels JetBrains en un clic.

Installer Jetbrains Toolbox

Téléchargez et installez Jetbrains Toolbox

Installer IntelliJ

Exécutez JetBrains Toolbox puis faites un clic droit sur son icône dans la barre d'état système de Windows. Trouvez ensuite IntelliJ et Datagrip dans la liste des applications proposée. Cliquez sur "*Install*" et attendez la fin du téléchargement.

Avec Chocolatey

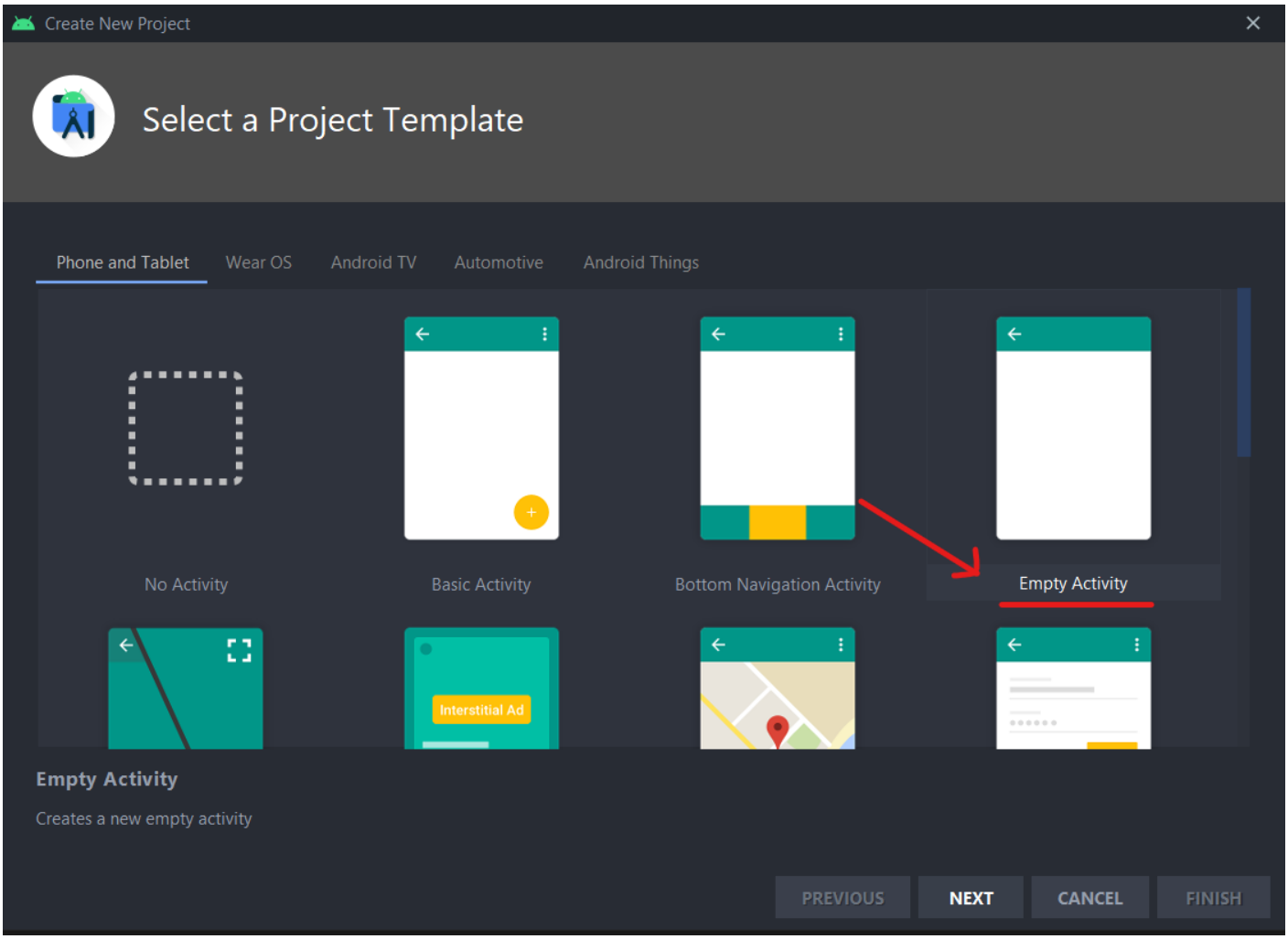
```
choco install AndroidStudio -y
```

Sans Jetbrains Toolbox ni Chocolatey

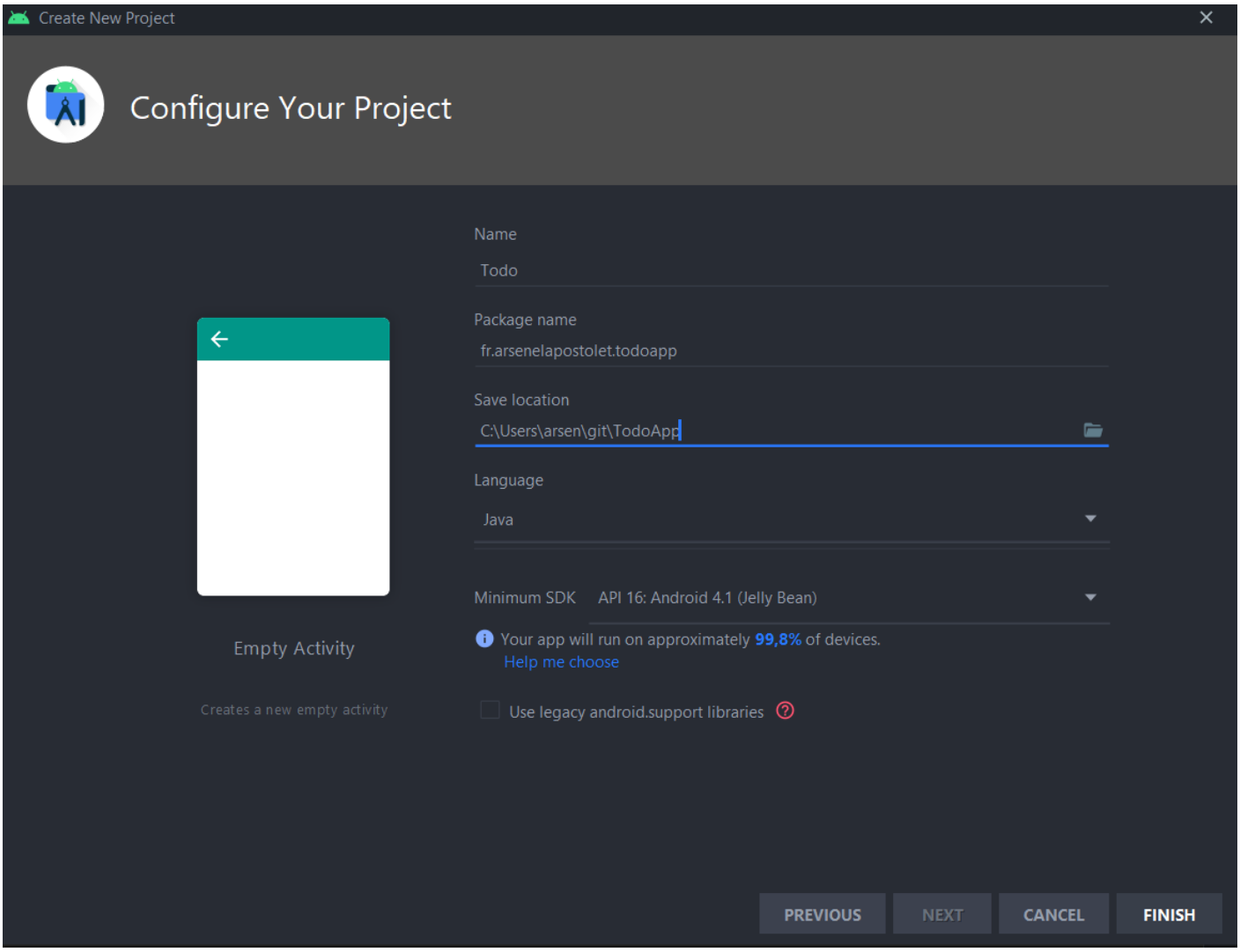
Téléchargez et installer via ce lien.

Créer un projet avec Android Studio

Lancez Android Studio et créez un nouveau projet. Séléctionnez "Empty Activity" comme template de projet :

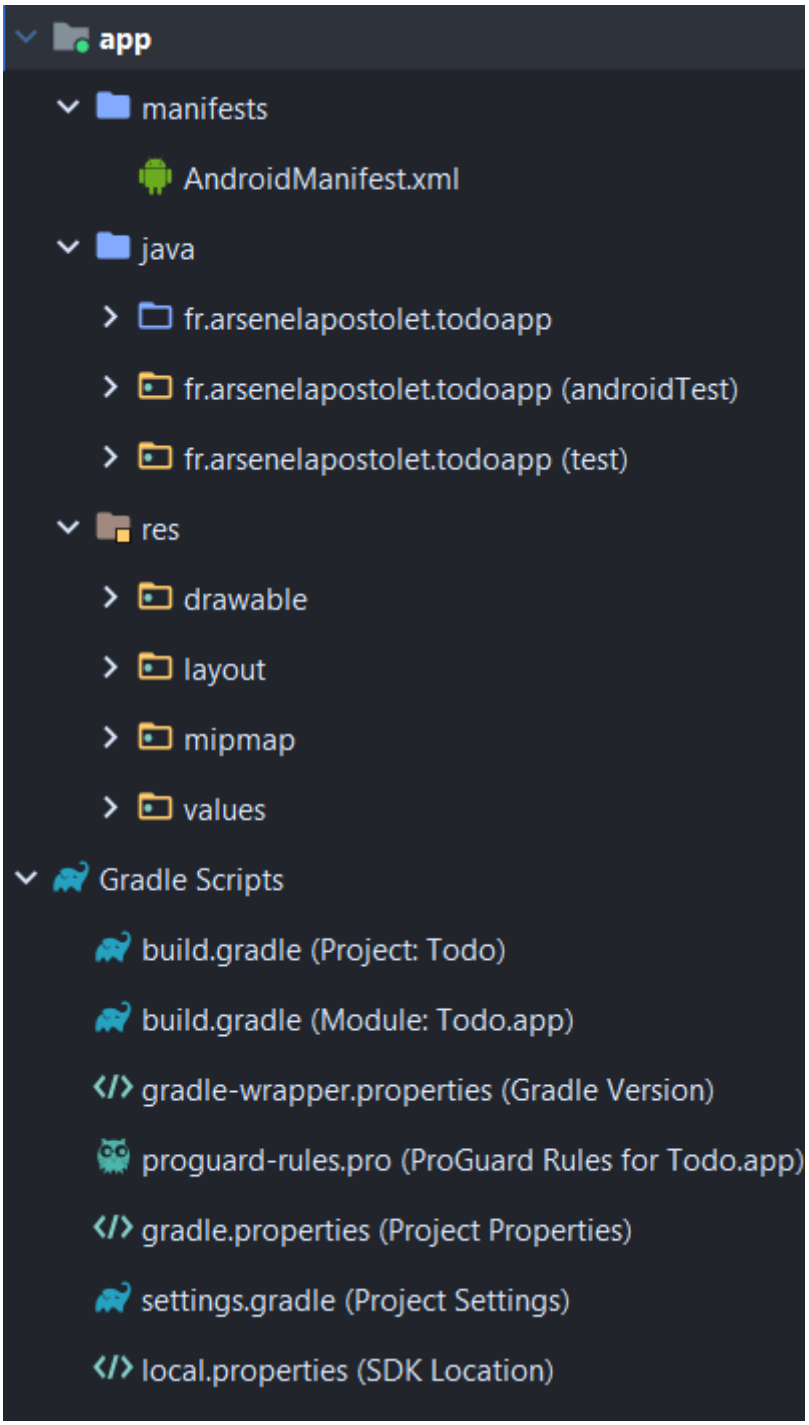


Remplissez ensuite les caractéristiques de votre projet :



Structure du projet

La structure de notre projet est la suivante :



On retrouve tout d'abord le fichier `AndroidManifest.xml` qui va contenir toutes les méta données de l'application, comme par exemple son nom son icone, ses permissions, etc ...

Ensuite dans le dossier `java` on va retrouver nos codes source Java. Ensuite dans le dossier `res` on va trouver toutes les ressources et assets de l'application :

- `drawable` : Images
- `layout` : Templates XML pour les vues
- `mipmap` : Icônes
- `values` :
  - `colors.xml` : Constantes de couleurs
  - `strings.xml` : Constantes chaines de caractère

- `themes` : Configurations des couleurs du thèmes

Enfin, on a deux fichiers `build.gradle` qui permettent de configurer les dépendances et la génération de l'application.

Configurer les bibliothèques JetPack

Pour ajouter les librairies de Android JetPack, nous allons justement utiliser ces fichiers `build.gradle`.

On va commencer par ajouter les versions des librairies que l'on utiliser dans des variables :

```
def lifecycleExtensionVersion = '1.1.1'
def butterKnifeVersion = '10.1.0'
def supportVersion = '28.0.0'
def retrofitVersion = '2.3.0'
def glideVersion = '4.9.0'
def rxJavaVersion = '2.1.1'
def roomVersion = '2.1.0-rc01'
def navVersion = '2.1.0-alpha05'
def preferencesVersion = '1.1.0'
```

Juste avante l'object `dependencies`.

Maintenant nous allons rajouter les déclarations dans `dependencies` pour réellement importer les librairies :

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

    // Libraries imports

    // Android
    implementation "com.android.support:design:$supportVersion"
    implementation "android.arch.lifecycle:extensions:$lifecycleExtensionVersion"

    // Butterknife
    implementation "com.jakewharton:butterknife:$butterKnifeVersion"
```

```
    annotationProcessor "com.jakewharton:butterknife-compiler:$butterKnifeVersion"

    // Room
    implementation "androidx.room:room-runtime:$roomVersion"
    implementation "androidx.legacy:legacy-support-v4:1.0.0"
    annotationProcessor "androidx.room:room-compiler:$roomVersion"

    // Navigation
    implementation "androidx.navigation:navigation-fragment:$navVersion"
    implementation "androidx.navigation:navigation-ui:$navVersion"

    // Material
    implementation "com.google.android.material:material:1.2.1"

    // Retrofit
    implementation "com.squareup.retrofit2:retrofit:$retrofitVersion"
    implementation "com.squareup.retrofit2:converter-gson:$retrofitVersion"
    implementation "com.squareup.retrofit2:adapter-rxjava2:$retrofitVersion"

    // RxJava
    implementation "io.reactivex.rxjava2:rxjava:$rxJavaVersion"
    implementation "io.reactivex.rxjava2:rxandroid:$rxJavaVersion"

    // Glide
    implementation "com.github.bumptech.glide:glide:$glideVersion"

    // Palette
    implementation "com.android.support:palette-v7:$supportVersion"

    // Preferences
    implementation "androidx.preference:preference:$preferencesVersion"



    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Ensuite pour configurer le plugin de génération de code pour la navigation, rajoutez la ligne :

```
plugins {
    ...
    id 'androidx.navigation.safeargs'
}
```

dans l'objet `plugins` tout en haut du fichier. Ensuite, afin de pouvoir utiliser des fonctionnalités récentes de Java comme les streams et lambdas, nous allons ajouter l'objet suivant dans l'objet `android` :

```
android {

    ...

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    dataBinding.enabled = true
}
```

Voilà, c'est tout pour le `build.gradle` de l'app. Maintenez, ouvrez le fichier `build.gradle` du projet, et ajoutez la ligne suivante dans `dependancies` :

```
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:2.1.0-alpha05"
```

Voilà ! C'est tout pour la configuration Gradle. Maintenant vous pouvez faire `File > Sync Project with Gradle` afin d'impacter la configuration gradle sur le projet.

La configuration des dépendances votre projet est maintenant terminée.

Structure du Code

Nous allons structurer notre code en plusieurs packages pour l'organiser plus clairement :

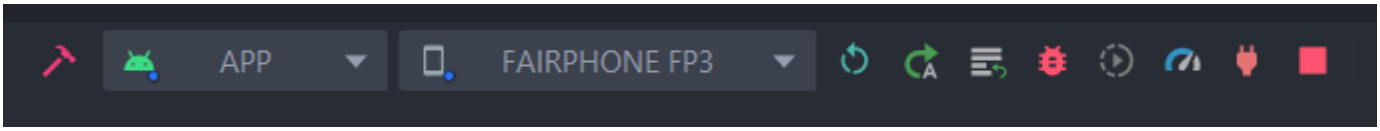
- `view` : les Activités et les Fragments, qui vont décrire la logique propre à l'interface utilisateur.
- `model` : le code qui corrspond à la logique métier de l'application
- `util` : les classes utilitaires

Vous pouvez donc déplacer votre classe `MainActivity` dans le package `view`.

Conseils pour le débogage

Je trouve que l'émulateur Android est pénible à utiliser : il prend beaucoup de ressource, ce qui rend le développement difficile sur des petites configs, et il bug souvent (sur une machine il me faisait des bluescreens systématiques au bout de qq minutes). C'est pourquoi j'utilise mon appareil physique pour le débogage. Cependant, on peut trouver ça pénible de devoir prendre ce dernier à chaque lancement de son appli et de devoir faire les manip sur le téléphone physique.

C'est pourquoi j'utilise ScrCpy pour controler mon téléphone depuis mon ordinateur. C'est un petit outil open source très performant, et simple à setup en USB. Branchez simplement votre téléphone à votre PC par USB et activer le débogage USB dessus. En faisant ça votre téléphone devrait être automatique détécté par Android Studio comme cible de déploiement :

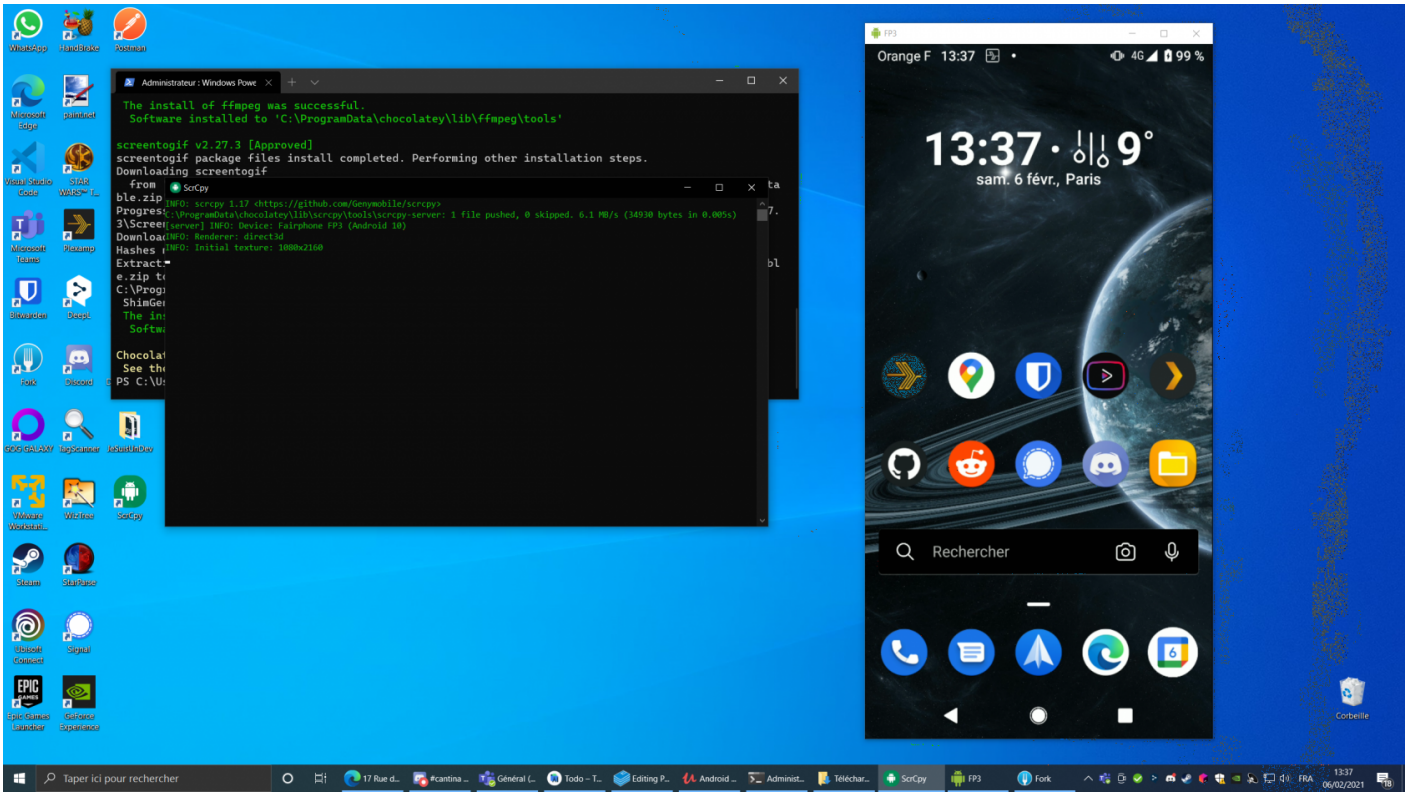


Ensuite, installez ScrCpy soit via Chocolatey :

```
choco install scrcpy -y
```

ou alors de télécharger le ZIP sur GitHub.

Une fois installé, lancez tout simplement pour ce résultat :



Vous pouvez interagir avec votre téléphone à l'écran avec votre souris et clavier aussi simplement qu'avec l'émulateur !

Revision #9
Created 2021-01-31 14:42:31 UTC by Arsène Lapostolet
Updated 2021-02-07 15:24:04 UTC by Arsène Lapostolet