

# Intégration des librairies tierces



**neopixl.**

A SMILE GROUP COMPANY

**Welcome on board**



# Sommaire

01

Introduction à Gradle

02

Les librairies tierces

03

Firebase Crashlytics

04

Fichiers JAR

05

Les modules

INTRO

3

**1**

# Introduction à Gradle

# Présentation

Gradle est le système de build utilisé par Android Studio.

Conseil, mettez à jour la version de distribution de Gradle autant que possible. Android Studio vous le proposera.

Gradle se compose de fichiers de configuration (que vous ne devrez normalement pas modifier)


Il se compose aussi d'au minimum deux scripts.

Le premier est lié au projet, le second à votre module app.


Dans le cas où votre projet contient d'autres modules, un script Gradle sera associé à vos modules complémentaires (par exemple une librairie interne)


## ▼ Gradle Scripts


 **build.gradle** (Project: HelloAndroid)

 **build.gradle** (Module: HelloAndroid.app)

 **gradle-wrapper.properties** (Gradle Version)

 **proguard-rules.pro** (ProGuard Rules for HelloAndroid.app)

 **gradle.properties** (Project Properties)

 **settings.gradle** (Project Settings)

 **local.properties** (SDK Location)



# Les plugins

Gradle vous permet d'utiliser des plugins.

Le premier plugin que vous retrouverez par défaut est "com.android.application"

Dans cet exemple, nous utilisons aussi des plugins spécifiques à Kotlin ainsi que des plugins liés à Google pour la gestion des notifications push et le crash collect.

Ces plugins permettent au système de build de générer le code nécessaire au fonctionnement de votre application.

Attention, certains plugins doivent être déclarés dans un ordre de priorité particulier. En effet, certains plugins dépendent d'autres plugins. **Consulter toujours la documentation avant d'en intégrer de nouveaux dans vos projets.**

```
apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-android-extensions'  
apply plugin: 'kotlin-kapt'  
apply plugin: "androidx.navigation.safeargs.kotlin"  
apply plugin: 'com.google.gms.google-services'  
apply plugin: 'com.google.firebase.crashlytics'
```



# Les plugins

Vous pouvez aussi utiliser la nouvelle syntaxe pour la définition des plugins

```
plugins {  
    id 'com.android.application'  
    id 'kotlin-android'  
    id 'kotlin-android-extensions'  
    id 'kotlin-kapt'  
    id 'androidx.navigation.safeargs.kotlin'  
    id 'com.google.gms.google-services'  
    id 'com.google.firebase.crashlytics'  
}
```



# Les plugins

Attention, certains plugins nécessitent de spécifier les "classpath" dans les dépendances du gradle du PROJET afin de pouvoir les résoudre.

```
buildscript {  
    ext.kotlin_version = "1.4.21"  
    ext.nav_version = "2.3.3"  
    ext.google_services_version = "4.3.5"  
    ext.crashlytics_version = "2.5.0"  
  
    repositories {  
        google()  
        jcenter()  
    }  
  
    dependencies {  
        classpath "com.android.tools.build:gradle:4.1.2"  
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"  
        classpath "com.google.gms:google-services:$google_services_version"  
        classpath "com.google.firebase:firebase-crashlytics-gradle:$crashlytics_version"
```



# Les tasks

Afin d'être en mesure de générer un build, Gradle devra exécuter différentes tâches.

Vous devrez définir les informations nécessaires afin que Gradle sache comment compiler votre application:

- Comment générer l'APK (version du SDK, version de l'application, version de la JVM Java,...)
- Quelles sont les dépendances de bibliothèques tierces à utiliser
- Faut-il obfusquer le code
- Quel environnement faut-il utiliser (dev, qa, production...)

Pas de panique, Android Studio va gérer la base pour vous!

```
apply plugin: 'com.android.application'  
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-android-extensions'  
apply plugin: 'kotlin-kapt'  
apply plugin: "androidx.navigation.safeargs.kotlin"  
apply plugin: 'com.google.gms.google-services'  
apply plugin: 'com.google.firebase.crashlytics'
```



```
android {...}
```

```
dependencies {  
    ...  
}
```

# La task Android

Dans cette tâche, vous pourrez configurer beaucoup de paramètres qui vont influencer votre build.

Dans un premier temps nous allons faire un focus sur les paramètres principaux:

- `compileSdkVersion`: version du SDK Android utilisé pour compiler votre application (devrait toujours être la plus récente release)
- `buildToolsVersion`: version des outils de compilation (même remarque, mais peut être ignoré pour un update "automatique")
- `defaultConfig` (configuration par défaut de votre application)
  - `applicationId` (package name, identifiant unique de l'application)
  - `versionName` (e.g. 3.2.0, mais peut être n'importe quel String)
  - `minSdkVersion` (Version de l'OS Minimum requis pour utiliser l'app)
  - `TargetSdkVersion` (Version du SDK pour laquelle l'app à été désignée, testée)
  - `versionCode`

```
android {  
    def code = 1  
  
    ndkVersion "21.3.6528147"  
    compileSdkVersion 30  
    buildToolsVersion "30.0.2"  
  
    defaultConfig {  
        applicationId "com.neopixl.helloandroid"  
        versionName project.ext.versionName  
        minSdkVersion 23  
        targetSdkVersion 30  
        versionCode code  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
        multiDexEnabled true  
    }  
}
```

# La task Android

Mais sachez que vous pourrez aller bien plus loin:

- compileOptions
- kotlinOptions
- buildFeatures
- buildTypes
- packagingOptions
- productFlavors et flavorDimensions
- lintOptions
- Et beaucoup d'autres encore...

```
android {  
    ndkVersion "21.3.6528147"  
    compileSdkVersion 30  
    buildToolsVersion "30.0.2"  
    def code = 1  
  
    lintOptions {  
        checkReleaseBuilds true  
        abortOnError false  
    }  
  
    defaultConfig {  
        applicationId "com.neopixl.helloandroid"  
        versionName project.ext.versionName  
        minSdkVersion 23  
        targetSdkVersion 30  
        versionCode code  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
        multiDexEnabled true  
    }  
  
    compileOptions {  
        sourceCompatibility = 1.8  
        targetCompatibility = 1.8  
    }  
  
    kotlinOptions {  
        jvmTarget = '1.8'  
    }  
  
    buildFeatures {  
        viewBinding = true  
    }  
  
    buildTypes {  
        debug {  
            buildConfigField 'boolean', 'CUSTOM_ALLOW_APP_RESTORE', 'false'  
            shrinkResources false  
        }  
    }  
}
```

# Les dependencies

Gradle vous permet aussi de définir des dépendances vers des librairies tierces.

Dans cet exemple vous pouvez constater que certaines dépendances sont soulignées en jaune.

Cette mise en forme correspond à une information nous avertissant qu'une version plus récente de la librairie est disponible.

Plus de détails vous seront fournis dans le chapitre suivant nommé "Les librairies"

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    // Kotlin libs  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlinVersion"  
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:$coroutinesVersion"  
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutinesVersion"  
  
    // Android libs  
    implementation "androidx.core:core-ktx:1.3.2"  
    implementation "androidx.appcompat:appcompat:1.2.0"  
    implementation "androidx.constraintlayout:constraintlayout:2.0.4"  
    implementation "androidx.legacy:legacy-support-v4:1.0.0"  
    implementation "androidx.multidex:multidex:2.0.1"  
    implementation "androidx.navigation:navigation-fragment-ktx:$navigationVersion"  
    implementation "androidx.navigation:navigation-ui-ktx:$navigationVersion"  
    implementation "androidx.dynamicanimation:dynamicanimation:1.0.0"  
    implementation "androidx.preference:preference-ktx:$preferenceVersion"  
    implementation "androidx.recyclerview:recyclerview:$recyclerviewVersion"  
    implementation "androidx.security:security-crypto:1.1.0-alpha02"  
    implementation "com.google.android.material:material:$materialVersion"  
  
    // Location libs  
    implementation 'com.google.android.gms:play-services-location:17.1.0'  
  
    // Analytics libs  
    implementation 'com.google.firebase:firebase-config-ktx:20.0.1'  
    implementation 'com.google.firebase:firebase-analytics:18.0.0'  
    implementation 'com.google.firebase:firebase-crashlytics:17.3.0'  
    implementation 'com.google.firebase:firebase-dynamic-links-ktx:19.1.1'  
    implementation 'com.google.android.gms:play-services-tagmanager:17.0.0'
```

# Les build flavors

Gradle vous permet aussi de définir différentes dimensions et flavors pour vos build.

L'idée est de pouvoir générer différents apk sur base de ces configurations afin par-exemple de connecter votre application sur différents environnements (dev, preprod, prod...)

Dans cette configuration, notre application pourra être compilée pour trois environnements. Les APK générés pourront être distribués soit pour du test interne, pour du beta testing ou pour une publication sur le Play Store.

```
flavorDimensions "ENV"

productFlavors {
    'dev' {
        dimension "ENV"
        applicationIdSuffix ""
        buildConfigField 'String', 'COOKIE_DOMAIN', '"https://dev.helloandroid.com"'
        buildConfigField 'String', 'API_BASE_URL', '"https://private-helloandroid.apiary-mock.com"'
        buildConfigField 'String', 'API_BASE_URL_PDF', '"https://dev.medias.helloandroid.com/pdf"'
        buildConfigField 'String', 'WEB_BASE_URL', '"https://dev.helloandroid.com"'
    }

    'preprod' {
        dimension "ENV"
        applicationIdSuffix ".pre"
        buildConfigField 'String', 'COOKIE_DOMAIN', '"https://pre.helloandroid.com"'
        buildConfigField 'String', 'API_BASE_URL', '"https://pre.helloandroid.com"'
        buildConfigField 'String', 'API_BASE_URL_PDF', '"https://pre.medias.helloandroid.com/pdf"'
        buildConfigField 'String', 'WEB_BASE_URL', '"https://pre.helloandroid.com"'
    }

    'prod' {
        dimension "ENV"
        applicationIdSuffix ""
        buildConfigField 'String', 'COOKIE_DOMAIN', '"https://www.helloandroid.com"'
        buildConfigField 'String', 'API_BASE_URL', '"https://helloandroid.com"'
        buildConfigField 'String', 'API_BASE_URL_PDF', '"https://medias.helloandroid.com/pdf"'
        buildConfigField 'String', 'WEB_BASE_URL', '"https://www.helloandroid.com"'
    }
}
```



# Les build flavors

Ci-dessous, un exemple d'utilisation dans le code de notre propriété **API\_BASE\_URL** qui sera modifiée dynamiquement en fonction de la variante choisie au build.

```
val retrofit = Retrofit.Builder()
    .addCallAdapterFactory(RxErrorHandlingCallAdapterFactory.create())
    .addConverterFactory(GsonConverterFactory.create())
    .baseUrl(BuildConfig.API_BASE_URL)
    .client(okBuilder.build())
    .build()

return retrofit.create(AddressService::class.java)
```

**2**

# **Les librairies tierces**



# Ajouter une librairie tierce #1

Dans le build.gradle (Module)

Pour les dépendances liées aux librairies : **implementation + dépendance**

Pour les dépendances liées aux tests unitaires et d'intégration : **testImplementation** ou **androidTestImplementation**

Les librairies pour lesquelles une mise à jour est nécessaire sont en surbrillance

Astuce : commenter chaque groupe de dépendance pour plus de lisibilité

```
//TEST DEP
testImplementation 'junit:junit:4.13.2'
testImplementation("org.junit.jupiter:junit-jupiter:5.7.0")
androidTestImplementation 'androidx.test.ext:junit:1.1.2'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
androidTestImplementation 'androidx.test:runner:1.3.0'
androidTestImplementation 'androidx.test:rules:1.3.0'
testImplementation "io.mockk:mockk:1.10.3-jdk8"
androidTestImplementation "io.mockk:mockk-android:1.10.3-jdk8"
implementation 'org.hamcrest:hamcrest-all:1.3'
androidTestImplementation 'androidx.test.ext:junit:1.1.2'
implementation 'androidx.test.espresso:espresso-idling-resource:3.3.0'
testImplementation "org.robolectric:robolectric:4.4"
implementation 'org.jetbrains.kotlin:kotlin-test-junit:1.4.21'
testImplementation 'org.amshove.kluent:kluent-android:1.64'

//kotlin
implementation 'org.jetbrains.kotlin:kotlin-stdlib:1.4.31'
implementation "org.jetbrains.kotlin:kotlin-reflect:1.4.31"

//kotlin coroutines
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.3'

//lifecycle
implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.0'
def lifecycle_version = '2.2.0'
implementation "androidx.lifecycle:lifecycle-extensions:$lifecycle_version"

//eventbus
implementation 'org.greenrobot:eventbus:3.2.0'
```



## Ajouter une librairie tierce #2

Dans le build.gradle (Project)

**classpath + dépendance**

Nécessaire pour certaines dépendances, lire la documentation de chaque librairie pour savoir où ajouter la dépendance dans gradle

```
buildscript {  
    repositories {  
        google()  
        jcenter()  
        mavenCentral()  
        maven { url 'https://maven.google.com' }  
        maven { url "https://www.jitpack.io" }  
        maven { url 'https://oss.sonatype.org/content/repositories/snapshots/' }  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:4.1.3'  
        classpath 'org.greenrobot:greendao-gradle-plugin:3.3.0'  
        classpath 'com.google.gms:google-services:4.3.5'  
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:1.4.31'  
        classpath 'org.jetbrains.kotlin:kotlin-android-extensions:1.4.31'  
        classpath 'com.google.firebase:firebase-crashlytics-gradle:2.5.2'  
        classpath 'de.mannodermaus.gradle.plugins:android-junit5:1.3.2.0'  
        // NOTE: Do not place your application dependencies here; they belong  
        // in the individual module build.gradle files  
    }  
}
```



## Les avantages

- Gain de temps (on ne réinvente pas la roue)
- Support de la communauté (pour les bibliothèques populaires)
- Code pré-testé (pour les bibliothèques populaires)



# Les points d'attention

Vérifier la licence (Apache Licence V2 par exemple, MIT, etc)

Attention aux Ads cachés dans une librairie -> peut créer des soucis lors de la publication de l'app sur les stores

Vérifier la date du dernier commit

Voici quelques librairies "non populaires". Pour chacune d'entre elles voyons si elles sont utilisables ou pas en 2021.

<https://github.com/kizitonwose/Time>

<https://github.com/thepacific/timer>

<https://github.com/loppower/CircleView>

Par le même développeur

<https://github.com/VaibhavLakhera/Circular-Progress-View>

<https://github.com/loppower/CircularProgressBar>

<https://github.com/akexorcist/GoogleDirectionLibrary>

<https://github.com/andkulikov/Transitions-Everywhere>

<https://github.com/amitshekhariitbhu/Android-Debug-Database>



# Les types de librairies #1

Prenons le temps de voir les types de librairies que l'on peut implémenter au sein d'un projet.



## Base de données

GREENDAO  
ROOM  
OBJECTBOX  
REALM DATABASE  
STETHO (Visualisation de la base de données)



## Appels réseaux

**RETROFIT**  
OKHTTP



## Composants personnalisés

[Timer \(dariobruX\)](#)  
[Material Range Bar \(Oli107\)](#)



## Affichage d'image

GLIDE  
PICASSO

# Les types de librairies #2

Suite des types de librairies.



## Débogage

TIMBER  
CRASHLYTICS  
LEAKCANARY



## Json

GSON  
JACKSON



## Injection de dépendances

DAGGER / HILT  
BUTTERKNIFE



## Autre

JODA TIME (Gestion du temps)  
**EVENTBUS**  
FASTADAPTER (Simplifier l'affichage des recyclerView)



# Ressources utiles

Liste de librairies “populaires”

<https://github.com/wasabeef/awesome-android-libraries>

<https://android-arsenal.com/>



3

# Firebase Crashlytics

## Qu'est-ce que c'est?

Outil analytique permettant de suivre les crashes survenants lors de l'utilisation de l'application

Déployé par Firebase de Google (outil gratuit)

D'autres outils du même type existent (AppCenter, Sentry...)

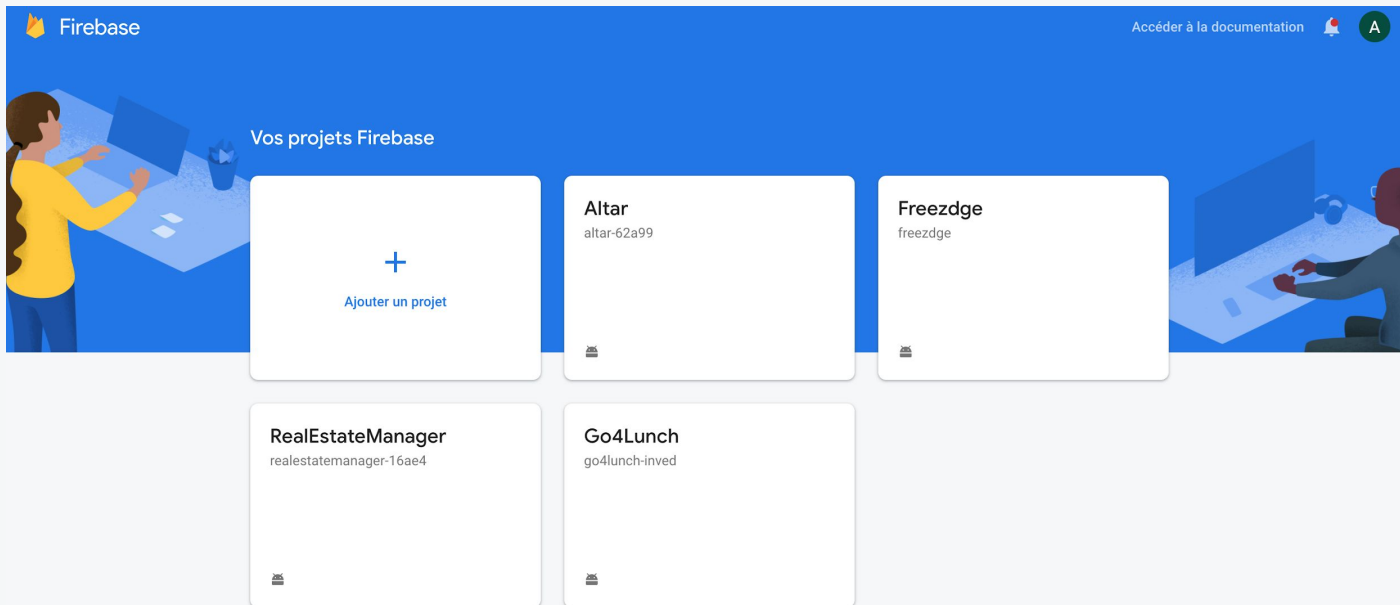




# Accès au projet

Accès depuis la Firebase Console.

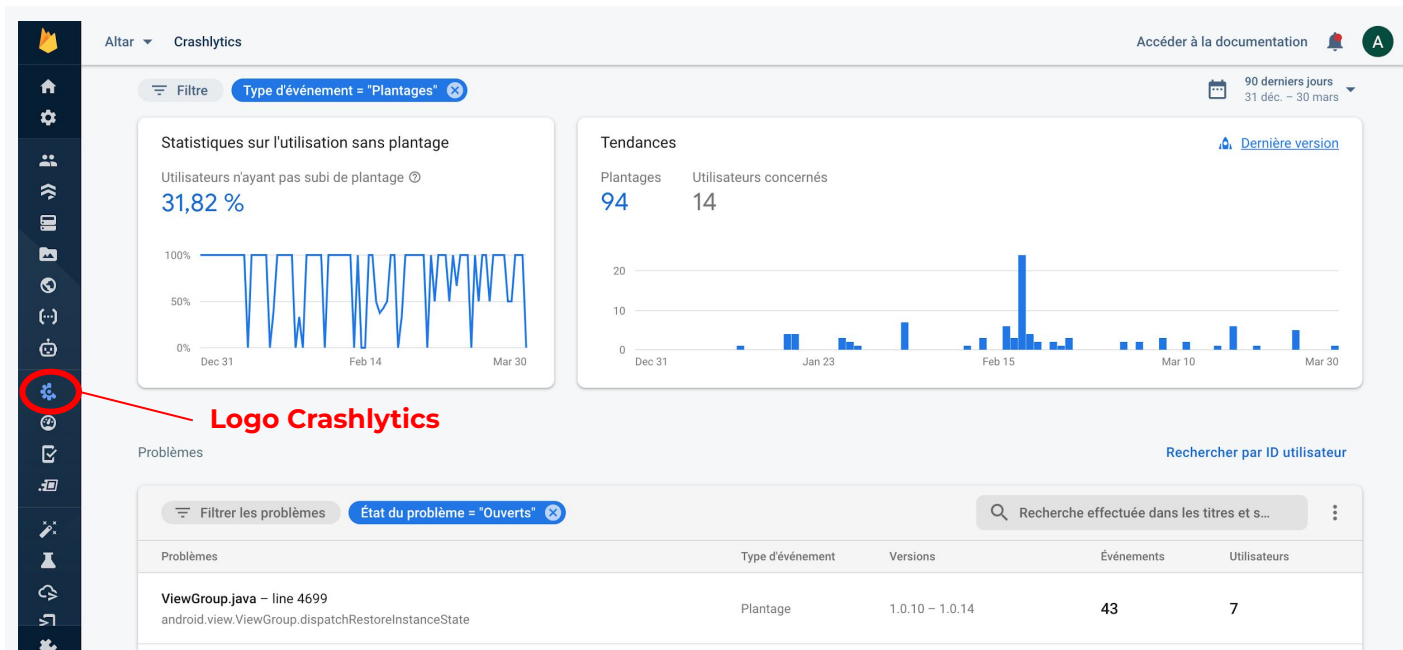
Vous pouvez ajouter un nouveau projet (voir exercice)



# Présentation d'un crashlytics

Choix possibles :

- Le laps de temps (60 dernières minutes, dernières 24h, 7 derniers jours, 30 derniers jours, **90 derniers jours**)
- Nombre de plantages
- Utilisateurs concernés (dans le détail on a les device concernés)



# Présentation du détail d'un crash

Informations visibles :

- Date du crash
- Version de l'app dans laquelle le crash s'est produit
- Device concerné (marque, modèle, version android)
- Classe et ligne dans lesquelles le crash s'est produit

The screenshot displays the Firebase Crashlytics web interface. At the top, the breadcrumb navigation shows 'Altar > Crashlytics > ViewGroup.java line 4699'. On the right, there are links for 'Accéder à la documentation', a notification bell, and a user profile icon labeled 'A'.

The main content area shows a summary of the crash event: 'Récapitulatif des événements', version '1.0.14 (1)', 10 crashes, on a 'Galaxy S9' device, dated '30 mars 2021, 08:40:22'. Below this, there are tabs for 'Trace de la pile' (selected), 'Clés', 'Journaux', and 'Données'.

The 'Trace de la pile' tab shows a stack trace for a 'Fatal Exception: java.lang.IllegalStateException' with the message 'Fragment no longer exists for key f#0: unique id de8b10ee-cf3c-4fac-b599-7f671037929d'. The stack trace includes the following frames (from bottom to top):

- com.android.internal.os.ZygoteInit.main (ZygoteInit.java:1100)
- android.view.View.restoreHierarchyState (View.java:21552)
- android.view.ViewGroup.dispatchRestoreInstanceState (ViewGroup.java:4699)** (highlighted with a blue arrow)
- androidx.viewpager2.widget.ViewPager2.dispatchRestoreInstanceState (ViewPager2.java:375)
- androidx.fragment.app.FragmentManager.getFragment (FragmentManager.java:772)

At the bottom of the stack trace, there is a link 'Afficher les 24 threads'.

**ex1**

# Exercise 1



# Exercice 1

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo1

Package name: com.technipixl.exo1

Placez le dans un dossier TECHNIFUTUR-AND9-EXO1





# Exercice : Installer CrashLytics

[Créez un compte gratuit avec Firebase](#)

Créer un nouveau projet dans la console Firebase et nommez le **And9 Exo1 (suivez les étapes)**  
Activez bien le Google Analytics dans votre projet (pour Crashlytics par la suite)

Ajoutez Firebase à votre projet dans Android Studio en [utilisant l'option 1 : ajouter Firebase à l'aide de la console Firebase](#)

Suivez [le tutoriel pour installer crashlytics](#) dans votre projet

Ajoutez un crash forcé [grâce à ce lien](#)

Lancez l'application et provoquez le crash.

Allez sur la console Firebase, choisissez votre projet, et vérifiez le crashlytics, que voyez-vous?


# Résultat

Si vous voyez un crash de ce type dans votre Crashlytics, **BRAVO** vous avez réussi à implémenter CrashLytics sur votre projet Android.

Problèmes

[Rechercher par ID utilisateur](#)

Filtrer les problèmes

État du problème = "Ouverts" 

 Recherche effectuée dans les titres et s...



Problèmes

Type d'événement

Versions

Événements

Utilisateurs

MainActivity.kt – line 20

com.technipixl.exo1.MainActivity\$forceCrash\$1.onClick

Plantage

1.0 – 1.0

1

1

4

# Fichiers JAR





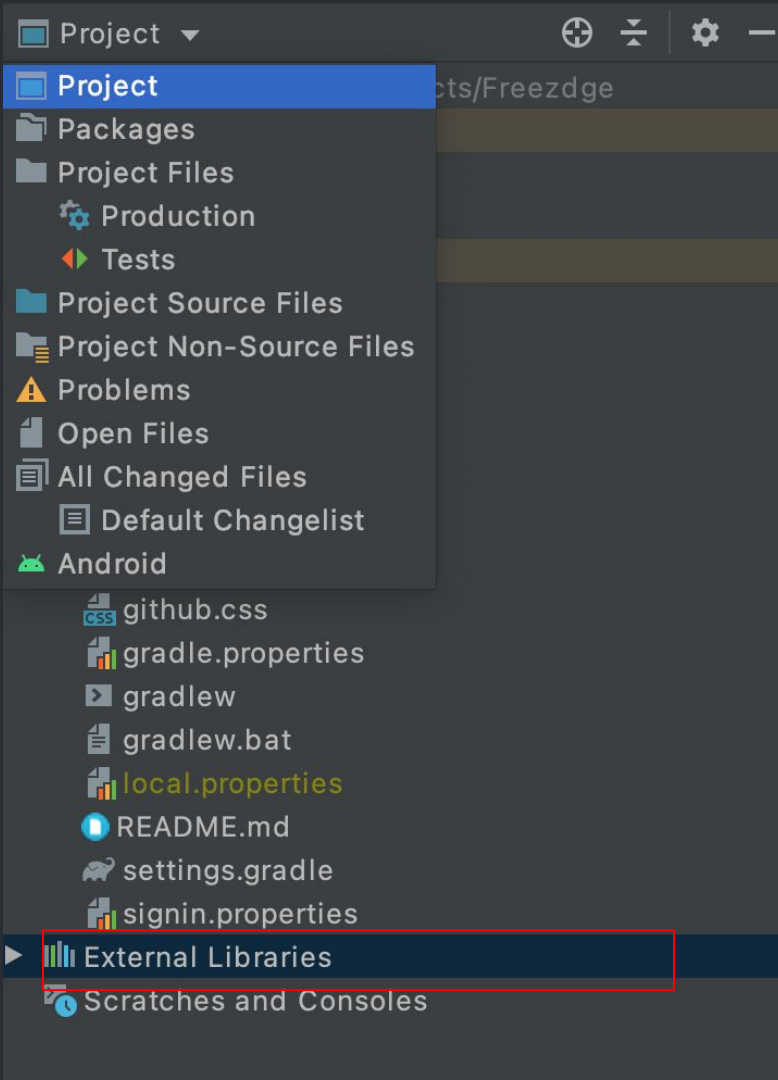
# Fichiers .jar

L'extension .jar est un abréviation de "Java Archive". Un dossier .jar contient une bibliothèque de plusieurs fichiers java qui définissent, dans notre cas, le fonctionnement d'une librairie tierce.

Si vous souhaitez savoir comment une librairie fonctionne en profondeur, alors vous pouvez aller voir dans les fichiers .jar de la librairie.

Pour cela, il faut se mettre en mode Projet dans Android Studio, puis on peut voir l'ensemble des Librairies Externes (External Libraries)

Choisissez celle qui vous intéresse et parcourez ses fichiers.



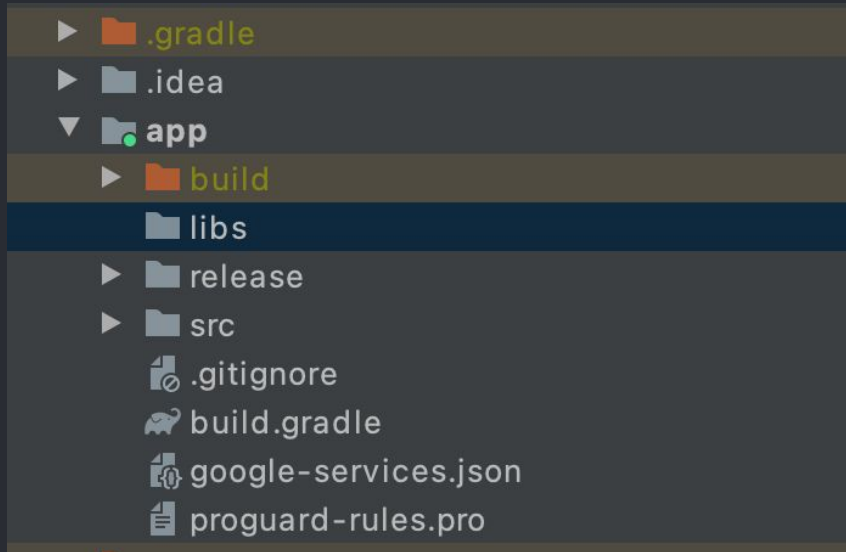
# Ajouter un fichier .jar

On peut également ajouter une librairie externe en ajoutant directement le fichier .jar en tant que librairie externe dans Android Studio. Cependant, je vous conseille de toujours passer par Gradle.

## Etape 1

On copie le fichier .jar que l'on veut ajouter (Ctrl+C ou Cmd+C). Ensuite, il faut se mettre en mode Projet dans Android Studio, puis aller dans **app** et faire un clic-droit sur **libs** puis on colle le fichier .jar (Ctrl+V ou Cmd+V)

Si le dossier libs n'existe pas, il est possible d'en créer un au même endroit.





# Ajouter un fichier .jar

Une fois que vous avez collé le fichier .jar dans le dossier libs vous avez deux options :

- Faites un clic droit sur le fichier jar et cliquez sur “Add as library”, choisissez le module dans lequel vous voulez ajouter la librairie. Cela se chargera d'ajouter les fichiers de compilation de la librairie ajoutée ('libs/nom\_de\_la\_bibliothèque.jar') dans build.gradle (et non de toutes les librairies .jar du dossier libs)

Exemple: **implementation files('libs/retrofit-2.9.0.jar')**

- Ou configurez votre build.gradle (module) pour ajouter tous les fichiers .jar du dossier libs à votre projet

```
dependencies {  
  
    implementation fileTree(dir: 'libs', include: '*.jar')
```

ex2

# Exercise 2

## Exercice 2

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo2

Package name: com.technipixl.exo2

Placez le dans un dossier TECHNIFUTUR-AND9-EXO2

Télécharger [le dernier fichier JAR](#) de la librairie externe Retrofit

Ajoutez le fichier .jar téléchargé en suivant les étapes du cours (diapos 34 et 35)



**5**

# **Les modules**

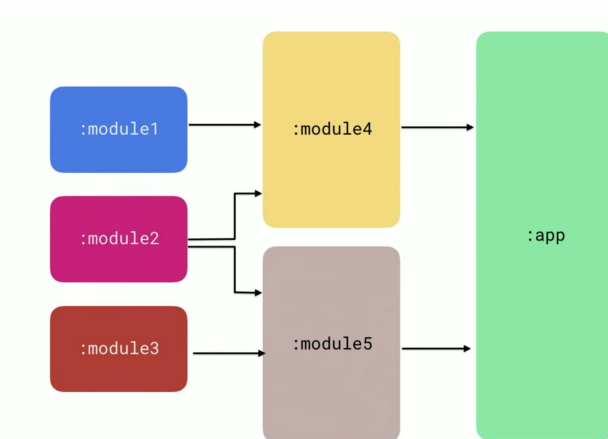
# L'objectif d'un projet modulaire

En général les développeurs utilisent le module app par défaut, créé lors de la création d'un nouveau projet Android.

Tout leur projet, toutes les dépendances qu'ils ajoutent ne tiennent que dans un seul projet. Cela peut produire, lorsqu'on compile l'app, des temps de chargement longs (10min), car tous les fichiers du module compilé sont vérifiés pour voir s'ils ont été modifiés.

Pour résoudre ce problème, de grandes applications comme Uber, choisissent de modulariser leur applications afin d'avoir les avantages suivants:

- Les rendres plus rapides
- Réutiliser des modules indépendamment les uns des autres pour d'autres applications
- Améliorer le travail d'équipe (chaque développeur peut travailler sur une fonctionnalité développée sur un module)
- Améliorer les flux git





# Créer un nouveau module

Pour créer un nouveau module  
Cliquez sur **File > New > New Module**

Choisir “Android Library” comme sur l’image à droite, et cliquez sur suivant (Next)

Puis, nommez votre nouveau module, par exemple “**core**”, et laissez les paramètres par défaut, cliquez sur terminer (Finish)

Un nouveau module nommé “core” apparaît

## Liaison du nouveau module “core” au module “app”

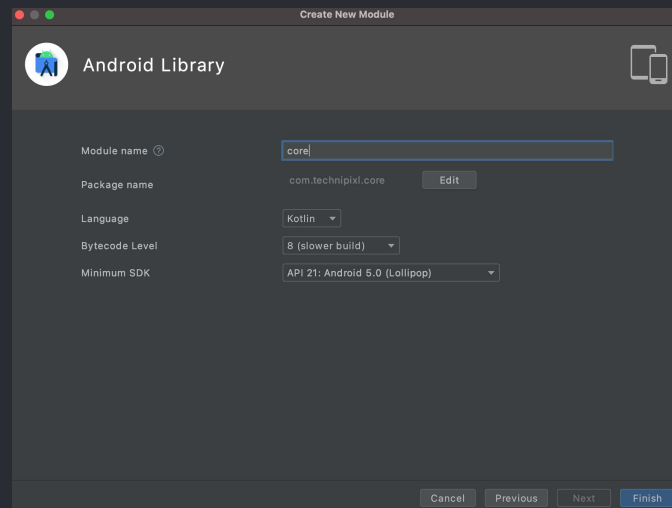
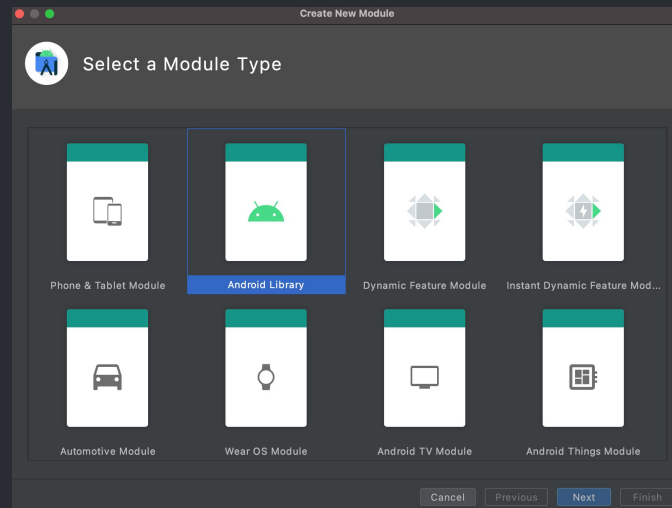
Vérifier dans le fichier settings.gradle que le nouveau module est bien listé

```
include ':core'  
include ':app'
```

Si c’est le cas, ajoutez la ligne suivante dans le build.gradle du module app, comme dépendance

```
implementation project(":core")
```

Pour finir, synchronisez le projet.





ex3

# Exercise 3



## Exercice 3 #1

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo3

Package name: com.technipixl.exo3

Placez le dans un dossier TECHNIFUTUR-AND9-EXO3



## Exercice 3 #2

Créer un nouveau module nommé “core” comme dans le cours.

Dans le module “core”, créez une classe Calculation dans laquelle vous créez quatre méthodes prenant en paramètre a et b (de type Double) :

- addition
- subtraction
- multiplication
- division

Chacune des méthodes retourne un type Double et réalise la bonne opération

Puis dans le module “app”, dans le fichier activity\_main.xml , créez 4 TextView l'une en dessous de l'autre dont les id sont respectivement nommés :

- textView1
- textView2
- textView3
- textView4

Activez le view binding dans le gradle du module “app”

Dans la MainActivity, liez la classe Calculation avec les textView puis lancez l'app :

- textView1 → addition
- textView2 → subtraction
- textView3 → multiplication
- textView4 → division





Avez-vous des questions?

**neopixl.**

A SMILE GROUP COMPANY



# Fil rouge



# Intégration fil rouge

Demandez à votre formateur de vous partager le projet de base de l'application Fil rouge Android ou utilisez votre projet si vous êtes parvenus à réaliser l'étape précédente

[https://github.com/technipixl/TECHNIFUTUR-AND-FILROUGE/releases/tag/advanced\\_login](https://github.com/technipixl/TECHNIFUTUR-AND-FILROUGE/releases/tag/advanced_login)

Depuis develop, créez une branche feature/FR005

Votre implémentation se fera sur cette branche

Une fois votre implémentation terminée, vous devrez “merge” votre branche sur **develop**

Demandez aussi à votre formateur de vous partager les fichiers drawable nécessaires à la mise en place de votre écran





# Intégration fil rouge

Implémenter un écran “home”:

- Ajouter les éléments graphiques (background, pictos...)
- Ajouter un view pager et son adapter...
- Implémenter les cellules à afficher (fragments contenant une cardview)
- Intégrer un indicateur de position
- Gérer le débordement sur les cellules



# Intégration fil rouge

Comment ajouter du débordement à votre pager (rendre visible le début de la carte précédente et suivante) ?

Astuce : Utilisez la méthode `setPadding` en combinaison avec la propriété `clipToPadding` (`false`)

```
private fun setupViewPager() {  
    val pagerAdapter = HomeAdapter(requireContext(), childFragmentManager,  
FragmentPagerAdapter.BEHAVIOR_RESUME_ONLY_CURRENT_FRAGMENT, this)  
    binding.viewPager.clipToPadding = false  
    binding.viewPager.setPadding(120, 0, 120, 0)  
    binding.viewPager.adapter = pagerAdapter  
    binding.tabLayout.setupWithViewPager(binding.viewPager, false)  
}
```





# Intégration fil rouge

Sur le clic d'une carte vous allez devoir afficher le fragment correspondant de votre menu bottom :

- FoodFragment
- FunFragment
- MoveFragment



# Intégration fil rouge

Vous allez devoir capter le clic dans le fragment représentant une cellule de votre view pager

L'idée est de remonter l'information grâce à un listener jusque votre HomeFragment

Dans le HomeFrament vous pourrez utiliser le composant de navigation pour vous positionner sur le bon élément de votre bottom navigation view

```
override fun onCardClicked(menuIndex: Int) {  
    when (menuIndex) {  
        1 -> findNavController().navigate(R.id.foodFragment)  
        2 -> findNavController().navigate(R.id.funFragment)  
        3 -> findNavController().navigate(R.id.moveFragment)  
    }  
}
```



# Intégration fil rouge

La définition d'un listener se base sur une interface

Vous allez devoir trouver un moyen de fournir l'index de l'élément cliqué afin que le HomeFragment puisse se positionner sur le bon élément de la bottom navigation view

```
class HomeCardFragment : Fragment() {  
  
    interface HomeCardClickListener {  
        fun onCardClicked(menuIndex: Int)  
    }  
  
    ...  
}
```

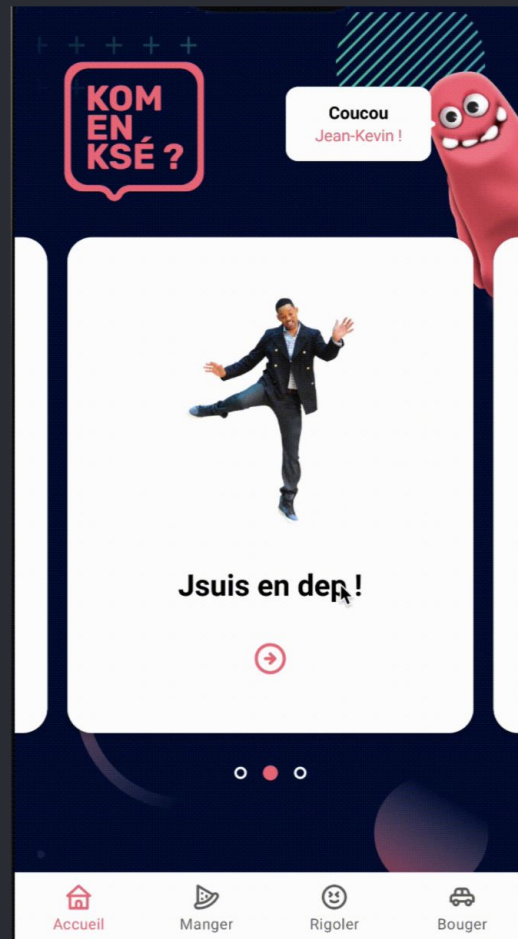
# Intégration fil rouge

Résultat attendu animé



# Intégration fil rouge

Résultat attendu





## Restons en contact

**neopixl.**

A SMILE GROUP COMPANY

115A, Rue Emile Mark  
L-4620 Differdange

(+352) 26 58 06 03  
[contact@neopixl.com](mailto:contact@neopixl.com)