Laboratoire 1

Les premiers pas

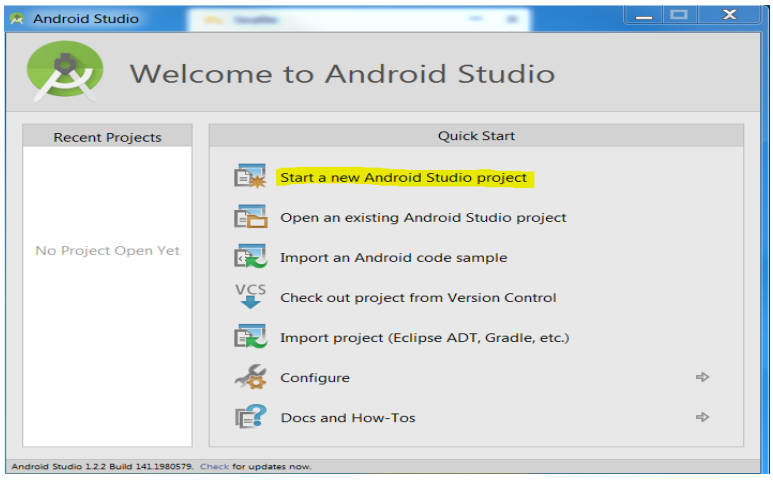
Objectifs d'apprentissage

- Avoir un premier contact avec la programmation mobile sous Android
- Réaliser une première application avec Android Studio
- Comprendre l'arborescence d'un projet Android
- Comprendre l'environnement d'exécution
- Comprendre la notion d'agencement des composants (Layout)
- Comprendre les différentes façons de créer un composant de base (XML, Programmation, graphiquement)
- Utiliser les composants à l'aide de leur identifiant
- Interaction avec l'application

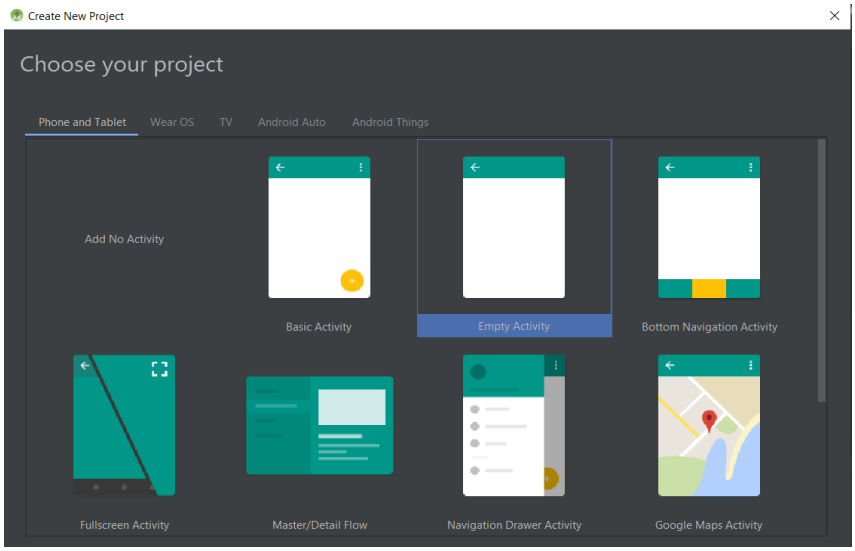
1. Création d'une première application

Créer l'application laboratoire 1

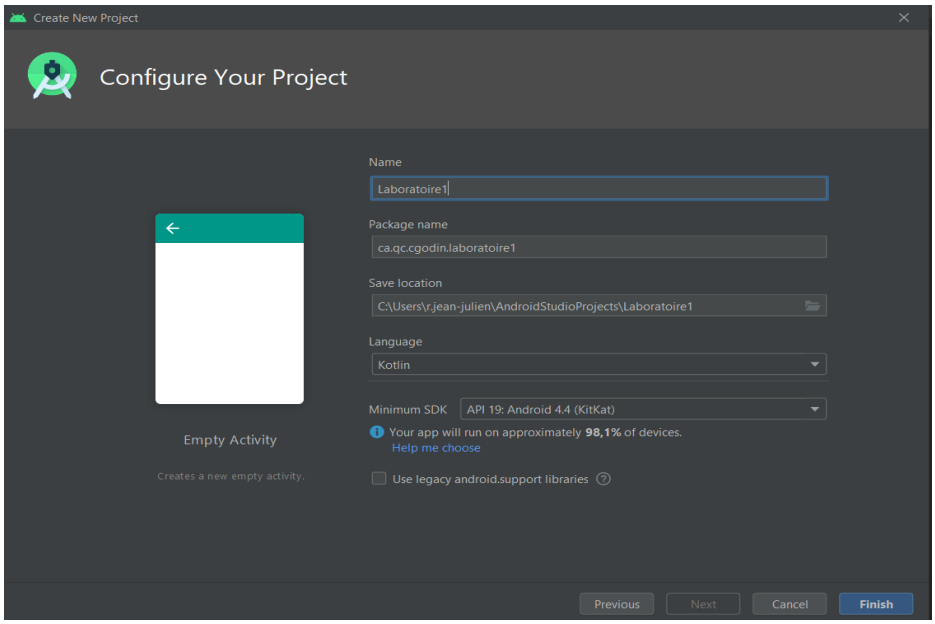
- À la première ouverture d'Android Studio, choisir **start a new Android Studio Project** dans l'écran de bienvenue



- Vous devez créer un projet avec le template "Empty Activity"



- Remplir les champs **Application Name** et **package name** selon le modèle suivant :

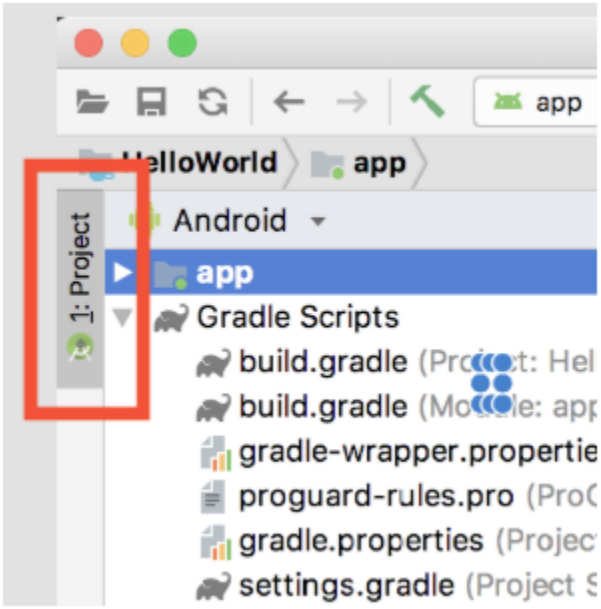


Important : le nom de l'application doit commencer par une majuscule. C'est ce nom qui apparaîtra pour l'utilisateur.

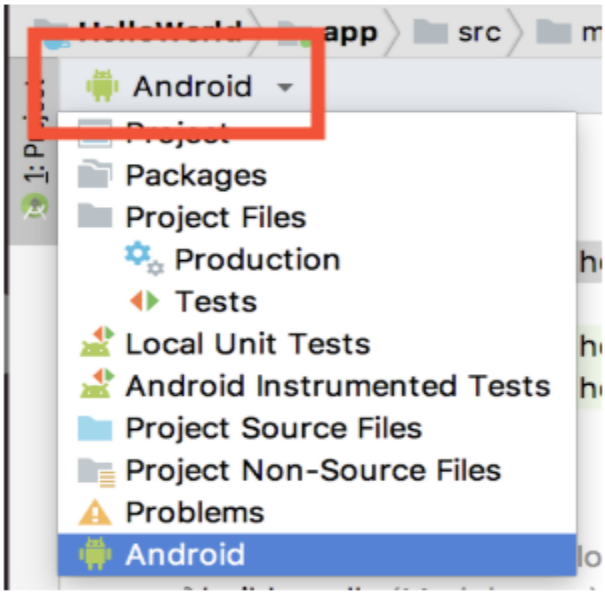
- **package Name** est généré automatiquement. Noter laboratoire1 commence par une minuscule et ne contient pas d'espace (règles des noms des packages dans Kotlin). Ce nom de package doit être unique. Modifiez le nom pour ca.qc.cgodin.laboratoire1 si ce n'est pas le cas.
- **Project Location :** choisir un emplacement pour votre projet
- Cliquer sur **Finish**

Explorer Android Studio

- Si l'onglet ***Projet*** n'est pas déjà sélectionné, sélectionnez-le.
- L'onglet ***Projet*** se trouve dans le panneau vertical à gauche de votre fenêtre dans Android Studio.

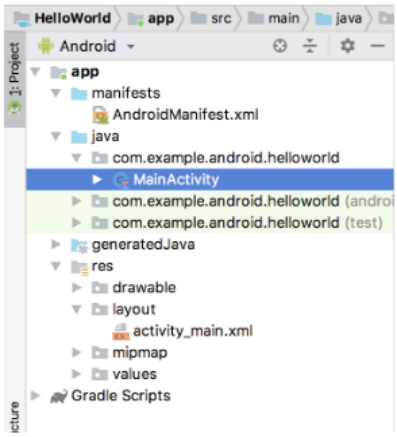


- Pour afficher le projet comme une hiérarchie de projet Android standard, sélectionnez **Android** dans le menu déroulant en haut du volet Projet. (**Android** est la valeur par défaut.) Vous pouvez afficher les fichiers de projet de différentes façons, y compris l'affichage des fichiers comment ils apparaissent dans la hiérarchie du système de fichiers. Cependant, le projet est plus facile à travailler avec l'utilisation de la vue Android.



Explorer le dossier de l'application

- Dans le **volet Projet > Android,** développez le dossier **de l'application.** À l'intérieur du dossier se trouvent trois ou quatre sous-dossiers :
  Manifests, java, res et dans certains cas generatedJava qui contient des fichiers générés par Android Studio à la construction de l'application.
- Développez le dossier **Java**, puis développez le dossier **ca.qc.cgodin.laboratoire1** pour afficher le fichier Kotlin **MainActivity.**

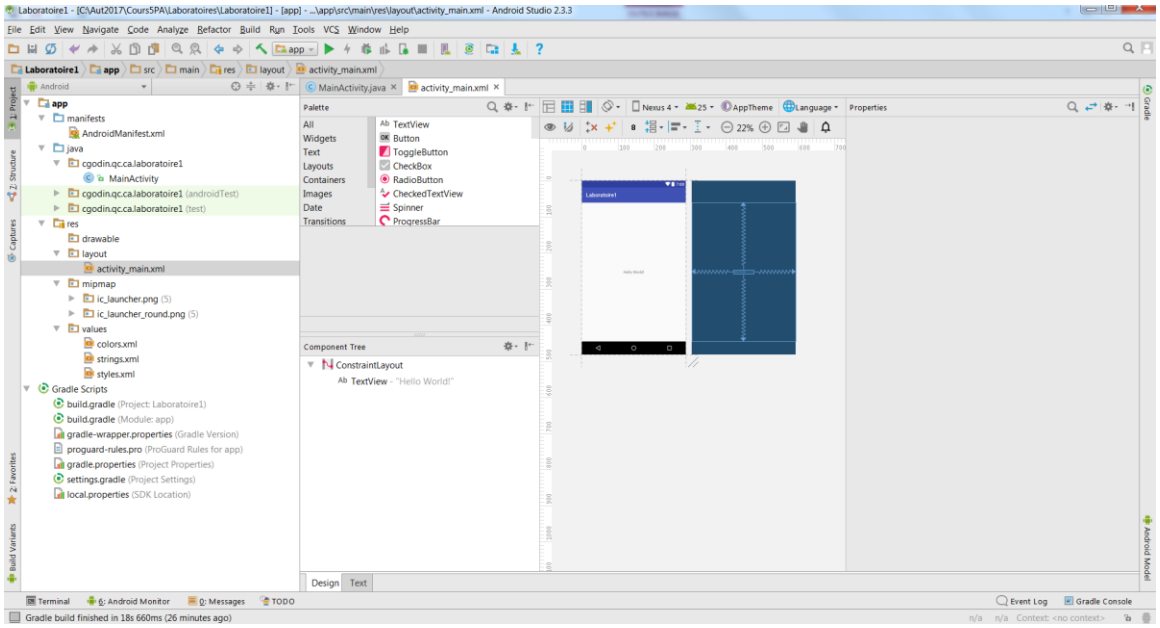


Le dossier Java contient tout le code Kotlin principal d'une application Android.
Il existe des raisons historiques pour lesquelles votre code Kotlin apparaît dans le dossier java. Cette convention permet à Kotlin d'interagir en toute transparence avec le code écrit dans le langage de programmation Java, même dans le même projet et application. Les fichiers de classe de votre application sont contenus dans trois sous-dossiers, comme le montre la figure ci-dessus. Le dossier ***ca.qc.cgodin.laboratoire1*** contient présentement le fichier MainActivity.kt qui est dans ce cas la porte d'entrée de votre application. Les deux autres sous-dossiers sont utilisés à des fins de test.

Explorer le dossier res

- Dans le volet Project>Android, développez le dossier res contenant les ressources de l'application.
  Les ressources d'une application Android sont du contenu statique. Elles incluent des images, du texte, des styles, etc. Les applications Android séparent le plus que possible le code des ressources. Cela permet de trouver assez facilement les textes ou icones utilisés dans l'interface utilisateur. En outre, lorsque vous modifiez l'un de ces fichiers de ressources, la modification prend effet partout où le fichier est utilisé dans l'application.

- Dans le dossier **res,** développez le dossier **layout** pour afficher le fichier activity_main.xml. Vous devriez avoir un écran semblable à celui-ci ou du code XML. Une activité est en général associée à un fichier de layout, défini comme un fichier XML dans le répertoire res/layout.



Remarque : si vous avez une erreur de rendu :

Rendering problems : missing styles....

- Cliquer sur rebuild si ce choix vous est offert.
- Sinon, utilliser le menu Fichier/Invalidate caches/restart

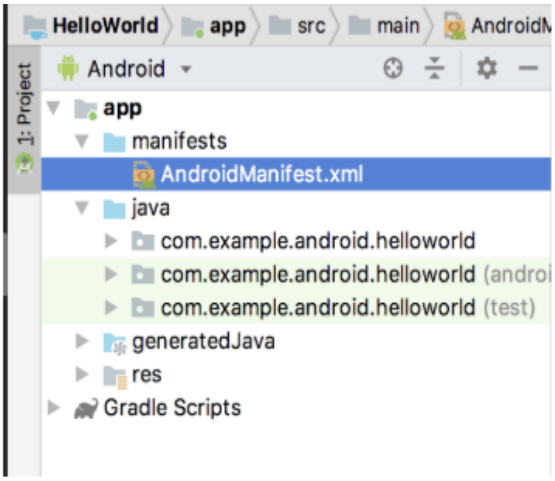
Les fichiers ressources sont classés dans des sous-dossiers selon leur type dont voici quelques sous-dossiers :

- mipmap-*dpi: contient les images au format png, gif, jpeg et bmp.
- layout : descriptions d'interfaces graphiques définies dans des fichiers au format xml. Noter la présence du fichier *activity_main.xml* pour notre première application.
- values : valeurs de différents types simples (exemple : chaines de caractères, entiers,...) contenus dans des fichiers *xml*. Chacune des valeurs est accessible via un identifiant unique.

Explorer le répertoire manifests et le fichier AndroidManifest.xml

1. Le dossier **manifests** contient des fichiers fournissant des informations essentielles à propos de votre application.

Développez le dossier **manifests** et double-cliquez sur **AndroidManifest.xml** pour l'ouvrir. Le fichier **AndroidManifest.xml** contient des détails dont a besoin le système Android pour exécuter votre application, y compris les activités qui font partie de l'application.



2. Notez que MainActivity est référencé dans l'élément <activity>. Toute activité faisant partie de votre application doit être déclarée dans le fichier **manifest**.
Voici un exemple de fichier manifest.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ca.qc.cgodin.laboratoire1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Laboratoire1"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

3. Notez l'élément <intent-filter> dans <activity>. Les éléments <action> et <category> indiquent à Android où commencer l'application quand l'utilisateur clique sur l'icone de démarrage.
Le manifeste est l'endroit où vous aurez à définir les permissions de votre application telles que accéder aux contacts de votre téléphone, accès à la caméra, etc.

Explorer le répertoire app/build.gradle

- **A**ndroid Studio utilise Gradle pour compiler l'application.
- Un fichier build.gradle est généré pour chaque module d'un projet Android.
- Un fichier build.gradle est aussi généré pour le projet au complet (le module d'application app). C'est ce fichier qui nous intéresse. Ce fichier contient les dépendances et les paramètres de configuration de l'application (compiledSdkVersion, applicatgionId, minSdkVersion, targetSdkVersion). Vous aurez assez souvent à le modifier.

2. Accès aux ressources

- **Depuis un programme Kotlin**

  Le dossier **project/app/build/generated/source/r/debug/<Nom de votre Package>** contient un fichier **R.java,** auto-généré. Il indexe toutes les ressources utilisées par le projet. Il est mis à jour automatiquement quand on ajoute ou supprime des ressources au projet (exemple : ajout d'un composant graphique). **Il ne faut jamais modifier ce fichier.**

  Depuis un code Java ou kotlin

  [package].R.type.nom

  type est le nom du sous-dossier du dossier **res** contenant la ressource.
  nom est le nom du fichier dépourvu de son extension.

  Exples : R.layout.activity_main (Voir fichier MainActivity.java ou MainActivity.kt)
  R.mipmap.icon

  Cette syntaxe fonctionne pour toutes les ressources à l'exception de celles du dossier res/values. Dans ce cas, le type de la ressource est le nom de la balise définissant la ressource et le nom de la ressource est le nom spécifié par l'attribut *name* de la balise de la ressource.

  Exples : R.string.message
  R.color.bleu

- **Depuis un code xml**

  @[package]type/nom

Exples : @mimmap/icon
@layout/activity_main
@string/message
@color/bleu

3. Exécution de l'application

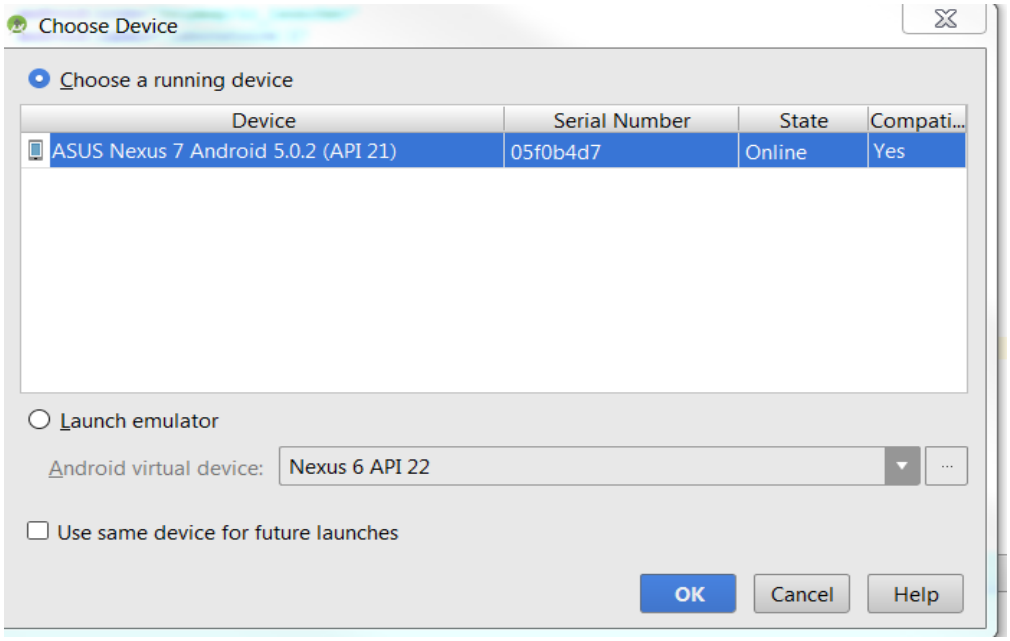
- **Dans un environnement simulé**

  Le SDK propose un émulateur **Android Virtual Device** (AVD), permettant de tester les applications en local.

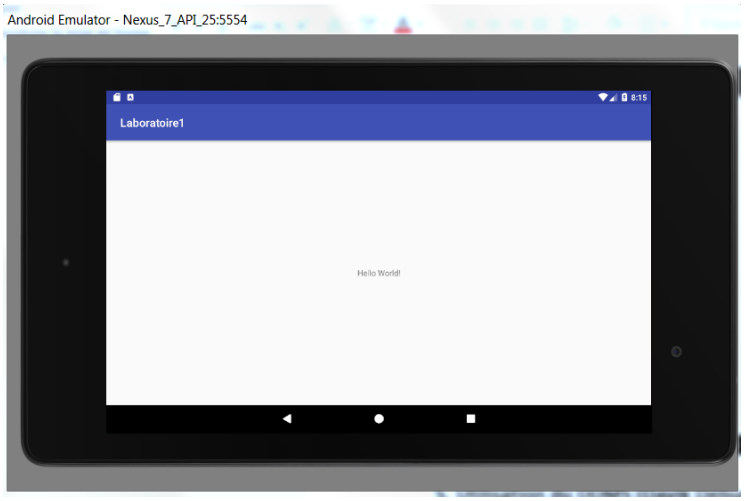
- **Directement sur un téléphone ou une tablette**

  Dans ce cas, il faut connecter l'appareil intelligent à l'ordinateur via un câble USB. Il faut s'assurer que le pilote adéquat a été installé auparavant. Dans **Paramètres/Options pour les développeurs** de votre tablette, vérifier que débogage **USB est coché**.

  - Exécuter l'application sur votre tablette



  - Vérifier que votre tablette est visible est qu'elle est *Online*. Cliquez sur *OK*
  - Le résultat de l'exécution devrait ressembler à ceci :

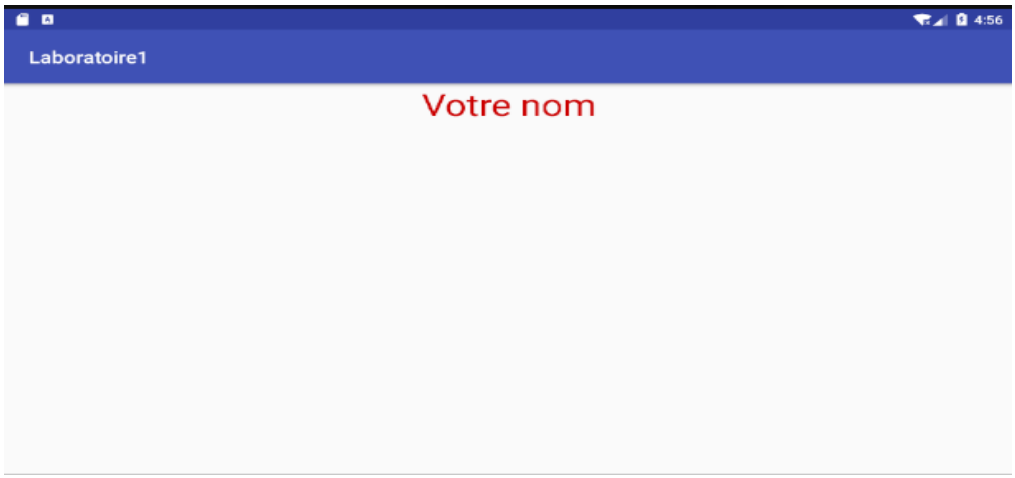


4. Modification de l'application

- Dans le dossier ressource (*res)*, double-cliquez sur *activity _main.xml* qui se trouve dans le sous-dossier *layout*. Noter la présence de trois onglets (Code, Split et Design). Vérifier que c'est l'onglet *Design* qui est activé.
- Notez la présence de l'arborescence des composants qui se trouvent dans le *layout* (Component tree dans le panneau de droite).
- Sélectionnez ***ConstraintLayout*** puis ***textView*** pour distinguer les deux composants.
- Cliquez sur la zone de texte (textView) pour afficher ses propriétés.
- Mettez la valeur de **id** à **@+id/textView1** (pour identifier la zone de texte)
- Mettre la valeur de *TextSize* à 36sp.
- Mettre la couleur(textColor) du texte a rouge.
  Noter les changements dans l'interface graphique et dans le fichier *activity_main.xml* (choisir onglet Text).
- Remarquer que la propriété **text** a pour valeur **Hello World**
  Note : Ce n'est pas une bonne pratique de placer directement une chaine dans ce fichier. Pour corriger ce problème, nous allons placer la chaine dans le fichier strings.xml et y faire référence.
- Remplacer android :text="**@string/votre_nom"**.
  Pour changer cette valeur, il faut donc ajouter la variable *votre_nom* dans le fichier strings.xml.
- Dans le sous-dossier **values** du dossier **res**, double-cliquez sur **strings.xml** et ajoutez la ligne suivante:

```
<string name="votre_nom">Votre nom!</string>
```

- Exécutez l'application et notez les changements dans l'interface graphique.



5. Ajout d'une zone de texte (TextView) avec des images en utilisant XML

Noter que le Layout créé par défaut est un **ConstraintLayout**. Les composants y sont repérés les uns par rapport aux autres. On le considère comme un RelativeLayout amélioré.

- Copier le bout de code suivant dans le fichier **activity_main.xml avant** la balise </Constraint>

```
<TextView
    android:id="@+id/textView_img"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:text="TextView"
    android:textAlignment="center"
app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.12"
    tools:text="@string/message_textView_img"
/>
```

A consulter pour mieux comprendre: https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html

- Ajouter la ligne suivante au fichier **strings.xml** :

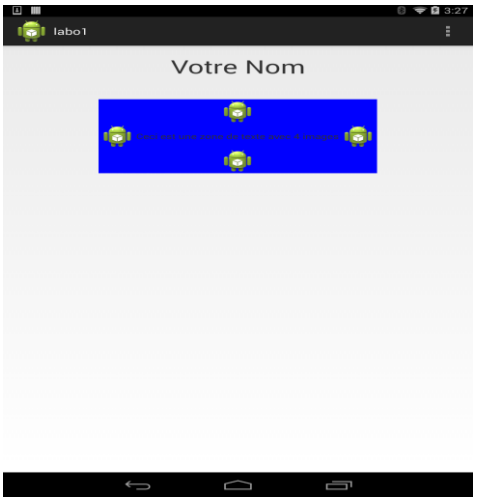
<string name="message_textView_img"> Ceci est une zone de texte avec 4 images</string>

Observer le résultat sur le *Design*. Vous pouvez exécuter à nouveau pour voir le résultat sur votre tablette. Le nouveau composant apparaît au-dessous de **Votre nom** grâce à la ligne,

app:layout_constraintVertical_bias="0.12"

, qui permet de positionner le textView juste en dessous. Utiliser la fenêtre de propriétés du nouveau composant pour comprendre les différents attributs.

- Exécuter votre application.
- Elle doit ressembler à ceci :



- Compléter le code XML ci-dessus pour changer la couleur du texte à blanc et la taille à 24sp

6. Ajout d'un champ de texte de saisie par programmation

- Pour réaliser cette opération, nous devons d'abord récupérer le **layout** de base. Pour ce faire, il faut donner un identifiant à notre Layout afin de pouvoir l'utiliser. Il faut donc ajouter la ligne en gras suivante au fichier activity_main.xml où est déclaré le composant.

```
<ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:id="@+id/layoutBase"
 ……
```

- Sauvegarder le fichier activity_main_xml

- Ajouter le code Kotlin en gras dans la classe *MainActivity.java*.

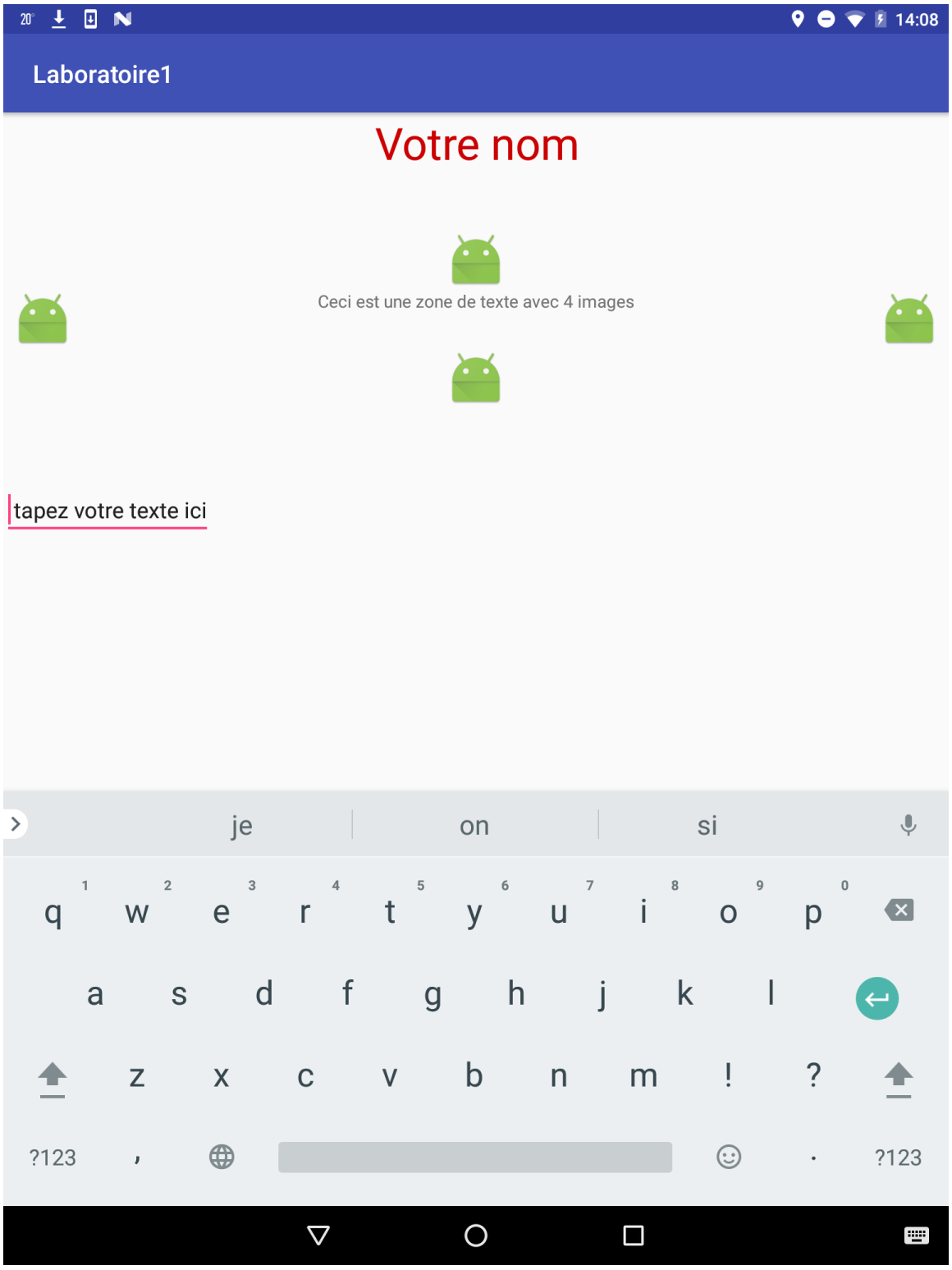
```
public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val constraintLayout = findViewById<ConstraintLayout>(layoutBase)
        val editText = EditText(this)
        editText.id = R.id.myEdText

        editText.hint="Tapez votre texte ici"
        editText.layoutParams = ConstraintLayout.LayoutParams(0, ConstraintLayou

        constraintLayout.addView(editText)
        val constraintSet = ConstraintSet()
        constraintSet.clone(constraintLayout)
        constraintSet.connect(editText.id,ConstraintSet.TOP, R.id.textView_img,
        constraintSet.connect(editText.id,ConstraintSet.LEFT, ConstraintSet.PARE
        constraintSet.applyTo(constraintLayout)
}
```

Noter l'utilisation de l'instruction :

```
val constraintLayout = findViewById<ConstraintLayout>(layoutBase)
```

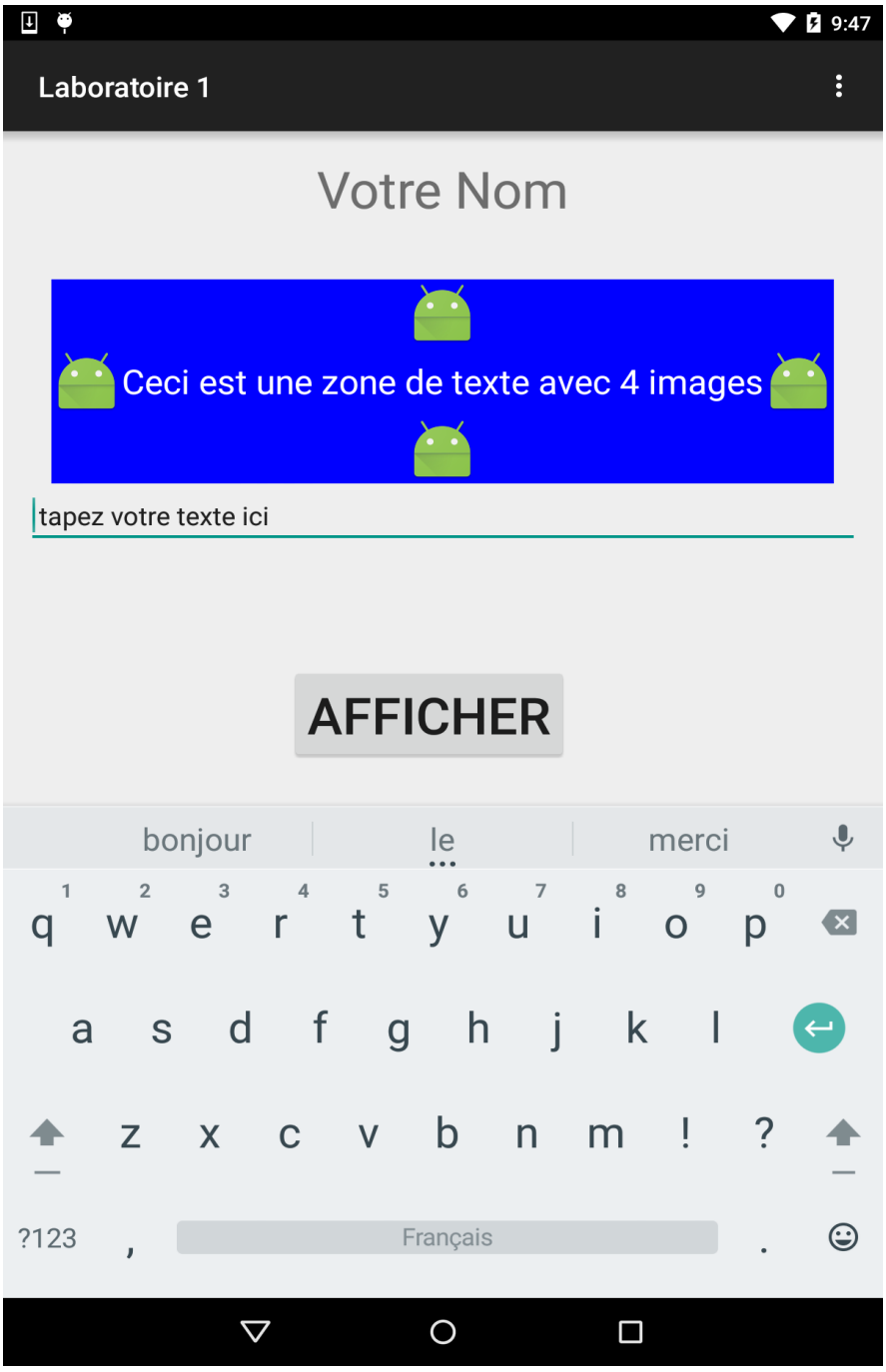
Elle permet de récupérer l'identifiant d'un composant afin de l'utiliser.

- Exécuter.
  Votre application doit ressembler à ceci : (le clavier apparaît de manière systématique pour la saisie)


14:08
Laboratoire1
Votre nom
Ceci est une zone de texte avec 4 images
tapez votre texte ici
je
on
si

7. Ajout d'un bouton en utilisant l'interface graphique (drag and drop)

- Cliquer sur activity_main.xml. Choisir **Design.**
- À partir du panneau ***Palette***, développer **Widgets,** sélectionner le composant **Button** et le glisser le sur le *layout*. Ajuster ses propriétés pour avoir ce qui suit à l'exécution :



8. Interactions avec les composants

Nous allons programmer le clic sur le bouton **Afficher**, pour afficher le message saisi dans la zone de texte *edText* créée par programmation. Dans les deux cas qui suivent, il faut d'abord récupérer l'identifiant du bouton, puis programmer l'écouteur d'événement qui réalisera l'affichage

8.1 Utilisation d'un objet Toast

Un Toast est un message qui s'affiche pendant quelques secondes en premier plan dans une fenêtre dont la taille est imposée par la taille du message. Intéressant dans le cas des messages purement informatifs.

- Copier le code suivant dans MainActivity.java

```
val btnAfficher = findViewById<Button>(R.id.button)
        btnAfficher.setOnClickListener {
                val context = applicationContext
                val text = editText.getText()
                val duree = Toast.LENGTH_LONG
                val toast = Toast.makeText(context, text, duree)
                toast.show()
        }
```

- Importer les classes nécessaires et définir la variable *edText* comme variable d'instance (sortir la déclaration de la méthode onCreate)
- Exécuter, entrer du texte dans la zone de saisie, puis cliquer sur le bouton Afficher. Observer le résultat.

Note: Dans les cas d'un bouton crée à l'aide de l'interface graphique, il est aussi possible d'associer, dans la fenêtre de propriétés, un nom de méthode à l'événement On Click, puis de programmer son corps dans l'activité correspondante. L'entête de la méthode serait :

public void nom_methode(View v).

8.2 Utilisation d'une boite de dialogue

Il s'agit d'utiliser une boite de dialogue d'alerte très simplifiée.

- Mettre en commentaires l'écouteur d'événement du bouton afficher.

- Copier et bien comprendre le code suivant dans MainActivity

```
btnAfficher.setOnClickListener{
            val alertDialog = AlertDialog.Builder(this)
            alertDialog.setTitle(" Texte entré ");
            alertDialog.setMessage(editText.text);
            alertDialog.setIcon(android.R.mipmap.sym_def_app_icon);

            alertDialog.setPositiveButton("OK") { _, _ ->
                val toast = Toast . makeText (applicationContext, "vous
                toast.show();
            }

            alertDialog.setNegativeButton("Annuler") { _, _ ->
                val toast = Toast . makeText (applicationContext, "vous
                toast.show();
            }

            alertDialog.show()
        }
```