



FORMATION

IT - Digital - Management

24/02/2024



m2iformation.fr

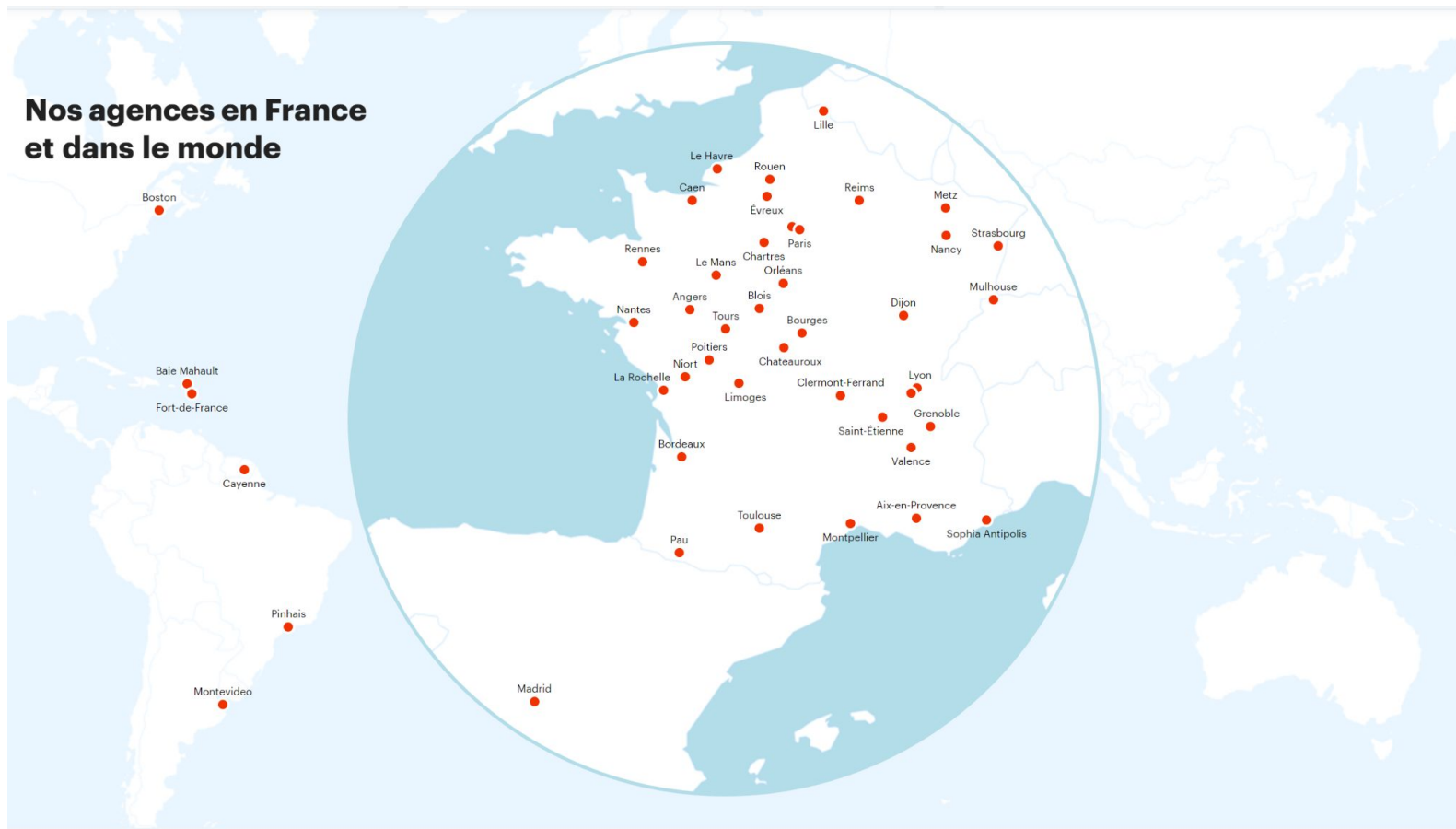


Bienvenue sur cette formation React

- Clément Hamon
- Développeur depuis 5 ans
- 3 ans de Java en entreprise
- 2 ans de formation
- `clement.hamon35@gmail.com`

Le réseau M2I Formation

Nos agences en France et dans le monde



Le groupe M2I

- Le groupe M2i est leader de la formation IT, Digital et Management en France depuis plus de 35 ans.
- L'engagement pour la qualité en étant certifié Qualiopi et Datadock.
- Plus de 300 collaborateurs dédiés à la montée en compétences de votre capital humain.
- Le catalogue M2I : <https://www.m2information.fr/catalogues/>
- La démarche qualité : <https://www.m2information.fr/demarche-qualite/>

Horaire et convocations

- 9h - 17h
- 15 mn de pause le matin
- 1h de pause déjeuner
- 15 mn de pause l'après midi

- Certifications
- Dernier jour à 16h30

Déroulé et structure de la formation et formalités

- Emargement le matin et l'après-midi
- Google Forms de demi-journée pour la validation des acquis et l'adaptabilité
- Évaluation du formateur le dernier jour au retour de la pause midi
- Notions théoriques suivis de mise en pratique

Tour de table et pré-requis

- Qui êtes-vous et quel est votre expérience dans l'informatique ?
- Vos attentes à l'issue de cette formation ?
- Droits admin
- Avoir suivi le cours JAV-SE "Java - Les fondamentaux et le développement Java SE"
- Ou avoir une connaissance pratique du langage Java.
- Mail avant 10h30

Structure et versionning Github

- Pour chaque notion montrée, un commit sera disponible sur ce git
- <https://github.com/ClementHamonDev/Formation-AND-PRG>
- Support de cours sur Teams et Github

Objectifs de la formation

1. Définir l'architecture des applications Android et leur cycle de vie
2. Concevoir une interface graphique pour terminal mobile
3. Interroger des services Web
4. Gérer les évènements Touch
5. Adapter un contenu pour tablettes avec les fragments
6. Utiliser les API multimédia
7. Déployer une application.

Installation des outils

- Android studio



Android Studio

Koala Feature Drop
2024.1.2



1. Présentation d'Android

Historique d'Android

- Création par Google en 2008
- Système d'exploitation open-source basé sur Linux

Modèle de développement :

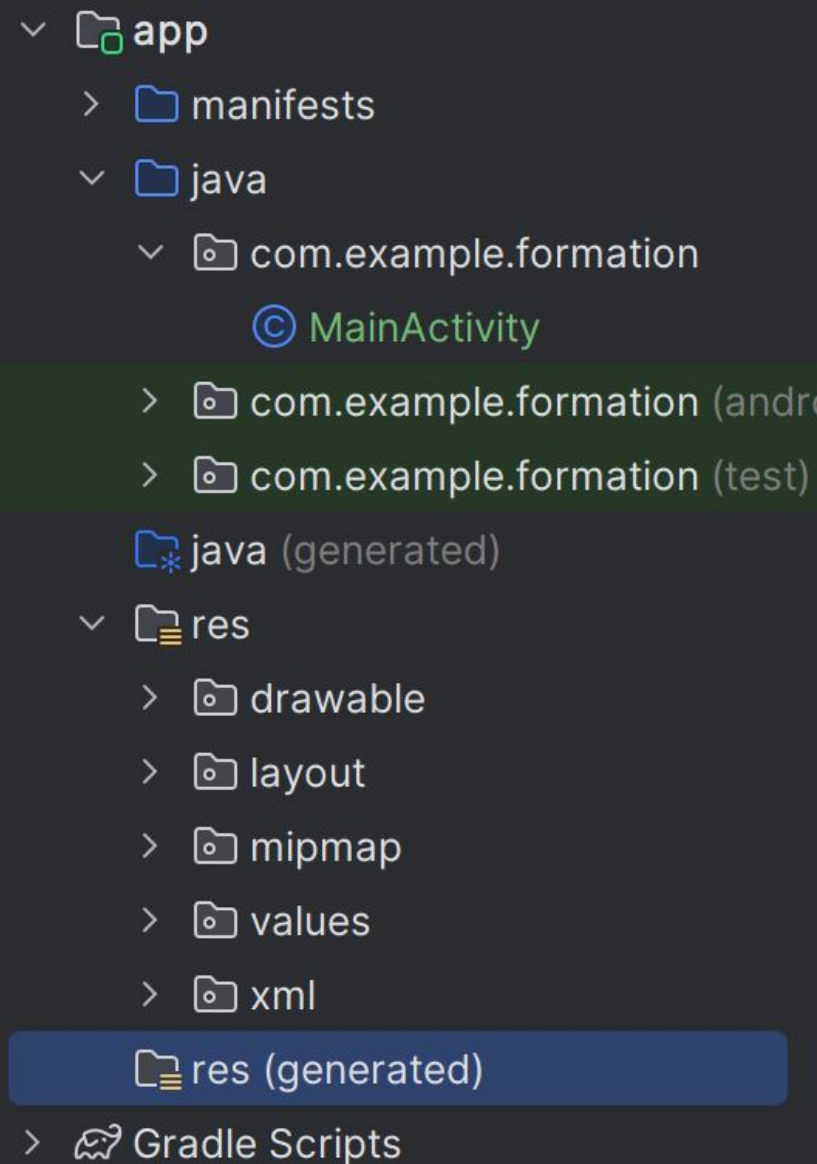
- Applications écrites en Java (ou Kotlin)
- Basé sur le SDK Android et Android Studio

Contexte et enjeux :

- Marché des smartphones, tablettes et objets connectés
- Plateforme dominante avec plus de 70% de parts de marché

Architecture d'une application Android

- **Manifest :**
 - Décrit les composants d'une application (activités, services, permissions)
- **Le dossier Java :**
 - Tous les fichiers source Java créés lors du développement de l'application, y compris les fichiers de test.
- **Les dossier res :**
 - Toutes les sources non liées au code, comme les images, les **mises en page XML** (dans layout) et les chaînes de texte



Fichier Manifest

- **manifest** : Contient le nom du package et inclut l'élément `<application>`.
 - **uses-permission** : Déclare les permissions nécessaires à l'application.
 - **uses-configuration** et **uses-features** : Précisent les configurations matérielles et logicielles requises.
 - **application** : Spécifie les métadonnées et les composants de l'application.
 - **uses-library** : Lien vers une bibliothèque partagée.
 - **activity** : Déclare une activité.
 - **intent-filter** : Spécifie les intentions que l'activité peut gérer.

Dossier res/values

- **colors.xml** : Définit les couleurs primaires, secondaires et personnalisées utilisées dans l'application.
- **dimens.xml** : Contient les dimensions des éléments d'interface, facilitant les changements globaux (ex : marges, hauteurs).
- **strings.xml** : Stocke les chaînes de texte pour éviter les chaînes codées en dur dans les layouts.
- **styles.xml** : Définit les thèmes et les styles personnalisés de l'application, avec un thème de base et des options de personnalisation.

build.gradle

- Automatise la compilation, le test, et le déploiement de l'application
- Gère les dépendances (bibliothèques et outils) et les configurations du projet (comme les versions du SDK et les options de build)

. Il existe deux types principaux de fichiers *build.gradle* :

1. **Top-level** : Partage les configurations globales entre tous les modules.
2. **Module-level** : Spécifie les détails propres à chaque module, comme les dépendances, versions, types de build...

Exercice

Exercice :

- Créer une application Android
- Ajouter une permission, une configuration matérielle (ex: la caméra) et une feature
- Ajouter une string et une couleur dans les fichiers correspondants

2. Différents fichiers du projet

- **Activités** : La couche de présentation de l'application, construite sur la classe Activity. Les activités gèrent l'interface utilisateur et répondent aux actions des utilisateurs.

```
public class ExempleActivity extends AppCompatActivity {  
  
    //code  
  
}
```

- **Services** : Composants en arrière-plan qui mettent à jour des données, déclenchent des notifications, et gèrent des tâches.

```
public class MyService extends Service {  
  
    //code  
  
}
```

Exercice

Exercice :

- Créer plusieurs activités depuis le menu de création et voir à quoi ils ressemblent dans leur layout
- Créer un service qui fait un log lors de la création et de la destruction
- Créer un service qui fait le calcul et l'affiche dans les logs

3. L'interface graphique

Les Views

- **TextView** : Affiche du texte à l'écran, idéal pour les labels et les messages.
- **ImageView** : Affiche des images dans l'interface utilisateur.
- **RecyclerView** : Affiche une liste d'éléments de manière optimisée, utilisé pour des listes volumineuses.
- **CardView** : Encadre le contenu dans une carte avec des ombres et des coins arrondis.
- **ScrollView** : Permet le défilement du contenu s'il dépasse la taille de l'écran.

Voici un aperçu des **contrôles principaux** en Android Java :

1. **Button** : Un bouton interactif que l'utilisateur peut cliquer.
2. **EditText** : Permet à l'utilisateur de saisir du texte, utilisé pour les formulaires.
3. **CheckBox** : Permet de sélectionner/désélectionner des options.
4. **RadioButton** : Choix exclusif entre plusieurs options dans un groupe.
5. **ToggleButton** : Contrôle de basculement entre deux états (activé/désactivé).
6. **Switch** : Comme un ToggleButton, mais avec un design moderne pour activer/désactiver une option.
7. **SeekBar** : Contrôle glissant permettant de sélectionner une valeur dans une plage définie.
8. **Spinner** : Menu déroulant pour sélectionner une option dans une liste.

Exercice

Exercice 1 :

1. Saisir les informations d'un utilisateur via un formulaire.

Instructions

1. **Formulaire de saisie :**
 - Ajoutez un champ `EditText` pour saisir le **nom** de l'utilisateur.
 - Ajoutez un champ `EditText` pour saisir le **prénom** de l'utilisateur.
 - Ajoutez une `CheckBox` pour indiquer si l'utilisateur souhaite recevoir une newsletter.
 - Ajoutez un groupe de `RadioButton` pour sélectionner le **genre** (Homme/Femme/Autre).
 - Ajoutez un `Switch` pour activer ou désactiver un mode premium.
 - Ajoutez un `Spinner` pour sélectionner une **catégorie d'image** (par exemple : "Animaux", "Paysage", "Technologie").
 - Ajoutez une `Button`.

Exercice

Bonus :

Afficher une image dans une `ImageView` en fonction d'une sélection effectuée dans un menu déroulant (`Spinner`).

Afficher un indicateur de progression (`ProgressBar`) lors du traitement des données.

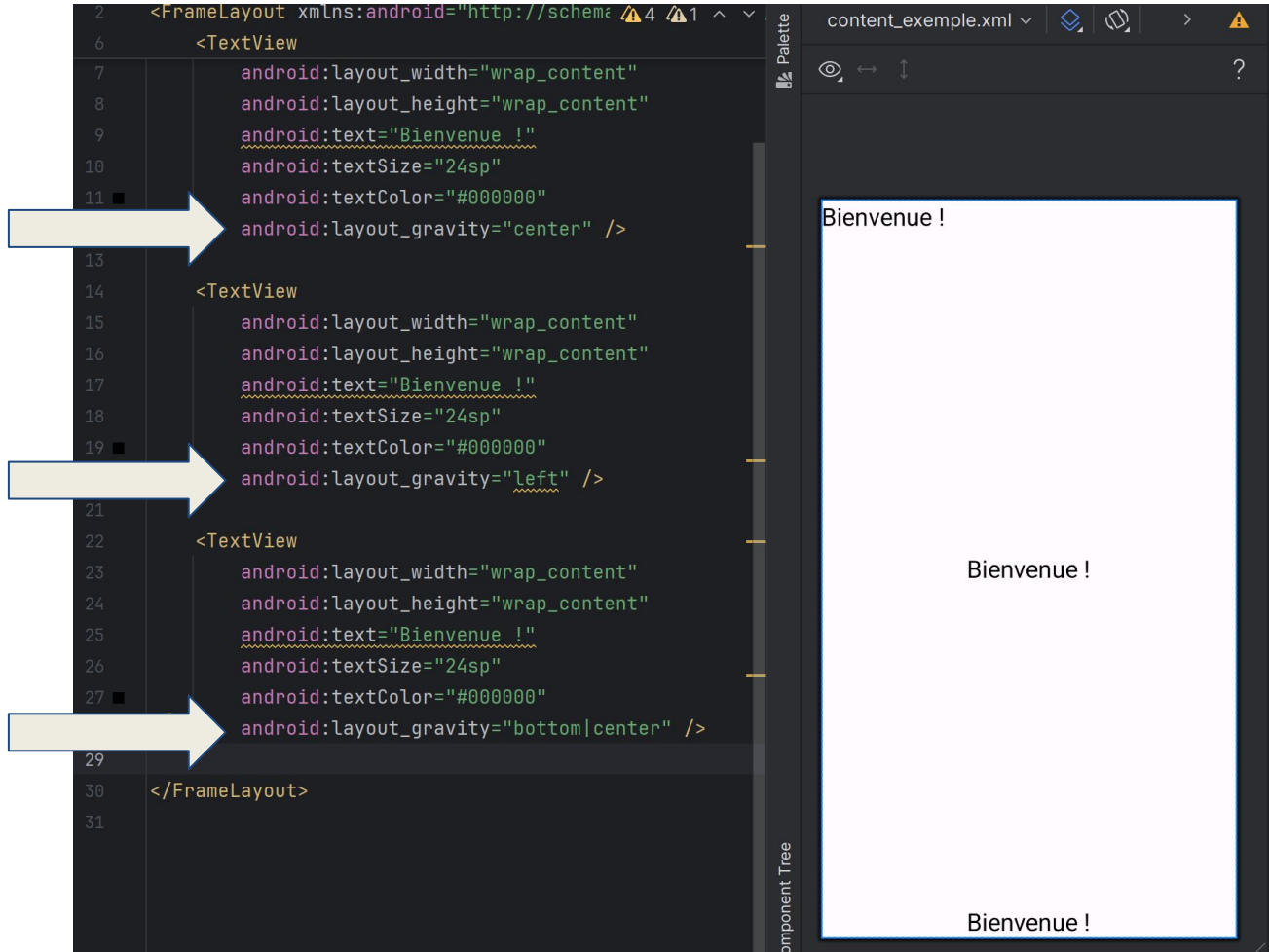
ListView

- **ListView** : afficher une liste défilante d'éléments dans Android. Il est particulièrement utile lorsqu'il faut afficher une grande quantité de données sous forme de liste. Il fonctionne avec un **Adapter**, qui convertit les données en vues que **ListView** peut afficher.

Les **Layout Managers** dans Android sont des extensions de la classe **ViewGroup**, utilisées pour positionner les vues enfants dans l'interface utilisateur. Les types courants incluent :

- **FrameLayout** : Positionne les vues dans un cadre, selon la valeur de l'attribut *layout_gravity*
- **LinearLayout** : Aligne les vues en ligne verticale ou horizontale.
- **RelativeLayout** : Positionne les vues relatives aux autres.
- **GridLayout** : Dispose les vues en grille.
- **ConstraintLayout** : De la même façon que RelativeLayout, il vient positionner les vues relativement aux autres, mais ici il faut préciser les 4 “côtés”

FrameLayout

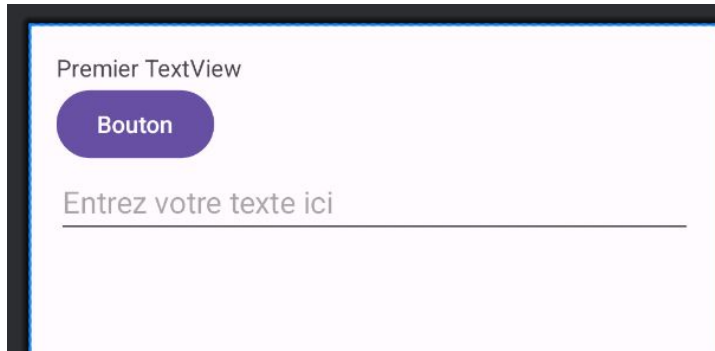


```

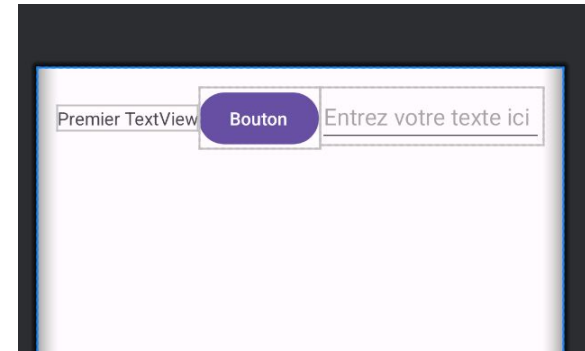
2   <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
6       <TextView
7           android:layout_width="wrap_content"
8           android:layout_height="wrap_content"
9           android:text="Bienvenue !"
10          android:textSize="24sp"
11          android:textColor="#000000"
12          android:layout_gravity="center" />
13
14      <TextView
15          android:layout_width="wrap_content"
16          android:layout_height="wrap_content"
17          android:text="Bienvenue !"
18          android:textSize="24sp"
19          android:textColor="#000000"
20          android:layout_gravity="left" />
21
22      <TextView
23          android:layout_width="wrap_content"
24          android:layout_height="wrap_content"
25          android:text="Bienvenue !"
26          android:textSize="24sp"
27          android:textColor="#000000"
28          android:layout_gravity="bottom|center" />
29
30  </FrameLayout>
31
  
```

The visual preview on the right shows a white rectangle representing the FrameLayout. Three yellow arrows indicate the positions of the TextViews: one at the top-left, one in the middle-left, and one at the bottom-center. The text "Bienvenue !" is visible in each of these positions.

`android:orientation="vertical"`



`android:orientation="horizontal"`



RelativeLayout

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">


    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Premier TextView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/textView1"
        android:layout_marginTop="54dp"
        android:text="Bouton"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignStart="@+id/button1"
        android:layout_marginLeft="120dp"
        android:text="Deuxieme texte"
        android:layout_centerHorizontal="true"
        />

```

content_exemple.xml



Component Tree

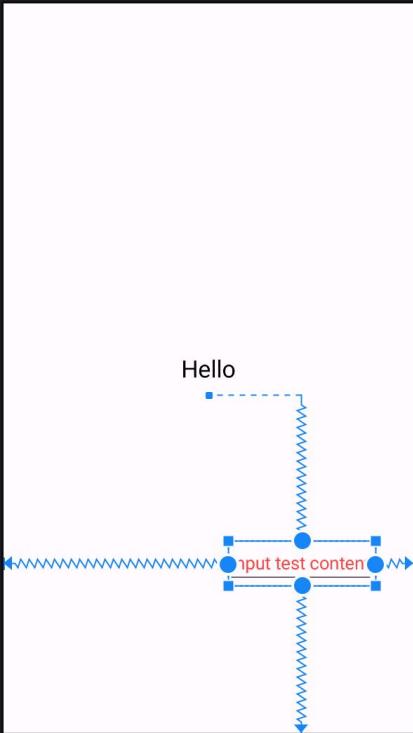
ConstraintLayout

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView"
        android:layout_width="80dp"
        android:layout_height="50dp"
        android:gravity="center"
        android:textSize="24dp"
        android:text="Hello"
        android:textColor="@android:color/black"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="input test content"
        android:inputType="phone"
        android:textColorHint="@android:color/ho_lo_red_light"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.85"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.5" />

```

activity_empty.xml



RelativeLayout VS ConstraintLayout

En quoi ConstraintLayout diffère-t-il de RelativeLayout ?

Disposition des contraintes	Disposition relative
Dans la disposition des contraintes, nous devons ajouter des contraintes à la vue sur les quatre côtés	Dans la disposition relative, nous pouvons simplement aligner notre composant d'interface utilisateur par rapport à son ID en utilisant les identifiants des composants d'interface utilisateur.
Dans la disposition Contrainte, si le composant de l'interface utilisateur n'est pas contraint, l'interface utilisateur ne ressemblera pas à celle de l'éditeur de conception.	Dans la disposition relative, l'interface utilisateur qui est réellement visible dans l'éditeur de conception d'Android Studio sera la même que celle que nous verrons dans l'application

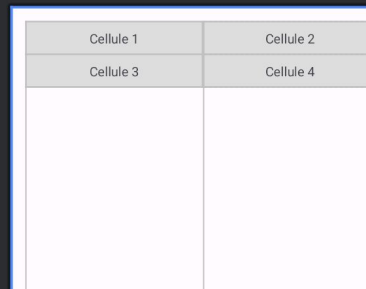
GridLayout

```

<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:rowCount="3"
    android:padding="16dp">

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Cellule 1"
        android:layout_columnWeight="1"
        android:gravity="center"
        android:background="#FFDDDDDD"
        android:padding="8dp"/>

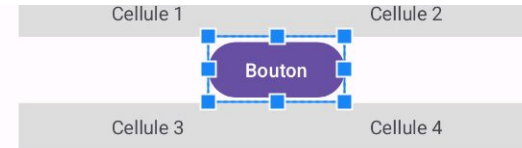
```



```

<Button
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="Bouton"
    android:layout_columnSpan="2"
    android:layout_gravity="center_horiz"
    android:padding="8dp"/>

```



Exercice

Exercice 1 :

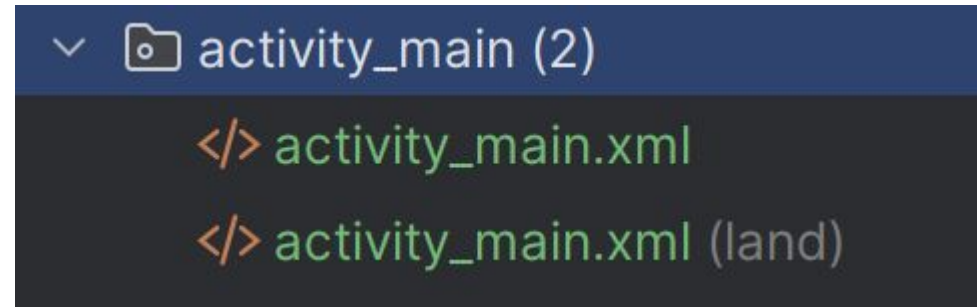
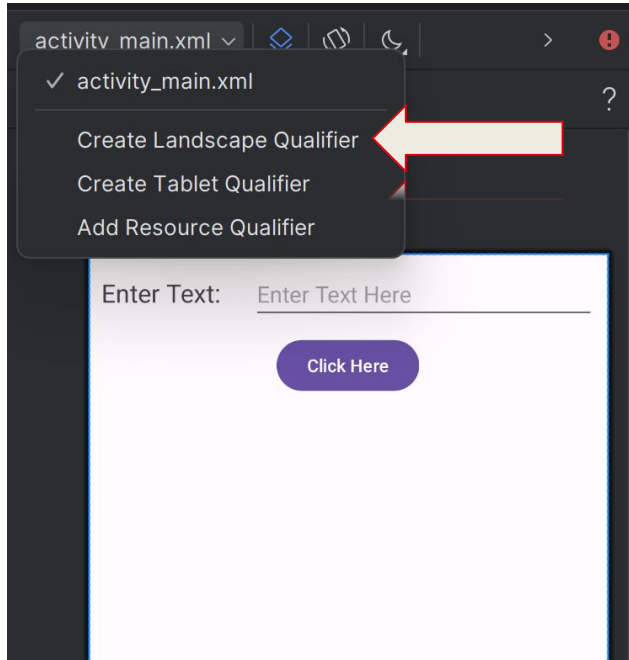
Créer une application "**Layout Explorer**"

Instructions

1. Créer une page pour chaque type de Layout et explorer les possibilités et spécificités de chaque.

Comment faire différents rendu selon l'orientation de l'écran ?

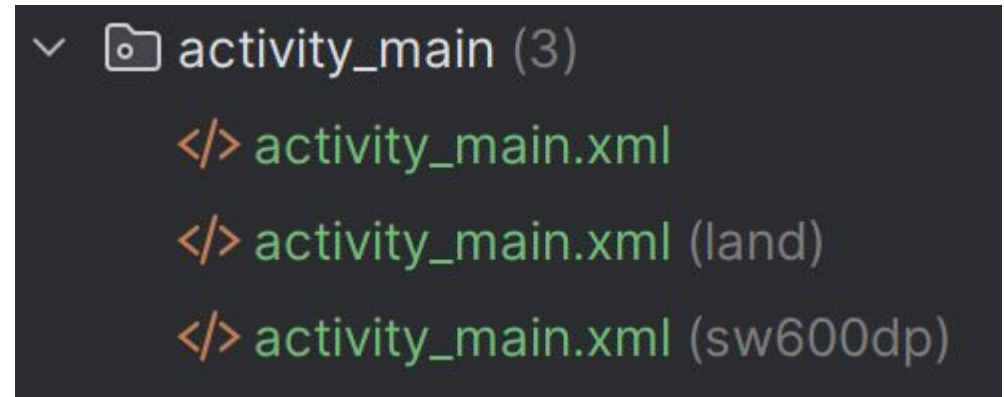
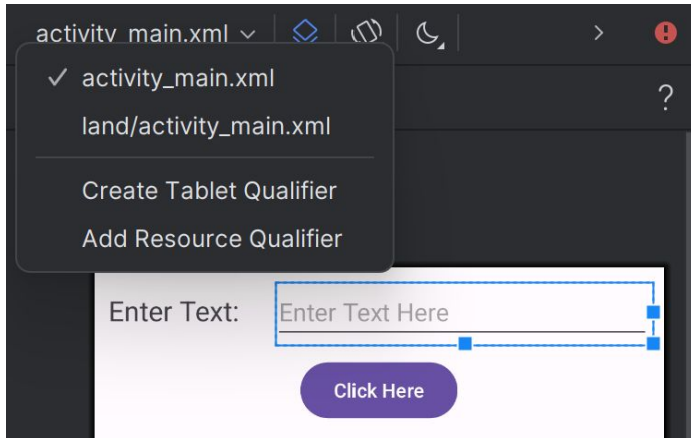
Créer des fichiers XML pour le landscape mode



Comment faire différents rendus selon la taille de l'écran ?

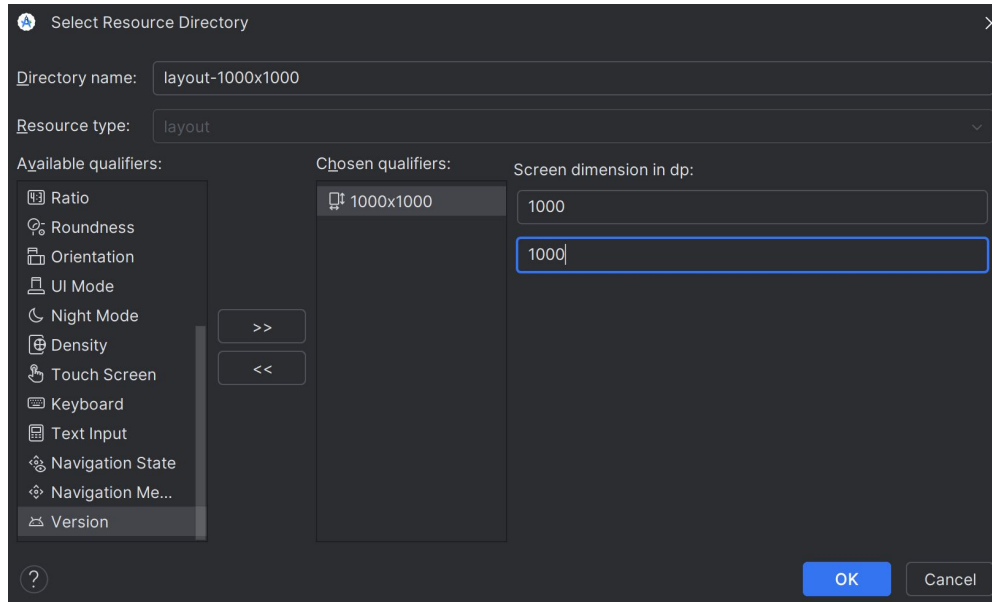
Créer des fichiers XML différents pour des téléphones, tablettes ou ordinateurs

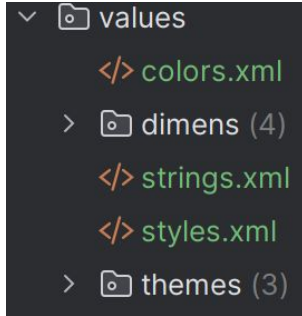
- **Téléphones** : `res/layout` (par défaut)
- **Tablettes** : `cliquer sur Create Tablet Qualifier`



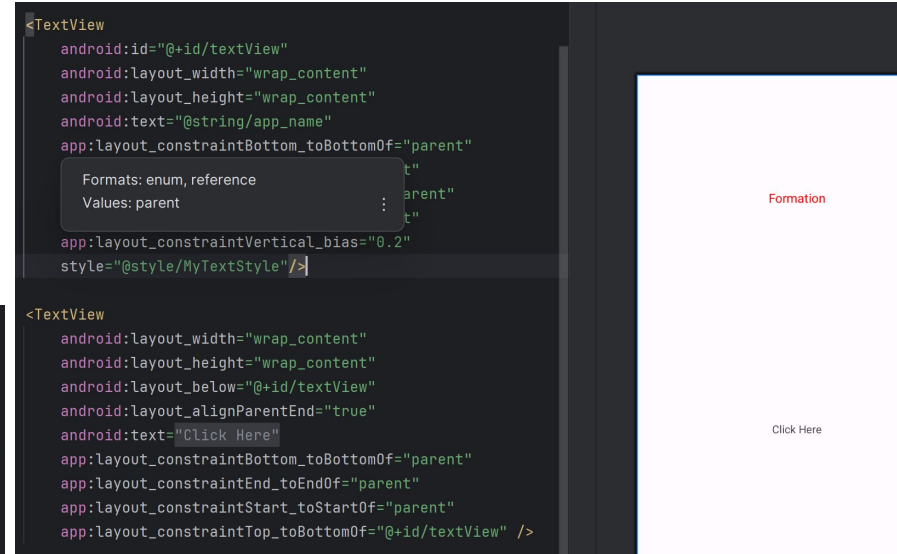
Comment faire différents rendus selon la taille de l'écran ?

Pour créer avec des dimensions custom

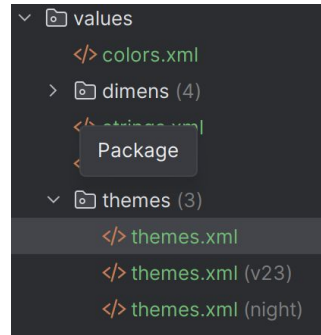




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MyTextStyle">
    <item name="android:textColor">#FF0000</item>
    <item name="android:textSize">16sp</item>
    <item name="android:padding">8dp</item>
  </style>
</resources>
```



Thème



```

M AndroidManifest.xml ×
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
14
15      <application
16          android:theme="@style/Theme.Formation"
  
```

```

    <activity
        android:theme="@style/AppTheme"
        android:name=".MainActivity"
  
```


Autres ressources

Vous pouvez inclure des polices personnalisées dans votre application. Vous pouvez les ajouter dans le dossier `res/font/`

Les images peuvent être ajoutées dans différents répertoires selon leur résolution pour s'adapter à différentes tailles d'écran :

- `res/drawable-mdpi/`
- `res/drawable-hdpi/`
- `res/drawable-xhdpi/`
- `res/drawable-xxhdpi/`
- `res/drawable-xxxhdpi/`

Les couleurs : `res/values/colors.xml`. Cela permet de centraliser les couleurs utilisées dans l'application.

Les animations : `res/anim/`. Vous pouvez créer des animations de translation, de rotation, de mise à l'échelle, etc.

Les menus : `res/menu/`. Cela permet de créer des menus d'options ou des menus contextuels pour vos activités.

Material Design

Nativement installé quand on crée un nouveau projet via Android Studio :

Dans build.gradle

```
dependencies {  
    implementation(libs.appcompat)  
    implementation(libs.material)
```

Dans themes.xml

```
<style name="Base.Theme.Formation" parent="Theme.Material3.DayNight.NoActionBar">
```

Liste des thèmes possible :

- Theme.MaterialComponents
- Theme.MaterialComponents.NoActionBar
- Theme.MaterialComponents.Light
- Theme.MaterialComponents.Light.NoActionBar
- Theme.MaterialComponents.Light.DarkActionBar
- Theme.MaterialComponents.DayNight
- Theme.MaterialComponents.DayNight.NoActionBar
- Theme.MaterialComponents.DayNight.DarkActionBar

Exercice

Exercice 1 :

Différents rendus selon l'orientation de l'écran

1. **Objectif** : Afficher un contenu différent en mode portrait et paysage.
2. **Instructions** :
 - Créez deux versions du layout principal :
 - Une pour le mode **portrait** avec une disposition verticale (exemple : une image au-dessus d'un texte).
 - Une pour le mode **paysage** avec une disposition horizontale (exemple : l'image à gauche et le texte à droite).
 - Placez les fichiers de layout dans les dossiers **res/layout** (portrait par défaut) et **res/layout-land** (pour paysage).
3. **Résultat attendu** : L'interface s'ajuste automatiquement à l'orientation de l'écran.

Exercice

Exercice 2 :

Différents rendus selon la taille d'écran

1. **Objectif** : Adapter la disposition pour les petits, moyens et grands écrans.
2. **Instructions** :
 - Créez plusieurs fichiers de layout :
 - Par exemple, pour un écran compact, affichez uniquement une image et un texte succinct.
 - Pour un écran large, ajoutez des boutons ou des éléments supplémentaires.
 - Utilisez les dossiers spécifiques à la taille :
 - `res/layout` (par défaut).
 - `res/layout-small`, `res/layout-large`, `res/layout-xlarge`.
3. **Résultat attendu** : L'application affiche des interfaces optimisées selon la taille de l'écran.

Exercice

Exercice 3 :

Objectif : Appliquer des styles pour uniformiser l'apparence des éléments UI (textes, boutons, etc.).

Instructions :

- Définissez un style dans le fichier `res/values/styles.xml` :
 - Un pour les `TextView` (exemple : police, couleur).
 - Un autre pour les boutons (exemple : forme, ombre).
- Appliquez les styles à différents éléments dans le layout principal.

Résultat attendu : Les éléments suivent un style cohérent défini globalement.

Exercice

Exercice 4 :

Utilisation de thèmes

1. **Objectif** : Modifier l'apparence globale de l'application grâce aux thèmes.
2. **Instructions** :
 - Créez deux thèmes personnalisés dans `styles.xml` :
 - Un thème clair avec des couleurs lumineuses.
 - Un thème sombre avec des couleurs atténuées.
 - Configurez un thème par défaut dans le fichier `AndroidManifest.xml`.
 - (Facultatif) Ajoutez un bouton pour permettre de basculer entre les deux thèmes dynamiquement.
3. **Résultat attendu** : L'apparence générale de l'application change avec le thème actif.

4. Intent et Fragment

Permet de passer d'une activité à une autre de manière explicite ou implicite

explicite :

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);  
startActivity(intent);
```

implicite (ici permet d'ouvrir le navigateur) :

```
Intent intent = new Intent(Intent.ACTION_VIEW);  
intent.setData(Uri.parse("https://www.example.com"));  
startActivity(intent);
```


Intent fréquents :

- ACTION_SET_ALARM
- ACTION_SET_TIMER
- ACTION_INSERT (permet d'ajouter un evenement à l'agenda)
- ACTION_IMAGE_CAPTURE
- ACTION_VIDEO_CAPTURE
- ...

Pour passer des informations supplémentaires (par ex à d'autres activités de l'application), on utilise des Extras :

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);  
intent.putExtra("message", "Hello from MainActivity!");  
startActivity(intent);
```

//Dans la seconde activité

```
String message = getIntent().getStringExtra("message");
```

Intent-filter (Manifest)

Les **intent-filters** sont utiles dans plusieurs situations :

- **Lancement d'activités externes** : Par exemple, si vous souhaitez permettre à votre application d'ouvrir un site web ou une autre application
- **Gestion de l'intention de partage** : Si votre application peut partager du contenu (comme des images ou du texte)
- **Recevoir des notifications ou des données** : Si votre application doit recevoir des données (comme des SMS ou des notifications) d'autres applications
- **Écoute d'intentions spécifiques** : Pour que votre application réagisse à des actions spécifiques du système (comme une mise à jour du réseau)

Exercice

Exercice 1 :

Lancer une nouvelle activité

1. **Objectif** : Créer une application avec deux activités. La première permet de naviguer vers la seconde via un bouton.
2. **Instructions** :
 - Créez deux activités : `MainActivity` et `SecondActivity`.
 - Ajoutez un bouton dans `MainActivity` pour naviguer vers `SecondActivity`.
 - Utilisez un **Intent explicite** pour démarrer `SecondActivity`.
3. **Résultat attendu** : En cliquant sur le bouton, la seconde activité s'affiche.

Exercice

Exercice 2 :

Passer des données entre activités

1. **Objectif :** Passer des informations de `MainActivity` à `SecondActivity` et les afficher.
2. **Instructions :**
 - Dans `MainActivity`, utilisez un champ de texte pour permettre à l'utilisateur de saisir un nom.
 - Ajoutez un bouton pour envoyer ce nom à `SecondActivity` via un **Intent explicite** avec des extras.
 - Affichez le nom dans un `TextView` de `SecondActivity`.
3. **Résultat attendu :** L'application affiche le nom saisi sur la seconde activité.

Exercice

Exercice 2 :

Appeler une activité externe

1. **Objectif :** Utiliser un **Intent** pour ouvrir un navigateur web.
2. **Instructions :**
 - Ajoutez un bouton dans votre activité principale.
 - Configurez l'intent pour ouvrir une URL (exemple : <https://www.google.com>).
3. **Résultat attendu :** En cliquant sur le bouton, le navigateur s'ouvre avec l'URL spécifiée.

Fragment

Un Fragment représente une partie réutilisable de l'UI de votre application.

Les fragments ne peuvent pas être autonomes. Ils doivent être hébergés par une activité ou un autre fragment.

Avantages des fragments :

- **Réutilisabilité** : les fragments peuvent être utilisés dans plusieurs activités.
- **Gestion d'écrans multiples** : idéale pour les interfaces adaptatives (ex. tablettes).
- **Modularité** : permet de séparer le code pour une meilleure organisation.

Cas d'utilisation courants :

- Création d'interfaces dynamiques (ex. : affichage en liste et détail).
- Gestion de l'interface utilisateur dans des écrans multi-tâches.

Exercice

Exercice 1 :

Création d'un Fragment Simple

Objectif : Créer un fragment qui affiche un texte simple.

1. Créez une nouvelle classe `MonFragment` qui hérite de `Fragment`.
2. Dans la méthode `onCreateView()`, affichez un texte "Bienvenue dans le fragment!".
3. Ajoutez ce fragment dans une activité existante.

Exercice

Exercice 2 :

Utilisation de Fragment dans une Activité

Objectif : Afficher plusieurs fragments dans une même activité.

1. Créez deux fragments : un fragment qui affiche un titre et un autre qui affiche une liste d'éléments.
2. Ajoutez ces deux fragments dans une activité à l'aide du `FragmentManager`.
3. Utilisez le `FrameLayout` pour organiser l'espace où les fragments seront placés.

Exercice

Exercice 3 :

Interaction entre un Fragment et son Activité

Objectif : Faire communiquer un fragment avec son activité hôte.

1. Créez un fragment qui possède un bouton.
2. Dans l'activité, récupérez ce fragment et mettez en place une interaction : lorsque le bouton est cliqué, affichez un message dans l'activité.
3. Utilisez une interface pour communiquer entre le fragment et l'activité.

Exercice

Exercice 4 :

Remplacer un Fragment Dynamique

Objectif : Remplacer un fragment à la volée dans l'application.

1. Créez deux fragments : l'un affiche un texte et l'autre une image.
2. Dans l'activité, ajoutez un bouton qui, lorsqu'il est cliqué, remplace le fragment affiché par un autre.
3. Utilisez la méthode `FragmentManager.replace()` pour effectuer ce remplacement.

5. Contrôles avancées

Liste des contrôles avancées

Composants UI Android : Dialog, WebView, ActionBar et Navigation avancée

Dialog

Définition

- Fenêtre modale temporaire affichée au-dessus de l'interface utilisateur active.
- Utilisée pour des interactions brèves comme des alertes, confirmations ou choix.

Exemples courants :

- **AlertDialog** : Afficher des messages ou des options simples (OK/Annuler).
- **DatePickerDialog** : Sélectionner une date.

Avantages

- Intuitif pour l'utilisateur.
- Ne perturbe pas l'activité principale.

Liste des contrôles avancées

Composants UI Android : Dialog, WebView, ActionBar et Navigation avancée

WebView

Définition

- Composant pour afficher des contenus web directement dans une application.
- Équivaut à un navigateur intégré.

Fonctionnalités clés :

- Affichage de pages HTML/CSS.
- Interaction avec JavaScript.

Cas d'utilisation

- Afficher des articles web, documents ou cartes intégrées.
- Fournir une expérience utilisateur homogène sans quitter l'application.

Liste des contrôles avancées

Composants UI Android : Dialog, WebView, ActionBar et Navigation avancée

ActionBar

Définition

- Barre supérieure de l'écran.
- Propose des actions et des options de navigation contextuelles.

Caractéristiques

- Bouton retour (Up Navigation).
- Menus d'options via trois points.
- Icône personnalisable pour représenter l'application.

Avantages

- Améliore l'expérience utilisateur.
- Centralise les actions essentielles.

Liste des contrôles avancées

Composants UI Android : Dialog, WebView, ActionBar et Navigation avancée

NavigationDrawer

Définition

- Menu latéral coulissant.
- Permet un accès rapide à différentes sections de l'application.

Caractéristiques

- Déclenché par un bouton ou un glissement latéral.
- Contient une liste d'options ou de destinations.

Cas d'utilisation

- Applications complexes avec plusieurs fonctionnalités (ex. : Gmail).

Liste des contrôles avancées

Composants UI Android : Dialog, WebView, ActionBar et Navigation avancée

NavigationDrawer

Définition

- Menu latéral coulissant.
- Permet un accès rapide à différentes sections de l'application.

Caractéristiques

- Déclenché par un bouton ou un glissement latéral.
- Contient une liste d'options ou de destinations.

Cas d'utilisation

- Applications complexes avec plusieurs fonctionnalités (ex. : Gmail).

Exercice

Exercice 1 :

Créer un **AlertDialog** pour confirmer une action

1. Créez une activité avec un bouton nommé "**Supprimer l'élément**".
2. Lorsque l'utilisateur clique sur le bouton :
 - Affichez un **AlertDialog** avec le message "*Voulez-vous vraiment supprimer cet élément ?*".
 - Ajoutez deux boutons : "**Oui**" et "**Non**".
 - Si l'utilisateur clique sur "**Oui**", affichez un **Toast** : "*Élément supprimé*".
 - Si l'utilisateur clique sur "**Non**", fermez simplement le dialog.

Bonus :

- Ajoutez une icône au Dialog (par exemple, une corbeille).
- Créer une vraie liste, et ajouter un bouton "Ajouter un élément" et pouvoir le supprimer

Exercice

Exercice 2 :

Charger une page web dans une WebView

1. Créez une activité contenant une **WebView** en plein écran.
2. Configurez la **WebView** pour charger la page suivante : <https://www.wikipedia.org>.
3. Ajoutez un champ de texte pour saisir une URL et un bouton "**Charger**".
 - Lorsqu'un utilisateur entre une URL, affichez cette page dans la WebView.

Bonus :

- Activez **JavaScript** pour interagir avec des pages dynamiques.
- Gérer les erreurs de chargement en affichant un **Toast** si l'URL est invalide.

Exercice

Exercice 3 :

Ajouter des actions à l'ActionBar

1. Créez une activité avec une **ActionBar**.
2. Ajoutez deux actions au menu :
 - **"Partager"** : Affiche un Toast avec le message *"Partage en cours..."*.
 - **"Rechercher"** : Affiche un **Toast** avec le message *"Recherche en cours..."*.

Bonus :

- Ajoutez un **icone** à chaque action.
- Créez un bouton retour (Up Navigation) qui ramène à une autre activité.

Exercice

Exercice 4 :

Implémenter un **NavigationDrawer**

1. Créez une activité avec un **NavigationDrawer**.
2. Ajoutez les sections suivantes dans le menu :
 - **Accueil**
 - **Profil**
 - **Paramètres**
3. Lorsqu'une section est sélectionnée :
 - Affichez un **Toast** avec le nom de la section.

Bonus :

- Ajoutez une **header view** personnalisée (par exemple, un avatar et un nom d'utilisateur).
- Affichez un nouveau **Fragment** pour chaque section sélectionnée.

API

Définition

- Les **API réseaux** permettent à une application de communiquer avec un serveur via Internet.
- Elles utilisent des protocoles comme HTTP ou WebSockets.

Exemples d'utilisation :

- **HTTP/REST** : Envoi de requêtes GET/POST pour récupérer ou envoyer des données.
- **WebSockets** : Maintenir une connexion bidirectionnelle pour des applications en temps réel comme le chat.

Outils courants :

- **HttpURLConnection** (basique, intégré à Java).

Multithreading

Définition

- Technique permettant d'exécuter plusieurs tâches en parallèle.
- Empêche le **thread principal** (UI) d'être bloqué par des opérations longues.

Exemples courants :

- Charger des données depuis une API en arrière-plan.
- Gérer des téléchargements ou des traitements lourds.

Outils :

- **Threads** natifs de Java.

Appel de services Web

Définition

- Interaction avec des **services web** pour envoyer ou recevoir des données.
- Repose sur des requêtes HTTP (GET, POST, PUT, DELETE).

Outils populaires :

- **Retrofit** : Simple et efficace pour travailler avec des API RESTful.
- **Volley** : Gère bien les requêtes fréquentes et les images.

Exemple avec Retrofit :

- Déclarer une interface pour l'API.
- Configurer le client Retrofit.
- Envoyer une requête et traiter la réponse.

Exercice

Exercice 1 :

Appeler une API publique pour récupérer des données

1. Utilisez une API publique (par exemple, l'API de Pokémon : <https://pokeapi.co>).
2. Créez une application Android avec un bouton **"Charger Pokémon"**.
3. Lors du clic, effectuez une requête **GET** pour récupérer les détails d'un Pokémon (par exemple, Pikachu).
4. Affichez les informations suivantes dans une **TextView** :
 - Nom
 - Type principal

Bonus :

- Ajoutez un champ de texte pour que l'utilisateur puisse rechercher un Pokémon en entrant son nom.
- Gérer les erreurs réseau et afficher un **Toast** en cas de problème.

Exercice

Exercice 2 :

Télécharger un fichier en arrière-plan

1. Créez une activité avec :
 - Un bouton "**Télécharger fichier**".
 - Une **ProgressBar** pour afficher l'état du téléchargement.
2. Lorsque l'utilisateur clique sur le bouton :
 - Téléchargez un fichier d'exemple
 - Effectuez cette opération dans un **Thread**.
 - Une fois le téléchargement terminé, affichez un message dans un **Toast** : "*Téléchargement terminé !*".
3. Actualisez la **ProgressBar** pendant le téléchargement.

Bonus :

- Affichez l'image téléchargée dans un **ImageView**.
- Gérer les interruptions de réseau en affichant une erreur.

Exercice

Exercice 3 :

Implémenter Retrofit pour appeler une API

1. Configurez Retrofit dans votre projet :
 - Ajoutez les dépendances nécessaires dans `build.gradle`.
 - Créez une interface pour appeler l'API <https://jsonplaceholder.typicode.com/posts>.
2. Créez une activité qui :
 - Affiche un bouton **"Charger les Posts"**.
 - Lors du clic, récupère les **posts** via une requête GET.
3. Affichez les informations suivantes dans une **RecyclerView** :
 - Titre du post.
 - Contenu (body).

Bonus :

- Ajoutez une barre de chargement pendant la récupération des données.
- Implémentez un système de pagination pour charger les posts par blocs de 10.

6. Persistance de données

Persistance des données

- **Stockage clé-valeur** : Utilise `SharedPreferences` pour stocker des données simples sous forme de paires clé-valeur. Idéal pour conserver les préférences utilisateur (ex. : thème, login, etc.).
- **Système de fichiers** : Pour stocker des fichiers dans la mémoire interne ou externe. Vous pouvez utiliser `FileOutputStream` et `FileInputStream` pour lire/écrire des fichiers.
- **SQLite** : Base de données relationnelle embarquée utilisée pour stocker des données structurées localement.
- **Room** : Une bibliothèque ORM pour SQLite, facilitant les opérations de base de données.

- **ContentProvider** : Permet aux applications de gérer, de partager des données et d'accéder à des données stockées dans des bases de données SQLite, des fichiers, ou même des données réseau.

```
public class ContentProviderExemple extends ContentProvider {  
    public void onCreate() {}  
    //code  
}
```

Exercice

Exercice 1 :

Utilisation de SharedPreferences

Objectif : Stocker et récupérer des préférences utilisateur simples avec SharedPreferences.

1. Créez une activité qui permet à l'utilisateur de choisir un thème (clair ou sombre) à l'aide d'un `RadioButton`.
2. Utilisez `SharedPreferences` pour enregistrer le choix de l'utilisateur.
3. Lors du lancement de l'application, l'activité doit récupérer cette préférence et appliquer le thème choisi.

Exercice

Exercice 2 :

Stockage et Lecture de Fichiers dans le Système de Fichiers

Objectif : Lire et écrire des données dans un fichier dans la mémoire interne.

1. Créez un fichier texte dans la mémoire interne en utilisant `FileOutputStream`.
2. Ajoutez des données simples (ex. : un nom d'utilisateur) dans le fichier.
3. Lisez le fichier avec `FileInputStream` et affichez son contenu dans un `TextView`.

Exercice

Exercice 3 :

Stockage de Données avec SQLite

Objectif : Utiliser SQLite pour stocker des données structurées (ex. : une liste de contacts).

1. Créez une base de données SQLite avec une table **contacts** (nom, téléphone, email).
2. Insérez plusieurs contacts dans la base de données.
3. Créez une interface qui permet de récupérer et d'afficher la liste des contacts à partir de la base de données.

Exercice

Bonus :

Utilisation de Room pour Manipuler une Base de Données

Objectif : Utiliser la bibliothèque Room pour interagir avec une base de données SQLite.

1. Créez une entité `Contact` avec des champs comme `id`, `nom`, `numéro` et `email`.
2. Créez un DAO pour ajouter, supprimer et récupérer des contacts.
3. Utilisez Room pour insérer quelques contacts dans la base de données et afficher la liste dans une activité.

7. Fonctionnalités multimédia

Outils multimédia

- **Affichage de documents** : Utilisation de **WebView** ou d'applications tierces pour visualiser des fichiers PDF, Word, etc.
- **Prise de photo** : Accès à l'appareil photo via **Camera API** ou **CameraX**, permettant de capturer des images.
- **Son et micro** : Enregistrement et lecture audio à l'aide de **MediaRecorder** et **MediaPlayer**, et gestion des flux audio.
- **Envoi et réception de SMS** : Utilisation de l'API SMS pour envoyer des messages via des applications tierces ou en mode natif.
- Fournisseurs de **géolocalisation** et API Google Maps V2

Exercice

Exercice 1 :

Affichage de Documents avec WebView

Objectif : Utiliser un **WebView** pour afficher un fichier PDF.

1. Créez une activité qui contient un **WebView**.
2. Chargez un fichier PDF local ou distant dans ce **WebView**. Vous pouvez utiliser une URL ou un fichier stocké dans la mémoire interne de l'appareil.
3. Assurez-vous que le fichier est affiché correctement dans l'interface.

Exercice

Exercice 2 :

Prise de Photo avec CameraX

Objectif : Capturer une image à l'aide de CameraX.

1. Créez une interface simple permettant de capturer une photo en appuyant sur un bouton.
2. Utilisez CameraX pour accéder à l'appareil photo et capturer l'image.
3. Affichez l'image capturée dans un `ImageView`.

Exercice

Exercice 3 :

Enregistrement Audio avec MediaRecorder

Objectif : Enregistrer un fichier audio à l'aide de **MediaRecorder**.

1. Créez un bouton "Démarrer l'enregistrement" et un bouton "Arrêter l'enregistrement".
2. Utilisez **MediaRecorder** pour démarrer l'enregistrement lorsqu'on appuie sur le premier bouton et l'arrêter lorsque le second bouton est pressé.
3. Sauvegardez le fichier audio dans la mémoire interne et affichez un message de confirmation.

Exercice

Exercice 4 :

Lecture Audio avec MediaPlayer

Objectif : Lire un fichier audio à l'aide de MediaPlayer.

1. Créez une interface avec un bouton pour lire un fichier audio stocké dans les ressources ou la mémoire interne.
2. Utilisez MediaPlayer pour lire le fichier audio lorsque le bouton est pressé.
3. Ajoutez des boutons pour mettre en pause, reprendre la lecture et arrêter le son.

Exercice

Exercice 5 :

Envoi de SMS via l'API SMS

Objectif : Envoyer un SMS à un numéro spécifique.

1. Créez un formulaire permettant à l'utilisateur d'entrer un message et un numéro de téléphone.
2. Utilisez l'API SMS pour envoyer le message au numéro indiqué.
3. Assurez-vous d'ajouter les permissions nécessaires dans le manifeste et de gérer les demandes d'autorisations à l'exécution.

Fournisseurs de géolocalisation

Définition

- Les fournisseurs de géolocalisation permettent de récupérer la position actuelle d'un appareil.
- Utilisés pour des applications comme le suivi GPS, les recommandations locales, ou la navigation.

Fournisseurs courants :

1. GPS

- Haute précision.
- Nécessite un ciel dégagé.
- Consomme beaucoup de batterie.

2. Réseaux mobiles (Cell Tower)

- Précision moyenne.
- Fonctionne en intérieur.
- Consomme moins de batterie.

3. Wi-Fi

- Précision élevée en zones couvertes.
- Dépend de l'accès à un réseau Wi-Fi.

API pour la géolocalisation :

- **Fused Location Provider** (recommandé) : Combine automatiquement les fournisseurs pour optimiser la précision et la consommation de batterie.
- Google Maps API V2

Obtenir une clé API Google Maps :

- Accédez à la console Google Cloud.
- Activez l'API Maps et générez une clé API.

Ajouter la clé à votre projet Android :

- Placez la clé dans le fichier `AndroidManifest.xml`.

Ajouter la dépendance à `build.gradle`

Afficher une carte :

- Ajoutez un **MapFragment** ou une **MapView** à votre activité.
- Configurez et chargez la carte dans le callback `onMapReady()`.

Ajouter un marqueur

Afficher la position actuelle

Exercice 1 :

Récupérer la position actuelle et l'afficher dans un Toast

1. Créez une application avec un bouton **"Obtenir ma position"**.
2. Lors du clic :
 - Utilisez le **FusedLocationProviderClient** pour récupérer les coordonnées latitude et longitude de l'utilisateur.
 - Affichez ces coordonnées dans un **Toast**.

Bonus :

- Ajoutez une vérification des permissions d'accès à la localisation.
- Gérez les erreurs, par exemple, si la localisation est désactivée.

Exercice 2 :

API Google Maps V2

Exercice : Ajouter une carte avec un marqueur

1. Créez une application avec une carte interactive.
2. Affichez un marqueur sur une position spécifique, par exemple :
 - **Latitude** : 40.7128
 - **Longitude** : -74.0060 (New York).

Bonus :

- Ajoutez un bouton "**Localiser ma position**" :
 - Lors du clic, récupérez la position actuelle et affichez-la sur la carte avec un marqueur.

Bonus :

Application de géolocalisation avec carte

1. Créez une application qui :
 - Affiche une carte centrée sur la position actuelle de l'utilisateur.
 - Ajoute un marqueur là où l'utilisateur clique sur la carte.
 - Trace un chemin (polyline) entre deux marqueurs ajoutés.

Bonus :

- Ajoutez un champ de recherche pour naviguer vers un lieu spécifique.
- Affichez des informations supplémentaires sur les marqueurs dans une fenêtre contextuelle (info window).

8. Déploiement

Prérequis au déploiement :

- **Google Play** : Inscription à Google Play Console pour publier des applications. Respect des politiques de contenu, de sécurité et de confidentialité.
- **En entreprise** : Compréhension des exigences de déploiement spécifiques, telles que la sécurité des données, l'intégration avec les systèmes d'entreprise et le support technique.
- **Internationalisation** : Adaptation des applications pour différents marchés, y compris la localisation linguistique et la prise en charge des formats de date et d'heure, ainsi que des devises.

9. Introduction à Kotlin

Introduction à Kotlin Android

- **Intérêts d'utiliser Kotlin en Android** : Kotlin offre une syntaxe concise, une interopérabilité avec Java, des fonctionnalités modernes qui simplifient le développement d'applications Android.
- **Éléments de syntaxe** : Compréhension des types de données, structures de contrôle (if, when, loops), et la gestion des nulls, ce qui réduit les erreurs courantes.
- **Les classes et les objets** : Apprentissage de la programmation orientée objet avec des classes, objets, héritage, et interfaces.
- **Android Studio pour Kotlin** : Installation et configuration du plug-in Kotlin dans Android Studio pour le développement d'applications.

Dossier pédagogique

- Feuilles d'émargement signées pour chaque journée.
- Feuilles d'émargement signées pour les passages de certifications (si certifications)
- Évaluations formateur

Formation suivante

- La formation Android avancée, pourquoi ?

<https://www.m2iinformation.fr/formation-atelier-android-avance/AND-AV/>

- La formation Kotlin, pourquoi ?

<https://www.m2iinformation.fr/formation-kotlin-developpement-mobile-android-et-ios-avec-kmp-kotlin-multiplatform/KOTL-DEV/>

Questions / réponses

- Revenons sur les questions hors plan de cours que vous m'avez posé durant la formation pour y répondre



Votre formateur

Clément Hamon

Formateur externe M2I

clement.hamon35@gmail.com

<https://www.linkedin.com/in/clément-hamon-135485209/>

**Merci d'avoir suivi cette formation
M2I et à très bientôt !**



Bilan formation et remerciements

- Merci d'avoir participé à cette formation M2I.
- Envoie du Bilan formation.

Annexe TP validation des acquis

L'objectif est de créer une application **"Task Manager"**, une application Android permettant de gérer des tâches avec persistance des données, navigation multi-écrans, et quelques fonctionnalités avancées.

Objectif final :

- Une application Android où l'utilisateur peut :
 - Ajouter, modifier, supprimer des tâches.
 - Voir la liste des tâches avec une interface multi-écrans.
 - Géolocaliser une tâche et l'afficher sur une carte.
 - Sauvegarder les tâches dans une base SQLite.

Annexe TP validation des acquis

Itération 1 : Création du projet et mise en place de l'interface de base

Objectifs :

- Configurer le projet Android.
- Implémenter une liste des tâches statiques.

Étapes :

1. **Création du projet :**
 - Démarrer un nouveau projet Android dans Android Studio avec un écran principal (`MainActivity`).
2. **Interface de liste statique :**
 - Ajouter une `ListView` pour afficher une liste de tâches statiques.
 - Créer une classe `Task` avec des attributs simples (`title`, `description`, `dueDate`).
3. **TP :**
 - Remplir la liste avec des données fictives et afficher les tâches dans la `ListView`.
4. **Bonus :** Mettre en place un style et gérer les différentes orientations et tailles

Annexe TP validation des acquis

Itération 2 : Ajout de tâches avec navigation multi-écrans

Objectifs :

- Ajouter la fonctionnalité de création de tâches.
- Naviguer entre plusieurs écrans.

Étapes :

1. **Écran de création de tâches :**
 - Ajouter une activité `AddTaskActivity` pour permettre à l'utilisateur d'ajouter une tâche.
 - Ajouter un formulaire (champs pour le titre, la description, et la date d'échéance).
2. **Navigation entre écrans :**
 - Ajouter un bouton dans `MainActivity` pour naviguer vers `AddTaskActivity`.
 - Récupérer les données du formulaire avec un `Intent` et les transmettre à l'activité principale.
3. **TP :**
 - Ajouter une tâche via le formulaire et afficher cette tâche dans la liste de `MainActivity`.

Annexe TP validation des acquis

Itération 3 : Persistance des données avec SQLite

Objectifs :

- Sauvegarder les tâches dans une base de données locale.
- Charger les tâches sauvegardées au démarrage.

Étapes :

1. **Mise en place de SQLite :**
 - Créer une classe `TaskDatabaseHelper` pour gérer la base de données SQLite.
 - Ajouter des méthodes pour insérer, lire, mettre à jour et supprimer des tâches.
2. **Intégration avec l'application :**
 - Modifier `MainActivity` pour charger les tâches depuis SQLite au démarrage.
 - Sauvegarder les nouvelles tâches dans SQLite depuis `AddTaskActivity`.
3. **TP :**
 - Persister les données dans la base SQLite et vérifier leur affichage après redémarrage de l'application.

Annexe TP validation des acquis

Itération 4 : Géolocalisation et Google Maps

Objectifs :

- Ajouter une localisation pour chaque tâche.
- Afficher les tâches sur une carte.

Étapes :

1. **Ajout de localisation :**
 - Modifier le formulaire de création de tâches pour inclure la position GPS actuelle.
 - Utiliser les API de géolocalisation pour récupérer les coordonnées GPS.
2. **Carte avec Google Maps :**
 - Ajouter une activité **MapActivity** avec l'intégration de l'API Google Maps.
 - Placer des marqueurs sur la carte pour chaque tâche géolocalisée.
3. **TP :**
 - Créer une tâche avec une localisation et la visualiser sur une carte.

Annexe TP validation des acquis

Itération 5 : Améliorations et fonctionnalités avancées

Objectifs :

- Ajouter des notifications pour les tâches à échéance.
- Ajouter la possibilité de modifier/supprimer des tâches.

Étapes :

1. **Notifications :**
 - Utiliser l'API `AlarmManager` pour planifier des notifications locales lorsque la date d'échéance d'une tâche approche.
2. **Modification et suppression :**
 - Ajouter une option dans la liste pour modifier ou supprimer une tâche.
 - Mettre à jour la base SQLite en conséquence.
3. **TP :**
 - Ajouter une tâche avec une échéance et vérifier la notification.
 - Modifier ou supprimer une tâche existante.