PROGRAMMATION MOBILE

Introduction à

ANDROID

charles.meunier@u-bourgogne.fr

I PRÉSENTATION DU COURS

OBJECTIFS

Android Studio
Prendre en main l'environnement de développement intégré (EDI) conçu pour Android

Interface utilisateur
Être en mesure de créer une interface utilisateur simple.

Capteurs embarqués
Accéder aux informations renvoyées par les capteurs présents sur le périphérique Android.

Stockage de données
Enregistrer des données localement, dans la mémoire du périphérique Android.

API Rest
Se connecter et échanger des informations à distance à l'aide d'une API Rest.

La base pour créer des applications Android

ENSEIGNEMENTS

1	**CM** Pour poser les bases
3	**TD** Pour découvrir l'environnement Android
3	**TP** Pour acquérir les dernières notions utiles au projet

ÉVALUATIONS

Évaluation théorique
Aléatoirement au début des séances de TD/TP.

Évaluation pratique
Mise en pratique des connaissances acquises sous la forme d'un projet.

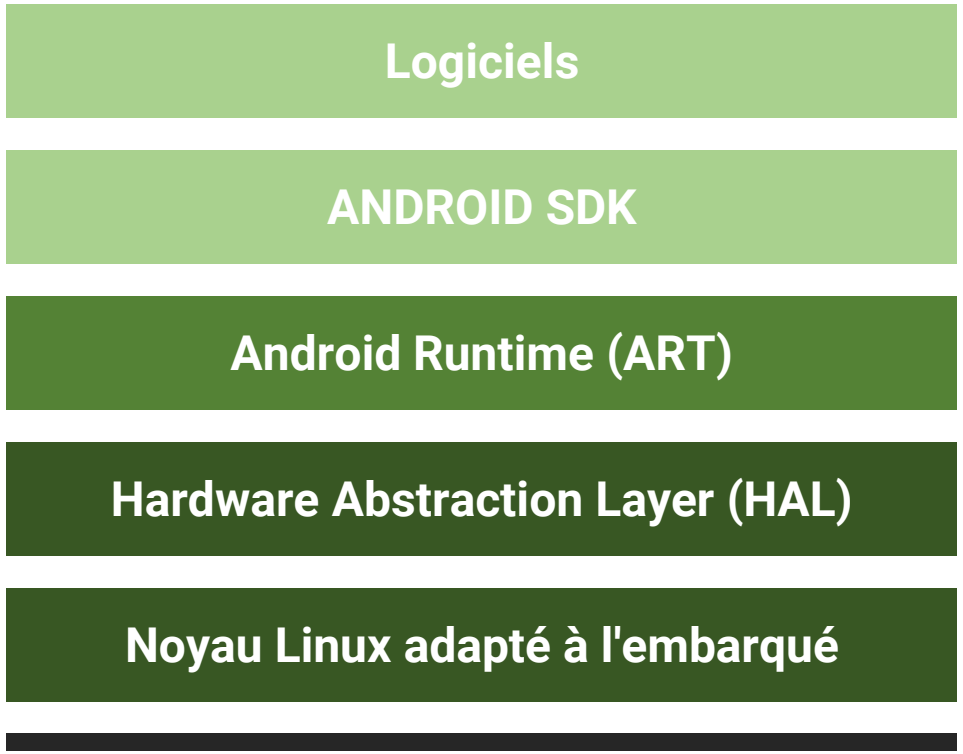
II ANDROID

PILE ANDROID

Ensemble de **logiciels indispensables au fonctionnement d'Android** : Luncher, Téléphone, Messagerie, Navigateur, Contacts, Paramètres...

Machine virtuelle en charge de l'exécution des applications Android. Utilise la compilation **Ahead of Time** (AOT) pour convertire le bytecode en code machine dès l'installation.

Android repose sur un **Noyau Linux légèrement modifié** pour les environnements mobiles (économie d'énergie, ...).



Ensemble de **bibliothèques et de services nécessaires au développement** des applications Android

Couche d'abstraction servant d'**interface entre la noyau Linux et le matériel**.

OPEN SOURCE ET GRATUIT ?

OPEN SOURCE

Le code source d'Android est disponible en Open Source. Chacun est donc libre de créer sa propre version d'Android gratuitement.

MAIS

Les Google Mobile Services (Play, Maps, GMail, Chrome, Google Drive, ...) sont des applications propriétaires de Google et nécessitent une licence.

La licence GMS est "gratuite" mais nécessite une certification qui elle est payante.

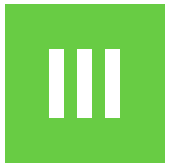
AVEC QUELLE VERSION TRAVAILLER ?

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.1 Jelly Bean	16	
4.2 Jelly Bean	17	99,9%
4.3 Jelly Bean	18	99,7%
4.4 KitKat	19	99,7%
5.0 Lollipop	21	98,8%
5.1 Lollipop	22	98,4%
6.0 Marshmallow	23	96,2%
7.0 Nougat	24	92,7%
7.1 Nougat	25	90,4%
8.0 Oreo	26	88,2%
8.1 Oreo	27	85,2%
9.0 Pie	28	77,3%
10. Q	29	62,8%
11. R	30	40,5%
12. S	31	13,5%

TOUT DÉPEND DU PARC CIBLE

Disponible sur Google Play ?
Devra être fonctionnel sur un maximum de périphériques Android.

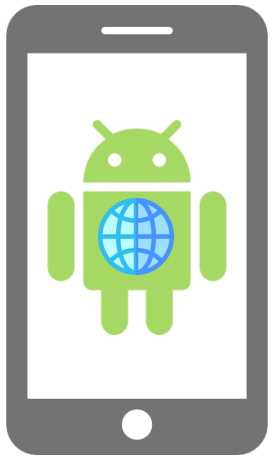
Maîtrise du parc matériel ?
A définir avec le client en fonction des API requises.



APPLICATIONS MOBILES

PLUSIEURS APPROCHES

NATIVE



HYBRIDE

PWA

APPLICATIONS NATIVES

NATIVE

- Développées spécifiquement pour la plateforme cible (iOS ou Android)
- Utilisent un langage de programmation pris en charge par la plateforme :
  - Swift pour iOS
  - Java ou Kotlin pour Android
- Offrent les meilleures performances
- Permettent un accès à toutes les fonctionnalités du périphérique (appareils photo, capteurs, contacts, ...)
- Distribuées via la boutique d'applications de la plateforme cible

APPLICATIONS HYBRIDES

HYBRIDE

- Développées sous la forme d'une application Web (HTML, CSS, JS) puis encapsulée dans une application native intégrant juste un navigateur configuré pour ne charger que l'application Web associée.
- Offrent de moins bonnes performances qu'une application native.
- L'accès aux fonctionnalités du périphérique (appareils photo, capteurs, ...) est possible via des appels JavaScript au runtime de l'application native.
- Distribuées via la boutique d'applications de la plateforme cible

PROGRESSIVE WEB APPLICATIONS (PWA)

PWA

- Développées sous la forme d'une application Web (HTML, CSS, JS)
- Utilisent un Service Worker pour mettre en cache les éléments nécessaires à leur fonctionnement de manière déconnectée.
- Offrent des performances intermédiaires entre celles des applications natives et hybrides.
- L'accès aux fonctionnalités du périphérique est limité et se fait via les API Web disponibles.
- Distribuées via le Web. Ne nécessitent pas d'installation. Peuvent être ajoutées à l'écran d'accueil.

FRAMEWORKS NATIFS MULTIPLATEFORMES

IV DÉVELOPPER UNE APPLICATION NATIVE ANDROID

ENVIRONNEMENT DE DÉVELOPPEMENT

Nécessitent d'être configurés pour la prise en charge du SDK Android

Android Studio est prêt à l'emploi

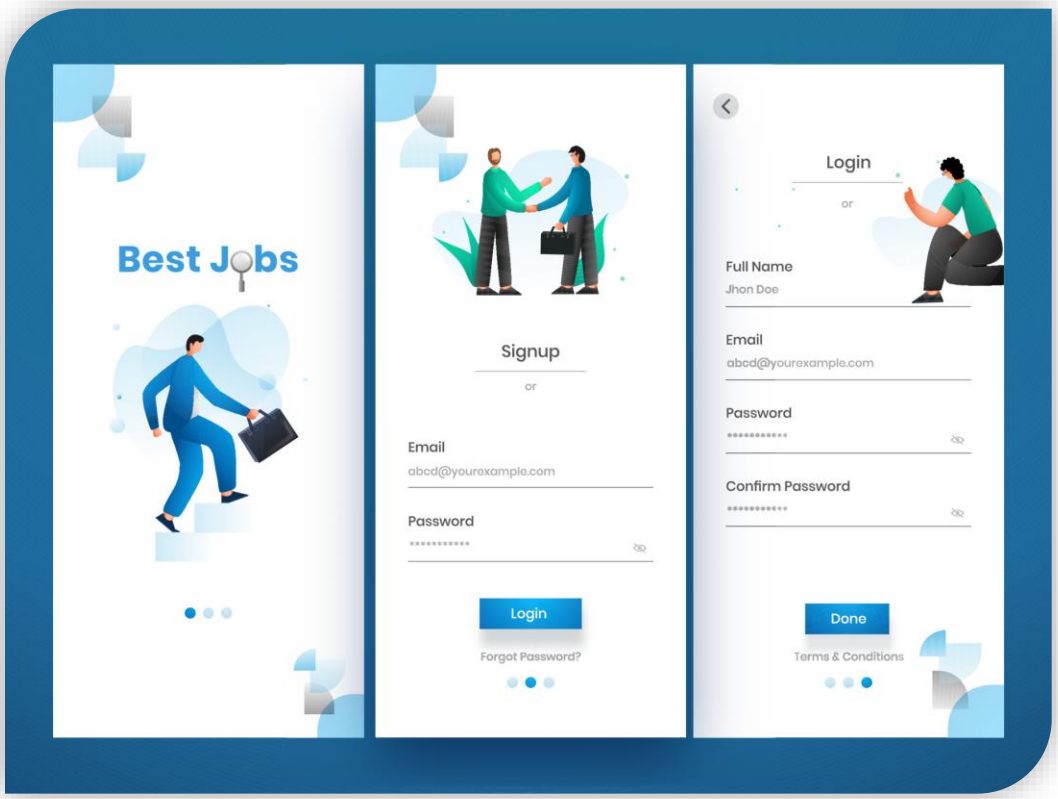
LANGAGES DE PROGRAMMATION

Langage historique pour le développement d'applications Android.

Officiellement pris en charge pour le développement d'applications Android en 2017 et mis en avant à partir de 2019.

V LES ACTIVITÉS

ACTIVITÉ (ACTIVITY)



UNE ACTIVITÉ

- représente un écran de l'application
- est responsable de l'interaction avec l'utilisateur pour cet écran
- est un élément fondamental de la navigation au sein de l'application

LAYOUT

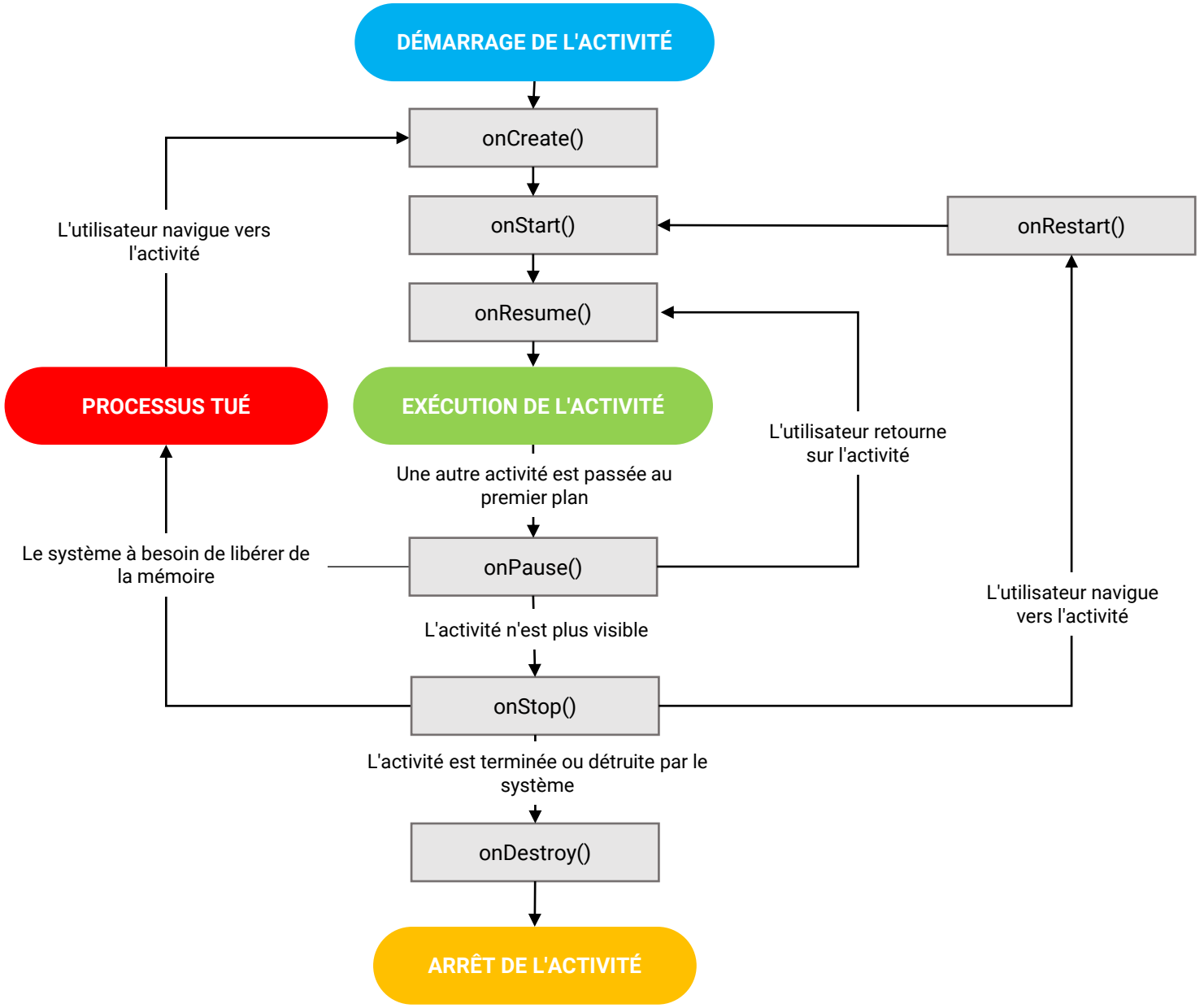
Le **layout** est un document XML qui décrit la structure d'une interface utilisateur :

- Les composants présents (champ texte, image, bouton, ...)
- L'aspect des composants (taille, couleur, bordure, ...)
- La position des composants (relative, absolue, ...)

Android Studio fournit un éditeur graphique permettant d'éditer un layout sans avoir à manipuler le XML directement.

Une activité est généralement associée à un layout, mais ce n'est pas toujours le cas.

CYCLE DE VIE


DÉMARRAGE DE L'ACTIVITÉ
onCreate()
onStart()
onRestart()
onResume()
L'utilisateur navigue vers l'activité
PROCESSUS TUÉ
EXÉCUTION DE L'ACTIVITÉ
L'utilisateur retourne sur l'activité
Une autre activité est passée au premier plan
Le système à besoin de libérer de la mémoire
onPause()
L'utilisateur navigue vers l'activité
L'activité n'est plus visible
onStop()
L'activité est terminée ou détruite par le système
onDestroy()
ARRÊT DE L'ACTIVITÉ

ACTIVITÉ ET LAYOUT

```kotlin
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val connectButton = findViewById<Button>(R.id.btnConnection)
        connectButton.setOnClickListener{ _ ->
            connect()
        }
    }

    private fun connect()
    {
        Toast
            .makeText(this, "Connexion en cours...", Toast.LENGTH_LONG)
            .show()
    }
}
```

MainActivity.kt


```xml
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#BFBFBF"
    android:orientation="vertical"
    android:padding="20dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.5">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Utilisateur" />

    <EditText
        android:id="@+id/txtLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dp"
        android:background="#FFFFFF"
        android:ems="10"
        android:inputType="text"
        android:padding="5dp" />
```

activity_main.xml

PILE D'ACTIVITÉS (BACK STACK)

La navigation au sein d'une application, mais également entre les applications se fait par empilement / dépilement des activités.

Le passage d'une activité à une autre se fait via :

- un **Intent** qui démarre une nouvelle activité
- le retour en arrière qui réactive l'activité précédente

PILE D'ACTIVITÉS - EXEMPLE

1. L'utilisateur démarre pour la première fois l'application *"Zombie Total War 3000 Full Extermination X And Again"*

   Un écran de connexion demande ses identifiants à l'utilisateur

PILE D'ACTIVITÉS - EXEMPLE

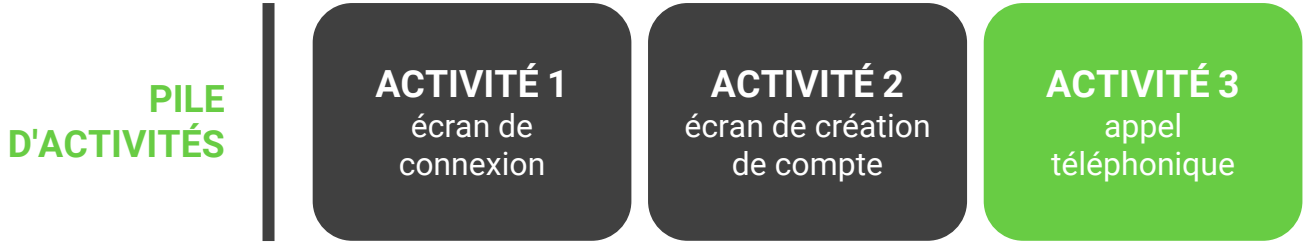
2. Comme l'utilisateur n'a jamais joué à ce jeu, il clique sur le bouton "Créer un compte"

   Un écran de création de compte apparaît

PILE D'ACTIVITÉS - EXEMPLE

3. L'utilisateur reçoit un appel pendant qu'il saisit ses informations

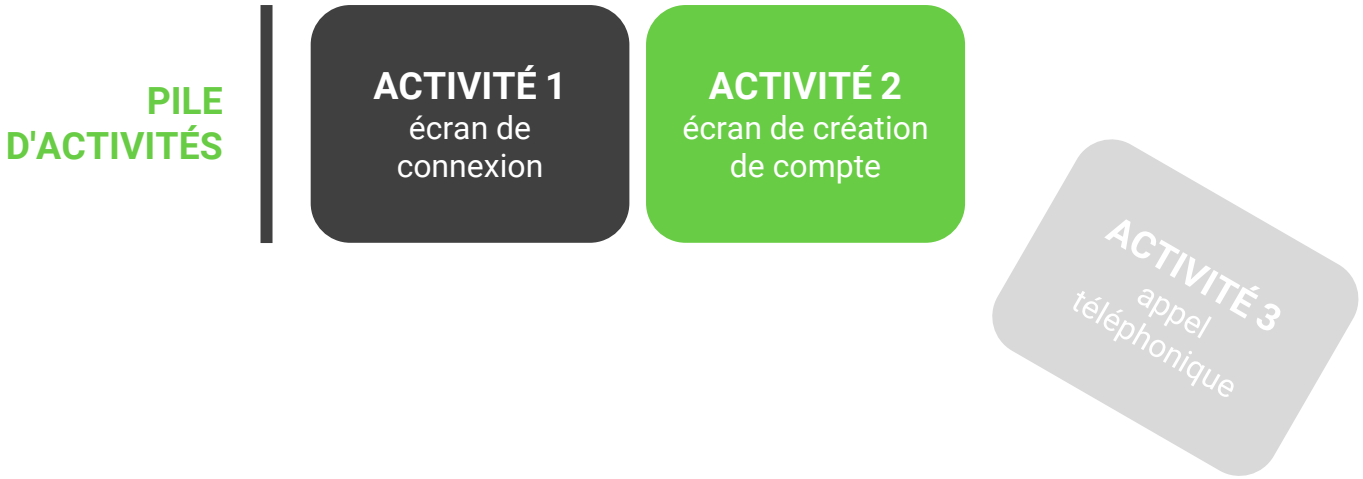
L'activité de l'application Téléphone est chargée sur la pile et masque l'écran de saisie du jeu



PILE D'ACTIVITÉS - EXEMPLE

4. L'utilisateur termine son appel

L'activité de l'application Téléphone est fermée et dépilée. L'écran de saisie des informations du jeu est de nouveau visible



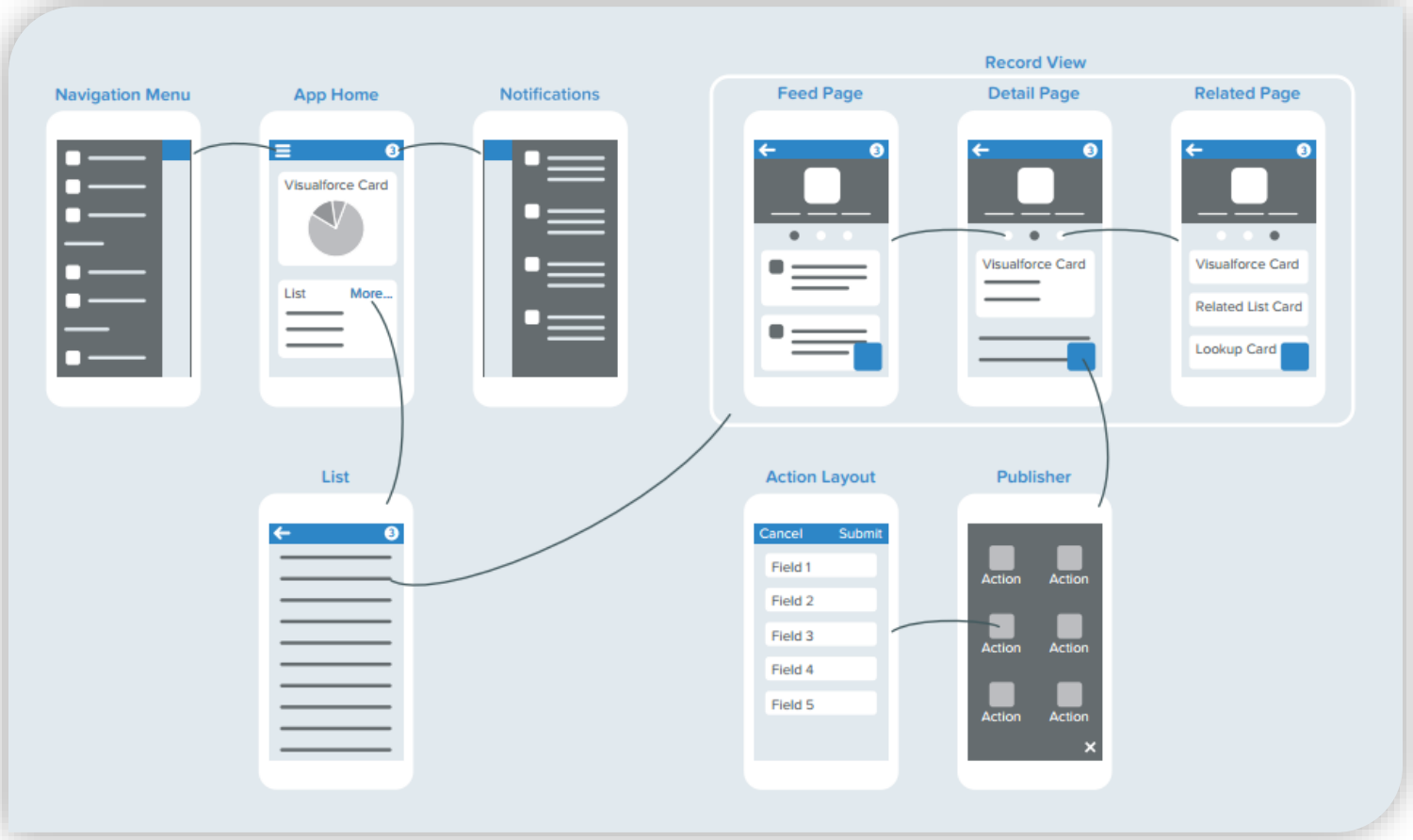
PILE D'ACTIVITÉS - EXEMPLE

4. L'utilisateur a terminé la création de son compte

   L'écran de saisie des informations du jeu est fermé et l'activité associée est dépilée. L'écran de login apparaît à nouveau

VI LES INTENTIONS

L'APPLICATION, UN ENSEMBLE D'ACTIVITÉS



Comment passer d'une activité à une autre ?

- Le bouton **BACK** qui permet de revenir à l'activité précédente sur la pile d'activités
- Les intentions (**Intent**) qui permettent de démarrer de nouvelles activités internes ou externes à l'application (exemple : la galerie photo)

INTENT EXPLICITE

Lorsque l'on passe d'une activité à une autre **au sein de la même application**, le développeur est en territoire connu. Il sait exactement quelle activité charger sur la pile d'activités.

La classe de l'activité à démarrer sera **explicitement** indiquée dans les paramètres de l'Intent :

```
//Création de l'Intent explicite
val intentToNextActivity = Intent(
        sourceActivity.this,     //Contexte de départ
        targetActivity.class     //Activité cible indiquée explicitement
);

// Démarrage de la nouvelle activité
startActivity(intentToNextActivity);
```

La classe de **l'activité cible doit faire partie du même package que l'activité source**.

INTENT IMPLICITE

Mais il existe d'autres cas dans lesquels le développeur ne sais pas quelle activité doit être démarrée.

Par exemple, si on souhaite que l'utilisateur charge une photo pour son profil, on ne sait pas s'il préfère utiliser l'application Galerie ou Photo ou Pinterest ou encore prendre un selfie avec Camera.

Dans ce cas, on indique au système Android la nature de ce que l'on souhaite faire et lui déterminera, en fonction des applications installées et des choix de l'utilisateur, l'activité à démarrer.

```
//Création de l'Intent implicite permettant de choisir quelque chose
val intentToPickPicture = Intent(Intent.Action_PICK)

//Précise que l'on souhaite choisir une image
intentToPickPicture.type = "image/*"

//Démarrage de la nouvelle activité avec attente de l'image sélectionnée
startActivityForResult(intentToPickPicture, "Identifiant_Resultat");
```

TRANSMETTRE DES INFORMATIONS

Les Intents permettent de transmettre des informations entre les activités :

activité source

```
//Création de l'Intent
val intent = Intent(...);

//Transmission de data_value sous
//le nom data_name
intent.putExtra("data_name", "data_value");

// Démarrage de la nouvelle activité
startActivity(intent);
```

activité cible

```
//Récupération de l'Intent qui a démarré
//l'activité
Intent i = getIntent();

//Récupération de la valeur de la donnée
//data_name
String value = i.getStringExtra("data_name");
```

Les données sont transmises sous la forme de paires clé/valeur, chacune étant des **chaines de caractères**.

VII TOUT LE RESTE

PLEIN DE CHOSES À DÉCOUVRIR

Si le développement mobile vous intéresse, il vous restera bien des choses à découvrir et à approfondir à l'issu de ce module :

- Les quantificateurs
- Les fragments
- Les ViewModel
- ...