

Fondamentaux du développement Android

Architecture, Composants et Publication

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1 Les Activities	1
2 Les Intents	1
2.1 Exemples d'Intents	2
2.2 Passage de données	2
3 Les Fragments	2
4 Les Services	2
4.1 Exemple d'Implémentation	2
4.2 Déclaration dans le Manifest	3
4.3 Démarrage d'un Service	3
4.4 Bonnes Pratiques	3
5 Le Manifest Android	3
5.1 Structure du Manifest	3
6 Cycle de Vie des Composants	4
6.1 Exemple d'Activité	4
6.2 États du Cycle de Vie d'une Activity	4
6.3 Fragments	5
7 Les Broadcast Receivers	5
8 Les Content Providers	5
9 Les ViewModels	6
9.1 MVVM (Model-View-ViewModel) . .	6
9.2 MVVM vs MVC	6
10 Interfaces Utilisateur	6
10.1 XML Layouts	6
10.2 XML Layout Example	7
10.3 Jetpack Compose	7
10.4 Jetpack Compose Example	7
10.5 Bonnes Pratiques UI	7

1 Les Activities

- **Définition**
 - Composant fondamental d'une application Android
 - Représente un écran unique avec une interface utilisateur
 - Point d'entrée principal pour l'interaction utilisateur
- **Caractéristiques principales**
 - Une seule activité active à la fois
 - Gestion automatique dans une pile (back stack)
 - Cycle de vie bien défini
 - Peut être démarrée par d'autres applications
- **Activité principale: `MainActivity`**
 - Déclarée dans le fichier `AndroidManifest.xml`
- **Etudier `MainActivity.kt` dans le projet Android**

2 Les Intents

- **Définition**
 - Mécanisme de messagerie pour communiquer entre composants
 - Permet de démarrer des activités, services, etc.
 - Support des communications inter-applications

- **Types d'Intents**

1. **Explicites**

- Cible spécifique (composant exact)
- Utilisés pour navigation intra-app

2. **Implicites**

- Action générique
- Système choisit le composant approprié

2.1 Exemples d'Intents

```
// Intent Explicite
val intent = Intent(this,
SecondActivity::class.java)
startActivity(intent)

// Intent Implicite pour sélectionner
un contact
val intent =
Intent(Intent.ACTION_PICK).apply {
    type = ContactsContract.CommonDataKinds.Phone.CONTENT_TYPE
}
startActivityForResult(intent,
PICK_CONTACT_REQUEST)
```

2.2 Passage de données

```
// Envoi de données
val intent = Intent(this,
DetailActivity::class.java).apply {
    putExtra("key_name", "value")
    putExtra("key_id", 123)
}
startActivity(intent)

// Réception de données
class DetailActivity :
AppCompatActivity() {
    override
fun onCreate(savedInstanceState:
Bundle?) {
    super.onCreate(savedInstanceState)
    val name =
intent.getStringExtra("key_name")
}
```

```
val id =
intent.getIntExtra("key_id", 0)
}
```

3 Les Fragments

- **Définition**

- Sous-composants modulaires d'une Activity
- Représentent une portion réutilisable de l'interface utilisateur
- Peuvent être combinés pour créer des interfaces complexes

- **Caractéristiques principales**

- Cycle de vie propre, lié à l'Activity hôte
- Peuvent être ajoutés/supprimés dynamiquement
- Réutilisables dans différentes Activities
- Adaptés aux interfaces flexibles et responsive

4 Les Services

- **Définition**

- Composant qui exécute des opérations de longue durée en arrière-plan
- Ne fournit pas d'interface utilisateur
- Continue à fonctionner même si l'utilisateur change d'application

- **Types de Services**

- **Services en Premier Plan (Foreground)**
 - Notification visible pour l'utilisateur
 - Exemple : lecteur de musique
- **Services en Arrière-plan (Background)**
 - Opérations invisibles pour l'utilisateur
 - Exemple : synchronisation de données

4.1 Exemple d'Implémentation

```
class MyBackgroundService : Service()
{
    override fun onBind(intent:
Intent): IBinder? {
```

```

        return null
    }

    override fun
onStartCommand(intent: Intent?,
flags: Int, startId: Int): Int {
    // Code à exécuter en arrière-
plan
    return START_STICKY
}
}

```

4.2 Déclaration dans le Manifest

```

<service
    android:name=".MyBackgroundService"
    android:exported="false" />

```

4.3 Démarrage d'un Service

```

// Démarrer un service
val serviceIntent = Intent(this,
MyBackgroundService::class.java)
startService(serviceIntent)

// Arrêter un service
stopService(serviceIntent)

```

4.4 Bonnes Pratiques

- Utiliser WorkManager pour les tâches périodiques
- Gérer efficacement le cycle de vie
- Respecter les restrictions de batterie
- Libérer les ressources correctement

5 Le Manifest Android

```

<?xml version="1.0"
encoding="utf-8"?>
<manifest xmlns:android="http://
schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.

```

```

android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/
data_extraction_rules"
        android:fullBackupContent="@xml/
backup_rules"
        android:icon="@mipmap/
ic_launcher"
        android:label="@string/
app_name"
        android:roundIcon="@mipmap/
ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/
Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/
app_name"
            android:theme="@style/
Theme.MyApplication">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>

```

5.1 Structure du Manifest

Le fichier `AndroidManifest.xml` est un composant crucial de toute application Android. Il contient les informations essentielles de l'application que le système Android doit

connaître avant de pouvoir exécuter le code. Dans cet exemple :

- `<application>` définit les propriétés globales de l'application :
 - `android:allowBackup` permet la sauvegarde des données
 - `android:icon` et `android:roundIcon` définissent les icônes de l'application
 - `android:theme` spécifie le thème visuel global
- `<activity>` déclare la `MainActivity` comme point d'entrée de l'application :
 - `android:exported="true"` permet son lancement depuis l'extérieur
 - L'`intent-filter` avec `android.intent.action.MAIN` et `android.intent.category.LAUNCHER` indique que cette activité est le point de départ et doit apparaître dans le lanceur d'applications

6 Cycle de Vie des Composants

Les composants Android suivent un cycle de vie strict géré par le système d'exploitation. Cette gestion est essentielle pour :

Objectifs du cycle de vie : - Optimiser l'utilisation des ressources système - Éviter les fuites de mémoire - Garantir une expérience utilisateur fluide - Gérer les changements de configuration

Principaux composants concernés : - Activités : gestion des écrans et interactions utilisateur - Fragments : sous-parties modulaires d'une interface - Services : opérations en arrière-plan - ViewModels : conservation des données lors des changements de configuration

Points clés : - Chaque état correspond à une méthode de callback spécifique - Le système

gère automatiquement les transitions entre états - Les développeurs doivent implémenter les comportements appropriés - Important pour la gestion de la mémoire et des ressources

Cas d'usage typiques : - Sauvegarde de données lors de la rotation de l'écran - Libération des ressources quand l'app passe en arrière-plan - Reprise d'état après un redémarrage de l'application - Gestion des permissions et des résultats d'intents

6.1 Exemple d'Activité

```
class MainActivity : AppCompatActivity() {  
    override  
    fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
  
    override fun onResume() {  
        super.onResume()  
        // L'activité devient visible  
    }  
  
    override fun onPause() {  
        super.onPause()  
        // L'activité n'est plus au premier plan  
    }  
}
```

6.2 États du Cycle de Vie d'une Activity

1. **Création** (`onCreate`)
 - Premier callback appelé
 - Initialisation des composants UI
 - Création des ViewModels
 - Configuration de base de l'Activity
2. **Démarrage** (`onStart`)
 - Activity devient visible

- Initialisation des ressources UI
 - Préparation des animations
3. **Reprise** (`onResume`)
 - Activity au premier plan
 - Interactive avec l'utilisateur
 - Capture des événements UI
 - Démarrage des animations
 4. **Active**
 - État où l'Activity est pleinement fonctionnelle
 - Interaction utilisateur possible
 - Exécution des opérations principales
 5. **Pause** (`onPause`)
 - Activity partiellement visible
 - Arrêt des animations
 - Sauvegarde de l'état léger
 - Libération des ressources non essentielles
 6. **Arrêt** (`onStop`)
 - Activity non visible
 - Sauvegarde de l'état complet
 - Libération des ressources lourdes
 - Arrêt des opérations en cours
 7. **Redémarrage** (`onRestart`)
 - Transition vers `onStart`
 - Restauration de l'état
 - Réinitialisation des ressources
 8. **Destruction** (`onDestroy`)
 - Nettoyage final
 - Libération de toutes les ressources
 - Désinscription des observers
 - Fin du cycle de vie

6.3 Fragments

```
class MainFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(
            R.layout.fragment_main,
            container,
```

```
        false
    )
}
```

7 Les Broadcast Receivers

- **Définition**
 - Composant qui réagit aux messages système et applicatifs
 - Permet de répondre aux événements à l'échelle du système
 - Fonctionne même si l'application n'est pas active
- **Types d'événements courants**
 - Changement d'état de connectivité
 - Niveau de batterie faible
 - Installation/désinstallation d'applications
 - Redémarrage du système
 - Messages personnalisés

8 Les Content Providers

- **Définition**
 - Interface standardisée pour le partage de données entre applications
 - Permet l'accès sécurisé aux données structurées
 - Encapsule les données et fournit des mécanismes de sécurité
- **Caractéristiques principales**
 - Utilise une structure similaire aux URI pour accéder aux données
 - Supporte les opérations CRUD (Create, Read, Update, Delete)
 - Gère les permissions d'accès aux données
 - Permet le partage de données entre applications
- **Cas d'Usage Courants**
 - Accès aux contacts du téléphone
 - Gestion des fichiers médias
 - Partage de données d'application

- Synchronisation de données avec le serveur

9 Les ViewModels

• Définition

- Composant d'architecture qui conserve et gère les données liées à l'UI
- Survit aux changements de configuration (rotation d'écran)
- Sépare la logique métier de l'interface utilisateur

• Caractéristiques principales

- Conserve les données pendant le cycle de vie de l'Activity/Fragment
- Évite la perte de données lors des changements de configuration
- Facilite le partage de données entre fragments
- Suit le pattern MVVM (Model-View-View-Model)

9.1 MVVM (Model-View-ViewModel)

• Architecture en 3 couches

- **Model**: Données et logique métier
- **View**: Interface utilisateur (Activity/Fragment)
- **ViewModel**: Pont entre Model et View
- Différent de MVC (Model-View-Controller)
- **MVC**

- Contrôleur gère directement les interactions UI
- Vue dépend directement du Model
- Risque de contrôleurs monolithiques
- Difficile à tester unitairement

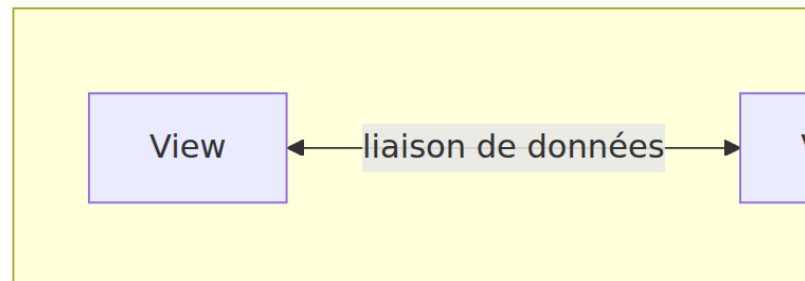
▸ MVVM Avantages

- ViewModel ne connaît pas la View
- Communication unidirectionnelle
- Meilleure séparation des responsabilités
- Plus facile à tester

9.2 MVVM vs MVC

```
graph TD
    subgraph MVC
        C[Controller] -->|met à jour| V1[View]
        V1[View] -->|notifie| C[Controller]
        C[Controller] -->|manipule| M1[Model]
        M1[Model] -->|notifie| V1[View]
    end

    subgraph MVVM
        VM[ViewModel] -->|met à jour| M2[Model]
        M2[Model] <-->|liaison de données| V2[View]
        M2[Model] -.->|notifie| VM[ViewModel]
    end
```



10 Interfaces Utilisateur

• Approches de Développement UI

- XML Layouts (Approche traditionnelle)
- Jetpack Compose (Moderne, déclarative)

10.1 XML Layouts

• Définition

- Fichiers XML décrivant la structure de l'interface utilisateur

- Utilisés pour définir les vues, les contraintes et les interactions
- Supportent les ressources, les styles et les thèmes
- **Caractéristiques**
 - Séparation claire entre UI et logique
 - Support historique et mature
 - Éditeur visuel disponible
 - Structure hiérarchique
- Intégré dans Android Studio et les outils de développement
- **Avantages**
 - API moderne et déclarative
 - Code plus concis
 - Preview en temps réel
 - État prévisible

10.4 Jetpack Compose Example

10.2 XML Layout Example

```
<?xml version="1.0"
encoding="utf-8"?>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android!" /
    >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```

```
@Composable
fun GreetingScreen(name: String) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Text(
            text = "Hello $name!",
            style =
                MaterialTheme.typography.h4
        )

        Button(
            onClick = { /* action */ }
        ) {
            Text("Click Me")
        }
    }
}
```

10.3 Jetpack Compose

- **Définition**
 - Toolkit moderne pour la création d'interfaces utilisateur Android
 - Basé sur le langage de programmation Kotlin
 - Utilise une approche déclarative pour la création d'UI

10.5 Bonnes Pratiques UI

- Respecter les guidelines Material Design
- Supporter différentes tailles d'écran
- Gérer les thèmes sombre/clair
- Optimiser les performances
- Implémenter l'accessibilité