Développement Android avec Kotlin

Cours - 02 - Les bases du système Android : Structure et Applications

Jordan Hiertz

Contact
hiertzjordan@gmail.com
jordan.hiertz@al-enterprise.com

Qu'est ce qu'Android ?

Système d'exploitation mobile : Développé par Google, Android est le système d'exploitation le plus utilisé dans le monde.

Plateforme Open Source : Basé sur Linux, Android est flexible et offre une grande liberté aux développeurs.

Écosystème Android : Téléphones, tablettes, montres connectées, TV, etc.

Historique d'Android

2003 : Création d'Android Inc.

2005 : Acquisition par Google.

2008 : Lancement du premier appareil Android (HTC Dream).

2011 : Android 4.0 Ice Cream Sandwich - Première grande refonte visuelle.

2014 : Android 5.0 Lollipop - Introduction du **Material Design.**

2017 : Android 8.0 Oreo - Améliorations de la gestion des performances et des notifications.

2019 : Android 10 - Abandon des noms de desserts, mode sombre natif.

Aujourd'hui : Android 15 - OS mobile le plus utilisé au monde avec plus de 2.5 milliards d'appareils actifs.

Fragmentation Android et surcouches des constructeurs

Diversité des appareils : Android est utilisé sur une vaste gamme de téléphones, tablettes, télévisions, montres, etc., provenant de différents fabricants (Samsung, Xiaomi, Huawei...).

Surcouches logicielles : Les constructeurs ajoutent leurs propres couches à Android, comme **One UI** (Samsung), **MIUI** (Xiaomi), ou **EMUI** (Huawei), ce qui modifie l'interface et certaines fonctionnalités.

Impact pour les développeurs :

- Tests sur plusieurs versions et appareils.
- Adaptation de l'interface.
- Support de fonctionnalités spécifiques à certains appareils.

Android : Un OS open source et centré sur l'utilisateur

Open source : Android est distribué sous une licence open source.

Interactions utilisateur : Conçu pour être intuitif, basé sur des **gestes tactiles** comme :

- **Balayage** pour naviguer.
- **Tapotement** pour sélectionner des éléments.
- **Pincement** pour zoomer.

Clavier virtuel et contrôleurs :

- Inclut un **clavier virtuel** pour la saisie.
- Prend en charge des **contrôleurs** de jeu, des **claviers physiques**, des **souris** et autres **périphériques**.

Android : Un OS optimisé par des capteurs

- **Accéléromètre** : Détecte l'orientation et les mouvements de l'appareil.
- **Gyroscope** : Mesure la rotation et les mouvements angulaires.
- **Capteur de proximité** : Détecte si un objet / une personne est proche de l'écran.
- **Capteur de lumière ambiante** : Mesure la quantité de lumière reçue.
- **Capteur d'empreintes digitales** : Pour la sécurité biométrique.
- **GPS** : Localisation précise et navigation.
- **Boussole (Magnétomètre)** : Détecte le champ magnétique terrestre.

Android : Plus qu'un OS, c'est aussi un SDK !

SDK Android : Kit de développement pour créer des applications Android. 🛠️

Outils de développement : 🖥️

- **Debugger** : Outil pour détecter et corriger les erreurs dans le code. 🐞
- **Monitoring** : Analyse des performances de l'application. 📊

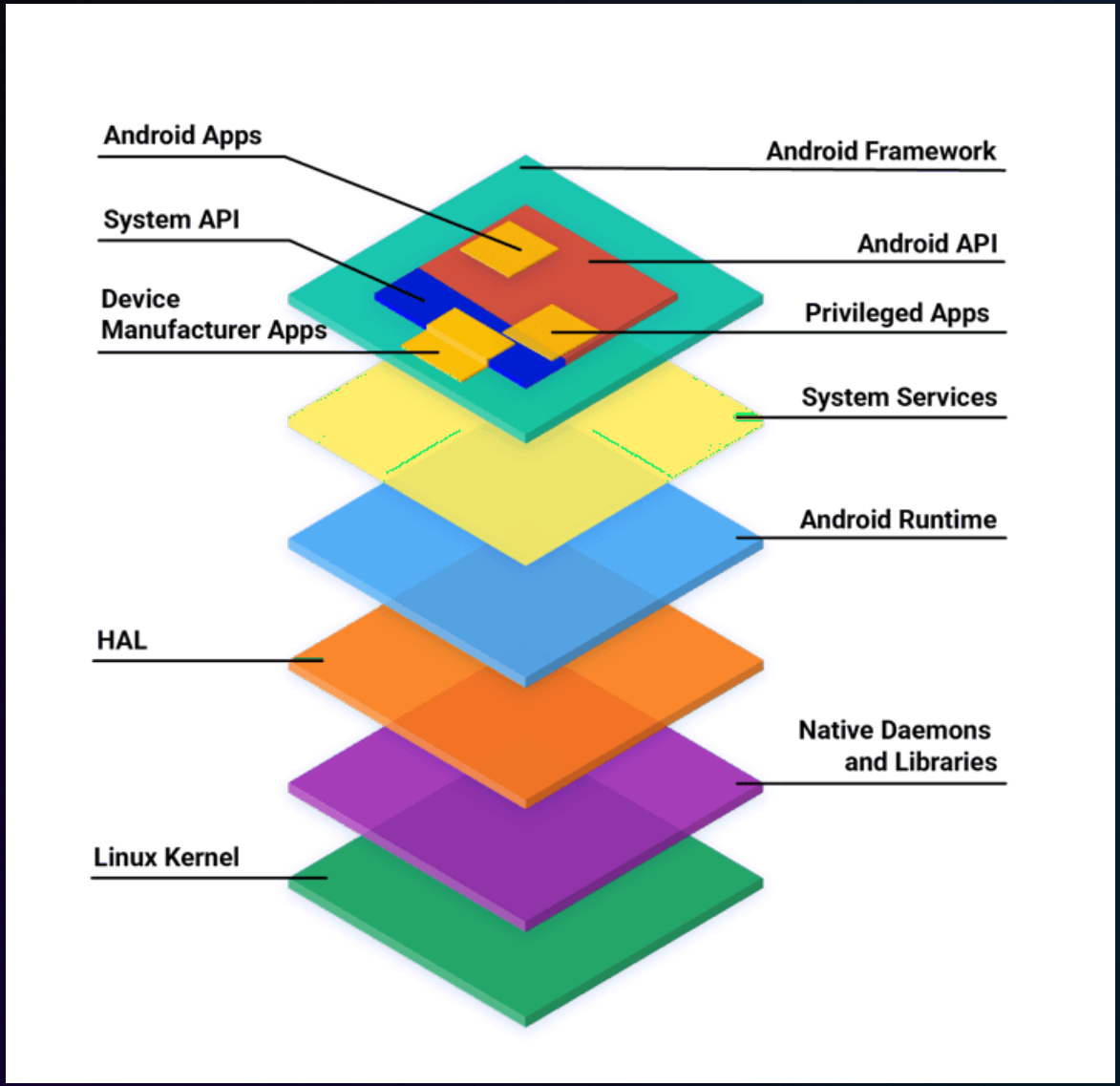
Bibliothèques logicielles :📚

- Android Jetpack : Simplifie le développement.
- Google Play Services : Intègre des services Google.

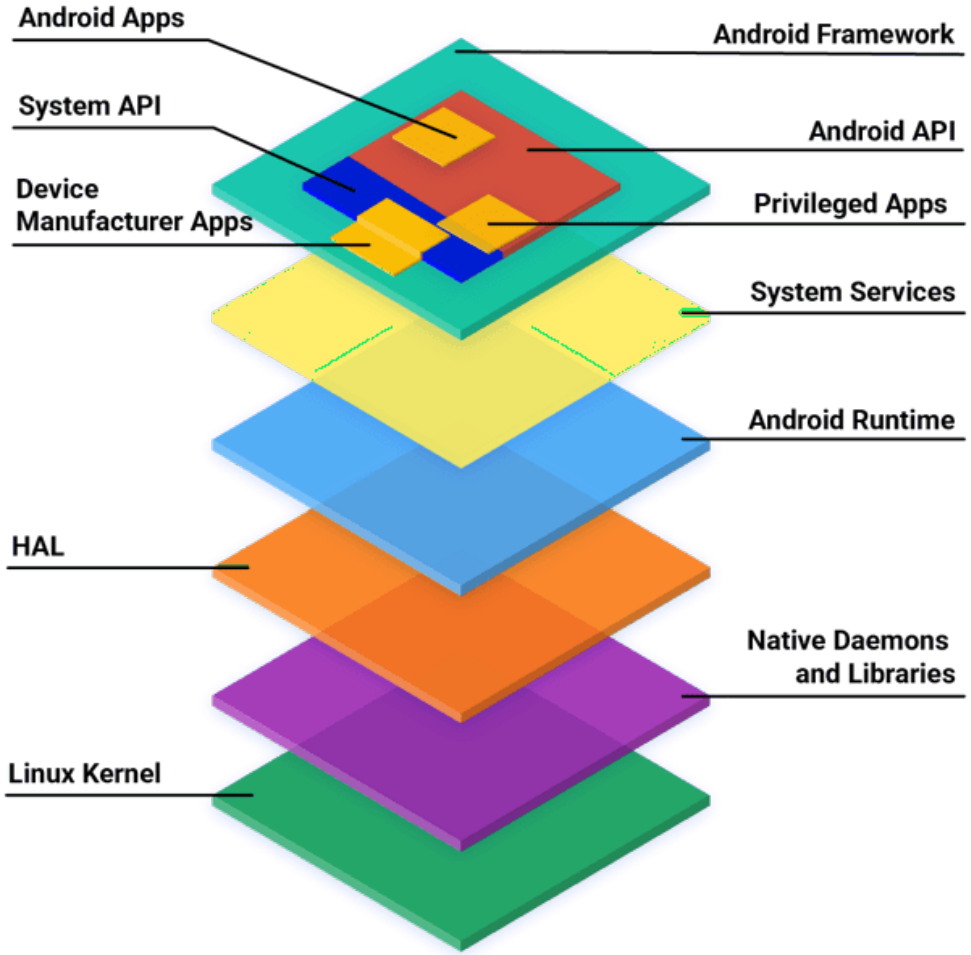
Documentation : 📖

- developer.android.com : Ressource officielle avec tutoriels et exemples.

La Stack Android : Composants Clés



La Couche Applicative d'Android



Application Android : Développée uniquement avec l'API publique, elles sont disponible sur le Google Play Store.

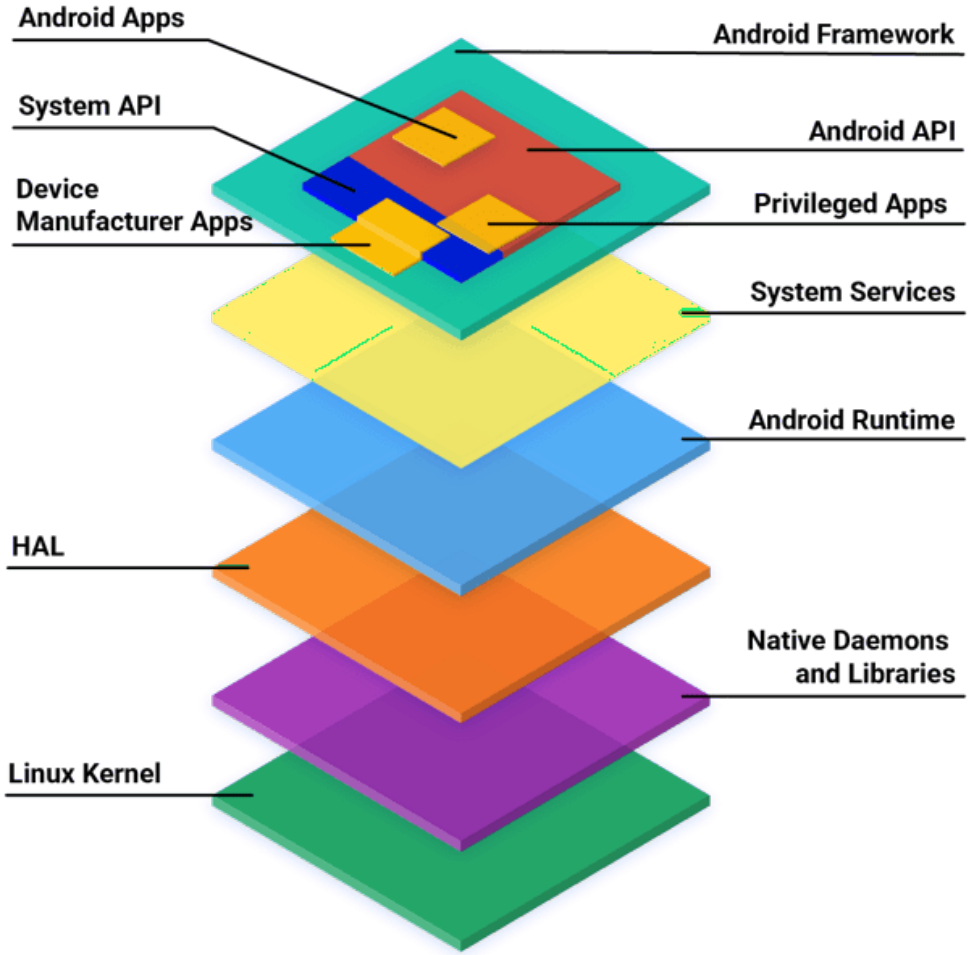
Application privilégiée : Utilise également les API systèmes, elles sont généralement préinstallées.

Application du fabricant : Accès direct à l'implémentation du framework Android. Elles sont mise à jour uniquement avec le système.

API système : APIs réservées aux partenaires et fabricants (OEM), uniquement disponibles pour les applications préinstallées.

API Android : API publique utilisée par tous les développeurs d'applications Android.

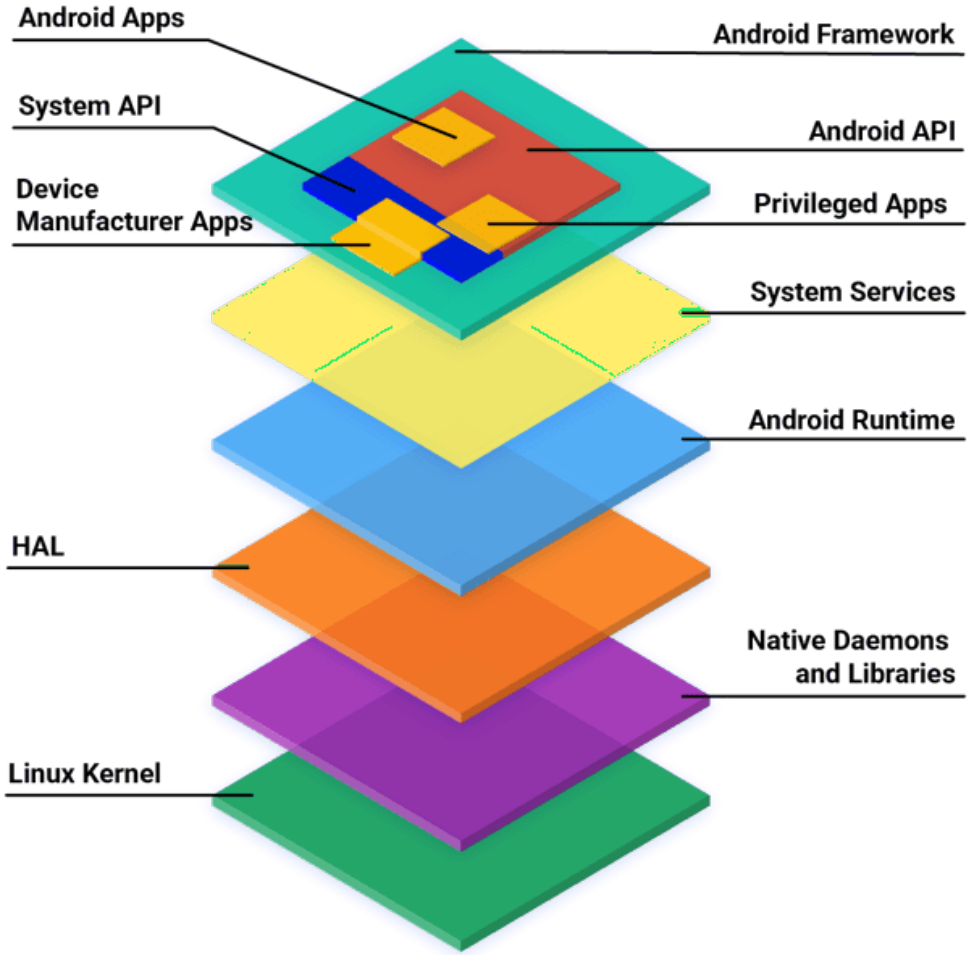
Framework Android et Services Système



Android Framework : Ensemble de classes et d'interfaces Java précompilées. Le code du framework s'exécute dans le processus de l'application.

System Services : Composants modulaires et spécialisés qui gèrent des fonctionnalités cruciales. Les API du framework Android communiquent avec ces services pour accéder au matériel sous-jacent et fournir les fonctionnalités nécessaires.

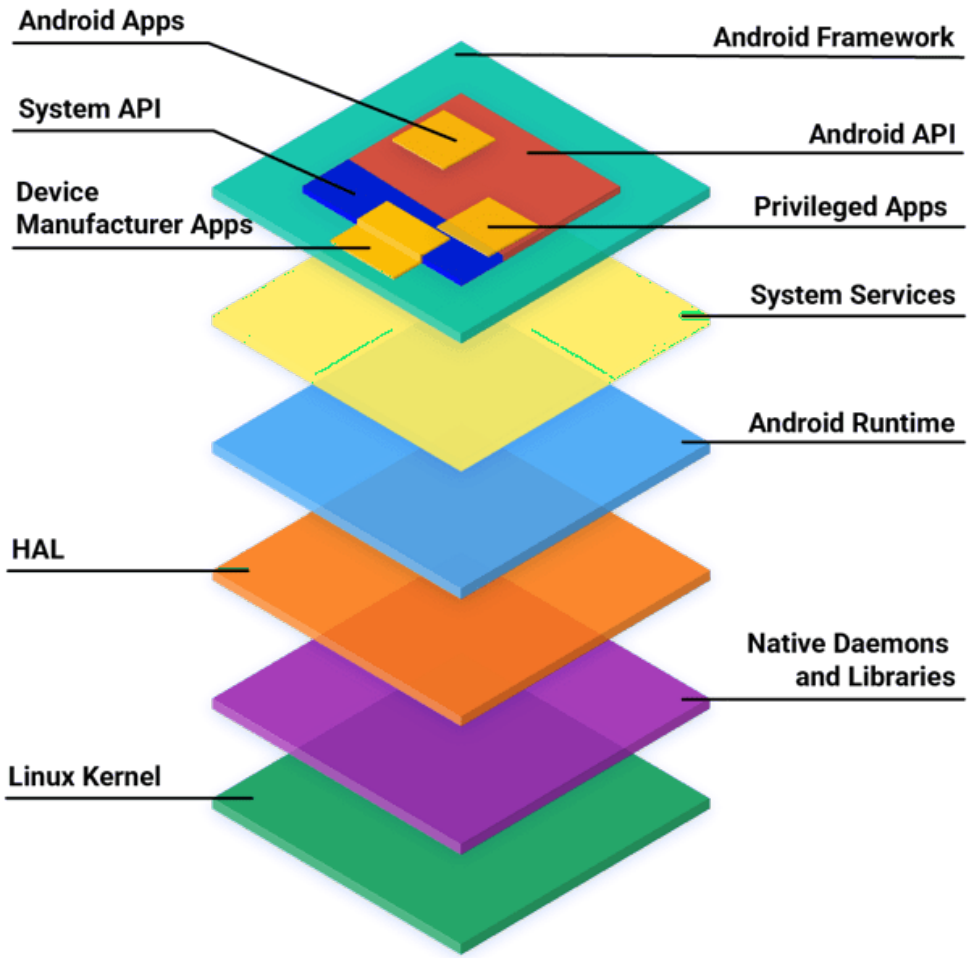
Android Runtime (ART)



Android Runtime : Un environnement d'exécution Java fourni par AOSP. **ART** effectue la traduction du bytecode de l'application en instructions spécifiques au processeur.

- **Optimisation des performances** : ART utilise la compilation Ahed-of-Time (AOT) pour optimiser la vitesse d'exécution des applications.
- **Gestion de la mémoire** : Améliore la gestion de la mémoire par rapport à son prédécesseur, Dalvik.
- **Exécution native** : Les applications sont exécutées de manière native sur le matériel, offrant une meilleure performance globale.

Hardware Abstraction Layer (HAL)

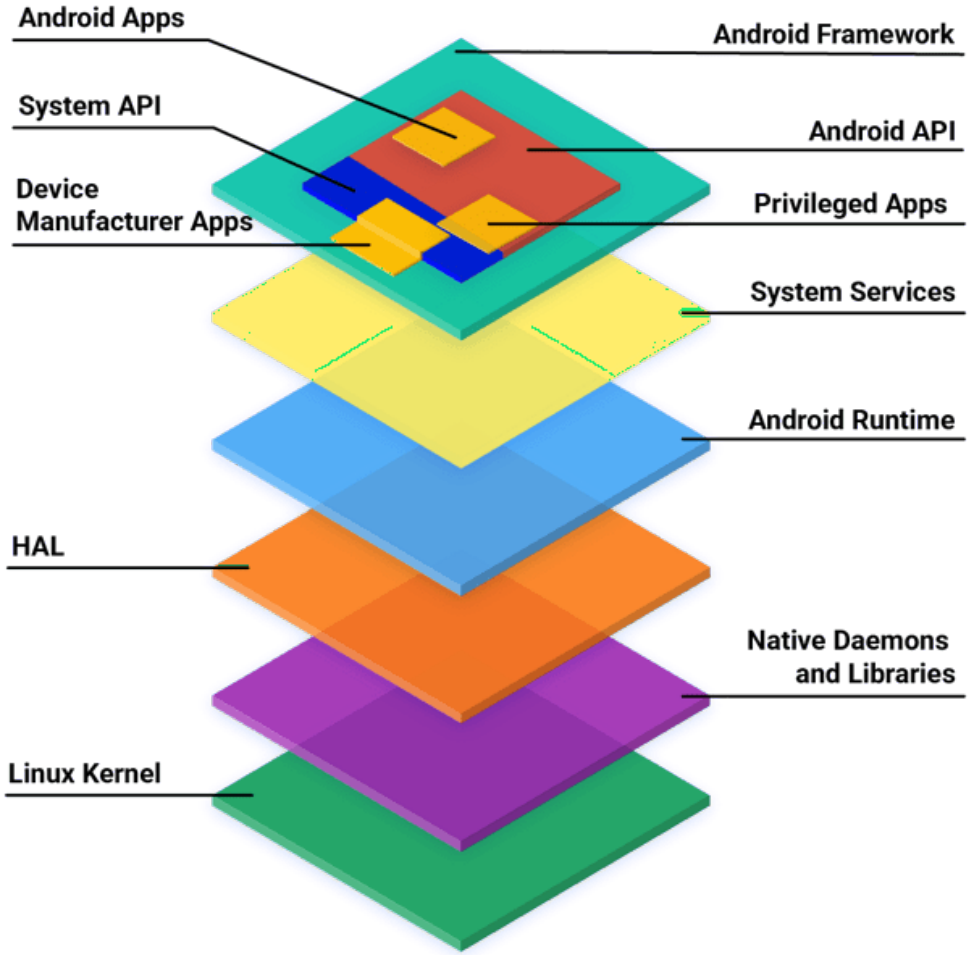


Un **HAL** est une couche d'abstraction avec une interface standard que les fabricants de matériel peuvent implémenter.

- Permet à Android d'être agnostique aux implémentations des pilotes de bas niveau.

- Utiliser un HAL permet d'implémenter des fonctionnalités sans affecter ni modifier le système de niveau supérieur.

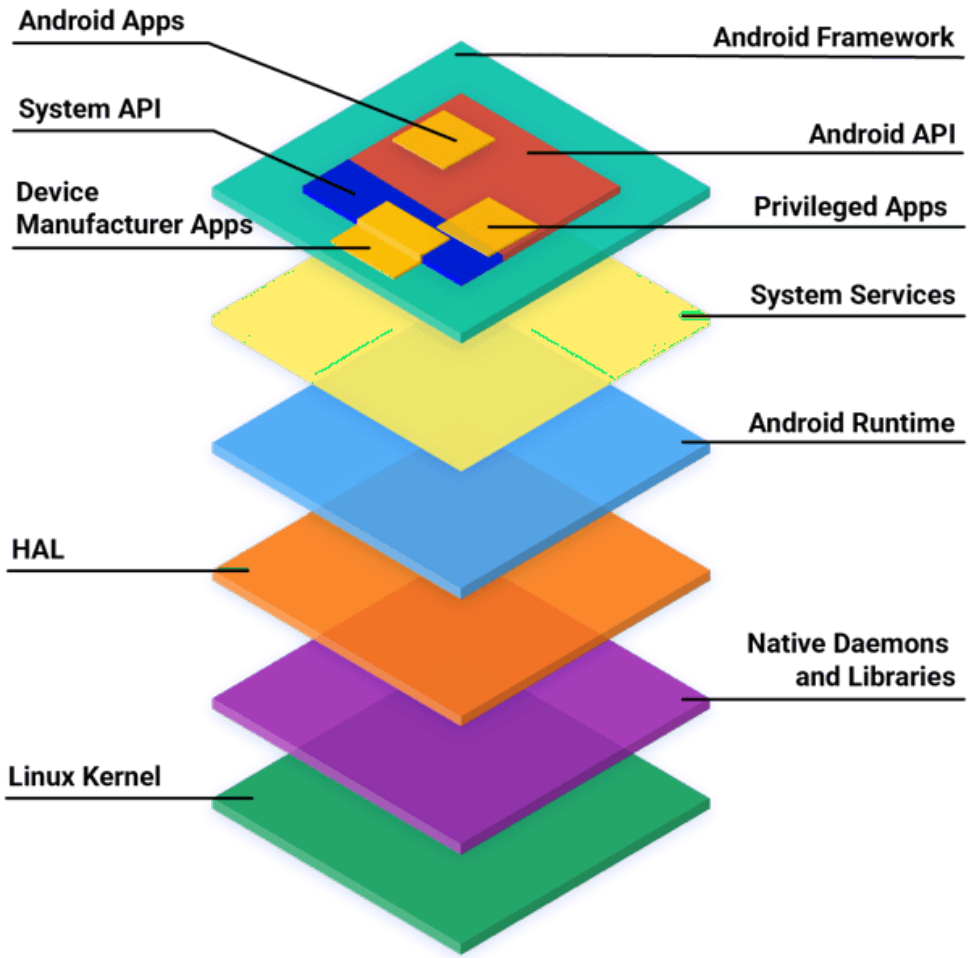
Daemons et Bibliothèques Natives



- **Daemons** : Inclut des processus tels que `init`, `healthd`, `logd`, `storaged`, qui interagissent directement avec le noyau ou d'autres interfaces, sans dépendre d'une implémentation HAL.

- **Bibliothèques Natives** : Comprend `libc`, `liblog`, `libutils` etc. qui interagissent également directement avec le noyau ou d'autres interfaces, sans dépendre d'une implémentation HAL.

Noyau Linux : Le Cœur d'Android



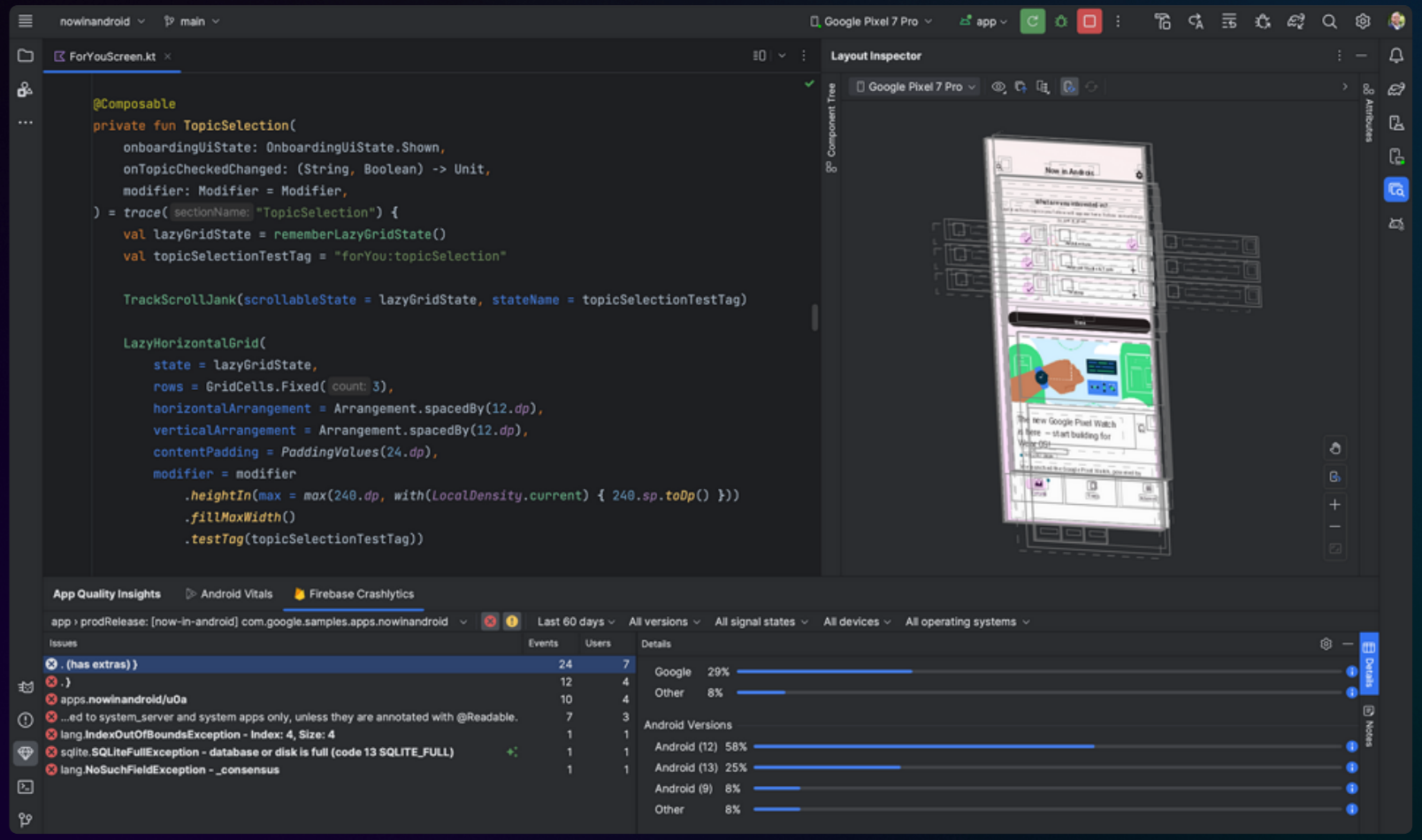
- **Fonction centrale** : Le noyau est la partie centrale de tout système d'exploitation, permettant la communication entre le matériel et le logiciel de l'appareil.

- **Modules agnostiques et spécifiques** : Le noyau AOSP (Android Open Source Project) est divisé en modules agnostiques au matériel et en modules spécifiques au fournisseur, permettant une flexibilité dans l'implémentation matérielle.

- **Interfaçage matériel** : Le noyau gère les ressources matérielles, telles que le processeur, la mémoire, les périphériques d'entrée/sortie et garantit que les applications peuvent interagir avec le matériel de manière sécurisée et efficace.

Android Studio

Environnement de développement intégré (IDE) officiel des applications Android.



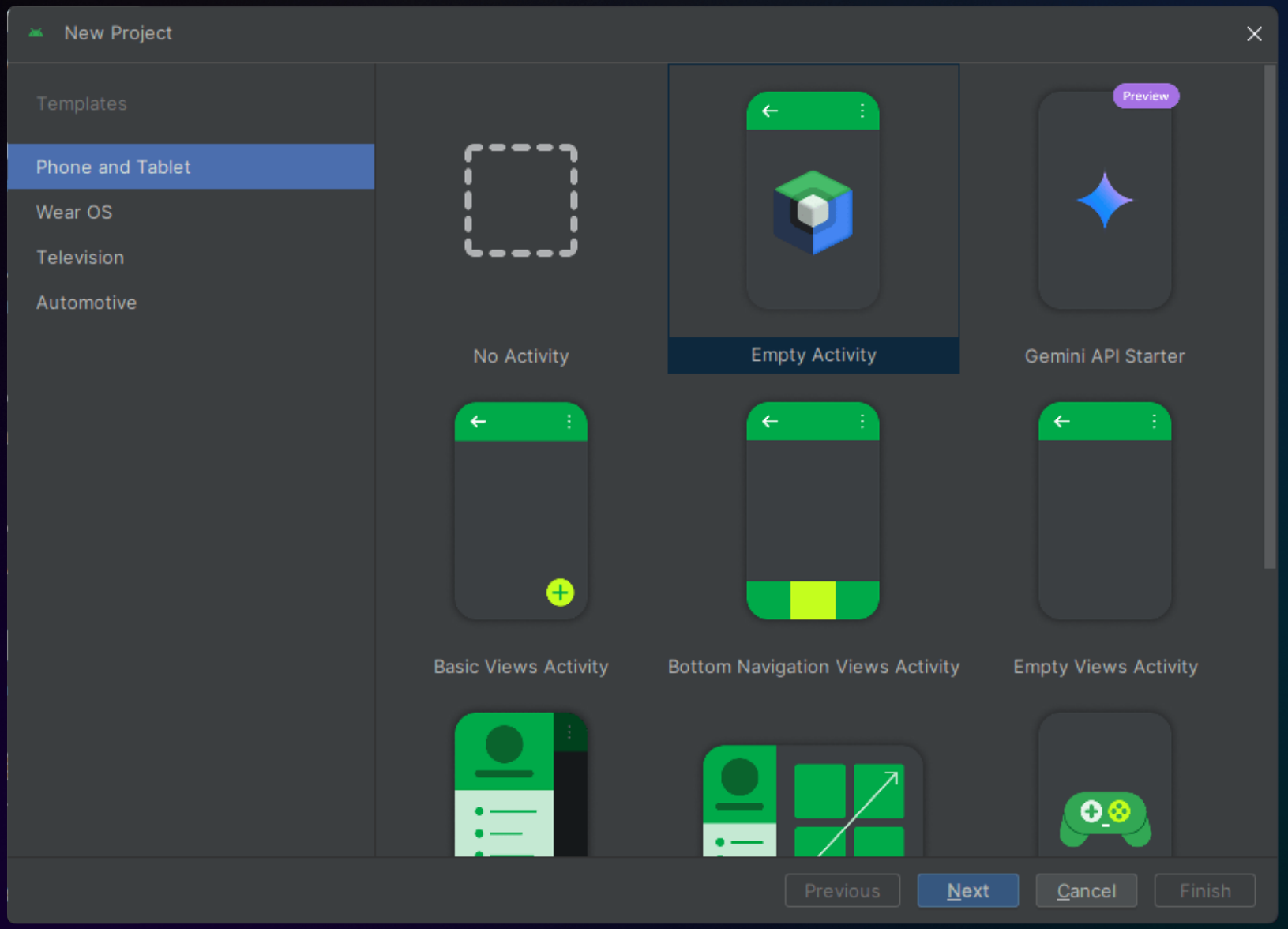
Android Studio - IDE

- Lien vers le téléchargement : developer.android.com/studio

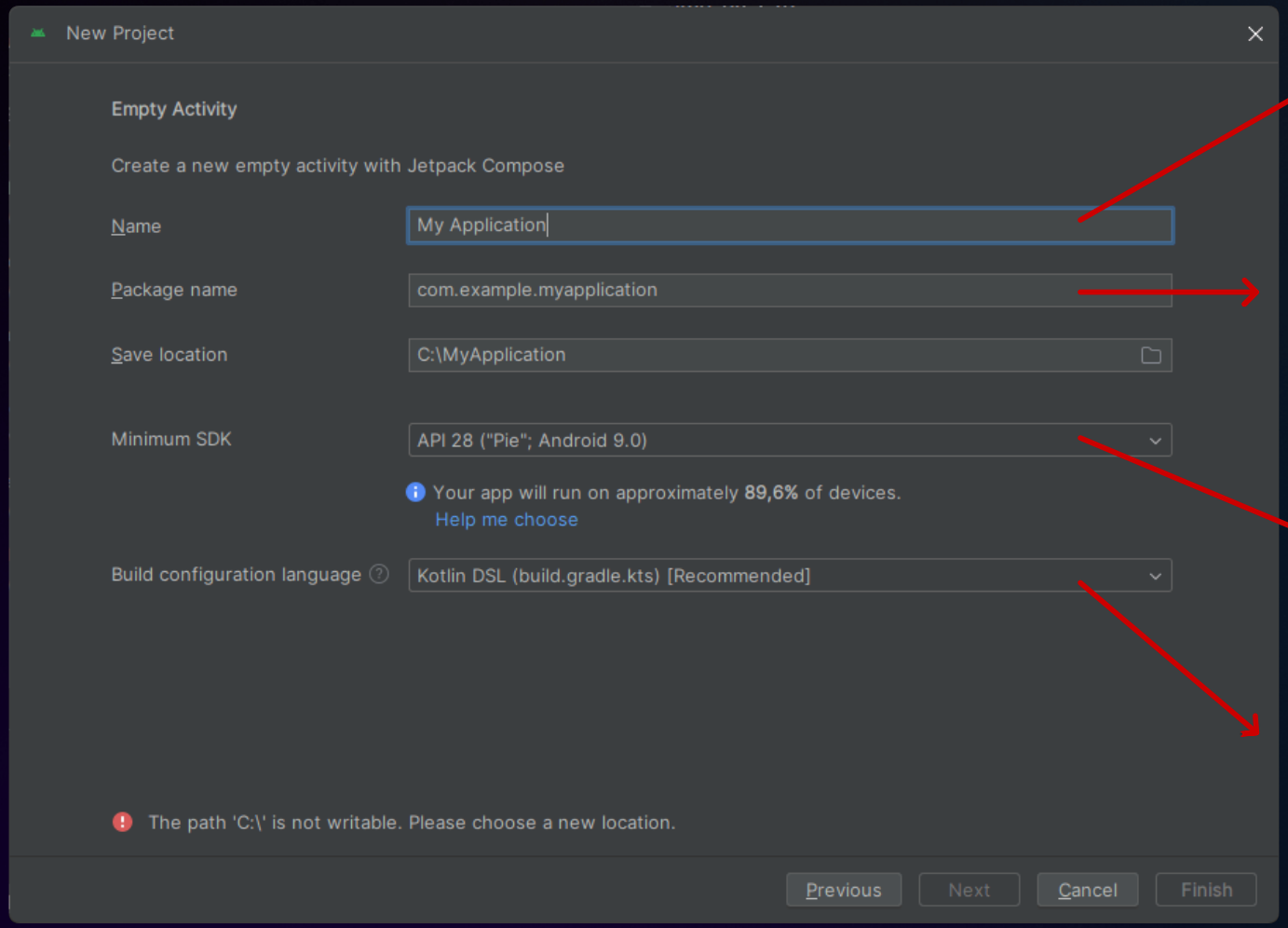
- Basé sur IntelliJ IDEA : Android Studio est construit sur la plateforme IntelliJ, un IDE populaire pour Java et Kotlin.

- **Fonctionnalités pour la productivité :**
  - **Compilation avec Gradle** : gestion simplifiée des builds.
  - **Émulateur Android intégré** : tester sans appareil physique.
  - **Outils de test complets** : pour les tests unitaires et UI.
  - **Linter intégré** : détecte les erreurs et problèmes.
  - **Support C++** : développement natif avec le NDK.
- **Extensible avec des plugins communautaires**

Choisir un Template de Projet Android



Configuration d'un Projet Android



C'est le nom visible de l'application. Choisissez un nom clair et significatif.

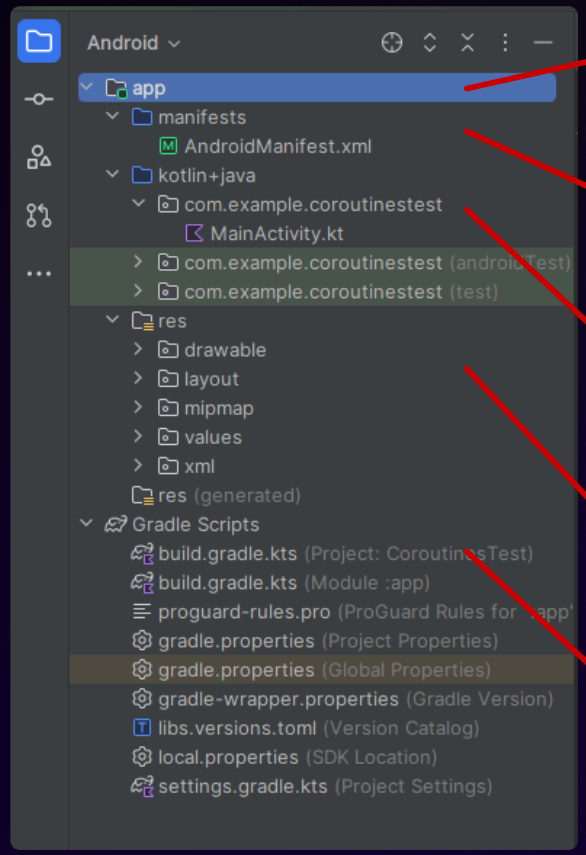
Le nom du package unique qui identifie l'application dans l'écosystème Android. Ce package est essentiel pour la publication sur le Play Store.

Détermine la version minimale d'Android que l'application supportera.

Kotlin DSL : Utilise la syntaxe Kotlin pour définir la configuration de build.
Groovy : Configuration de build traditionnelle.

Structure d'un Projet Android

Chaque projet dans Android Studio est composé d'un ou plusieurs modules, chacun contenant des fichiers de code source et de ressources.



app : Module d'application par défaut contenant le code principal.

manifests : Contient le fichier `AndroidManifest.xml` utile pour déclarer les composants.

kotlin+java : Contient les fichiers de code source (classes, activités, tests).

res : Contient toutes les ressources non liées au code (strings, images, icônes etc.).

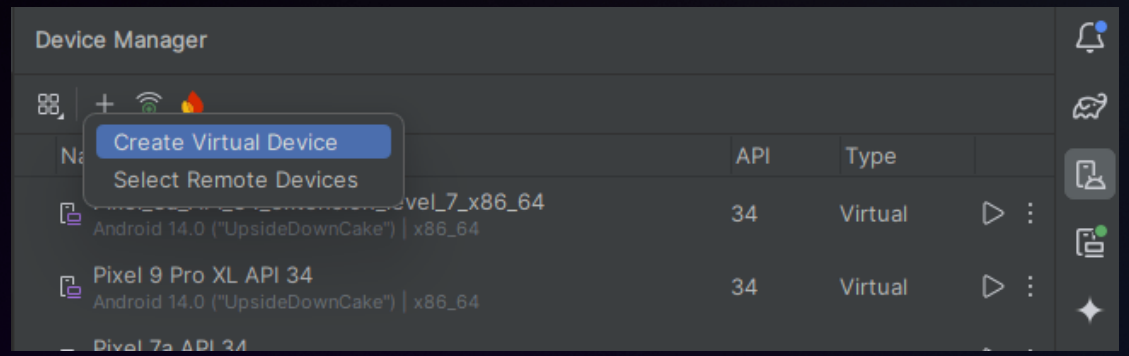
Gradle Scripts : Configuration de build (dépendances, versions de SDK etc.).

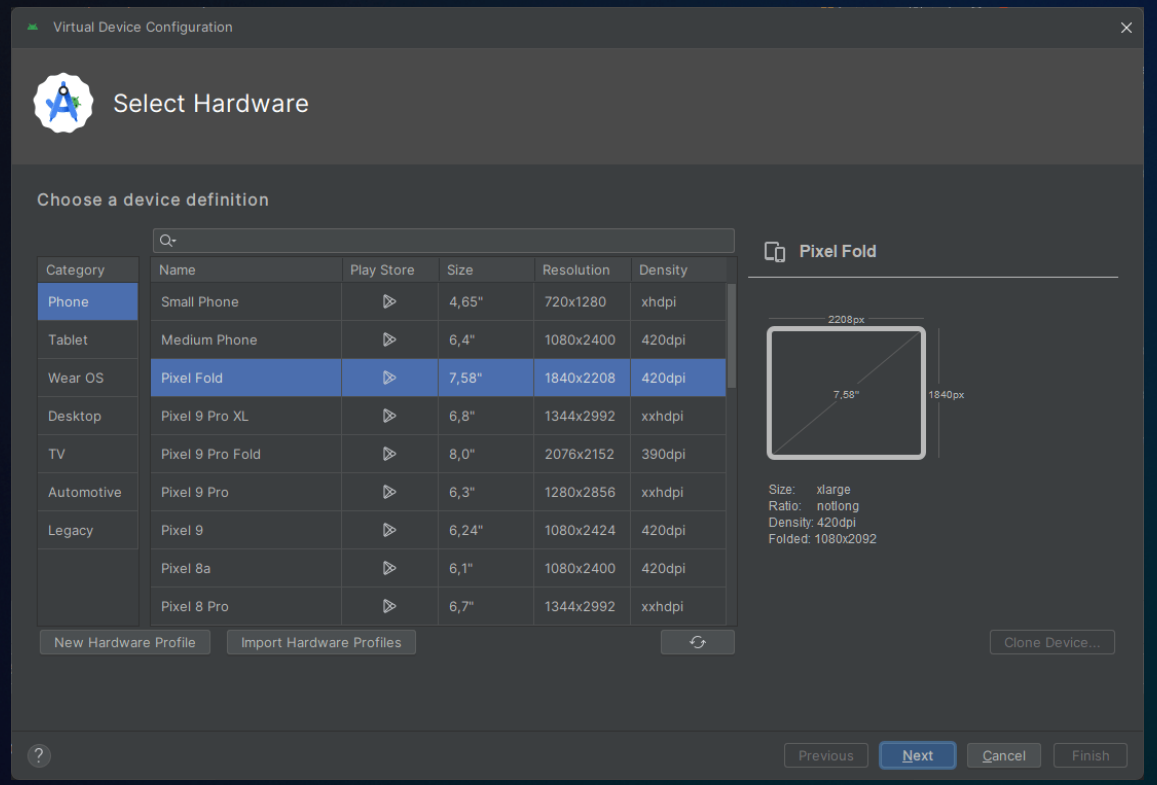
Exécution d'un Projet Android

Deux possibilités pour exécuter votre projet :

- Sur émulateur : Testez votre application sans appareil physique en utilisant un émulateur Android intégré.
- Sur smartphone : Exécutez votre application directement sur un appareil Android connecté.

Exécution sur émulateur



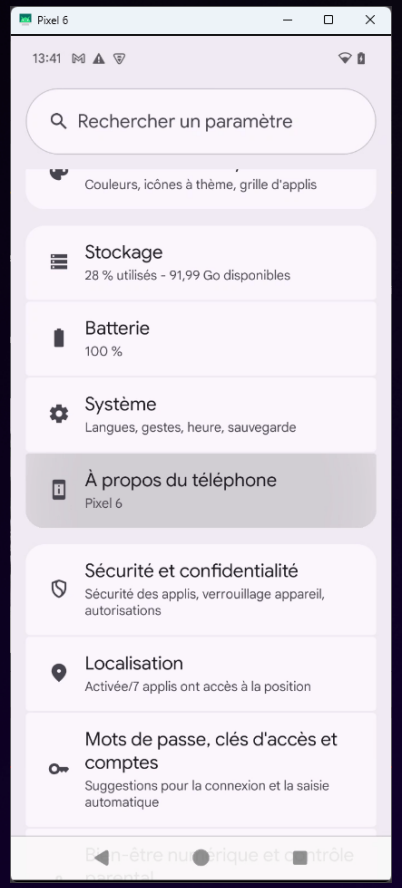


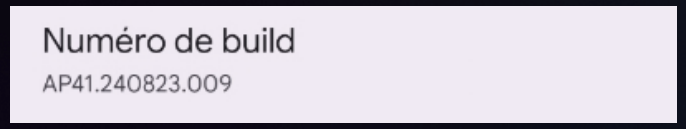
1. **Configurer l'émulateur**
   - Accéder au Device Manager
   - **Créer un appareil** : Séléctionnez le type d'appareil et la version d'Android souhaitée.

2. **Lancer l'émulateur** : Démarrez l'émulateur que vous venez de configurer.

3. **Exécuter le projet** : Cliquez sur le bouton "Run"

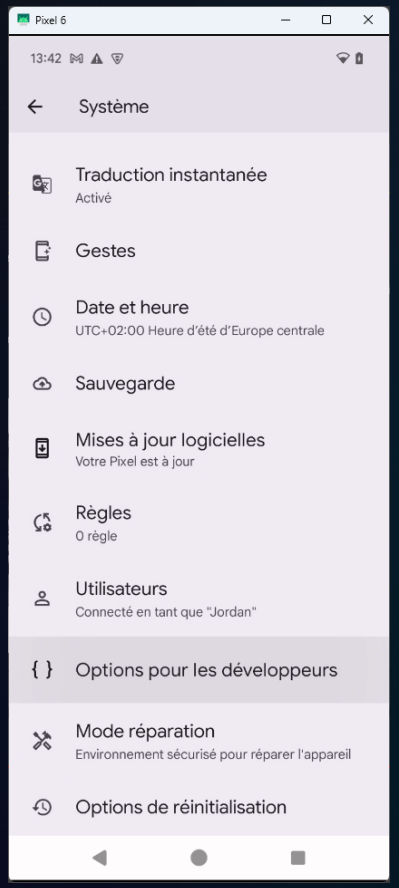
Exécution sur smartphone

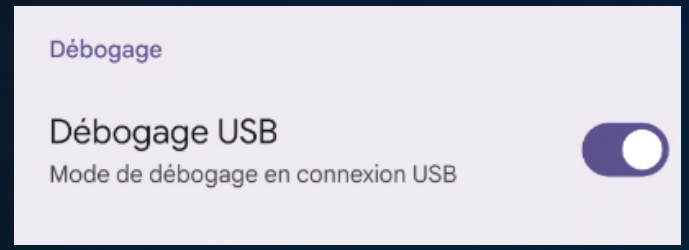




1. Activer les options développeurs :

Dans les "Paramètres" > "À propos du téléphone" > tapotez "Numéro de build" 7 fois pour activer les options développeur.





2. Activer le débogage USB :

Dans les "Options pour les développeurs"

Principes de base des applications

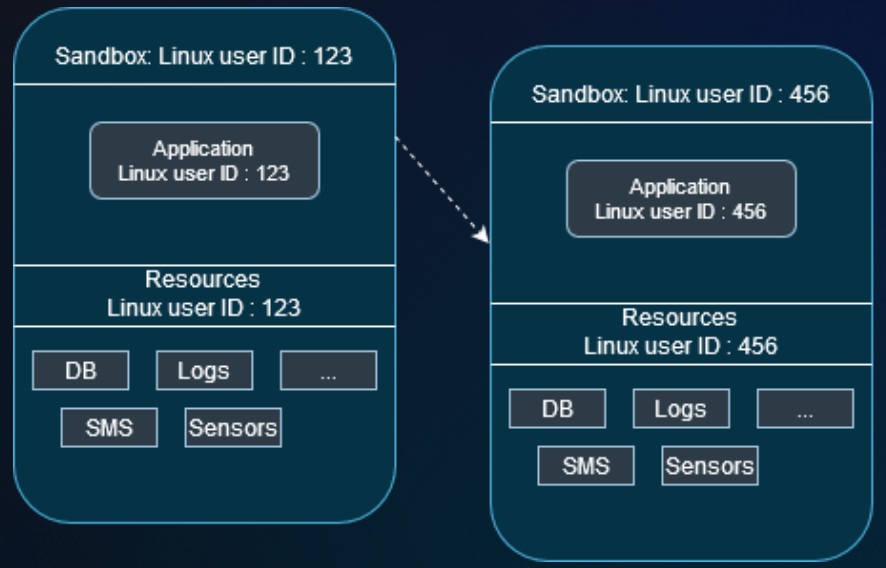
- **Langages de développement** : Les applications Android peuvent être écrite en Kotlin, en Java ou en C++.

- **Formats de distribution** :

  - **APK (Android Package)** : Fichier archive avec l'extension `.apk`, contenant tous les éléments nécessaires au fonctionnement de l'application lors de son exécution. Utilisé pour installer l'application sur les appareils Android.

  - **AAB** (**Android App Bundle**) : Fichier archive avec l'extension `.aab`, incluant les contenus d'un projet Android avec des métadonnées supplémentaires. C'est un format de publication non installable directement sur les appareils. Lors de l'installation depuis le Play Store, les serveurs génèrent des APK optimisés contenant uniquement les ressources et le code nécessaires pour l'appareil qui demande l'installation.

Sécurité des Applications Android



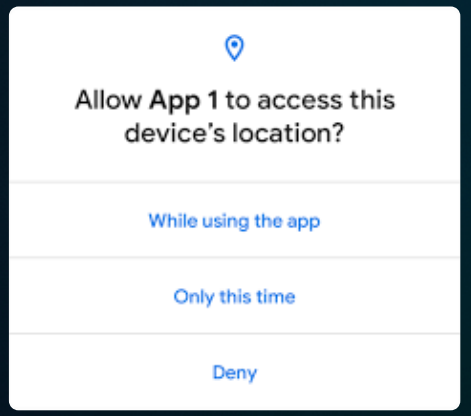
- **Sandbox de sécurité** : Chaque application Android fonctionne dans son propre environnement sécurisé (sandbox).
- **Isolation par utilisateur** : Android est un système Linux multi-utilisateurs, où chaque application est traitée comme un utilisateur distinct.
- **Identifiant utilisateur unique** : Le système attribue à chaque application un identifiant utilisateur (UID) unique. Cet UID est utilisé uniquement par le système et garantit que seuls les fichiers de l'application peuvent être accédés par elle-même.
- **Isolation des processus** : Chaque application s'exécute dans une machine virtuelle dédiée (VM), ce qui garantit que son code est isolé de celui des autres applications.
- **Processus indépendant** : Par défaut, chaque application est exécutée dans un processus Linux distinct, démarré et arrêté par le système Android en fonction des besoins en ressources.

Sécurité et Permissions dans Android

- **Principe de moindre privilège** : Chaque application a seulement les accès nécessaires à ses fonctionnalités, limitant les risques de sécurité.

- **Partage de ressources, deux applications peuvent** :
  - Partager un même **UID** pour accéder aux fichiers de l'autre.
  - Utiliser le même processus et VM, si elles sont signées avec le même certificat.

- **Permissions spécifiques** :
  - Pour accéder à des fonctionnalités comme la **caméra**, la **localisation**, ou **Bluetooth**, l'application doit obtenir l'autorisation explicite de l'utilisateur.



Les composants d'une application Android

Les composants sont les éléments de base d'une application Android. Ils permettent au système ou à l'utilisateur d'interagir avec l'application.

Chaque composant à un rôle spécifique et un cycle de vie qui lui est propre.

Activités (Activities)

Une **activité** représente un écran unique avec lequel l'utilisateur interagit. C'est un composant fondamental dans une application Android.

Exemple dans une application de shopping :

- Affichage du catalogue de produits
- Détails d'un produit spécifique
- Validation et paiement de la commande

Fonctionnalités principales :

- Gère ce que l'utilisateur voit à l'écran (interface active).
- Assure la restauration de l'état si l'application est interrompue.
- Facilite la navigation entre différentes activités ou applications (ex. : partager un produit avec une autre app).

Les activités sont définies en tant que sous-classe de `Activity`.

Services

Un service est un composant qui permet à une application de continuer à s'exécuter en arrière-plan sans nécessiter d'interface utilisateur.

Exemple d'utilisation :

- Lire de la musique en arrière-plan pendant que l'utilisateur utilise une autre application.
- Synchroniser des données en arrière-plan.

Deux types de services :

- **Services démarrés (Started services)** : Continuent à s'exécuter jusqu'à ce que le travail soit terminé. Exemple : lecture de musique ou synchronisation de données.
- **Services liés (Bound services)** : Fournissent une API pour qu'un autre processus interagisse avec eux. Exemple : un processus A utilisant un service B.

Les services sont définis en tant que sous-classe de `Service`.

Broadcast Receivers

Un composant qui permet à l'application de recevoir des événements en dehors du flux normal de l'utilisateur, réagissant à des évènements globales du système.

Fonctionnement :

- Recevoir des notifications même si l'application n'est pas en cours d'exécution.
- Exemple : Un alarme pour notifier l'utilisateur d'un événement à venir

Origine des broadcasts :

- Événements système : écran éteint, batterie faible, changement de réseau.
- Événements applications : les applications peuvent initier des broadcasts pour informer d'autres applications.

Les broadcast receivers sont définis comme des sous-classes de `BroadcastReceiver` et chaque broadcast est délivré sous forme d'objet `Intent`.

Content Providers

Un content provider gère un ensemble de données partagées qu'une application peut stocker dans le système de fichiers, une base de données SQLite etc. Les autres applications peuvent interroger ou modifier ces données via le content provider.

Exemple : Le content provider des contacts permet aux apps d'accéder aux informations de contact, comme les noms et numéros de téléphone.

Sécurité :

- Les données sont accessibles via des URIs, permettant un accès sécurisé. Par exemple, un app peut partager une image via un URI tout en gardant le content provider protégé.

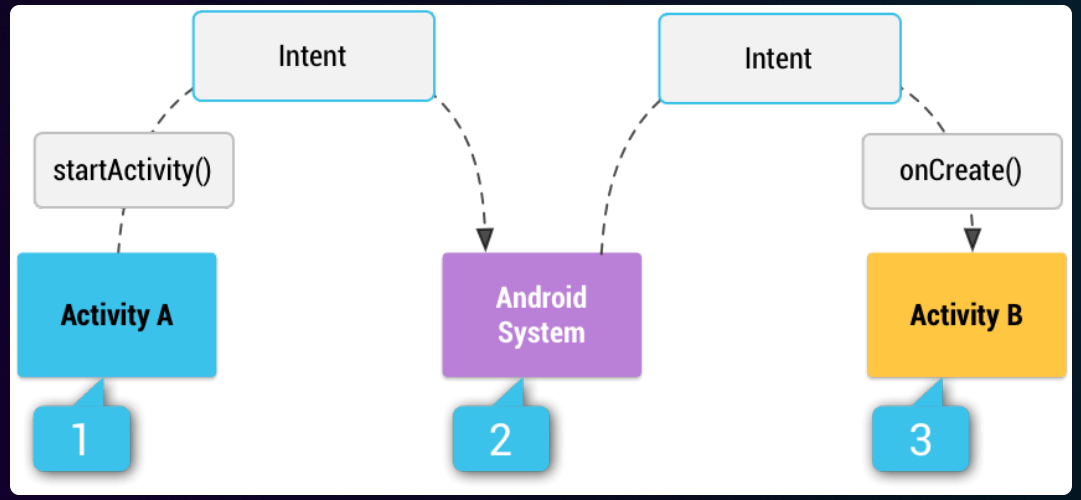
Les content providers sont définis comme des sous-classes de `ContentProvider` et doivent implémenter un ensemble d'API permettant d'effectuer les transactions.

Interaction entre applications Android

L'une des particularités d'Android est la possibilité pour une application de démarrer un composant d'une autre application :

- **Exemple pratique :** Si votre application a besoin de prendre une photo, elle peut déclencher l'activité caméra d'une autre app déjà installée, sans avoir à coder cette fonctionnalité. Une fois la photo capturée, elle est renvoyée à votre app comme si la caméra faisait partie intégrante de celle-ci.

- **Fonctionnement interne :** Bien que chaque application soit isolée dans son propre processus, le système Android peut activer les composants d'autres apps en réponse à des messages d'**intention** (`Intent`). Par exemple, si l'appareil photo est lancé, il s'exécutera dans le processus de l'application caméra, non dans celui de votre app.



Activer des composants avec les Intents

Un **intent** est un message asynchrone utilisé pour activer des **activités**, des **services**, ou des **broadcast receivers**.

Il est créé à l'aide d'un objet Intent qui définit un message qui active soit un composant spécifique (**explicite**), soit un type de composant (**implicite**).

- **Explicite** :
  - Déclenche un composant spécifique.
  - *Exemple* : Lancer l'activité de connexion de votre application.

- **Implicite** :
  - Déclenche un type de composant basé sur une action.
  - *Exemple* : Ouvrir un lien web.

Activer des composants avec les Intents

```
1 // Démarrer une activité
2 val intent = Intent(this, TargetActivity::class.java)
3 startActivity(intent)
```

```
1 // Démarrer un service
2 val intent = Intent(this, MyService::class.java)
3 startService(intent)
4
5 // Lier à un service
6 bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE)
```

```
1 // Envoyer un broadcast
2 val intent = Intent("com.example.ACTION")
3 sendBroadcast(intent)
```

Intents implicites : Exemples d'utilisation

```
1 val webIntent = Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com"))
2 startActivity(webIntent)
```

```
1 val emailIntent = Intent(Intent.ACTION_SENDTO).apply {
2     data = Uri.parse("mailto:example@example.com")
3     putExtra(Intent.EXTRA_SUBJECT, "Subject Here")
4     putExtra(Intent.EXTRA_TEXT, "Body of the email")
5 }
6 startActivity(emailIntent)
```

```
1 val mapIntent = Intent(Intent.ACTION_VIEW, Uri.parse("geo:37.7749,-122.4194"))
2 startActivity(mapIntent)
```

```
1 val callIntent = Intent(Intent.ACTION_DIAL).apply {
2     data = Uri.parse("tel:123456789")
3 }
4 startActivity(callIntent)
```

Pour que tous ces éléments fonctionnent ensemble, il faut un point de liaison essentiel...

Le Manifeste

Le fichier `AndroidManifest.xml` est crucial pour le fonctionnement de votre application Android. Il permet au système de reconnaître les composants de l'application et joue plusieurs rôles essentiels :

- **Déclaration des composants** : Identifie toutes les activités, services, et autres composants de l'application.
- **Permissions utilisateur** : Énonce les autorisations requises, comme l'accès à Internet ou aux contacts.
- **Niveau d'API minimum** : Indique le niveau d'API requis pour l'application, en fonction des API utilisées.
- **Fonctionnalités matérielles et logicielles** : Spécifie les fonctionnalités nécessaires, comme la caméra ou le Bluetooth.
- **Bibliothèques API** : Déclare les bibliothèques externes avec lesquelles l'application doit être liée.

Déclaration des composants

Le manifeste a pour tâche principale d'informer le système des composants de l'application. Par exemple, un fichier manifeste peut déclarer une activité comme suit :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest ... >
3     <application android:icon="@drawable/app_icon.png" ... >
4         <activity android:name="com.example.project.ExampleActivity"
5                   android:label="@string/example_label" ... >
6         </activity>
7         ...
8     </application>
9 </manifest>
```

Dans l'élément `<activity>`, l'attribut `android:name` spécifie le nom de la classe de l'Activity

Déclaration des composants

Tous les composants de l'application doivent être déclarés à l'aide des éléments suivants :

- `<activity>` pour les activités
- `<service>` pour les services
- `<receiver>` pour les broadcast receivers
- `<provider>` pour les content providers

Les activités, services et fournisseurs de contenu inclus dans votre code source mais non déclarés dans le manifeste ne seront pas visibles par le système et ne pourront donc jamais s'exécuter.

Déclaration des capacités des composants

Pour qu'un **Intent** puisse démarrer un composant, celui-ci doit indiquer ses capacités au système.

Lorsque le système reçoit un Intent, il identifie les composants susceptibles de répondre en comparant l'Intent avec les filtres d'intent déclarés dans les manifestes des applications installées.

En déclarant un composant dans le manifeste, on peut inclure des filtres d'intent pour exposer ses capacités.

Cela se fait en ajoutant un élément `<intent-filter>` en tant qu'enfant du composant concerné.

Exemples de filtre d'intent

Pour illustrer comment déclarer un filtre d'intent, prenons l'exemple d'une application de messagerie. Si vous avez une activité dédiée à la rédaction d'un nouvel email, vous pouvez déclarer un filtre d'intent pour répondre à des intents "SEND"

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Cas particulier : Catégorie Launcher

La catégorie **Launcher** indique au système que cette activité est la principale de l'application. Cela permet à l'utilisateur de la trouver facilement dans le lanceur d'applications.

```
<manifest ... >
    ...
    <application ... >
        <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>
```

Sans cet **Intent Filter**, l'application n'apparaîtra pas dans le lanceur d'applications du téléphone.

Conclusion : Un aperçu d'Android

Récapitulatif des points abordés :

- Architecture d'Android (couches du système, interaction avec le hardware).
- Les 4 composants principaux des applications Android (activités, services, broadcast receivers, content providers).
- Le rôle central du fichier `AndroidManifest.xml` dans la configuration des composants.
- Les **Intents** comme moyen de communication entre les composants.
- Importance des permissions et du sandboxing pour la sécurité.

Pourquoi Android est un OS unique :

- Modularité et capacité d'interaction entre les apps.
- Flexibilité grâce à l'open source et la grande diversité de matériels.