DOCUMENTATION TECHNIQUE

BILL–CUTTING MOBILE

3PROJ

RYAN DORDAIN / SERAPHIN DUBAIL / TRISTAN TOURBIER / THOMAS LAPERE

Table des matières

1- Choix des technologies :

Le projet a explicitement exclu l'utilisation de technologies hybrides comme Flutter, React Native, etc. Donc nous avons choisi de poursuivre le développement sur Android.

Android est un système d'exploitation mobile développé par Google, utilisé principalement sur les smartphones et les tablettes. Basé sur le noyau Linux, il offre une plateforme ouverte aux développeurs pour créer des applications en utilisant des langages comme Java et Kotlin.

Nous avons privilégié Kotlin à Java, qui est le langage de programmation le plus récent pour Android.

Kotlin est un langage de programmation moderne développé par JetBrains, conçu pour interopérer avec Java et principalement utilisé pour le développement Android. Kotlin est officiellement supporté par Google comme langage privilégié pour le développement d'applications Android.

Nous avons choisi Ktor pour gérer nos requêtes API car nous avons trouvé que c'était le plus simple à utiliser.

Ktor est un framework asynchrone développé par JetBrains pour créer des applications web et des microservices en Kotlin. Il sert à gérer les communications réseau de mon application Android, facilitant l'envoi de requêtes HTTP et la réception de réponses d'une API.

Le sujet nous a demandé d'implémenter une solution de connexion par tiers (Google). Ainsi, nous avons choisi OAuth2.

OAuth 2.0 est un protocole d'autorisation qui permet à une application tierce d'accéder aux ressources d'un utilisateur sur un serveur sans exposer les informations de connexion de l'utilisateur. Il utilise des jetons d'accès pour accorder des permissions spécifiques, améliorant ainsi la sécurité et le contrôle d'accès. OAuth 2.0 est couramment utilisé pour l'authentification sur des sites web et des applications mobiles, permettant des intégrations sécurisées avec des services tiers comme Google, Facebook, et autres.

Nos utilisateurs peuvent s'envoyer des messages sur l'application Android, mais pour que tout cela fonctionne bien, nous avons mis en place des websockets.

Les WebSockets pour Android permettent une communication bidirectionnelle en temps réel entre une application et un serveur. Ils établissent une connexion persistante, idéale pour des applications nécessitant des mises à jour instantanées, comme les messageries ou les jeux en ligne. En utilisant des bibliothèques telles que Ktor, les développeurs peuvent facilement intégrer les WebSockets dans leurs applications Android.

1.1- Avantages :

Android

Le fait d'avoir choisi Android pour développer notre application mobile présente plusieurs avantages :

Open Source : Android est un système d'exploitation open source, ce qui permet aux développeurs d'accéder au code source, de le modifier et de l'adapter à leurs besoins. Cela favorise l'innovation et la personnalisation, permettant de créer des expériences utilisateur uniques.

Large Base d'Utilisateurs : Android est le système d'exploitation mobile le plus répandu dans le monde, avec une part de marché significative. Cela offre aux développeurs une

large audience potentielle pour leurs applications, augmentant ainsi les chances de succès et de rentabilité.

Diversité des Appareils : Android fonctionne sur une variété d'appareils, des smartphones aux tablettes en passant par les montres connectées et les téléviseurs. Cela permet aux développeurs de créer des applications polyvalentes et d'explorer différents marchés au sein de l'écosystème Android.

Kotlin

Concision : Kotlin permet de réduire la quantité de code nécessaire, rendant le développement plus rapide et le code plus facile à lire et à maintenir par rapport à Java.

Coroutines : Les coroutines de Kotlin simplifient le développement asynchrone et multithread, facilitant la gestion des opérations longues comme les requêtes réseau ou les accès à la base de données.

Fonctionnalités modernes : Kotlin offre des fonctionnalités modernes telles que les extensions de fonctions, les types génériques et les expressions lambda, améliorant la flexibilité et la puissance du langage.

Support officiel : Kotlin est officiellement supporté par Google pour le développement Android, garantissant une intégration fluide avec les outils de développement Android et une forte communauté de support.

Ktor

La raison principale du choix de Ktor est la raison suivante :

Support de coroutines : Intégrant les coroutines de Kotlin, Ktor permet une gestion asynchrone et non bloquante des requêtes, ce qui améliore la réactivité et la scalabilité des applications.

Interopérabilité : Ktor s'intègre facilement avec d'autres bibliothèques Kotlin et Java, permettant aux développeurs de tirer parti de l'écosystème existant et d'utiliser des outils et des bibliothèques déjà familiers.

Performances optimisées : Ktor est conçu pour être léger et performant, ce qui permet de gérer efficacement un grand nombre de requêtes API simultanées sans surcharger le serveur. Cela garantit des temps de réponse rapides et une meilleure expérience utilisateur.

2- Collaboration :

Le développement d'une application complexe sur Android est difficile à réaliser seul. Il était évident que notre équipe devait se diviser les tâches sur le projet. Pour cela, nous avons travaillé avec un outil de collaboration qui est Git.

Git est un système de contrôle de version distribué qui permet aux développeurs de suivre les modifications du code source, de collaborer efficacement et de gérer différentes versions d'un projet. Il permet de créer des branches pour développer des fonctionnalités indépendamment, puis de fusionner ces changements dans la branche principale. Git est largement utilisé pour le développement de logiciels en raison de sa flexibilité et de sa robustesse.

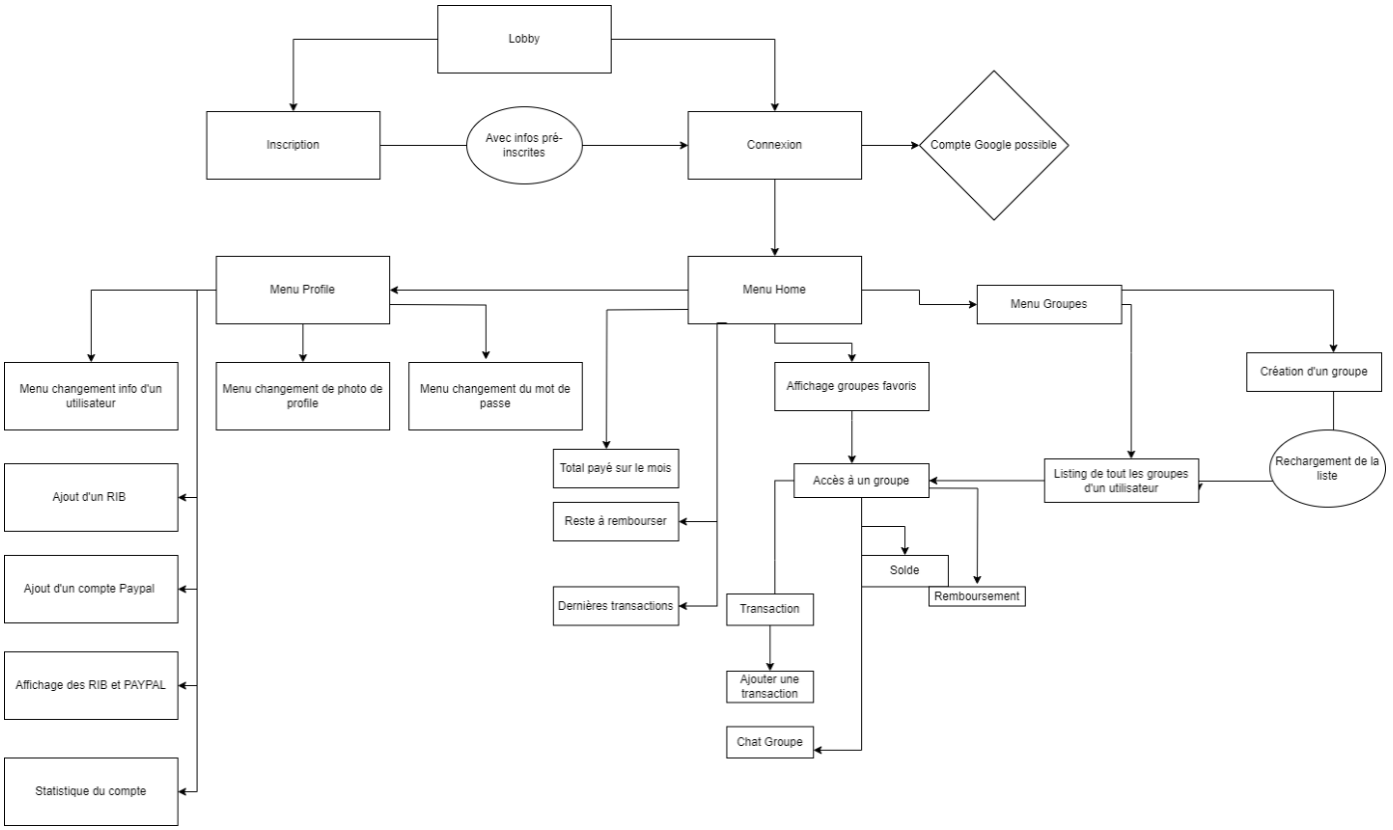
Cela nous a notamment permis d'avancer ensemble sur le projet sans pour autant causer des problèmes de compatibilité entre les codes.

Pour ce qui est des requêtes, il était évident de ne pas utiliser des requêtes vers une base de données mais bel et bien utiliser une API REST développée par notre équipe afin d'accroître la sécurité de notre application et d'éviter certaines failles.

3- Architecture de l'application :

Cheminement de l'application

Afin de mieux comprendre le cheminement de notre application, vous trouverez ci-dessous un schéma :

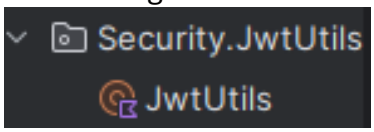


Structure de l’application

Pour notre application, nous avons une structure permettant de mieux organiser nos différentes méthodes et de gérer l'affichage.

JWT

Le décodage des tokens :



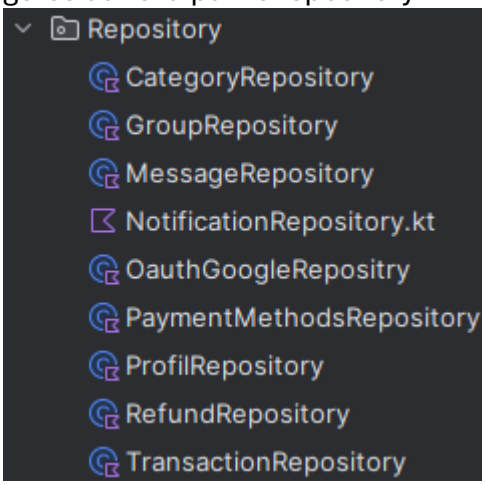
```
object JwtUtils {
    Tristan Tourbier
    fun decodeJWT(jwt: String): String {
        val parts = jwt.split( ...delimiters: ".")
        if (parts.size != 3) {
            throw IllegalArgumentException("Invalid JWT token")
        }

        val decodedBytes = Base64.decode(parts[1], Base64.DEFAULT)
        return String(decodedBytes, Charsets.UTF_8)
    }
}
```

Nous stockons notamment le token récupéré lors de la connexion dans le MainActivity afin de pouvoir l'utiliser dans toute l'application :

```
var LocalJwtToken = compositionLocalOf<String> { error("No JWT Token provided") }
```

Repository

La gestion des requêtes vers l'API et le traitement des résultats sont principalement gérés dans la partie repository :



Nous pouvons nous pencher sur le cas de la création d'un groupe dans GroupRepository :

L'association de Kotlin et de Ktor pour gérer les requêtes a permis de réaliser ce code.

Nous allons expliquer les points importants d'une requête basique :

`@OptIn(InternalAPI::class) ` :
- Indique que la fonction utilise des fonctionnalités marquées comme internes ou expérimentales, nécessitant une autorisation explicite pour être utilisées.

Déclaration de la fonction `suspend fun createGroup` :
- La fonction est déclarée avec le mot-clé `suspend`, ce qui signifie qu'elle est conçue pour être appelée dans une coroutine et peut suspendre son exécution sans bloquer le thread actuel.

Paramètres de la fonction :
- `jwtToken` : Jeton JWT utilisé pour l'authentification.
- `groupName` : Nom du groupe à créer.
- `groupDescription` : Description du groupe à créer.

5. Bloc `withContext(Dispatchers.IO) ` :
- Change le contexte d'exécution à `Dispatchers.IO`, optimisé pour les opérations d'entrées/sorties comme les requêtes réseau.

6. Requête HTTP POST :
- `httpClient.post("https://3proj-back.tristan-tourbier.com/api/groups")` : Envoie une requête POST à l'URL de notre API, c'est le lien de prod.

- `contentType(ContentType.Application.Json)` : Définit le type de contenu de la requête comme JSON.
- `header ("Authorization", "Bearer $jwtToken") ` : Ajoute un en-tête HTTP pour l'authentification avec le jeton JWT.
- `body = """{ "name" : "$groupName", "description" : "$groupDescription"}"""` : Définit le corps de la requête avec le nom et la description du groupe en format JSON.

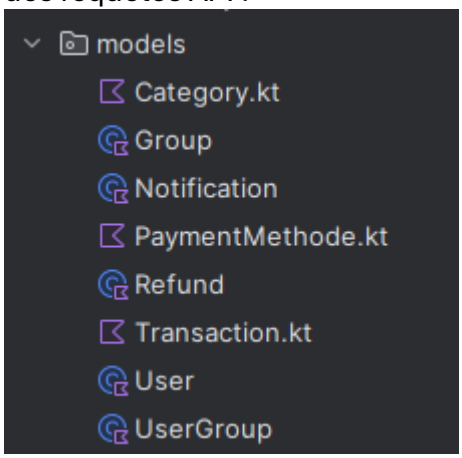
En résumé, cette fonction `createGroup` envoie de manière asynchrone une requête HTTP POST pour créer un groupe sur le serveur spécifié, tout en gérant les éventuelles erreurs de manière appropriée.

Et le modèle de données qui se trouve dans le dossier models -> Group.

Nous avons ici la classe Group

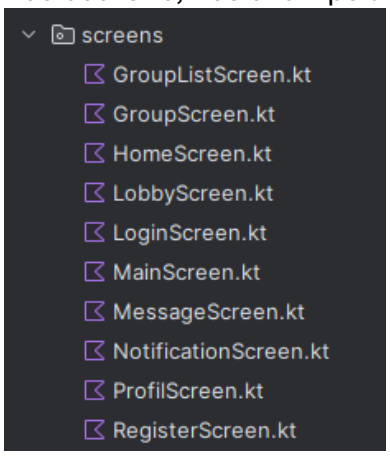
```kotlin
@Serializable
data class Group(
    val id: String? = "",
    val name: String? = "",
    val description: String? = "",
    val picture: List<String?>? = emptyList(),
    val ownerId: String? = "",
    val createdAt: String? = "",
    val updatedAt: String? = "",
    val activeUsersCount: Int? = 0,
    val isFavorite: Boolean? = false,
    val Users: List<User> = emptyList(),
)
```

Et l'utilisation de différents modèles de données est rendue possible grâce aux résultats des requêtes API :

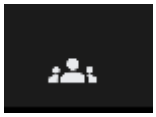


Screens

Notre application est composée de Screens, qui sont des vues permettant d'afficher nos boutons, nos champs de texte, etc., tout ce qui concerne l'aspect visuel.



Nous allons maintenant examiner la continuité avec GroupListScreen, qui permet l'affichage des groupes d'un utilisateur ainsi que la création d'un nouveau groupe.

Nous allons cliquer sur 

Et nous voyons notre écran de liste des groupes :

Nous cliquons ensuite sur le bouton + pour créer un groupe.

Le pop-up ci-dessus permet, avec le bouton "Création du Groupe", de créer un groupe grâce à l'appel de ce code :

```
confirmButton = {
    Button(
        onClick = {
            CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
                groupRepository.createGroup(jwtToken, groupName.value, groupDescription.value)
                showDialog.value = false
                groups.value = groupRepository.getUserGroups(userId, jwtToken, favorite: false)
            }
        },
        colors = ButtonDefaults.buttonColors(
            backgroundColor = Color( color: 0xFFFFA31A)
        )
    ) { this: RowScope
        Text( text: "Création du Groupe")
    }
},
```

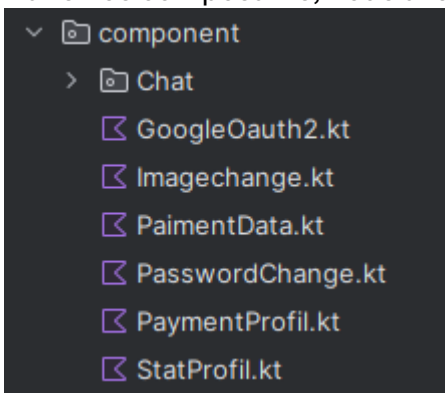
La phase la plus importante de ce code est :
groupRepository.createGroup

Cela appelle une fonction suspendue pour créer un groupe. Cette fonction envoie une requête réseau pour créer un groupe avec les informations fournies, en utilisant le code de notre fonction `createGroup` située dans le repository.

Composant

Nous utilisons principalement les components dans notre application afin de mieux segmenter notre affichage afin d'éviter d'avoir des fichiers screen trop volumineux qui gère le style ce qui rendrais l'application difficilement maintenable, dans certains cas exceptionnels nous stockons un peu de logique et quelques requêtes API si nous avons eu certains problèmes de débug

Dans nos composants, nous avons :



- La partie de chat dans les groupes.
- Le changement d'image de profil.
- La récupération des différents moyens de paiement enregistrés sur un compte.
- Le changement de mot de passe du compte.
- L'ajout de moyen de paiement sur le compte.
- L'affichage des statistiques du compte.

4- Affichage des écrans

Pour naviguer à travers les écrans, il est inévitable de faire des liens entre eux.
Ainsi, toute la logique d'affichage des différents écrans se retrouve dans MainActivity.kt.

Nous allons maintenant examiner l'affichage de l'accueil appelé "home" dans notre application.

Le code ci-dessous se situe dans MainActivity et permet de gérer l'affichage de nos Écrans.

```
composable( route: "home") { it: NavBackStackEntry
    MainScreen(navController) {
        HomeScreen(HttpClient(), navController)
    }
}
```

Composable("home") {Définition de la destination de navigation : Cette ligne indique qu'il y a une destination de navigation nommée "home". Lorsque l'utilisateur navigue vers cette destination, le contenu défini à l'intérieur du bloc sera affiché.

Ainsi, dans le code de LoginScreen.kt, nous retrouvons cette ligne, qui permet de naviguer vers l'accueil (home) :

```
navController.navigate( route: "home")
```

Ce qui va permettre par la suite d'afficher notre MainScreen, le MainScreen englobe notre Footer et notre Header.
Et ajouter l'affichage de notre HomeScreen qui comporte l'affichage des Groupes favoris, les différents soldes.

Nous avons notamment pour HomeScreen en paramètre HttpClient, ce qui va permettre de faire nos différentes requêtes API.

Dans certains cas, nous passons aussi notre token de connexion pour pouvoir l'utiliser, comme pour le cas du login :

```
composable( route: "login") { it: NavBackStackEntry
    LoginScreen(navController, HttpClient(), jwtToken, user, username: "", password: "")
}
```

Nous apercevons que nous passons le jwtToken mais aussi les infos utilisateur. Dans ce cas-ci, c'est pour récupérer les informations mises dans le registre afin que l'utilisateur final n'ait pas besoin de tout réécrire.

5- Requêtes API

Comme expliqué précédemment, nous utilisons Ktor pour effectuer nos différentes requêtes sur notre URL d'API.

Nous allons rapidement décortiquer une requête pour mieux comprendre en détail :

```
val response: HttpResponse = withContext(Dispatchers.IO) { this: CoroutineScope
    httpClient.get( urlString: "https://3proj-back.tristan-tourbier.com/api/users/${userId}/transactions/thisMonth")
        contentType(ContentType.Application.Json)
        header("Authorization", "Bearer $jwtToken")
    }
}
return if (response.status == HttpStatusCode.OK) {
    val responseBody = response.body<String>()
    val jsonObject = JSONObject(responseBody)
    val amount = jsonObject.getInt( name: "amount").toFloat()
    val transactions = jsonObject.getInt( name: "transactions")
    Pair(amount, transactions)
} else {
    println("Error: ${response.status}")
    Pair(0.0f, 0)
}
catch (e: Exception) {
    println("Error: $e")
    return Pair(0.0f, 0)
```

Ici, nous allons voir la récupération du total payé en 1 mois :

Nous utilisons donc httpClient issu de Ktor pour envoyer une requête GET. La particularité de cette requête est que nous incluons userId, ce qui permet de récupérer

l'identifiant de l'utilisateur connecté pour envoyer la requête. Nous envoyons également le token de connexion dans l'en-tête, qui est jwtToken.

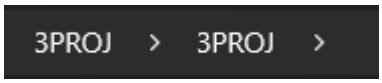
return if (response.status == HttpStatusCode.OK)

Cela permet d'exécuter une action si la requête renvoie une réponse OK (200). Ensuite, nous avons des valeurs qui vont lire le corps de la réponse HTTP et les convertir en chaîne (String). Sinon, nous envoyons la réponse d'erreur.

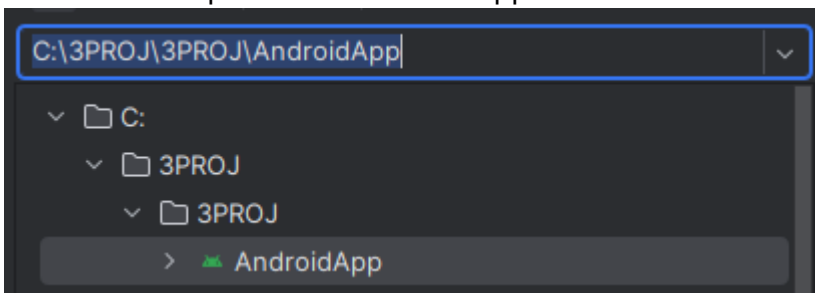
6- Lancement du projet

Pour lancer notre projet, il est nécessaire d'exécuter le projet dans un environnement qui prend en compte Android. Le plus simple est de télécharger le logiciel Android Studio.
[Télécharger Android Studio et les outils pour les applications – Développeurs Android | Android Studio | Android Developers]

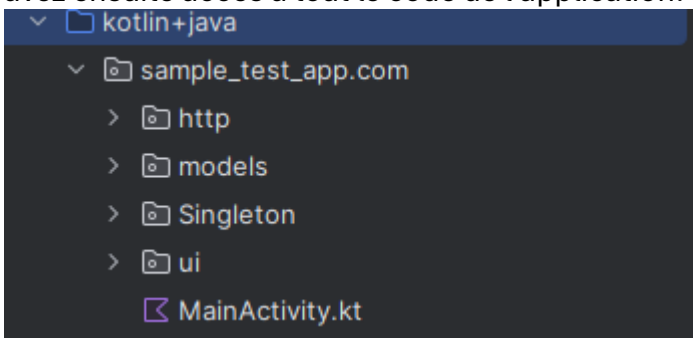
Par la suite, vous ouvrez Android Studio et suivez le chemin :



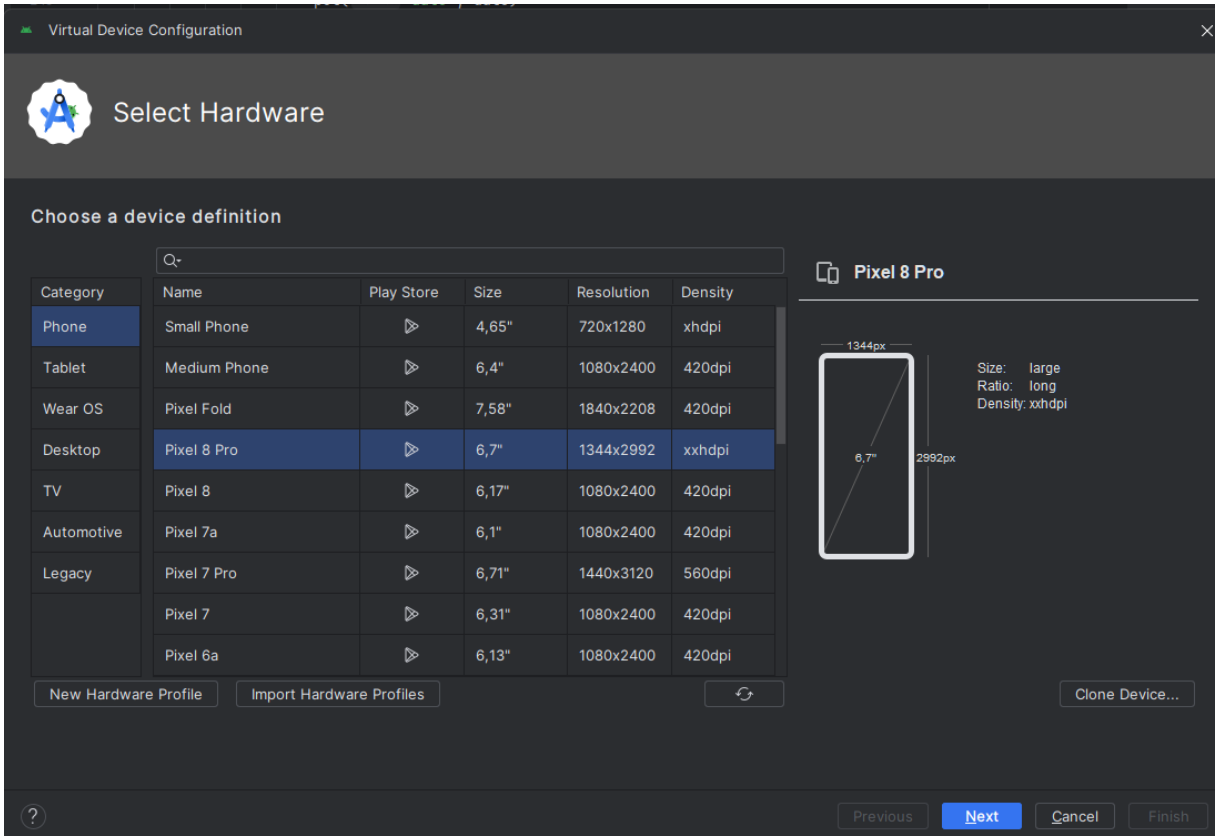
Et vous ouvrez par la suite AndroidApp



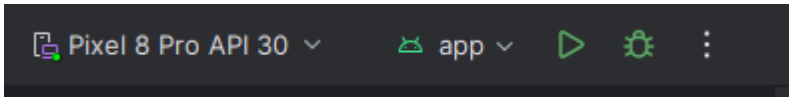
Attendez que le logiciel télécharge les dépendances et les différents packages. Vous avez ensuite accès à tout le code de l'application.



Installer par la suite un téléphone d’émulation



Le Pixel 8 Pro est le meilleur choix pour lancer l'application. Ensuite, cliquez sur "Play".



Et l'application se lancera.