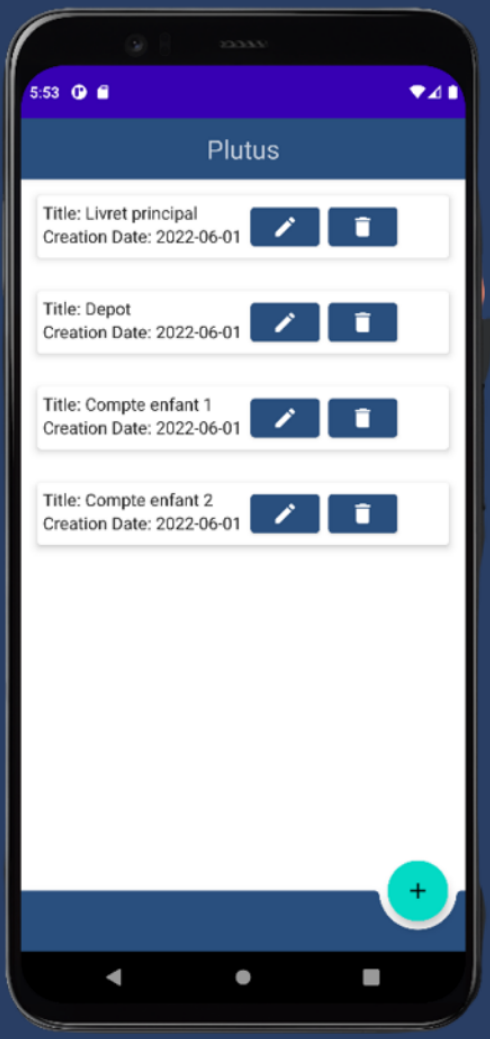
PLUTUS

Documentation développeur

JUIN 2022

BARROUX LÉO
RODRIGUEZ MAXIME



Esipe
INGÉNIEURS PARIS EST

INFO
INGÉNIEUR
ESIPE

Introduction

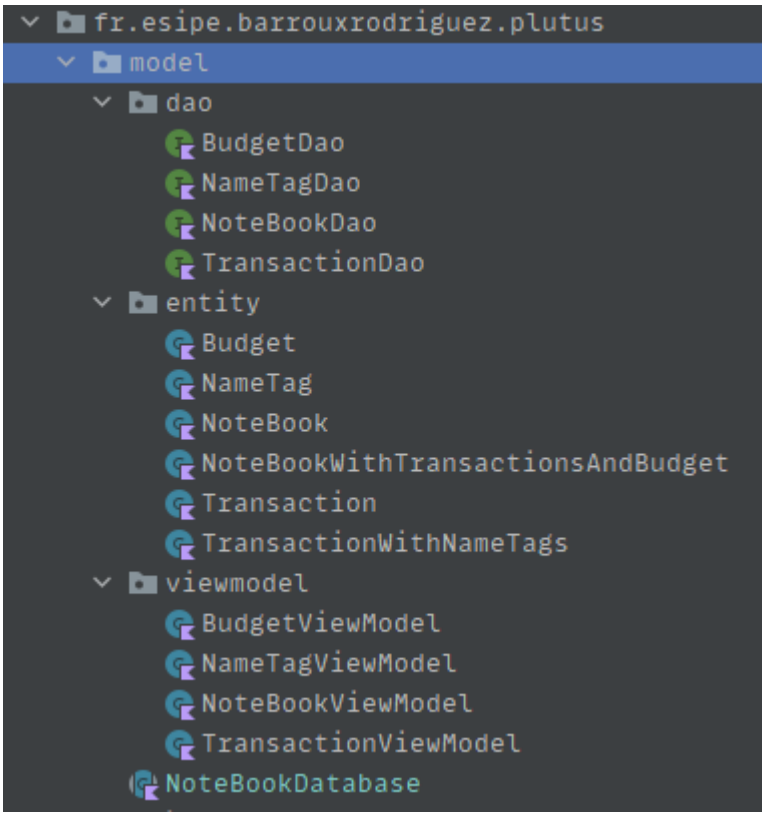
Ceci est la documentation développeur, elle permet à un developpeur de se retrouver au sein de notre application.
Nous allons détailler notre architecture de code.

Table des matières

Architecture du code

Notre code se divise en une petite architecture MVC, nous avons alors les packages de model, de controlleur et de vue ainsi qu'un package utils, bien utile.

Model



DAO

Nos interfaces DAO qui vont nous servir à faire des appels à notre base de donnée pour récupérér, ajouter, modifier ou supprimer des données;

Entity

Glossaire:

- NoteBook: Carnet de compte qui contient un titre et une date de création
- NoteBookWithTransactionsAndBudgets: Une entité qui contient un carnet de compte, une liste de transaction et une liste de budget associées à ce carnet
- Transaction: Une transaction avec un titre, une date, un montant
- TransactionWithNameTags: Une entité qui contient une transaction et sa liste de NameTag associée
- NameTag (Etiquette): Etiquette qui contient un titre, et un boolean désignant s'il sagit d'une étiquette par défaut ou non

- Budget: Budget qui représente contient un titre, un montant, une date de fin et de début, ainsi qu'une étiquette (NameTag)

Nos entity, ce sont nos classes ORM qui s'équivaut à nos tables en bases de données.
Notez que l'utilisation de ROOM nous impose de créer des entity spéciale pour les relations entre tables.
Room ne permet d'avoir du fetch Lazy pour des soucis de rapidité d'execution, pour cela nous avons pour les NoteBook ou les Transactions des classes intermédiaires qui vont récuéprerer une entité avec toutes ses données.
L'application direct de cela est que pour nos NoteBook,pour afficher la liste des notebook de l'utilisateur, nous n'avont pas besoin de leurs liste de transactions. Seulement de leurs nom et leurs dates de créations. Cela nous évite de récupérer un flot de donnée important quand nous n'en avons pas besoin.

ViewModel

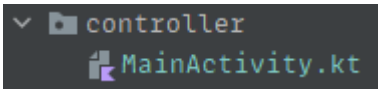
Nos viewmodel qui font office de transition entre la vue et le model. Elles sont utilisé dans les vues pour faire appel au modèle de donnée (les données en bases de données);
Elle utilisent les interfaces pour être créer et utiliser.

Pour la récupération de données, nous avons des LiveData sur les données.
Pour l'ajout, modification et suppression de données, ce sont des méthodes effectuer sur le dispatcher.IO d'android, elles doivent donc être placer à l'interieur et doit appeller des méthodes des DAO suspend.

Database

Notre base de donnée au format d'une classe, selon la documentation ROOM. Cela nous permet de l'instancier en tant que singleton pour la mise en place des viewModel. De plus elle nous permet de gérer la version de notre base de données, sa migration ou son export.

Controller

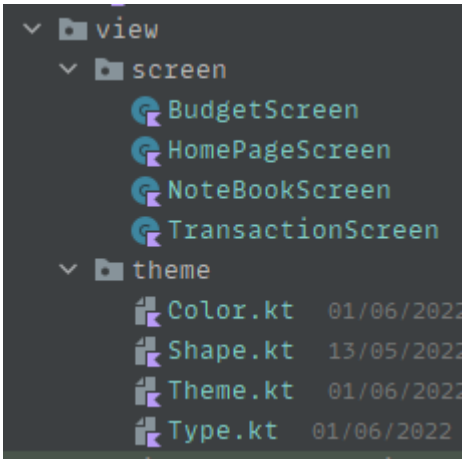


Ce package controller contient notre MainActivity, et contraindra nos futur service et activités que nous mettrons en place.

La mainActivity est un fichier kotlin qui permet d'initialiser les view models et les différentes routes de notre application (grâce à un NavController).

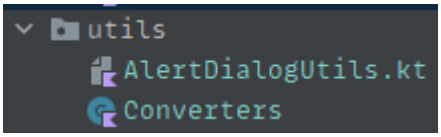
Elle mène ensuite vers la page principale de notre application

View



Le package view va alors contenir les pages de notre applications, chaque pages correspond à une page de notre application. Ces pages contiennent les fonctions Composables qui permettent l'affichage de chaques pages.
De plus nous y avons placer le package theme des project jetpack compose.

Utils



Enfin, le package Utils contient les classes qui nous seront utile pour effectuer des opérations repetitive, comme ici par exemple la convertion de données entre la BDD et le modèle (Surtout les dates).
Mais il y a également un AlertDialogUtils qui nous à permis de factoriser du code pour faciliter la création d'AlertDialog.