Projet du module “Programmation mobile Java”

Application réalisée par **INTISSAR BENNOUNA** GLSI 2023-2024 ENSAF

**PlanetsApp** : Application Crud réaliser avec le langage Java et Room database sous Android studio

**Fonctionnalités** :

* Afficher la liste des planètes avec leurs propriétés : Nom, Distance et Description
* Bouton pour ajouter une nouvelle planète redirige vers le formulaire d’ajout
* Menu avec 3 icons :
  + Actualisation
  + Infos
  + Recherche qui ouvre un Alertdialog pour saisir le nom du planète à rechercher
* Click long sur les éléments de la liste des planètes ouvre un context menu avec deux options :
  + Modifier qui redirige vers le formulaire de modification déjà rempli avec les données de la planète à modifier
  + Supprimer qui ouvre un Alertdialog pour confirmer la suppression du planète choisi

Explication des étapes suivis pour la réalisation de l’application :

1. **Entity Planet** :
   1. Data model class qui représente un objet **Planet**
   2. la notation **@Entity** représente une table “**planet\_table**”, avec une clé primaire **id** généré automatiquement et 3 attributs name, distance et description
   3. Constructeur qui accepte 3 paramètres (name, distance, description)
   4. getters et setters pour tous chaque attribut
2. **PlanetDAO :** 
   1. c’est une interface Data Object Access pour traiter les données des **Planètes**
   2. la notation **@DAO** indique que cette interface contient des méthodes qui interagissent avec la base de donnés  **:** 
      1. la méthode insert avec la notation **@Insert** qui indique qu’il s’agit d’une opération d’insertion
      2. la méthode deleteAll avec la notation **@Query** qui indique qu’il s’agit d’une requête SQL pour la suppression de toutes les planètes
      3. la méthode getAll avec la notation **@Query** qui indique qu’il s’agit d’une requête SQL pour récupérer toutes les planètes **LiveData<List<Planet>>** signifie qu'il fournira des données de manière asynchrone en tant List
      4. la méthode delete avec la notation **@Delete** qui indique qu’il s’agit d’une opération de suppression
      5. la méthode update avec la notation **@Update** qui indique qu’il s’agit d’une opération de modification
      6. la méthode getSearchResult avec la notation **@Query** qui indique une requête SQL pour récupérer les enregistrements de planète qui correspondent à certains critères de recherche basés sur le nom de la planète

**NB** : **LiveData** comme type de retour dans certaines méthodes permet d'observer les modifications dans la base de données et de mettre à jour l'interface utilisateur automatiquement lorsque les données changent.

1. **PlanetDatabase** : représente la base de données avec la notation **@Database**
   1. La méthode abstraite PlanetDAO permet d’obtenir un objet de type PlanetDAO
   2. La constance INSTANCE : utilisée pour stocker l'instance unique de la base de données. Cela permet d'éviter d'ouvrir plusieurs connexions à la base de données, en créant une seule instance partagée entre différentes parties de l'application.
   3. NUMBER\_OF\_THREADS : C'est le nombre de threads utilisés pour exécuter les opérations d'écriture dans la base de données.
   4. databaseWriteExecuto : C'est un ExecutorService utilisé pour exécuter des opérations d'écriture dans la base de données en arrière-plan sur les threads.
   5. La méthode getDatabase est utilisée pour obtenir une instance unique de la base de données
   6. RoomDatabase.Callback : lorsque la base de données est créée pour la première fois insère deux objets Planet par défaut dans la base de données.
2. **PlanetRepository** : pour gérer l'accès aux données pour le reste de l'application, avec deux attributs planetDAO et allPlanets de type **LiveData<List<Planet>>**
   1. **Constructeur** : pour initialiser les attributs, il prend en paramètre une Application qui sera utilisé pour obtenire une instance de la base de données avec la méthode getDatabase
   2. La méthode **getAll** qui renvoie l’objet allPlanets qui contient la liste des planètes
   3. les méthodes **insert, update, delete, deleteAll** pour ajouter, modifier et supprimer une planète dans la base de données, Pour ces opérations, on utilise un **databaseWriteExecutor** pour exécuter les opérations de base de données de manière asynchrone et éviter de bloquer le thread principal de l'application.
   4. classe interne **deletePlanetAsyncTask** c’est une sous classe de AsyncTask utilisé pour supprimer une planète d’une manière asynchrone
   5. la méthode **getSearchResult** qui retourne une liste des planètes ayant le nom passé en paramètre
3. **PlanetViewModel** : joue un rôle essentiel dans la séparation des préoccupations entre les composants de l'interface utilisateur et les données de l'application, il est responsable de la gestion de la communication entre l’interface utilisateur et le repository, a deux attributs planetRepository et allPlanets
   1. **Constructeur** : pour initialiser les attributs
   2. La méthode **getAllPlanets** qui retourne allPlanets
   3. Méthodes **insert, update, delete, et deleteAll, et searchPlanetsName** pour appeler les méthodes correspondantes du repository
4. **PlanetViewHolder** : utilisé avec un RecyclerView pour améliorer les performances lors du recyclage des vues dans une liste ou une grille. Il agit comme un cache pour les vues d'un élément d'une liste, ce qui permet d'éviter de chercher les vues à chaque fois qu'un élément est affiché ou mis à jour.
   1. Avec 3 attributs de type TesxtView planetName, planetDistance et planetDescription, ces attributs sont utilisés pour référencer les TextViews du layout **recyclerview\_item.xml**, qui affiche les informations concernant chaque planète dans la liste.
   2. Le **constructeur** prend une vue (itemView) en paramètre, qui est la vue qui représente chaque élément de la liste, les trois TextViews sont initialisés en utilisant findViewById pour les lier aux éléments correspondants du layout recyclerview\_item.xml. En plus, un OnCreateContextMenuListener qui gère la création d'un menu contextuel pour chaque élément de la liste lorsqu'il est longuement cliqué.
   3. La méthode **bind** utiliser pour lier les données d’une planète spécifique aux TextView correspondants
   4. La méthode **OnCreateContextMenu** est appelé lorsque l’utilisateur effectue un clic long sur une planète
   5. La méthode **create** utiliser pour créer une instance de PlanetViewHolder
5. **PlanetListAdapter** : pour lier une liste des planètes à un recycleview
   1. Le **constructeur** de PlanetListAdapter prend en paramètre un objet DiffUtil.ItemCallback<Planet> qui est utilisé pour comparer les éléments de la liste et déterminer les différences entre la liste précédente et la liste actuelle lors des mises à jour.
   2. La méthode **onCreateViewHolder** responsable de la création d’un nouveau ViewHolder pour chaque élément de la liste
   3. La méthode **onBindViewHolder** responsable de la liaison des données d'un élément spécifique de la liste au ViewHolder correspondant.
   4. La méthode **getPlanetAtPosition** pour obtenir l'objet Planet à une position donnée dans la liste.
   5. **PlanetDiff** class interne utilisé pour déterminer la différence entre 2 Planet
6. **AddNewOlanetActivity** : permet à un utilisateur d’ajouter une nouvelle planète
   1. Constantes sont définies pour les clés des extras (données supplémentaires) qui seront utilisées pour passer des informations entre cette activité et l'activité appelante.
   2. 3 attributs de type EditText planetName, planetDistance et planetDescription
   3. Dans la méthode **onCreate** 
      1. Les trois EditText et le bouton de sauvegarde (button\_save) sont initialisés en utilisant findViewById.
      2. Lorsque le bouton est cliqué on récupère les valeurs saisies , ensuite on vérifie si les champs sont vides si oui une erreur est afficher sinon une nouvelle instance Planet sera créer avec les valeurs saisies et la méthode savePlanet sera appelé
      3. enfin l’activité se termine en retournat un résultat positif
   4. La méthode **savePlanet** utiliser pour enregistrer une planète dans la base de données et affiche un Toast de confirmation
7. **UpdatePlanetActivity** : permet à un utilisateur à modifier une planète existante, L'activité reçoit les données de la planète à mettre à jour depuis une autre activité via un Intent, puis les affiche dans des champs EditText pour permettre la modification
   1. 3 attributs planetNameEditText, planetDistanceEditText, planetDescriptionEditText et planetId
   2. dans la méthode **onCreate** 
      1. Les trois EditText sont initialisés en utilisant findViewById
      2. L’Intent est récupéré et la planète à mettre à jour est extraite de l'Intent en utilisant la clé "planet", remplir les EditText avec les données récupérés
      3. Lorsqu'un utilisateur clique sur le bouton modifier, on récupère les valeur saisies, ensuite on vérifie si les champs sont remplies si oui on créer une nouvelle instance Planet avec les valeur saisies et l’identifiant planetID
   3. La méthode **updatePlanet**  pour modifier les informations d’une planète dans la base de données
   4. observePlanetList : est utilisée pour observer la liste de planètes mise à jour en utilisant le ViewModel (PlanetViewModel) et l'Observer.
8. **MainActivity** : est l'activité principale de l'application qui affiche la liste de planètes. L'utilisateur peut ajouter de nouvelles planètes en cliquant sur le bouton d'ajout, mettre à jour ou supprimer des planètes en utilisant le menu contextuel, rechercher des planètes par leur nom en utilisant le menu d'options, et rafraîchir la liste de planètes. L'activité utilise un RecyclerView pour afficher la liste de planètes et un PlanetViewModel pour gérer les données des planètes et les mises à jour de l'interface utilisateur.