Sujet de TP N°8 - Stockage de données avec Room

VUE D'ENSEMBLE

Dans ce TP, vous allez créer une application de gestion de recettes de cuisine. L'application permettra à l'utilisateur d'ajouter, de modifier et de supprimer des recettes, avec la possibilité de consulter une liste de recettes sauvegardées dans une base de données locale. Vous utiliserez Room pour gérer la persistance des données et Jetpack Compose pour l'interface utilisateur.

À la fin de ce TP, vous serez capable de créer des entités, des DAO, de configurer une base de données avec Room et de restituer ses données.

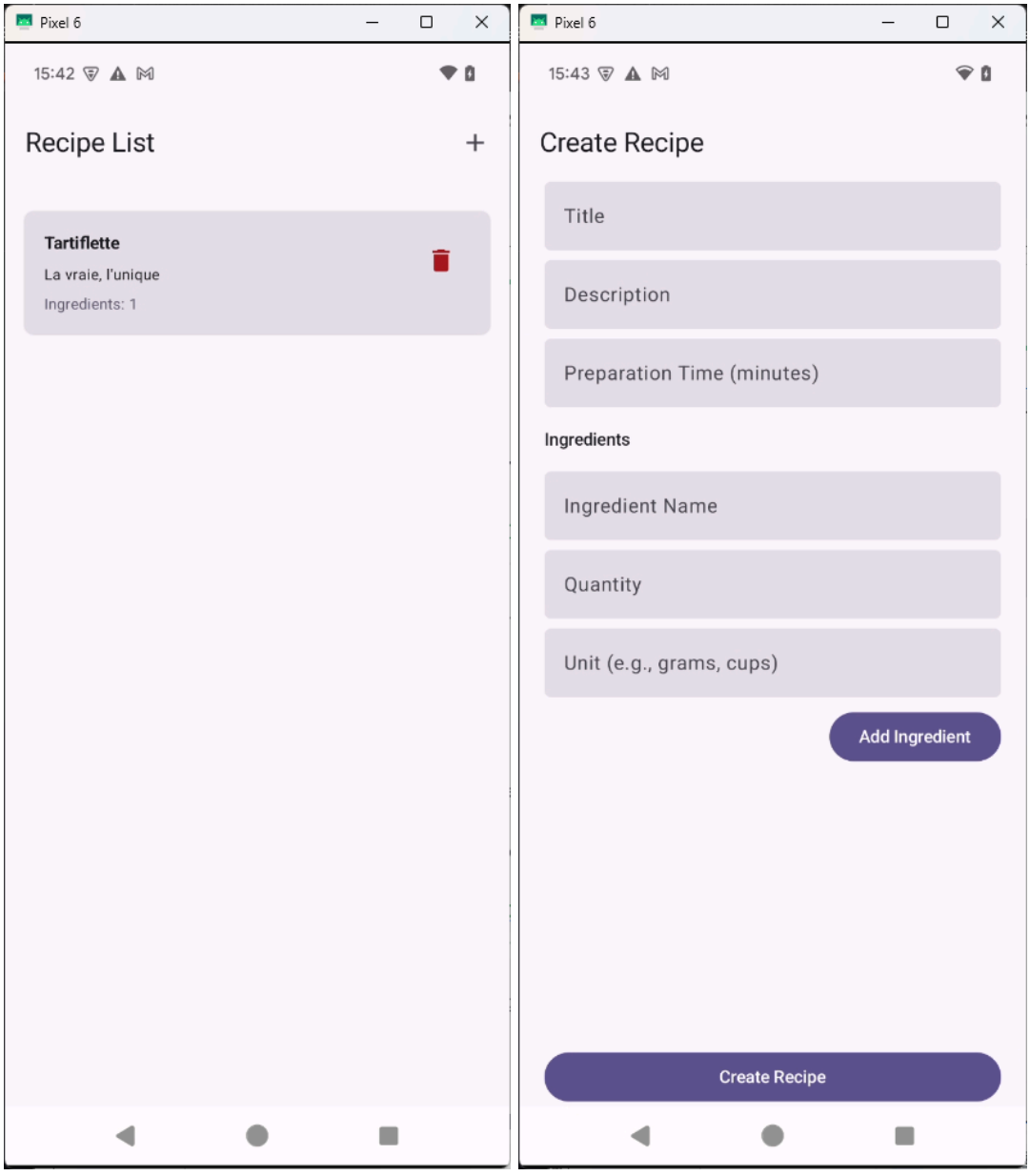
OBJECTIFS

- Comprendre le fonctionnement de Room et la gestion des données locales dans Android.
- Apprendre à créer et manipuler des entités dans Room.
- Utiliser les DAO pour effectuer des opérations sur la base de données (CRUD).
- Intégrer une fonctionnalité de persistance des données dans une application Android.
- Exécuter des requêtes de manière asynchrone pour éviter de bloquer l'interface utilisateur.

Fonctionnalités à mettre en œuvre :

L'application doit permettre à l'utilisateur de :

1. **Ajouter une recette** : Un formulaire doit permettre à l'utilisateur d'entrer un titre, une description, une liste d'ingrédients..
2. **Afficher une liste de recettes** : L'écran principal doit afficher une liste de toutes les recettes sauvegardées avec leur titre et une partie de leur description.
3. **Supprimer une recette** : L'utilisateur doit pouvoir supprimer une recette de la base de données.


Pixel 6
15:42
Recipe List
Tartiflette
La vraie, l'unique
Ingredients: 1
Pixel 6
15:43
Create Recipe
Title
Description
Preparation Time (minutes)
Ingredients
Ingredient Name
Quantity
Unit (e.g., grams, cups)
Add Ingredient
Create Recipe

Étapes du TP

Étape 1 : Configuration

Vous pouvez **partir du squelette de l'application** disponible dans **TP - 8 - Ressources**, ou vous en inspirer pour créer votre propre projet.

Ajout des dépendances
Si vous créez une nouvelle application, ajoutez les dépendances nécessaires pour :

- **navigation-compose**,
- **kotlinx-serialization**,
- **room-runtime**, **room-ktx**, et **room-compiler**.

Ces ajouts sont à effectuer dans les fichiers suivants :

1. **`libs.versions.toml`** : ajoutez les versions des bibliothèques.
2. **`build.gradle.kts` (Module : app)** : référencez ces dépendances en utilisant les alias définis.

Synchroniser le projet
Une fois les modifications appliquées, synchronisez votre projet pour télécharger et intégrer les nouvelles dépendances.

Étape 2 : Création de la base de données

1. Créez le package `database`

Dans votre projet, créez un package nommé **database**. Ce package contiendra l'ensemble des classes nécessaires pour gérer la base de données Room.

2. Créez la classe de base de données

Créez une **classe abstraite** nommée `RecipeDatabase` dans le package **`database`**.

Cette classe doit remplir les conditions suivantes :

- Annotée avec **`@Database`**.

- Inclure un tableau **`entities`** qui référence les classes d'entités de la base de données.
- Étendre **`RoomDatabase`**.
- Définir des méthodes abstraites pour accéder aux DAO.

Référez-vous au **cours** ou à la documentation officielle si nécessaire.

Étape 3 : Singleton pour la base de données

1. Créez la classe `DatabaseProvider`

Dans le même package `database`, ajoutez une classe `DatabaseProvider`. Cette classe implémentera un singleton pour créer et stocker l'instance de la base de données Room.

2. Implémentez le Singleton

Utilisez `Room.databaseBuilder` pour créer l'instance.

Étape 4 : Création des entités

1. Créez le package `entity`

Dans votre projet, ajoutez un package nommé `entity`. Ce package contiendra toutes les classes représentant les entités de la base de données.

2. Créez l'entité `Recipe`

Cette entité représente une recette et sera annotée avec **`@Entity`**.

- La table doit contenir :
  - un **id** (clé primaire auto-générée).
  - un **nom** pour la recette.
  - une **description** de la recette.

3. Créez les entités `Ingredient` et `RecipeWithIngredients`

- **`Ingredient`** représente un ingrédient lié à une recette.
- **`RecipeWithIngredients`** permet de gérer la relation **un à plusieurs** entre les recettes et les ingrédients.

Instructions :

Faites de même que pour `Recipe` en vous référant au cours ou à la documentation pour gérer les relations.

Étape 5 : Création du DAO

1. Dans le package `database`

Ajoutez une interface **`RecipeDao`** pour gérer les opérations sur la base de données.

2. Créez les méthodes du DAO

Ajoutez les méthodes nécessaires pour manipuler les recettes, les ingrédients et les relations. Utilisez les annotations `@Insert`, `@Delete`, `@Query`, et `@Transaction`.

- **Insérer une recette**
- **Insérer un ingrédient**
- **Insérer une recette avec ses ingrédients** (utilisez une transaction)
- **Sélectionner toutes les recettes avec leurs ingrédients** (utilisez une transaction)
- **Sélectionner les ingrédients d'une recette donnée**
- **Supprimer une recette**
- **Supprimer des ingrédients**
- **Supprimer une recette avec ses ingrédients** (utilisez une transaction)

Étape 6 : Création du ViewModel

1. Créez le package `viewmodel`

Ajoutez un package **`viewmodel`** dans le projet pour centraliser la logique de gestion des données.

2. Ajoutez le fichier `RecipeViewModel`

Dans ce package, créez un fichier **`RecipeViewModel`** qui sera responsable d’interagir avec le DAO et de fournir les données nécessaires à l'interface utilisateur.

3. Structurez le `RecipeViewModel`

Le ViewModel devra inclure les méthodes suivantes pour gérer les opérations de la base de données :

- **createRecipe** : Ajouter une recette avec sa liste d'ingrédients (utilisez la méthode transactionnelle du DAO).
- **deleteRecipeWithIngredients** : Supprimer une recette ainsi que ses ingrédients.
- **loadRecipesWithIngredients** : Charger toutes les recettes avec leurs ingrédients.

Gestion asynchrone : Assurez-vous d'exécuter les opérations de la base de données en arrière-plan pour ne pas bloquer le thread principal. Vous pouvez utiliser ExecutorService ou coroutines si vous êtes à l'aise.

Pour la mise à jour de l'UI en fonction des modifications de la base de données, référez-vous au squelette qui gère cette mise à jour à l'aide des `MutableStateFlow` dans le `ViewModel` et de la méthode `collectAsState` dans l'activité.