Technical Project Report - Android Module

TastyBite

Subject:	Introdução à Computação Móvel
Date:	Aveiro, 19/6/2024
Students:	107322: Bernardo Marujo 108342: Henrique Coelho
Project abstract:	This project focuses on the development of an innovative mobile application designed for efficient inventory management in a culinary context. The primary objective is to streamline the organization, tracking, and configuration of ingredients and recipes, providing users with a user-friendly interface to manage their kitchen inventory effectively.

Report contents:

1 Application concept

Individuals who regularly cook at home and manage their kitchen inventory. It provides an easy and intuitive way to keep track of their ingredients and recipes info so that they never forget how to prepare their favorite meals.

2 Implemented solution

Architecture overview (technical design)

Presentation Layer:

- **UI Components**: Built using Jetpack Compose for creating declarative UIs.
- **ViewModel**: Adheres to the MVVM (Model-View-ViewModel) pattern, separating the UI logic from business logic.

Domain Layer:

Encapsulates the business logic which is called by the ViewModel. For example, fetching ingredients, adding a new item, etc.

Data Layer:

- **Repository Pattern**: Provides a clean API for data access to the rest of the application. Acts as a mediator between different data sources (local database and remote APIs).
- **Data Sources**:
    - **Local Data Source**: Utilizes Room for local storage.
    - **Remote Data Source**: Fetches data from remote APIs if applicable.

Database Components:

1. **Entities**:
    - **Recipe** and **Ingredient** classes represent the database tables. These classes define the schema of their respective tables.
2. **Data Access Objects (DAOs)**:
    - **IngredientDao**: Provides methods for interacting with the `Ingredients` table. It includes methods for inserting, updating, deleting, and querying ingredients.
    - **RecipeDao**: Provides methods for interacting with the `Recipes` table. It includes methods for inserting, updating, deleting, and querying recipes.
3. **Database Class**:
    - **RecipesDatabase**: The abstract class that extends **RoomDatabase**. It provides DAOs for the **Recipes** and **Ingredients** tables and ensures only one instance of the database is created using a singleton pattern.

Repository:

- **OfflineRecipesRepository**: Implements the **RecipesRepository** interface and provides a clean API for data access to the rest of the application. It acts as a mediator between the DAOs and the ViewModel. The repository pattern is used to abstract the data layer, providing a clean API for data access and ensuring the rest of the application is not concerned with the source of the data.

ViewModel:

- **RecipeViewModel**: Manages the UI-related data in a lifecycle-conscious way. The ViewModel uses the repository to fetch data and exposes it to the UI via **LiveData** and **StateFlow**.

Insert, Update, Delete: These methods allow for adding, modifying, and removing ingredients in the database.

Query Methods: Methods like **getAllIngredients**, **getIngredient**, and **getIngredientsByNames** provide ways to retrieve ingredients, either all, by ID, or by a list of names, respectively. These methods return **Flow** objects, making the data retrieval reactive.

Similar to **IngredientDao**, **RecipeDao** provides methods for inserting, updating, and deleting recipes.

- It also has query methods for retrieving all recipes or a specific recipe by ID.
- **StateFlow**: Used to manage and observe state in a reactive way.
- **viewModelScope**: Ensures coroutines are tied to the lifecycle of the ViewModel.

Android Architecture Guidelines Implementation:

- **Model-View-ViewModel (MVVM)**: Ensures separation of concerns by dividing the application into distinct layers (UI, Domain, Data).
- **LiveData/Flows**: Used for reactive programming, allowing the UI to react to data changes in real time.
- **Room**: Manages local data storage, providing a robust and SQLite-based database.

This architecture leverages the benefits of Room, ViewModel, and Repository patterns to provide a clean, maintainable, and scalable solution approach, leveraging the benefits of the Room persistence library and the repository pattern.

Firebase Data Models/Data Structures

Recipe: This is a data model class that represents a recipe. It contains fields such as name, ingredients, description, category, calories, timetoCook and imageUrl.

Ingredient: This is a data model class that represents a recipe. It contains fields such as name, imageUrl, measurement and quantity.

Data Persistence

Firebase Storage: Firebase Storage is used to upload images and videos. The **StorageUtil** class has a method **uploadToStorage** that uploads a file to Firebase Storage and returns the download URL.

Firebase Firestore: Firebase Firestore is used to store the Recipe and Ingredients objects. Firestore is a NoSQL document database that allows you to easily store, sync, and query data for your mobile and web apps at a global scale.

Content Update Strategies

Real-time Updates: Firebase provides real-time updates out of the box. Any changes made to the data in Firebase Firestore are immediately synced to all connected clients.

Offline Support: Firebase Firestore also supports offline data persistence. This means that your app will continue to work even when it's offline. Any changes made while offline are synced back to Firebase Firestore when network connectivity is restored.

Synchronization Strategies

Automatic Synchronization: Firebase Firestore automatically handles data synchronization. When changes are made to the data in Firestore, those changes are automatically synced across all clients in real-time.

“Advanced” app design strategies

Automatic Synchronization: Firebase Firestore automatically handles data synchronization. When changes are made to the data in Firestore, those changes are automatically synced across all clients in real-time.

Integration with Internet-based External Services

Firebase Firestore is used for data persistence and Firebase Storage for storing images. Firestore is a NoSQL cloud database that lets you store and sync data between users in real-time. Firebase Storage is used to store and retrieve user-generated files like photos and videos.

Reading from Sensors

Our project uses the light sensor and ambient temperature sensor of the device. The data from these sensors is used to provide feedback to the user about the lighting conditions and ambient temperature.

Deferred Processing

The project uses Kotlin coroutines for asynchronous programming. Coroutines allow writing asynchronous code in a sequential manner. This is particularly useful when performing operations such as network requests or database transactions, which can block the main thread and cause the UI to freeze.

Text-to-Speech (TTS) Functionality

We use Android's Text-to-Speech engine to read out the recipe instructions. This is a great accessibility feature and can also enhance the user experience by allowing users to listen to the instructions hands-free while they are cooking.

Notification for Timer

Our project includes a timer feature that uses notifications to alert the user. The user is prompted to grant permission for notifications, as this is essential for the timer functionality. If the user does not grant permission, the timer cannot be used.

Implemented interactions

The application features a clear and intuitive navigation flow, starting from the Home Screen, which provides access to both Recipes and Ingredients. Users can view lists of recipes or ingredients, select individual items to view detailed information, and add or edit items through dedicated screens. For example, the user starts on the Home Screen and navigates to the Recipe List Screen by clicking the "Recipes" button. From there, they can view a detailed Recipe Detail Screen by selecting a recipe, or add a new recipe via the Add Recipe Screen. Similarly, navigating to the Inventory Screen from the Home Screen allows users to view and manage ingredients. This structured flow ensures a user-friendly experience, facilitating easy management of recipes and ingredients.

3 Conclusions and supporting resources

Lessons learned

During the implementation process, one of the major challenges we encountered was managing data persistence and retrieval using Room. Properly setting up the database schema, handling migrations, and ensuring seamless synchronization between the local storage and the UI was complex and required careful attention to detail. Another significant challenge was managing variables passed through navigation components. This was particularly messy because deep linking and ensuring that the correct data was passed and retrieved correctly between fragments and activities often led to unexpected behaviors and bugs. However, by thoroughly understanding the Room architecture and utilizing safe arguments for navigation, we were able to overcome these issues, enhancing the app's stability and user experience.

Android Development Insights: Some aspects of Android development were surprisingly useful and powerful, such as the integration of a ViewModel, which significantly simplified state management and UI updates.

Work distribution within the team

Taking into consideration the overall development of the project, the contribution of each team member was distributed as follows: Bernardo Marujo contributed with 50% of the work, and Henrique Coelho contributed with 50%.

Project resources

Resource:	**Available at:**
Code repository:	https://github.com/BMarujo/Kotlin_project
Ready-to-deploy APK:	https://github.com/BMarujo/Kotlin_project/blob/main/app-debug.apk
App Store page:	<put URL, only if applicable>
Demo video:	https://drive.google.com/file/d/1IchIloHpNWXgqqok4LTtM34vh4VvftNy/view?usp=sharing