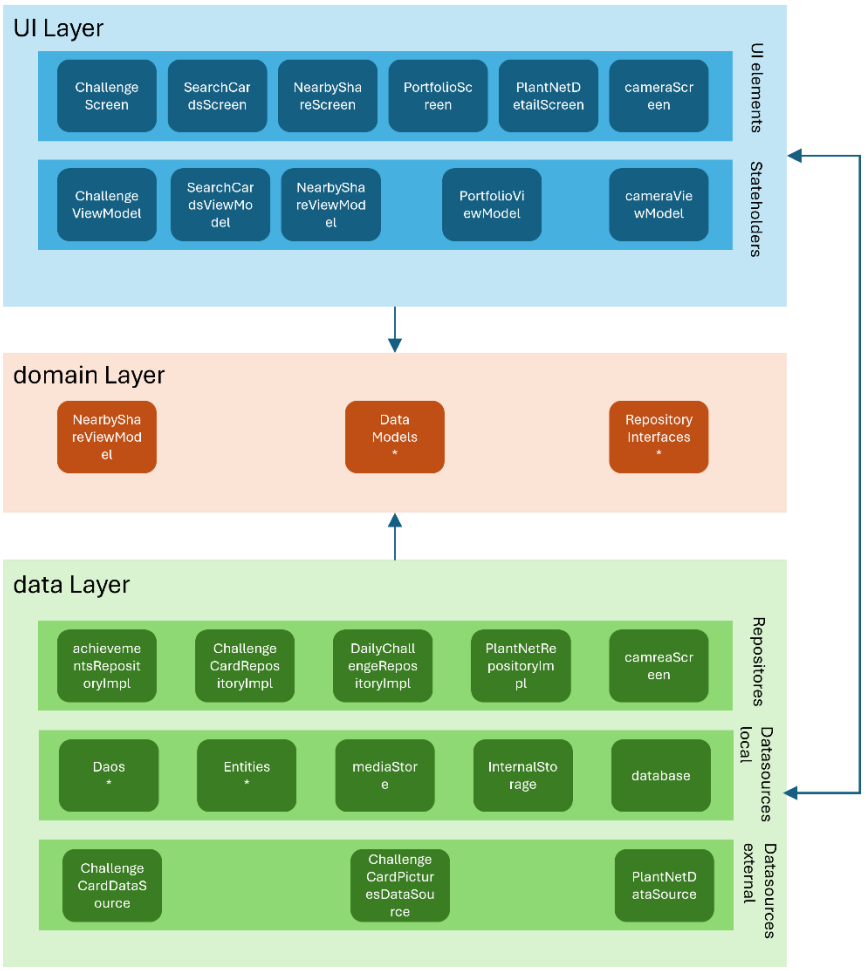
App Architecture

The App consists of 3 Layers, namely the UI Layer, the domain Layer, and the data Layer which are structured in the way of an MVVM Architecture. In the following each Layer is described in Detail:



UI Layer:

The UI Layer related files can be found in the UI Folder. The UI layer is splits into UI elements and state holders.

The UI elements exclusively contain code written for displaying the Interface of the App to the user. This part was realized using Jetpack Compose and is structured in screens. All files in the screens Folder are each dedicated to a single screen that is visible to a user.

The logic and data is placed in classes that inherit from the ViewModel class and act as screen level state holders. Each screen is associated with one ViewModel.

Domain Layer:

The domain contains the Usecases which encapsulate complex business logic, or simple business logic that is reused by multiple ViewModels. Since it was decided to build the App on

UI independent, persistent data models representing the data that drives the UI, respective files are also included in the domain layer, as well as the repository Interfaces.

Data Layer

The data layer is split into repositories and data sources.

The Repositories are responsible for interacting with the Data Sources.

Multiple of the Options Provided by the Android file System were chosen to realize the data sources in the Layer. Generally, they are divided into local sources, and external sources. The former make use of:

- the MediaStore API to save pictures to the gallery
- the internal Storage files to save pictures of downloaded challenges and information that don't make sense to be saved in a table
- a rooms database to save data in SQL form

the later handles:

- reading from and writing to a Cloud Firestore database that saves data on the challenge Cards
- downloading and uploading pictures from/ to a Firebase Cloud Storage that is used to save the challenge Cards pictures
- calling the PlantNet API to identify plants

Utils

Additionally, to the 3 main Layers, Dagger Hilt was used for managing dependency Injection, Jetpack Navigation to handle App internal navigation and lastly Kotlin Flow and Coroutines for asynchronous operations.

Dependencies/ third party software

Gson

Jetpack Compose

Jetpack navigation

Room

Android Material Design

Android graphics

Firebase-firestore

Firebase-storage

Okhttp3

Junit

Google Services

Additional Remarks

To test the App in the emulator one may load an image of a plant into the emulated camera environment. To do so, find the Android SDK location in your files and go to \Sdk\emulator\resources. Save the picture of the plant you want to identify in this folder. Then open the Toren1BD.posters file and replace any existing code with:

poster custom

size 1 1

position 0 0 -1.5

rotation 0 0 0

default hibiskus.jpg

make sure that in the camera setting of the Android Virtual Device you are using are set correctly. Go to the Device Manager, press the option buttons and select edit. Open the advanced settings and scroll down to the camera settings. Set Front to Emulated and Back to virtualScene. After restarting android studio (...the entire app not only the emulator!...) and opening the App, a plant poster should be visible in the camera view.

When the Emulator crashes it sometimes results in the database containing errors, in such a case, uncomment the .fallbackToDestructiveMigration() method in the AppModule file where the database is build. After running the app once, it can be commented out again.