Oryon App Architecture

Paul Kamp **Nils Rodenburg** **Sinan Thoese**

Oryon is a mobile running tracking app that allows users to create and participate in challenges. Users can automatically record their running activities. They can also view their progress. They can compete with others in various challenge formats.

The app offers the following functions:
- Register, log in and log out
- Live run tracking with GPS on a map
- Record distance, time and pace
- Participate in individual or group challenges
- Calculation of progress per challenge type (e.g. kilometres, minutes or number of runs)
- Display of rankings and statistics (for yourself and in the challenges)
- Data storage in Firebase (Firestore)
- Background tracking via a Foreground-Service

LogIn-Data

- paul@mail.de, pas123
- nils@mail.de, pas123
- sinan@mail.de, pas123
- lasse@mail.de, pas123

Since the distances of the individual runs and the progress in the challenges have been adjusted for demonstration purposes on Paul's account, there are some discrepancies, such as the pace of a run and the difference in distance between the activity screen and the challenges. To obtain correct data, we recommend either using a user without saved runs and creating a new challenge.

App Architecture

The app's architecture is based on the MVVM (Model-View-ViewModel) design, ensuring clear separation of responsibilities. This has been expanded to include a domain layer that bundles together the app's core functions, making them easily accessible to multiple ViewModels. The three main layers are:

UI-Layer

The UI layer forms the presentation level of the app and has been fully implemented using Jetpack Compose and Material 3.
The UI reacts to state changes provided by the relevant ViewModel via StateFlow or LiveData. The ViewModel responds to changes in the underlying data, such as new location information, saved runs or updated challenges. These changes are updated directly by flows into the ViewModel, which automatically leads to an update of the UI components.
On the other hand, when a user interacts with the user interface, the interaction is sent to the relevant ViewModel. The ViewModel then calls the relevant methods in the repository or domain layer to read or save data.
The app uses the Mapbox Maps SDK for Android in conjunction with the Mapbox Compose Extensions to provide users with an interactive map during their run. This enables the user's current position to be displayed in real time and their route to be tracked visually. Mapbox is used due to its many customisable styles and simple integration, which does not require a Google Developer account.

Structure of the UI-Layer

The ui package is divided into three main areas:

Screens: Each app screen is organised in its own sub-package. Each sub-package contains one or more composables for the screen, as well as the corresponding ViewModel. In some cases, a single ViewModel is used for two screens for logical reasons. For example, the ChallengeScreen and the ChallengeDetailScreen only use one ChallengeViewModel.

Components: This is where reusable and large UI components are stored, such as Top- and Bottombar, as well as dialogues and permission queries. It would be useful to store additional components that would be used for several screens here.

Theme: This package contains the app's core design elements, including its colour palette, custom typography, and other features specific to Material 3.

Domain-Layer

The RunTrackUseCase is located in the domain layer. It represents the business logic operations for tracking runs. As this is the app's main function, it is outsourced to the domain layer to keep the use case code clearer. This also allows access to the tracking function from several screens. It obtains location data in real time via the location repository. This data is then processed, e.g. to calculate the distance travelled or the route. If required, it transfers the results to the Firestore repository in order to save running data. At the same time, it communicates with the ViewModel, which forwards the current tracking data to the UI.

Data-Layer

The data layer is responsible for accessing persistent data sources and external systems. In the MVVM pattern, it is considered the model. This includes access to Firebase Firestore for storing and retrieving challenges, run sessions, and user profiles. Firebase Authentication is used for identifying and authenticating users. The Fused Location Provider (Google Play Services) is used for continuous location tracking while running.

Structure of the Data-Layer

The data package is organised thematically:

Firebase: This contains the central FirestoreRepositoryImpl, which handles all interactions with Firestore and implements the FirestoreRepository interface. It also contains the AuthRepository, which is responsible for login and UID management.

Location: This package contains the location logic implementation. In addition to the interface and its implementation, the LocationTrackingService is located here. This service provides location updates in the Foreground-Service.

This package also contains all data classes such as RunSession, ChallengeData, ChallengeParticipant and ChallengeGoal.

The app uses a sealed class structure to flexibly and systematically map different types of challenges (e.g. distance or time targets). Each challenge type is modelled as a clearly defined variant within a closed type system. This means that there are subclasses for each challenge type, making it possible to quickly add further challenge types. The app is based on the strategy pattern, whereby a separate 'strategy' is used to calculate and validate the unit and data depending on the challenge type.

Communication with Firebase and other asynchronous sources uses Kotlin coroutines and flows (e.g. callbackFlow and suspend functions). This allows for a reactive response to real-time Firebase changes, implemented in the code using snapshot listeners.

Structure of Firestore

Firestore organises data into two main collections. One is for users and the other is for challenges. The user collection contains documents with user information. Each document ID corresponds to a user's

UID. Each document contains the user's display name and email address. Further data could be stored here in the future. In addition, each user has their own 'run' collection. This sub-collection contains all of the user's run sessions. These contain the date, distance, duration, pace and route points of the run. Each challenge in the challenge collection has an ID and contains information about its name and type in the form of a string and a data map containing the challenge's goal. Additionally, all users participating in the challenge are stored in the participantIds array. This is used to query for users' challenges later on. There is also a 'participants' sub-collection, where each user is stored as a map containing information about their progress in the challenge.

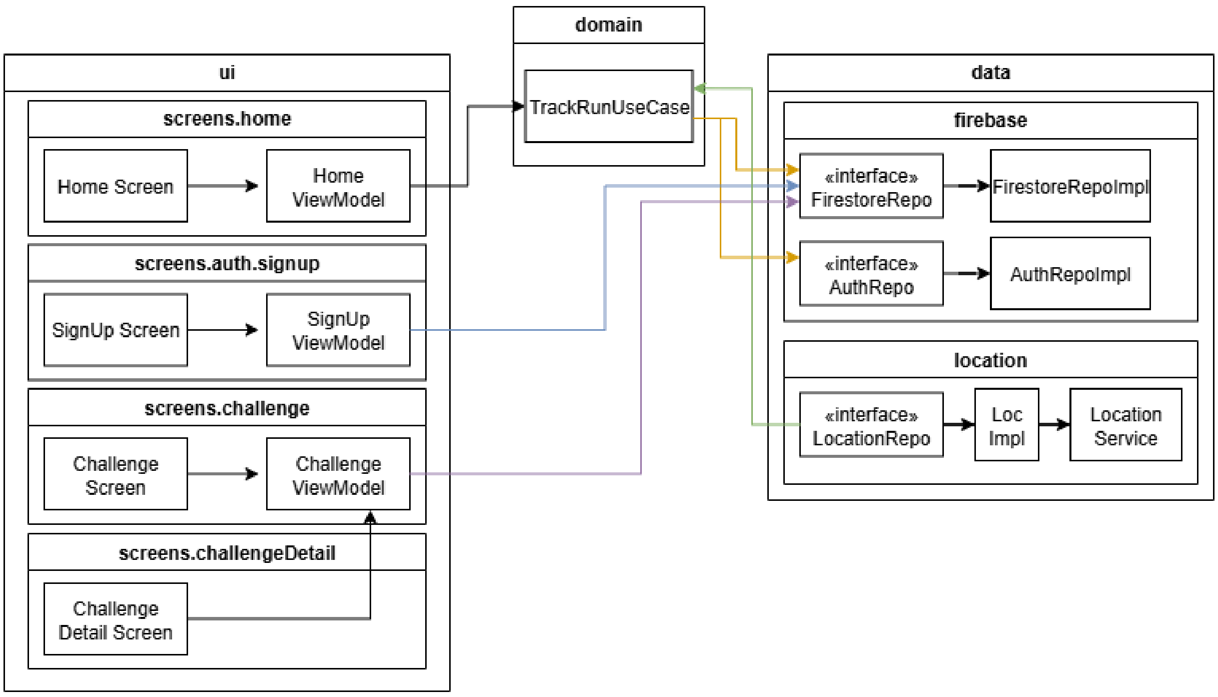

Figure 1: Shows some selected classes in there packages and their simplified data flow.