Technical Project Report - Android Module

SnapQuest – Peddy-Paper Aplication

Subject:	Universidade de Aveiro - Computação Móvel
Date:	4/04/2025
Students:	113655: Abel José Enes Teixeira 114429: José Maria Mendes
Project abstract:	SnapQuest helps users discover more about a place by learning in a creative and fun way. It turns sightseeing into a real-life treasure hunt. To learn more about a place, users will need to take several photos, each of different areas of the place, where each area will have some information to learn about the place, like a treasure hunt.

Report contents:

1 Application concept
- Overview
- Essential journey map

2 Implemented solution
- Architecture overview (technical design)
- Implemented interactions
- Project Limitations

3 Conclusions and supporting resources
- Lessons learned
- Work distribution within the team
- Reference materials
- Project resources

1 Application concept

Overview

→ The SnapQuest app was developed with the purpose of innovating the way people discover and learn about a place. Users of the app range from children on a field trip to people attending a large-scale event or even tourists in a new city.
The main objective is users learning, and to achieve that, they resort to the use of their cellphone camera and location to complete little challenges, discovering bit by bit the new place.

Essential journey map

→ Let's take the example already discussed in class of a high school student visiting the university open day. First, the event administrator, who could be a teacher, for example, creates a Quest in the app with various Challenges around the university. Then, when the student arrives at the event, he or she will be asked to join the Quest on SnapQuest. After logging in and joining, he or she can start discovering the university in an innovative treasure hunt style way. When he or she arrives at the location of the first Challenge, he or she will have to take a photo of the location to complete it, and after that, he or she will receive information about the location in the app and will unlock the next Challenge. This is how you do it until you complete the Quest.

2 Implemented solution

Architecture overview (technical design)

The SnapQuest application follows a layered MVVM (Model-View-ViewModel) architecture with clear separation of concerns, combined with the Repository pattern for data access. Lets breakdown the detailed implementation:

Layers:

- Presentation Layer:

- Activities (`MainActivity.kt`)
- Composable Screens (various UI screens)
- ViewModels (`HomeViewModel`, `QuestViewModel`, `SignInViewModel`)
- Navigation (Compose Navigation)

- Domain Layer:

- Business logic (handled in ViewModels)
- Data models (`User`, `Quest`, `Challenge, UserQuest and Notification`)

- Data Layer:

- Repositories (`FirestoreUserRepository`, `FirestoreQuestRepository and FirestoreStorageRepository`)
- Data sources (Firebase Firestore Database and Firebase Storage Database)
- Services (NotificationService)

Data Models and Persistence:

- Data Models:

- `User`: Represents app users with basic profile info
- `Quest`: Adventure quests with location, dates, and creator info
- `Challenge`: Individual tasks within quests with geolocation
- `UserQuest`: Tracks user progress through quests
- `Notification`: System notifications for user actions

- Persistence Strategy:

1. **Firebase Integration**:
    - Primary data storage in Firestore (NoSQL database)

- Firebase Storage for image uploads (quest/challenge photos)
- Collections: `users`, `quests`, `challenges`, `userQuests`, `notifications`

2. **Data Flow**:
    - ViewModels request data from repositories
    - Repositories interact with Firebase services
    - Data flows back through StateFlow to update UI reactively

Advanced App Design Strategies:

- Google Sign-In Integration:

- Uses Google Identity Services for authentication
- `GoogleAuthUiClient` handles OAuth flow
- User data synchronized with Firestore

- Location-Based Features:

- Requests `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions
- Uses FusedLocationProviderClient for device location
- Quests and Challenges have geolocation requirements

- Notification System:

- Implements `POST_NOTIFICATIONS` permission (Android 13+)
- `NotificationService` handles showing notifications
- Firestore stores and manages notification records

- Camera and Media Integration:

- Requests `CAMERA` and storage permissions
- Uploads photos to Firebase Storage during challenge completion
- Uses FileProvider for secure file sharing

- Background Processing:

- Declares foreground service for location tracking
- Uses coroutines for async operations (viewModelScope)

- Real-time Updates:

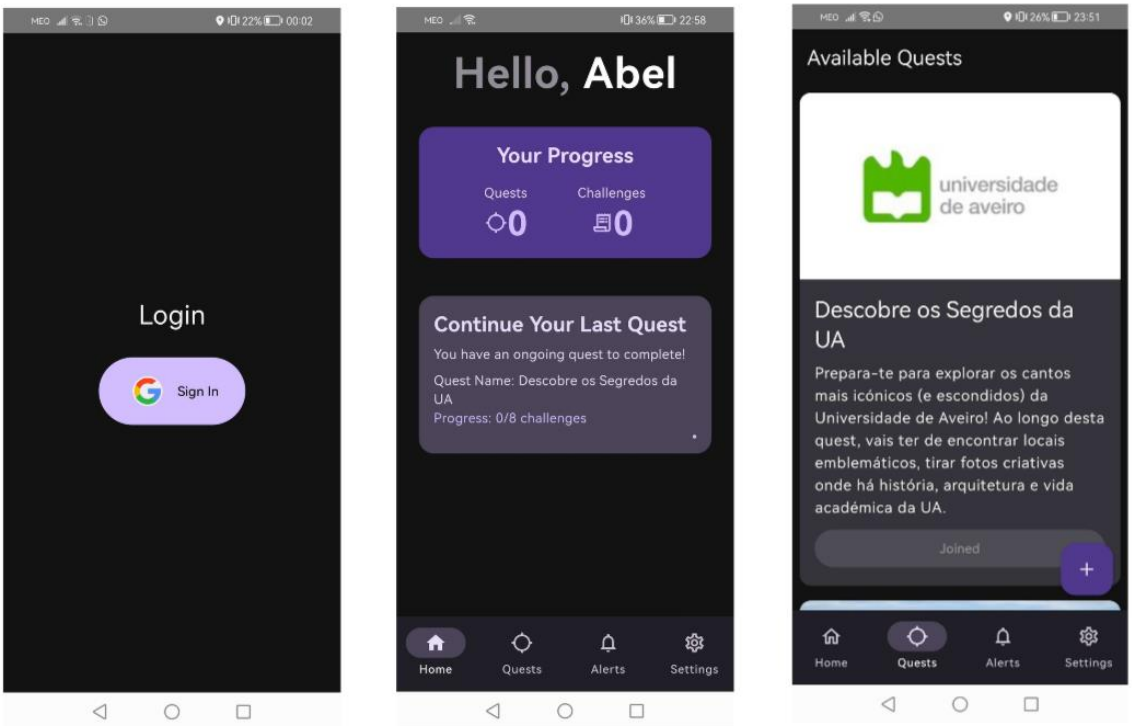
- Firestore listeners provide real-time data synchronization, eg: `fetchQuests()` continuously updates quest list

Component Interaction Flow:

1. **Authentication Flow**:
    - User signs in via Google → `GoogleAuthUiClient` → Firestore user record created/updated → ViewModel updates state → UI navigates to home
2. **Quest Management Flow**:
    - User creates quest → ViewModel calls repository → Firestore updates → StateFlow updates → UI reflects changes
    - Other users see new quests in real-time via Firestore listener
3. **Challenge Completion Flow**:
    - User takes photo → uploads to Storage → marks challenge complete → Firestore updates → notifications sent → progress tracked in UserQuest

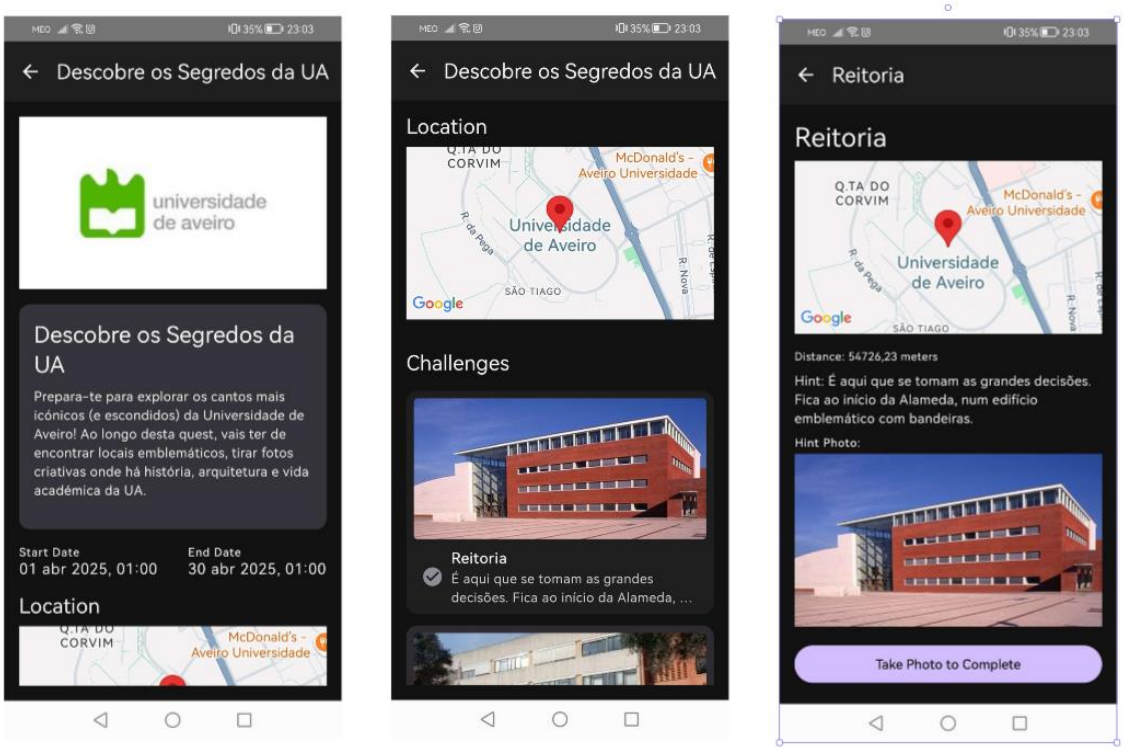
Implemented interactions

Let's take a look at the flow of a user:



→ A user is participating in university open day event and the event is in SnapQuest.
- First the user logs in and joins the quest:

- After joining, now can star selecting challenges to complete them, one by one, taking pictures of the places:



Project Limitations

It was planned to add daily challenges, in which challenges would appear to take a photo of a random object in an area near the user. To complete these challenges, the verification was supposed to be done by the ML Kit image recognizer.

3 Conclusions and supporting resources

Lessons learned

To conclude this report, I will start by talking about one of the biggest problems encountered during development, which was designing an architecture that would support data storage so that all users could access it. The solution found was to use cloud storage (Firestore) to store any information related to users, including images.

One aspect that surprised me was how easy it was to use JetPack Compose to develop the UI. Compose makes the job easier in many ways, but the most striking is the way in which the components are ready to be used (e.g.: Text(), Image(), Row(), Column(), …).

A recommendation for future students of the course would be to master JetPack Compose as quickly as possible and study the layered architecture to facilitate the development of the app.

Work distribution within the team

Taking into consideration the overall development of the project, the contribution of each team member was distributed as follow: Abel Teixeira did 90% of the work, and José Mendes contributed with 10%.

Reference materials

Learning how to implement the maps sdk for android:
https://developers.google.com/codelabs/maps-platform/maps-platform-101-compose#0

How to set up Google SignIn in firebase and your app:
https://firebase.google.com/docs/auth/android/google-signin?hl=pt-br

Get to know firebase realtime database:
https://firebase.google.com/docs/database?hl=pt-br

Setting up the storage database in firebase:
https://firebase.google.com/docs/storage?hl=pt-br

Learning how to develop in android:
https://developer.android.com/

Android app architecture:
https://developer.android.com/topic/architecture

Managing android permissions:
https://developer.android.com/guide/topics/permissions/overview

Building an apk:
https://www.youtube.com/watch?v=3FujlwQoKuk

Project resources

Resource:	**Available from:**
Code repository:	https://github.com/ttabelhaxd/Projeto1-ICM
Ready-to-deploy APK:	https://github.com/ttabelhaxd/Projeto1-ICM/blob/main/output.zip
App Store page:	---
Demo video:	