Instituto Superior de engenharia de Lisboa

CrossBoard Application

March 2025

Luís Reis, Nº 48318, E-mail: A48318@alunos.isel.pt, Tel: 932116540
Rúben Louro, Nº 48926, E-mail: A48926@alunos.isel.pt, Tel: 962902998

Supervisor: Pedro Pereira, E-mail: palex@cc.isel.ipl.pt

Introduction

In today's software development landscape, creating applications for multiple platforms often requires maintaining separate codebases for each platform. This approach leads to significant code duplication, increased development effort, and higher maintenance costs.

To address this challenge, we aim to explore **Kotlin Multiplatform (KMP)** [1] - a technology that enables developers to share common code across multiple platforms while still allowing access to platform-specific functionality when needed.

With KMP, business logic, data handling, and other core functionalities can be written once and reused across Android, iOS, web, and desktop applications. At the same time, developers retain the flexibility to implement platform-specific UI components and features where necessary. This hybrid approach helps reduce redundancy, improve code consistency, and accelerate development time while maintaining native performance.

Our study will focus on evaluating how Kotlin Multiplatform can help in multi-platform development, enhance code maintainability, and optimize resource utilization, ultimately leading to more efficient and scalable applications.

System Requirements

The application, built using **Kotlin Multiplatform (KMP)**, will have a set of **functional and non-functional requirements** to fully explore the potential of this technology in developing a **multi-platform application**. These requirements will ensure that the application is both feature-rich and optimized for the best user experience across different platforms.

Function Requirements

- ➢ Users will be able to use the application with or without an account in the application, although the unregistered users will have limited functionalities.
- ➢ Unregistered users will be able to play a single player match against the computer and choose the difficulty.
- ➢ Registered users will be able to play a match against the computer or against another user.
- ➢ Registered users can check their account information and their statistics.
- ➢ Registered users will have a rank system to play ranked matches or unranked matches.
- ➢ Administrators of the application will be able to check statistics of other players and will be able to manage the data information of users of the application.
- ➢ Figure 1 represents the functionalities described for all the types of users, but assumes there's only 1 implemented turn based board game.

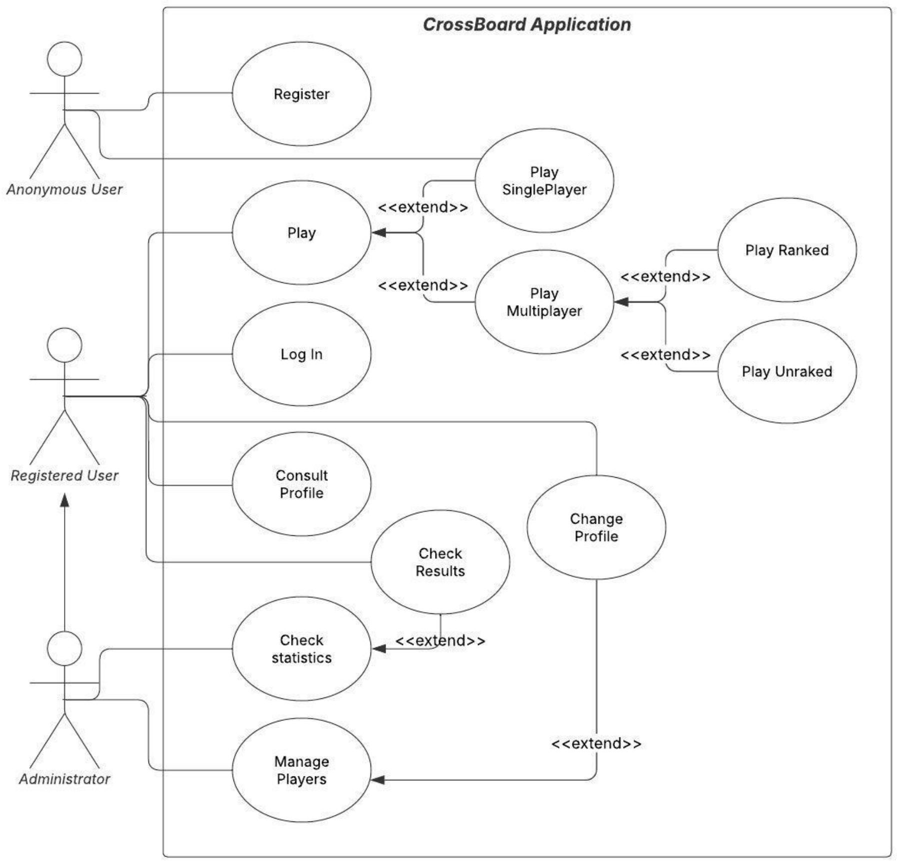

Figure 1 - UML Use case diagram.

Non-Functional Requirements

- ➢ Users will be able to use the application on different devices (Android, IOS, Desktop, etc)
- ➢ The application will have a user-friendly experience.
- ➢ Application documentation is complete and easy to expand for other uses and functionalities of this application or for future applications.
- ➢ Application testing with good coverage of the multiple functionalities to ensure security for the user while using the application.

Navigation Flow

To implement a user-friendly experience it's necessary for the screen flow to be as easy to understand as possible. Figure 2 shows how the screen navigation will take place as soon as the application is opened.

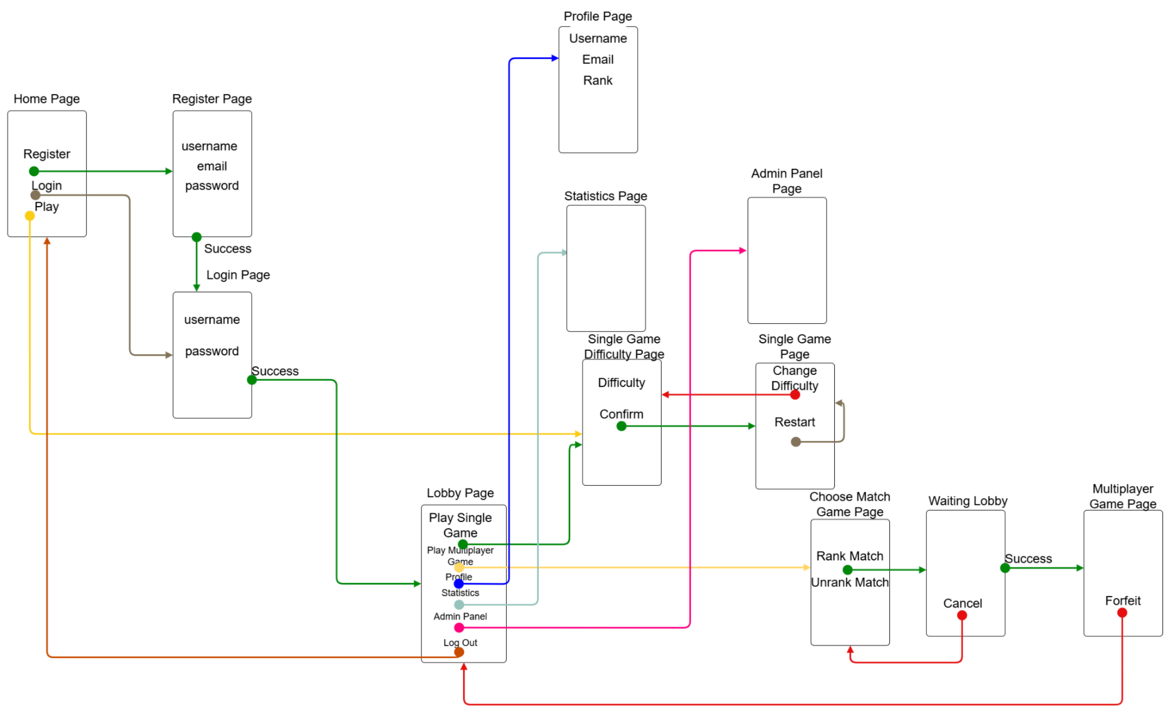

Figure 2 - Screen flow.

System Architecture

The system will have an architecture where Windows, iOS, and Android clients interact with a shared codebase. The shared code includes a UI layer for user interaction, a Business Logic Layer for processing workflows, a Data Access Layer for database interactions, and a Domain Model defining core entities. A centralized database ensures data consistency across platforms. Figure 3 demonstrates this architecture.

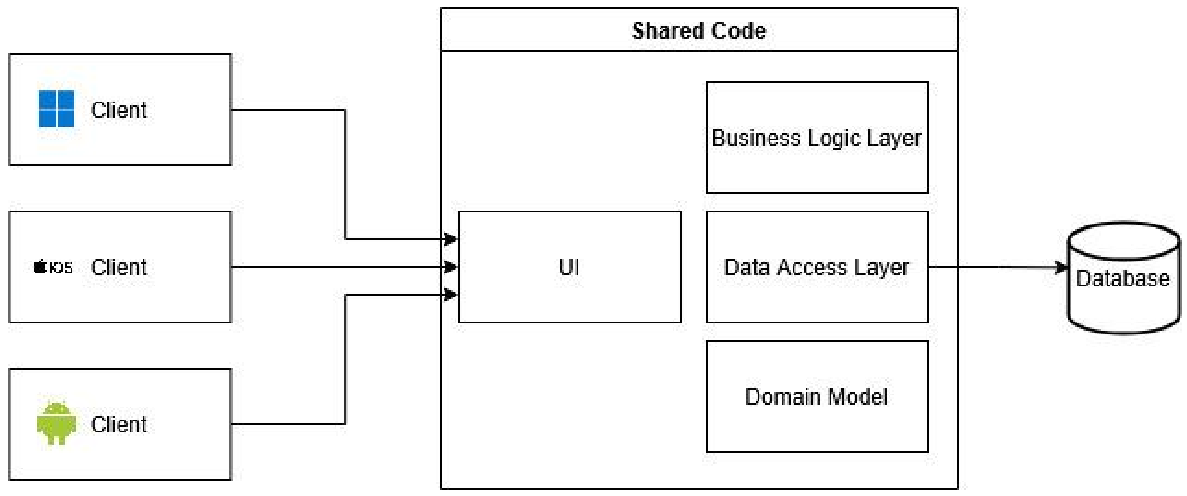

Figure 3 - System Architecture.

Risks

One of the main risks of this project is the **learning curve associated with Kotlin Multiplatform (KMP) and its ecosystem**. Since we need to implement various cross-platform technologies, we must become familiar with key libraries such as **Ktor** [2] **(for networking), SQLDelight** [3] **(for database management), and platform-specific APIs**. Understanding how to effectively use these tools is essential for ensuring seamless functionality across different platforms.

Learning all the required libraries, platform specific language and implementing the system will be highly time-consuming, which will be a risk to complete the task before the deadline.

Schedule

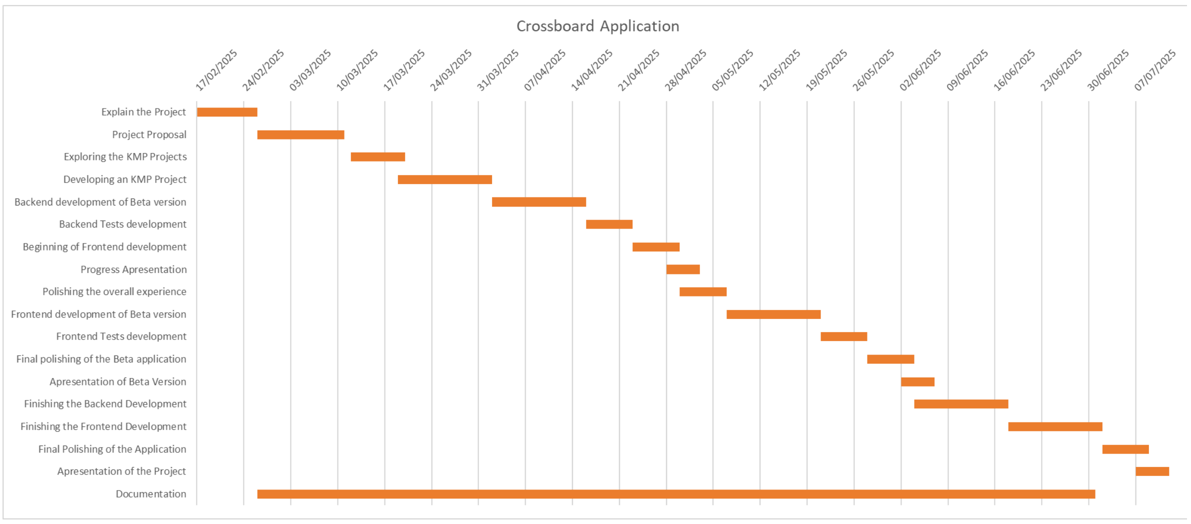

Figure 4 - Proposed Project Schedule

References

[1] "Kotlin Multiplatform." *Kotlin*, https://kotlinlang.org/docs/multiplatform.html.

[2] *Ktor: Build Asynchronous Servers and Clients in Kotlin*, https://ktor.io/.

[3] SQLDelight - Generates typesafe Kotlin APIs from SQL." *GitHub*, https://github.com/sqldelight/sqldelight.