João Miguel Simão da Costa

DEVELOPMENT OF AN ASYNCHRONOUS MOBILE GAME

1 2 9 0 UNIVERSIDADE D COIMBRA

UNIVERSIDADE D
COIMBRA

João Miguel Simão da Costa

# DEVELOPMENT OF AN ASYNCHRONOUS MOBILE GAME

**Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Professor João Nunes Lopes Barata and Sara João and presented to**
**Faculty of Sciences and Technology / Department of Informatics Engineering.**

June 2020

This page is intentionally left blank.

## Acknowledgements

First of all, I want to thank my family for all the support given throughout my University years and my life in general. I especially thank my sister Joana that was always there for me and gave me the encouragement I needed to finish this dissertation.

I wish to express my sincere gratitude to my advisor, professor João Barata, who was always available to help and provide quality advice, in each step of the way.

Finally, I would like to thank the HYP Team, especially Daniel and Sara, for all the support given throughout this internship and for the opportunity they gave me to learn about things I was so interested in.

This page is intentionally left blank.

## Abstract

Since their first appearance, mobile devices have had an exponential technological growth and have gained other functionalities that go beyond the standard calls and messaging services. With the advent of smartphones, the mobile gaming market has grown enormously to become the most widespread form of gaming across the world, reaching more corners than any other gaming market.

The primary purpose of this project is to develop a Minimum Viable Product (MVP) of an online multiplayer asynchronous mobile game using telepathy as a theme. In asynchronous gameplay, one player plays the game while the other has to wait until their turn comes.

In this particular mobile game, players interact through different mobile devices and take turns attempting to guess the answers given by other players to a given set of questions. Additionally, there are no established time limits between turns, allowing for higher flexibility on the part of individual players. Players are awarded points if their answers match the answers previously selected by the other players.

The present report envisages describing the whole engineering process leading to the realization of this particular mobile game, including a representation of the planning stages of the project, a review of the state of the art, a system description, and an overview of the development and testing phases.



## Keywords

mobile game, mobile application, asynchronous game, multiplayer game, telepathy game, software engineering.

This page is intentionally left blank.

## Resumo

Desde a sua génese, os telemóveis foram objeto de um desenvolvimento tecnológico exponencial e adquiriram outras funcionalidades para além das triviais chamadas telefónicas e serviço de mensagem. Com o advento dos smartphones, o mercado de jogos para telemóveis cresceu muito significativamente e converteu-se na forma de jogo mais difundida em todo o mundo, alcançando mais utilizadores do que qualquer outro segmento do mercado de jogos.

O objetivo principal do presente projeto é o desenvolvimento de um Produto Viável Mínimo (MVP) de um jogo para telemóveis, online, assíncrono e para múltiplos jogadores, inspirado na temática da telepatia. No jogo assíncrono, um dos jogadores interage com o jogo enquanto o outro tem que aguardar pela sua vez de jogar.

Neste jogo para telemóvel em particular, os jogadores interagem através de dispositivos móveis distintos e, alternadamente, tentam adivinhar as respostas dadas pelos outros jogadores a um determinado conjunto de questões. A acrescentar, não existe qualquer limite temporal definido entre turnos, assegurando uma maior flexibilidade. Aos jogadores são atribuídos pontos se as respostas estiverem corretas de acordo com as respostas previamente selecionadas pelos outros jogadores.

O presente relatório visa descrever todo o processo de engenharia conducente ao desenvolvimento do jogo com as características descritas, incluindo uma representação das diferentes etapas de planeamento do projeto, uma revisão do estado da arte, uma descrição do sistema e uma análise das fases de desenvolvimento e de teste.



## Palavras-Chave

jogo de telemóvel, aplicação de telemóvel, jogo assíncrono, jogo multijogador, jogo de telepatia, engenharia de software

This page is intentionally left blank.

# Contents

This page is intentionally left blank.

# Acronyms

**API** Application Programming Interface. 42

**ER** Entity-Relationship Diagram. 3, 43

**HCI** Human-Computer Interaction. 13

**IDES** Integrated development environments. 19

**ISO** International Organization for Standardization. 12, 13

**MVC** Model-View-Controller. 25

**MVP** Minimum Viable Product. 2

**OS** Operating System. 1

**REST** Representational State Transfer. 42

**SDK** Software Development Kit. 21

**ToS** Threshold of Success. 27, 84

**UI** User Interface. 13

**URL** Uniform Resource Locator. 42

**UX** User Experience. 13

This page is intentionally left blank.

# List of Figures

This page is intentionally left blank.

# List of Tables

This page is intentionally left blank.

# Chapter 1

# Introduction

The subject of this report is the development of an asynchronous mobile game by a student of the Masters in Informatics Engineering as part of the curricular subject Dissertation/Internship at the Faculty of Sciences and Technology of the University of Coimbra (FCTUC), building upon a project proposal put forward by HYP.

The aim of this chapter is to describe the work developed, explaining its context and motivation, the goals to be achieved, and the structure of this document.

## 1.1 Context and Motivation

Mobile technology has evolved significantly from a simple device used for phone calls and messaging into a multi-tasking device that can be used for many useful purposes such as internet browsing, GPS navigation, transfer of files, and much more [25]. Nowadays, almost everyone owns a mobile device and uses it regularly.

Since the appearance of smartphones and their respective Operating System (OS) (mostly Android, IOS, and Windows), anyone that owns such a device can easily download apps from a vast collection available in their specific OS app market. For example, Android and IOS - that own nearly 99 % of the Mobile Operating System Market Share Worldwide [4] - have an app market available for multiple purposes, including games. A great part of these games are free and normally revenues come from advertisement or in-game purchases [18]. This has opened a whole new market opportunity for games, and as phones become even smarter and more powerful, so do mobile games. As a result, the mobile gaming market is growing enormously, and has been the largest segment of gaming since 2017 [77], reaching more corners than any PC, console, or handheld has ever reached before [56]. In 2019 mobile games are reported to own 45% of the global games market, generating 68.5 billion dollars (Figure 1.1), and by 2022 it is predicted to reach 95.4 billion dollars, 49 % of the global games market [91].

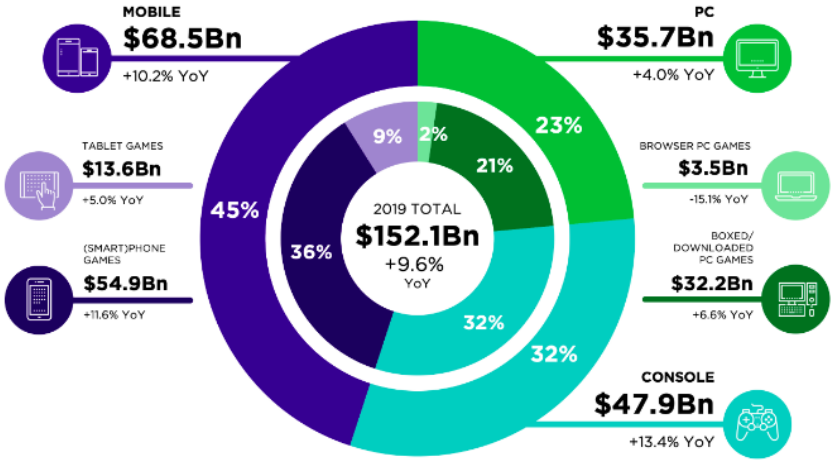



Figure 1.1: Gaming revenues per device [91]

HYP, an acronym for Handling Your Problems, is a software company created in 2015. The company's primary focus is the design and development of websites, mobile apps, and the creation of graphic design projects and illustrations [20]. In the context of the above-mentioned Dissertation/ Internship, HYP presented a project proposal towards the creation of an online asynchronous multiplayer mobile game using telepathy as the theme. This report builds on that project proposal.

The project proposal presented itself as very auspicious from the author's perspective. People embed a natural inclination to share information with their peers and are easily moved by the feelings of wanting to connect and get involved in a positive and effortless manner. The project's leading theme – telepathy – links intimately with those aspirations. People connect with other people through their own devices with the purpose of sharing and connecting. While playing the game, positive emotions are generated as people feel engaged and entertained.

## 1.2 Goals

As mentioned before, the leading idea of the project proposal is the development of an asynchronous multiplayer online mobile game using the telepathy theme, which can be accessed from IOS and Android mobile phones. In this game, players will have to answer questions about other players, based on what they think the other players responded previously to the same questions. For example, for players X and Y to be awarded points, player Y will have to give the same answer player X did before when asked for a car brand; or, if given a set of four options regarding what player X would choose to do in a given situation, player Y will have to match the same options previously selected by player X. Considering the format of the game, as described, creativity, diversity, and market orientation will also be essential for the development of ideas regarding the questions to be inserted, in order to keep the players entertained and engaged.

Building on the above-mentioned theme, the overall goal of the project is to create a Minimum Viable Product (MVP) of the proposed mobile game, fulfilling at least all the "Must Have" requirements. Achieving such goal will require going through a whole engineering process involving several different stages. However, it is important to mention that the game user interface look shall not be considered a priority, as the HYP team will develop

it at a later stage.

Apart from the mobile game that needs to be built for the client-side, a server application also needs to be developed so it can communicate with the mobile game and save players' information in the database. The author of this dissertation has both the roles of developer and contributor for the system requirements and description.

## 1.3 Structure of the Report

This document is divided into the following chapters:

- **State of the Art (Chapter 2)** The aim of this chapter is to conduct a competition analysis, focusing on different mobile games that match certain criteria, selected according to the scope of the project under development. Subsequently, the importance of usability is highlighted and explained, since it is one of the main quality attributes behind the success of an app.
- **Project Management (Chapter 3)** This chapter describes the temporal planning of the first and second semesters, followed by a description of the chosen methodology. Subsequently, descriptions regarding the decisions concerning the different technological tools to use for the client-side, server-side, database, and also support tools, are presented. Finally, the different risks found during the development of the project are identified, along with the mitigation plans.
- **System Description (Chapter 4)** Chapter 4 starts by identifying the user stories in order to assist the team in understanding all the features required for the development of the mobile game. Later on, the use cases and the functional and non-functional requirements for the mobile game are introduced. Finally, the System Architecture is presented, followed by the Entity-Relationship Diagram (ER), the Navigation Diagram and the Wireframes.
- **Development (Chapter 5)** This chapter starts by identifying the organizational methodologies employed by the author, followed by a description of the project structure. Afterward, the details of the implemented functionalities are specified joined by the respective mobile game layouts, as well as the problems encountered.
- **Testing (Chapter 6)** This chapter focuses on the techniques used by the author to validate the final MVP. Functional testing techniques are introduced, followed by an overview of the load testing conclusions and a description of the usability test results.
- **Conclusion (Chapter 7)** In this chapter are drawn the final conclusions of the project, as well as the future work.

This page is intentionally left blank.

# Chapter 2

# State of the Art

The aim of the present chapter is to review the state of the art, focusing on games that can relate to the scope of this project, selected according to a set of criteria determined in the first meeting of the semester between the author and the HYP team.

Different asynchronous mobile games are reviewed and weighed against each other, meaning that their various features are analysed and compared. That knowledge can then be used to build an asynchronous mobile game that can fit today's mobile gaming market. Finally, the importance of usability, utility and user experience are highlighted, since they are main attributes behind the success of many apps.

## 2.1 Competition Analysis

On this section, different mobile games that have the features of being asynchronous, multiplayer based, and/or that have the telepathy as a theme are studied. Before starting analysing asynchronous multiplayer mobile games, it is essential to define what does it mean to be asynchronous.

Asynchronous gaming means that one player is taking part on the game while the other(s) can ignore(s) it to do something else until their turn comes. Just like synchronous gameplay, in asynchronous gameplay, the number of players can go from two to a lot more, depending on the game. However, in asynchronous play, players never play at the same time. Instead, they play in sequence, one after another. In some cases, they might even have time limits in each turn. Depending on the asynchronous variant used, the turns progression may vary, but turns are typically used to give players more time to think about their actions and not focus so much on keeping track and controlling several things at once in real-time. This means that asynchronous gameplay allows for more flexibility on the part of individual players and demands fewer time synchronies, making it more casual [64] [87].

There are many types of asynchronous gameplay. For example, two friends may play chess against each other using the same computer or, alternatively, they may play chess against each other using different devices and with time limits; both are examples of asynchronous gameplay since the players play alternately. That being said, the type of asynchrony that characterizes the mobile game of the project under development has the following attributes:

- **It is supposed to be played by two players in different mobile devices**,

meaning that each user plays on their own device.

- **There are no time limits between turns**, meaning that a user can play their turn whenever they intend to.

Each of the following sections will focus on a mobile game with the characteristics identified earlier, followed by an analysis of what is different about these games and which features they emphasize. Since the requirements have not been fully identified at this point, this analysis will help the author realize what has been done on this particular market while extracting positive and negative aspects from each one of the mobile games.

### 2.1.1 Words with Friends 2

Words with friends is a two-player mobile game developed by Newtoy (a video games development company) for both IOS and Android Operating Systems, released in July 2009 [23]. This game is very similar to the board game Scrabble.

In this game, both players take turns forming words either vertically or horizontally on the board (Figure 2.1 (b)) with the letters presented on the screen. Depending on the word and where the player places it, they win a certain number of points. The player's aim is to score as many points as possible. At the end of the game the player with the higher number of points wins.

In this game, players can invite their Facebook friends or play with random people. There are daily challenges and bonuses that allow players to win different rewards, making the game more entertaining. There is a store (Figure 2.1 (d)) in which players can spend their coins in in-game items (that they won playing the game and doing the challenges). Players are also able to buy these coins using real money. In the ranking screen, players can see how good they are comparing with others, and in their profile screen, users can see different statistics regarding their games. As represented in Figure 2.1(a), players can view a tutorial so they can know how to play the game.


(a) Tutorial (b) Home screen (c) Gameplay (d) Store

Figure 2.1: Words with Friends 2

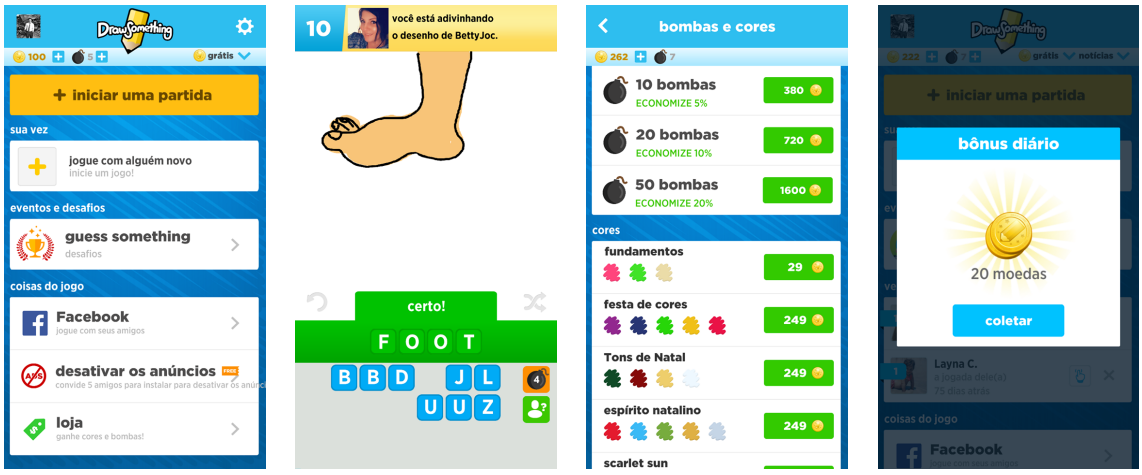
### 2.1.2 Draw Something

Draw Something is a two-player turn-based mobile app developed by OMGPop, launched on February 6 2012, that is available for IOS, Android and Windows Phone [11].

A game can be created with a friend (invited by email or Facebook) or with a stranger. After that, a player chooses one word out of three words that are displayed, where the hardest words to draw give the most coins. Then, they draw the word they chose, whereas they can change the thickness of the pencil and its color to draw it. In the end, they send the drawing for the other player to guess.

The other player will view a stroke by stroke replay of the drawing (without pauses in between) and has to guess the word with a set of given letters (Figure 2.2 (b)). If the player has difficulties guessing the word, there is a help option available, that will allow decreasing the letters or adding letters in the correct positions. After correctly guessing, both players win coins, and a new turn begins, where all the explained game flow repeats.

This game also has daily bonuses (Figure 2.2 (d)) and daily challenges, where, for example, players have to guess different drawings that allow them to win more coins. Players can buy more coins with real money, and these can be spent in the game store to buy more colors and bombs (which helps players guess drawings) as represented in Figure 2.2 (c). Users that have this game installed also receive notifications about new game features or current on-going games.




(a) Home screen (b) Gameplay (c) Store (d) Daily bonus

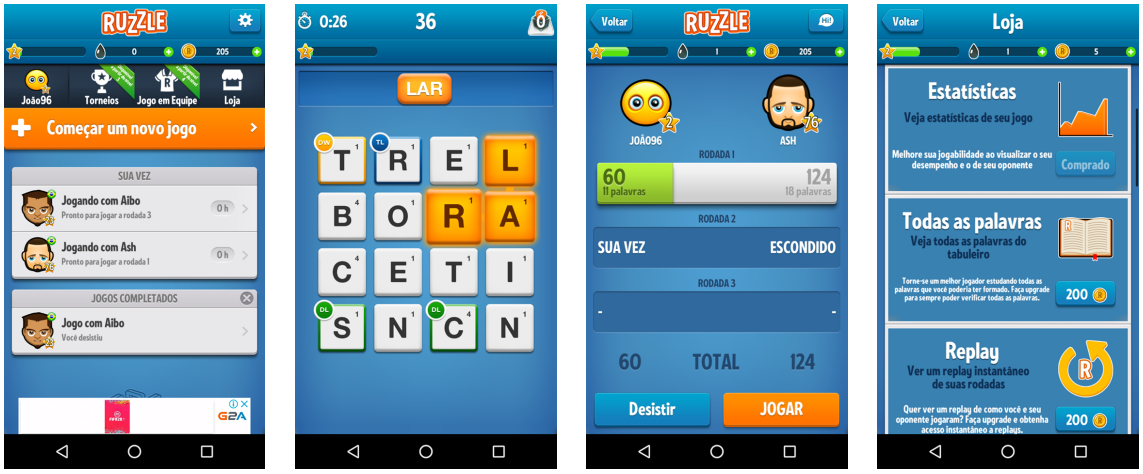
Figure 2.2: Draw Something

### 2.1.3 Ruzzle

Ruzzle is a mobile game developed by a company named MAG Interactive that was firstly published in the Apple Store in March 2012 and then a month later in Android, reaching up to one million downloads three weeks after launch [85]. This game can be played against a random online user, a friend in the friend's list or chosen among Facebook friends. The game is divided into three rounds. In each round, a player has to form the maximum numbers of words from a 4x4 grid on the screen within two minutes (Figure 2.3(b)). The number of points depends on the number of words found and the value of each one of them. As shown in Figure 2.3 (c), at the end of each turn the round statistics appear with the

number of words found by each player and the points they made. The winner is the player awarded the higher number of points at the end of the three rounds.

In the Ruzzles home page (Figure 2.3(a)), a player can see when it is their turn or the opponents' to play, and also the already completed games. In any case, it is possible to see the game statistics by clicking on the game. The game offers other gaming modes, such as tournaments and a teamwork mode. It also has a store where players can buy items to upgrade their performance (Figure 2.3 (d)).



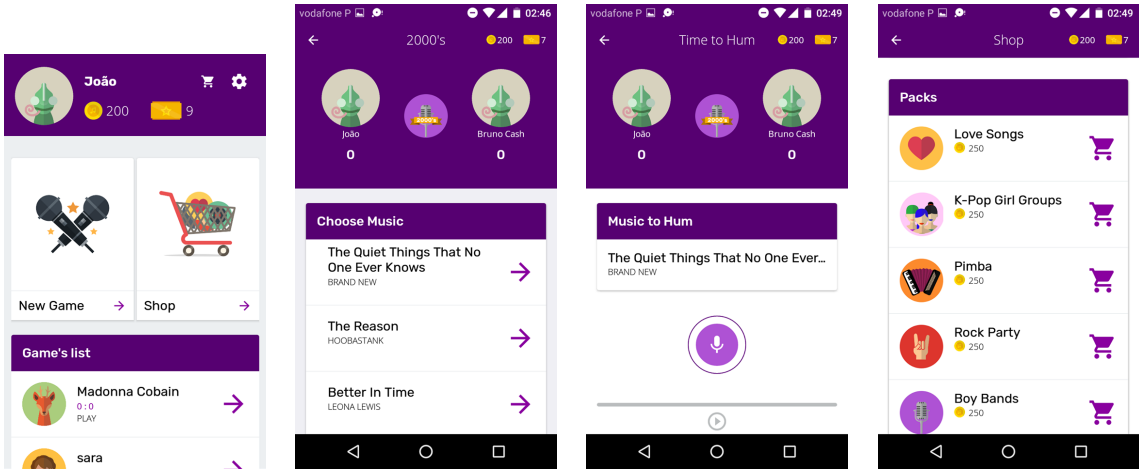

(a) Home screen (b) Gameplay (c) Statistics (d) Store

Figure 2.3: Ruzzle

### 2.1.4 Lalaoke

Lalaoke is a two-player turn-based multiplayer mobile game developed by HYP that is available for Android and IOS. To create a game, a player starts by inviting a friend, searching by their name, or choosing to play with a random user. After that, they select a music category out of three options, followed by a song choice also out of three available options (Figure 2.4 (b)). In case the player does not recognise any of the three songs, they can use a re-roll and new songs to choose from will appear (if he has re-rolls). Then, as shown in Figure 2.4 (c), by clicking in the play button, this player has to hum the chosen song and, when finished, click in the button send. Subsequently, the other player receives a notification and has to guess which was the song hummed by the other player. If this player gets it right, they win a point. After this, roles are inverted.

As illustrated in Figure 2.4 (a), there is a game list where players can continue their games and see their results. The game has a shop where players can buy new song categories and re-rolls (Figure 2.4 (d)). In Lalaoke, players receive a notification every time it is their turn to play.




(a) Home screen (b) Choose Song (c) Sing (d) Shop

Figure 2.4: Lalaoke

### 2.1.5 Trivia Crack

Trivia Crack is a mobile game developed by Etermax and released in March 2013, available for Android, iOS, Facebook and Windows Phone. It became the most downloaded game in December 2014 from the Apple App Store as well as the most viewed advertisement on all of the mobile phone services worldwide [90].

There are different modes available to be played in this game. In classical-mode, players may start a new game by pressing the new game button on the app (Figure 2.5(a)). They can choose to play against a random opponent, one of their Facebook friends, or search for a specific player. After that, the player spins a wheel in which one of six different knowledge categories can come up: Entertainment, Art, Sports, History, Science, and Geography (Figure 2.5(b)).

Subsequently, a question appears regarding that category (Figure 2.5(c)), and if the player gets it right, he can re-spin the wheel. If the player answers a question incorrectly, their turn is over, a notification to their opponent is sent, and control of the game passes to them. If a player does not know the answer to a question, they can use one of the items represented in Figure 2.5 (c) underneath (if the player has any available). The first item allows discarding two of the four possible answers; the second allows the player to play two times in a row, and lastly, the final item gives the player the correct answer.

If the player gets the crown in the wheel, they can choose a category and, if they get the answer right, they win the representative character. In-between rounds, players can see the games statistic, which show the percentage of correct answers in each category (Figure 2.5(d)).

A game is over after either one of the players obtains all six characters, someone surrenders or resigns, or they have played a total of 25 rounds.

The game has a shop where players can buy different items and also daily challenges and bonuses that allow players to win coins.

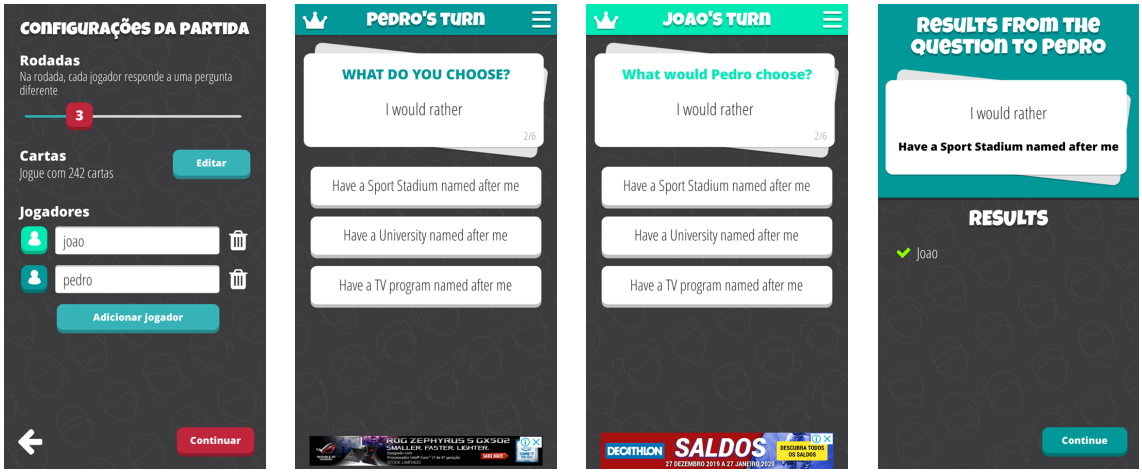

(a) Home screen (b) Category (c) Question (d) Statistics

Figure 2.5: Trivia Crack

### 2.1.6 Noumi: Do you know your friends

Noumi is an offline asynchronous mobile game developed by Treebit Technologies [26]. The objective of this game is to guess what a friend answered on a previous question about themselves. Because of this, the gameplay is very similar to the mobile game that is going to be developed, with the exception that this one is offline and, therefore, must be played on the same mobile phone.

The first step to start the game is too choose the number of players, insert their respective names, and choose the desired rounds (Figure 2.6(a)). Then, one of the players gets a question about themselves and answers it according to what fits them best (figure 2.6(b)). After that, each one of the other players (one at a time), gets that same question about that user and answers accordingly to what they think it fits them better (Figure 2.6(c)). After all the players have answered, at the end of the round, the correct answer appears on the screen, and whether the players got it right or not (Figure 2.6(d)). Then, papers are inverted. This happens until all players have answered a question about themselves, and after that, another round starts. At the end of the game, the overall results appear with the players' points.




(a) Game configuration (b) Home screen (c) Gameplay (d) Statistics

Figure 2.6: Noumi

### 2.1.7 Comparisons and Conclusions

Some of the features that prevail in the studied asynchronous mobile games are identified below, followed by the author's opinion regarding their importance. Afterward, a comparative table is presented.

- **Game history**. Game history corresponds to the presence of a games list where players can keep track of their games. It is essential in these type of asynchronous mobile games because players want to see their on-going games, know whether it is their turn to play or not, and also see the results.

- **Notifications**. The identified mobile games that have notifications, use them to warn the players when it is their turn to play, about new game features or daily challenges. It can be a crucial feature in asynchronous multiplayer mobile games to remind players of their respective turns to play.

- **Shop**. Having a shop where players can spend the coins they won playing the game is a common feature not only in this type of mobile games but in many others. It is not only a way of keeping users playing the game so they can buy new features, but also a way of making them spend real money so the company can profit.

- **Facebook**. Many of the apps that require a login have Facebook integration. Facebook login allows users to login into the apps without having to lose time in registering first. In the majority of mobile games where players can play with friends, they can also invite Facebook friends. This way, they can view which Facebook friends already play the game and invite them.

- **Tutorial**. In some mobile games there are tutorials that can guide players throughout different aspects of the game.

- **Game statistcs**. Every studied mobile game provided game statistics, such as the current game results or inbetween round statistics. This way, players can see how well they are performing against their opponents.

- **Daily bonuses and challenges**. Increasing the players interaction with daily challenges and daily bonuses can be interesting features for players to get back in the game, stay entertained, and possibly win coins to spend in the shop.

Table 2.1 compares the games under review in accordance with the features described above.

| Feature \ Game | Words With Friends | Draw Something | Ruzzle | Lalaoke | Trivia Crack | Naoumi |
|---|---|---|---|---|---|---|
| Game History | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Notifications | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Daily Challenges | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Daily Bonus | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Shop | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Game Statistics | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Facebook | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Tutorial | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ |

Table 2.1: Game features comparison

The main purpose behind the exploratory study of multiplayer asynchronous mobile games is to be able to extract their most inherent features, in order to determine whether they should be part of the mobile game to be developed.

As shown in Table 2.1, Naomi is the only studied mobile game that does not have most of the described features. As mentioned earlier, this game is played offline and on the same phone. Therefore, it does not need features such as Facebook integration (since it does not require authentication) and notifications. However, its gameplay thematic is very identical to the one that will be developed, since a player needs to guess what the other has answered to a previous question. This provided some very useful insights about the design format and structure when already in-game. The studied multiplayer online asynchronous mobile games have many identical features, that are considered relevant functionalities for the mobile game that is going to be developed.

This study will serve as a base for the Requirements Analysis and Specification phase.

## 2.2 Usability and Utility

Usability is defined in International Organization for Standardization (ISO) 9241-11 [79] as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use", and is important for every piece of software, including mobile apps. It is related with how easy it is for a user to use a specific user interface. There are five quality components that define usability[78]:

- **Learnability**: "How fast and easily can a user learn to use a system sufficiently well?"

- **Efficiency**: "Once a user has learned how to use the system, how fast can they accomplish the tasks?"
- **Memorability**: "When users return to the system after a period of not using it, how easily can they reestablish proficiency?"
- **Errors**: "How many errors do user make while using the system, how serious are these errors, how do users recover from these errors?"
- **Satisfaction**: "How pleasant is it to use the system?"

All of the five attributes mentioned above have high importance to ensure user satisfaction, but there is another quality attribute that is also very important: utility. Utility refers to whether the system provides the needed features, whereas usability refers to how easy and pleasant these features are to use. These attributes together are what make a useful system [78] [65] .

To ensure that all five usability quality attributes are met, it is essential to evaluate a system by testing it with representative users, generally done on a stable version of the product. During a usability test, typically, participants try to complete tasks while observers watch, listen, and take notes. The main goal is to identify any usability problems, collect qualitative and quantitative data, and determine whether the participant is satisfied with the product [35]. This can lead to necessary changes and improvements in the User Interface (UI) to make sure the product is suitable for deployment.

Before performing a usability test is very important to map out the goals for the test and discuss what areas of the system or product will be evaluated, as well as the desired qualities of participants and characteristics of users or customers of the product that match the target audience to provide the most accurate results possible [36]. After gathering all the information needed, a good test plan can be designed.

Usability tests must be done at least at the end of the development phase of the mobile game for the reasons already specified. To gather data, different metrics can be used in the usability tests for the mobile game, depending on which features are going to be evaluated, such as success rates, task time, error rates, and satisfaction questionnaire ratings.

## 2.3 User Experience

ISO 9241-210 [12] defines User Experience (UX) as the "user's perceptions and responses that result from the use and/or anticipated use of a system, product or service" and that user experience "is a consequence of brand image, presentation, functionality, system performance, interactive behaviour, and assistive capabilities of a system, product or service. It also results from the user's internal and physical state resulting from prior experiences, attitudes, skills, abilities and personality; and from the context of use". This basically implies that user experience involves features such as human factors, design, ergonomics, Human-Computer Interaction (HCI), accessibility, marketing as well as usability [76].

In light of the above, it is clear that user experience is a much broader concept than usability. While usability answers the question "Can the user accomplish their goal?", user experience answers the question, "Did the user have as delightful an experience as possible?" [7].

Designing a mobile game can be a complicated process. The team must keep in mind that they are not creating a game for themselves but for a specific end-user, to produce great

gaming experiences. While crafting a game is essential to continually guess how players might learn, think, perceive, and react in order to inform the design of the game, and therefore retain the user and make them come back [46].

# Chapter 3

# Project Management

This chapter focuses on the main aspects relevant to the project management: describing the tasks and how they were divided throughout the semesters, which methodology was used and why, the risks that were identified and how they will be controlled, and which tools were used.

## 3.1 Temporal Planning

It was agreed at the beginning of the first semester, that the intern would have weekly meetings (every Monday) with the HYP team with the purpose of presenting the work performed in the previous week and preparing the work to be completed during the following week. Throughout the next sections, the temporal planning of the first and second semesters are presented. For that purpose, the author made use of Gantt Charts, a project management tool used for the planning and scheduling of projects that uses a bar chart to define tasks and deadlines [40].

### 3.1.1 First Semester Expectation and Outcome

A Gantt Chart is presented in Figure 3.1, illustrating the expectations regarding the temporal planning of the first semester.

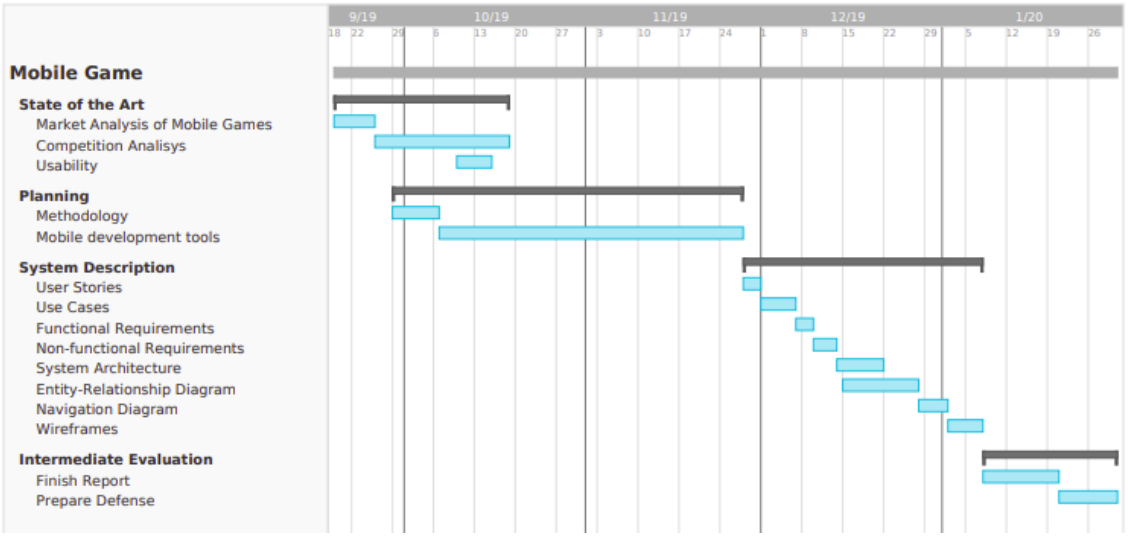



Figure 3.1: First semester expectation

In the first meeting with the HYP team, it was defined that the first subject to be studied would be the State of the Art. This phase would extend for a month in which a market analysis of mobile games should be completed, followed by the study and comparison of different asynchronous mobile games.

Due to the author's inexperience in mobile development tools, it became clear that the study should start as soon as possible. This process involved a theoretical and practical study of mobile frameworks to decide which framework would be the most suitable for this particular project. Due to its importance, this stage was extended until the end of November.

Afterward, the requirements phase would start with the creation of user stories, followed by the use cases. Based on the user stories and the use cases, the functional and non-functional requirements should be identified and prioritised. The expectation was that this phase would extend until mid-December.

Subsequently, the effort would be centered on the system architecture, the entity-relationship, the navigation diagram, and lastly, the wireframes.

During January, the focus would be to finish the report, filling up other aspects not specified in Figure 3.1, such as the introduction, risk management, support tools, and temporal planning. Finally, and after the delivery of the intermediate report, the author should begin preparing for the intermediate presentation.

As illustrated in Figure 3.2, a temporal adjustment regarding the Competition Analysis in the State of Art phase was required, as it had to be extended until the beginning of November. The reason for this extension was the fact that the author needed more time to explore the features of the mobile games studied in this section.

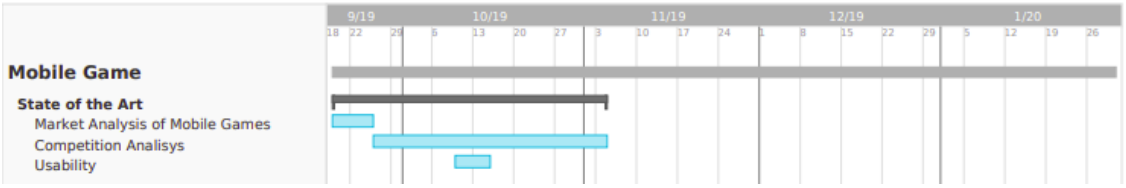



Figure 3.2: First semester outcome

### 3.1.2 Second Semester Expectation and Outcome

The temporal planning for the second semester is represented in Figure 3.3. The main focus was in developing and testing the mobile game.

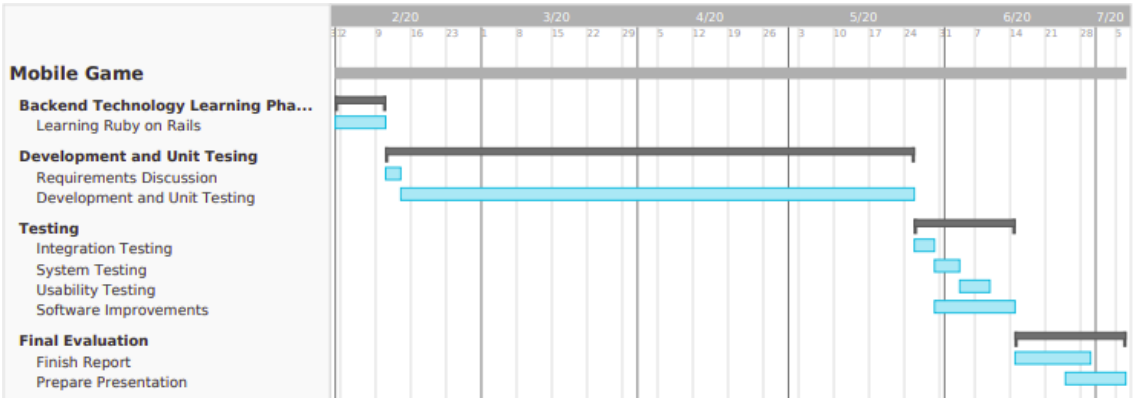



Figure 3.3: Second semester expectation

Throughout the first and second weeks of February the intern was expected to become familiar with Ruby on Rails (which is the framework that is going to be used for the server-side), and learn the best practices.

The next phase was expected to last three months. It began with a discussion with the HYP team aimed at determining the order by which the requirements should be developed. The discussion also focused on the organisation of the project. Subsequently, the author started developing the mobile game. Unit testing was executed every time a set of features was completed.

The testing phase was expected to begin in the last month before the delivery of the final project. Different types of testing methodologies took place in order to validate the application.

The last two weeks of June were reserved for concluding and reviewing the final thesis report. Subsequently, the author prepared the final presentation.

As illustrated in Figure 3.4, the testing phase had some alterations, since the functional testing and the usability testing took longer than predicted.

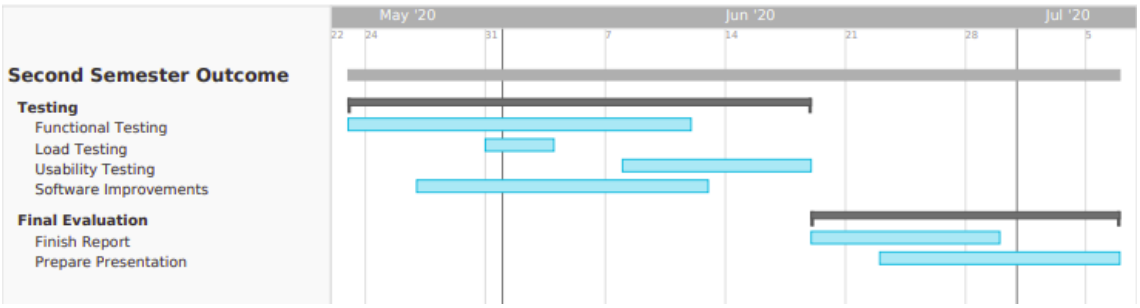



Figure 3.4: Second semester outcome

However, it is worth highlighting that the above-mentioned alterations did not entail significant deviation to the expected planning, considering that all tasks ended up being timely developed.

## 3.2 Methodology

In the temporal planning section, there is a noticeable separation in the type of tasks outlined in the first and second semesters. The first semester was divided mainly in evaluation, requirements analysis and identification, and design, while the second semester was divided between development and validation. As determined since the beginning of the internship, the process model to use is the Waterfall model. The Waterfall approach was the first software development life cycle (SDLC) model to be used widely in Software Engineering, firstly introduced by Benington [32].

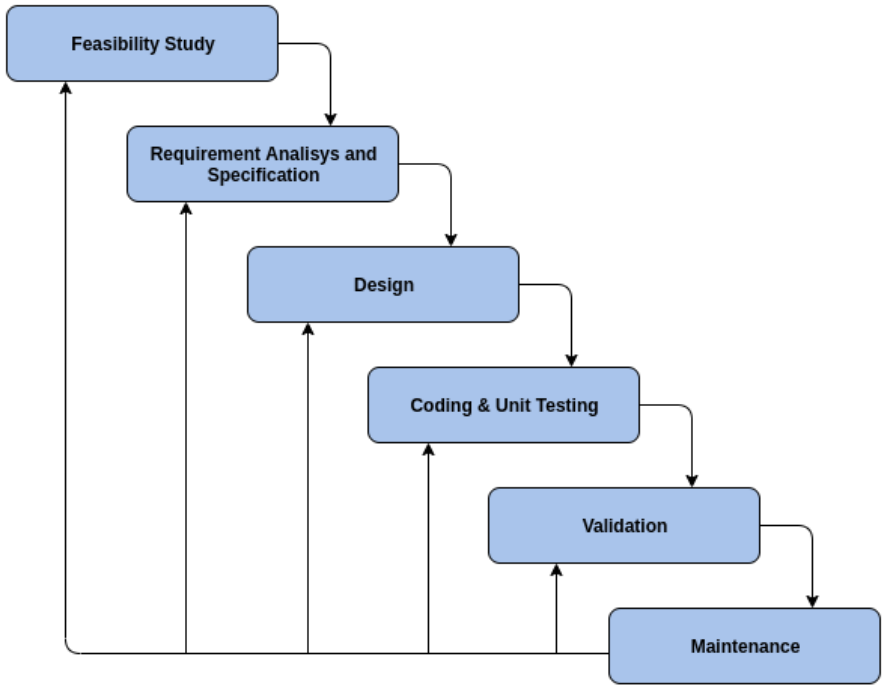



Figure 3.5: Waterfall model with feedback (adapted from [59])

As presented in Figure 3.5, in the Waterfall approach the whole process of software development is divided into separate phases: Requirements Analysis and Specification, Design, Coding and Unit Testing, Validation, and Maintenance. However, this project started with

the feasibility study, which includes the state of the art and the evaluation of technological tools. In the Waterfall model, the outcome of one phase acts as the input for the next phase sequentially and do not overlap [32]. Winston Royce enhanced this model by providing a feedback loop so that each preceding stage could be revisited, making it iterative [84].

The iterative Waterfall Model allows making necessary changes to the classical Waterfall Model so that it becomes applicable to practical software development projects [59]. When errors are detected, changes are needed, or new information uncovered, the feedback paths allow their correction or update during some phase by going back to previous process steps. For example, if during testing a design error is identified, the feedback path allows the team to go back to the design phase and make the necessary changes. Even though there can be reiterations, these can cause significant delays in the development of the project, so it is preferable to make sure that these do not happen by having each phase clearly defined.

One of the main criticisms to the Waterfall Model when compared with Agile Methodologies, is that change is unwelcome. When customers request requirement changes, it can be very expensive to return to development or/and to redesign the project, whereas Agile Methodologies allow changes because of the flexibility it offers [52]. However, in this project, there is no specific customer in mind that are external to HYP; therefore, all the requirements are discussed between the author and the HYP team, reducing the likelihood of occurring changes. Furthermore, this methodology makes it easier to set milestones and for the team to track progress, since all the detailed requirements are defined at the beginning of the process [80].

## 3.3 Tools

Developing a mobile application can become a real challenge for someone that has little experience in the field. That is why it is really important to study and decide which technological tools to use, regarding the project.

In this section, different tools that can be used for the development of mobile games are compared, in order to subsequently identify the ones that were chosen and the reasons underlying the choice. The final decision about the tools was jointly made by the author and the HYP Team.

### 3.3.1 Mobile Development Frameworks

For the client-side, the project proposal of the HYP team required the development of an asynchronous multiplayer mobile game for both IOS and Android operating systems. For this purpose, two different paths could be followed: implement native code for each one of the OS, or use a framework able to release for both of them. The last-mentioned are named cross-platform frameworks. A cross-platform framework is a development tool that allows developers to use a single code base and release to both Android and IOS operating systems (App Store and Play Store, respectively) [83].

Implementing native code for each OS instead of using cross-platform frameworks can have its advantages, such as better performance and speed, and easier connection to the device hardware features [68]. However, it has a significant disadvantage: the development time. In native development, it is necessary to write different code for each OS using distinctive languages (Java or Kotlin for Android and Swift for IOS) and two Integrated development environments (IDES) (Android Studio and XCode). On the other hand, using

a cross-platform framework not only allows the use of a single code base to release for both platforms (and therefore decreasing the development time) but also facilitates when it comes to testing and maintenance [68].

Taking into account the limited time frame for the development of this mobile game, and that during such period the author could not afford to learn two different languages and write two different codebases, it became apparent that using a cross platform framework would be a wiser choice.

There are different options regarding the frameworks to use for the game development on the client-side such as Ionic, Native Script, Xamarim, React Native, and Flutter. In the next section, a feature comparison of all these frameworks is presented.

### Comparison of cross-platform frameworks

Table 3.1 summarizes a comparison of different cross-platform frameworks with the purpose of choosing the best option for this project. The selected features are:

- **Language**. Which language needs to be used for the mobile application development.
- **Performance**. Performance of the apps developed with that framework, that can be crucial for a mobile game;
- **Community Support**. Amount of online support available, such as tutorials online, third-party libraries available, projects (e.g., in GitHub) and number of answers to possible problems (e.g., stackoverflow). These metrics can be used to see how popular an open-source framework is.
- **Documentation**. Documentation can be an important feature, since it is where the most trust-worthy information can be found regarding the specific framework and it is the first guide to be used throughout the development.
- **Hot Reload**. Allows to inject new versions of the files that were edited at runtime while the app is still running. It can be an essential feature for saving time [14].

The information contained in Table 3.1 was taken from multiple sources, namely [75] [61] [60] [58].

| | **Ionic** | **Xamarim** | **Flutter** | **React Native** | **Native Script** |
|---|---|---|---|---|---|
| Language | Angular JS | C# | Dart | React | TypeScript |
| Performance | Medium | High | Very High | High | High |
| Community | Big | Small | Medium | Very Big | Small |
| Documentation | Good | Good | Good | Good | Good |
| Hot Reload | Yes | No | Yes | Yes | No |

Table 3.1: Frameworks comparison

Of the five frameworks described in the table above, Native Script and Xamarim reveal the weakest community support. In almost every project, there are problems that developers struggle to solve. As a result, community support can be an essential feature that leads to solving problems and saving time. Choosing Native Script or Xamarin would be a risk because it could jeopardise the timely achievement of the project's goals since the author

does not have experience in any of these frameworks, and their community support is weak. Furthermore, neither of these frameworks has Hot Reload.

Ionic is a framework that uses HTML, CSS, JavaScript, and Angular for application development, presenting strong community support. However, the fact that it renders its graphic elements via a browser can make a more complex app perform weakly [63]. This situation cannot happen in online mobile games.

For the reasons mentioned above, in one of the first meetings with the HYP team it was defined that the primary frameworks to choose from would be React Native and Flutter.

With the main objective of assisting the author in understanding the development complexity on both Flutter and React Native, it was suggested by the HYP team the development of a Sample App, by implementing some core concepts to give some insight about the right one to use. The structure of this app and the applicable requirements are outlined in Appendix A.

In the sections below, these frameworks are introduced through a description of how each one works. Subsequently, the final decision is presented.

### Flutter

Flutter is an open-source mobile Software Development Kit (SDK) created by Google that had its initial release in May 2017. As a cross-platform framework, it can be used to create native-looking Android and iOS apps from the same code base that is written in Dart [15]. Dart, also developed by Google, is a client-optimised programming language for building apps for multiple platforms [9]. Unlike other frameworks that separate views, view controllers, layouts, and other properties (such as React Native), Flutter has a consistent, unified object model: the widget [15]. A Widget can be, for example, a text, a button, or even a screen layout. The main idea behind it is to have widgets inside other widgets to build the User Interface (Figure 3.6). Flutter is available with a considerable component library, which means that there are a large amount of Widgets that can be used.

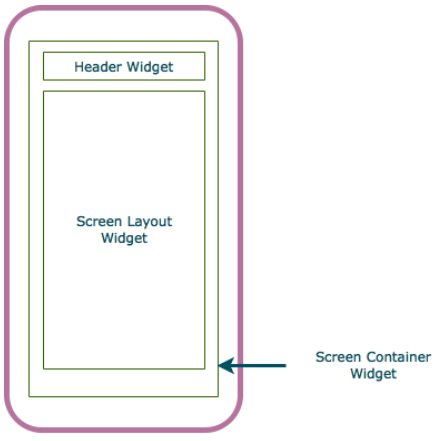



Figure 3.6: Widgets inside widgets [13]

Flutter works quite differently than other mobile frameworks. Apps built in flutter can look like native iOS or Android applications, simply by using the right libraries. Cupertino is for iOS, and Material is for Android, and each one has its own widgets. Because of this, an app can look precisely the same for both OS because Flutter does not have native controls or components. Flutter draws the UI output on a Skia Canvas and then sends it

to the platform [82].

To get started with a Flutter project, the first step is to download the flutter package from Flutters website `https://flutter.dev/showcase`, unzip it, and then create an environment variable pointing to a folder inside of the unzipped folder. Just like any other framework, to be able to run a first flutter project, an Android or IOS emulator(or device) and a IDE (in this case Visual Studio Code was used) are required. Then, it is necessary to run the following commands in the terminal:

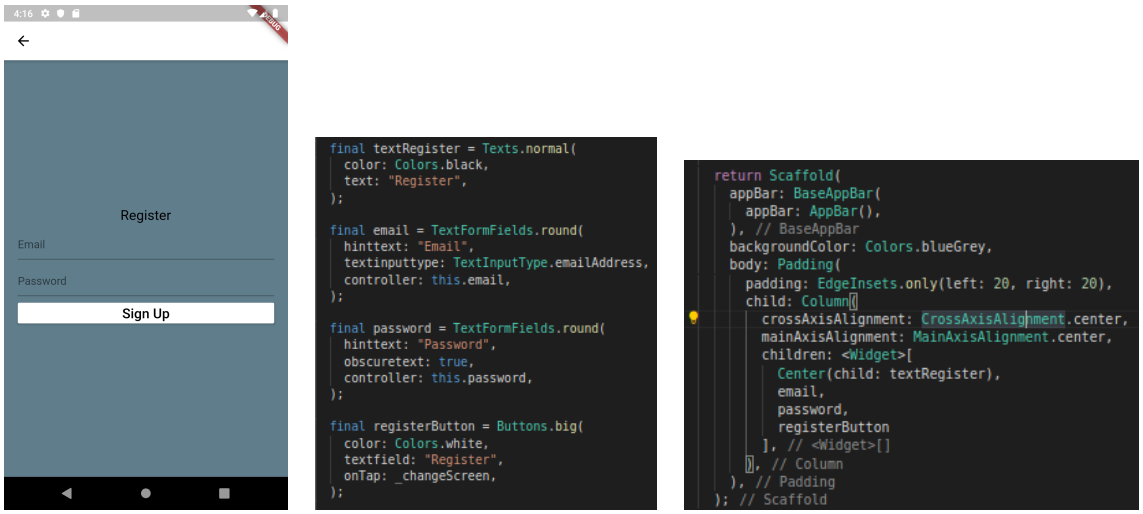
Listing 3.1: Create and run Flutter project

```
flutter create <name of app> # to create a project
cd <name of app> # go to project folder
flutter run <main file> # run project
```

Many frameworks and other technologies rely on the installation of external packages to reach some development objective, and many times it is not easy to get them to work, leading to time being wasted. While developing the Sample App with Flutter, the author did not have to install many external packages because most of them already came by default. However, if needed, a line with the dependency must be inserted on a project file named "pubspec.yalm" and, by running the project, it gets installed automatically.

Code organization is very important during development because code needs to be reusable, readable, and should use the least amount of lines possible. However, implementing, for example, a button in Flutter, can take more lines of code than expected. Fortunately, Flutter uses Dart, which is an object-oriented language. For that reason, a class can be created to return a button after passing it the variables one wants to use (e.g., width, color, text), as illustrated in appendix B. This can be done for every widget to re-use.

Figure 3.7 shows the UI of the register page of the Sample App using Flutter and part of the code that represents it.




(a) Register UI (b) Components (c) Layout return

Figure 3.7: Register Sample App Flutter

Figure 3.4(c) stands for the part of the code that returns the layout for the register screen, which uses the variables from Figure 3.4(b). These variables use constructors in other classes to build that particular component (as mentioned before for the button). Even though it allows this type of re-usability, the author felt that the framework lacks code readability. As can be seen on the examples, Flutter code can involve building fairly deep

tree-shaped data structures where there is no real separation between styles and other properties.

### React Native

Facebook released React Native in 2015 and has been maintaining it ever since. React Native is a JavaScript framework based on React (a Facebook's JavaScript library for building user interfaces), but instead of targeting the browser, it targets mobile platforms [30].

React Native invokes Objective-C APIs to render to iOS components, and Java APIs to render to Android component, unlike other cross-platform app development options which often end up rendering web-based views (such as Ionic). This is all possible because React Native has a "bridge" which provides React with an interface into the host platform's native UI elements [81].

To set up everything, the "Getting Started" documentation was followed in the React Native official website (`https://facebook.github.io/react-native/docs/getting-started`), Several installations may be needed, for example, node.js, Android Sdk, Java SE Development Kit, an emulator/device, an IDE etc.

Once the required components are set up, the project can start by running the following commands:

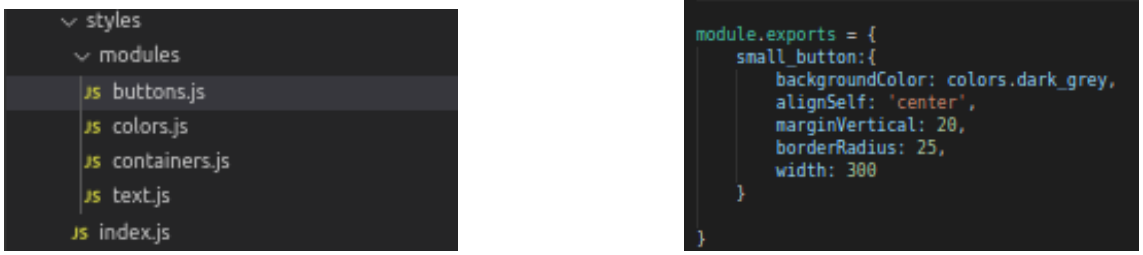
Listing 3.2: Create and run react native project

```
npx react-native init <name of project> # to create a project
cd <name of project> # go to project folder
npx react-native run-android # run project for android
```

Some members of the HYP team already had experience with this framework. As a result, in case any difficulties arose during the development of the app, the author would easily be able to get proper guidance. Before starting the development of the Sample App, the HYP team taught the author some principles to apply when it comes to code structure and organization of a React Native project.

The major problem faced in an early stage in the development of the Sample App in React Native was the fact that it required the installation of many third-party libraries (libraries that do not come by default with React Native). Unlike Flutter, React Native has a small core library and relies heavily on third-party libraries to fill in significant functionality aspects [75]. To install any library, a terminal must be opened in the projects folder and run the line of code that corresponds to the installation of that library. But, for many of them, it requires doing a lot more configuration, consuming a considerable amount of time. For example, "react-navigation" - the one responsible for allowing the transition between screens, and therefore necessary in almost every project -, took a long time to set up.

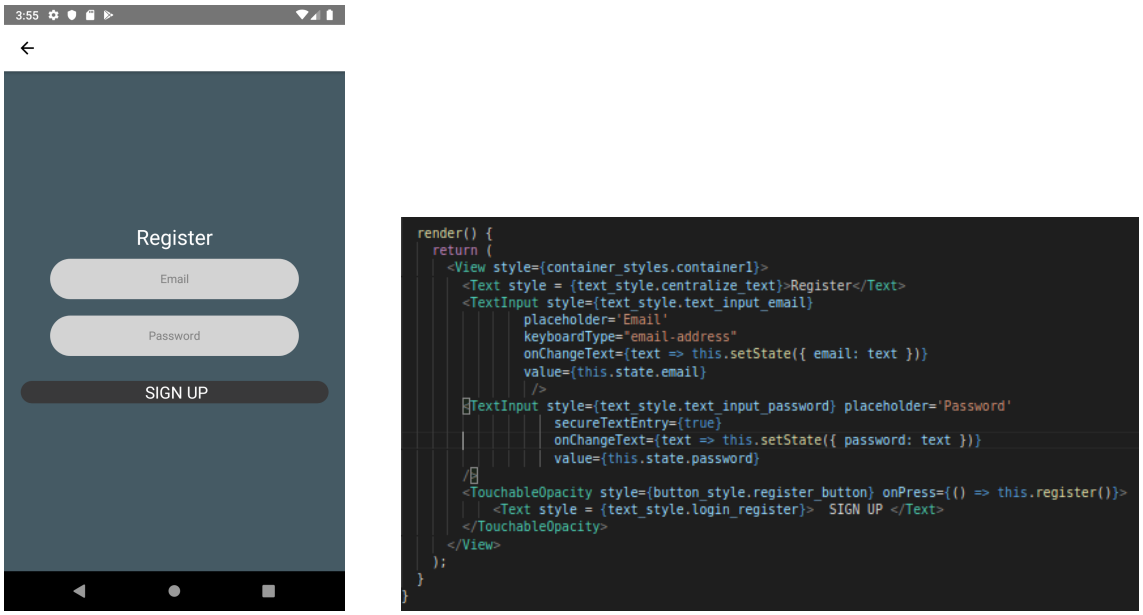
Since React Native uses JavaScript, all the core components accept a propriety named "style". The style names and values usually match how CSS works on the web, except the names are written in camel case (e.g., "fontSize" instead of "font-size"). One can have these styles in particular JavaScript files for each component needed (Figure 3.8 (a)), and then export them to be utilised for styling a component (Figure 3.8 (b)). This way, there is no need to re-write the same code every time a specific styles are used, optimising development time, and code organisation.




(a) Styles folder (b) Button styles

Figure 3.8: React Native re-using styles

The register screen and the code of the render function (responsible for drawing the UI) of the Sample App are presented in the Figures below.




(a) Register UI (b) Register code

Figure 3.9: Register Sample App React Native

The latter illustrates the clear separation between the styles and the other properties, that improve code organisation and readability.

### Decision

The final decision was made in a meeting between the author and members of the HYP team in the last week of November. The weaknesses and strengths of each framework were discussed, most pointed out before. Even though it was balanced, it was decided to go with React Native. The fact that some of the members already have experience in this framework was also heavily taken into consideration because, this way, they could better assist the author throughout the development of the app. In the case of Flutter, even though in the development of the Sample App there were not many complications, one would not know what to expect in a more complex app and, if problems arose, the weaker community support could be a significant risk. That could be a potential risk. Since the

author was also interested in expanding web development skills, going with React Native would also contribute to that aim, since the framework is based on JavaScript and the library React.

### 3.3.2 Server-side/Backend Framework

The mobile game to be developed will be online multiplayer based and asynchronous. Therefore, different player actions will have to go through a server first so that the latter can respond to the requests made by the users and possibly store data in the database. For example, when a user fills a registration form and clicks on the button "Sign Up", a request to the server containing the filled information is sent, and if the information is valid, the server will store the information on the database, and a successful response is sent back to the user. On that account, a framework that can make it easier to develop and maintain server-side code is necessary. One of the restrictions imposed by the HYP Team was that the framework to be used in the server-side should be Ruby on Rails.

Ruby on Rails is a web application development framework written in the programming language Ruby, an object-oriented programming language similar to Perl and Python, and it is designed to simplify programming web applications and also to encourage habits that increase productivity. Ruby on Rails stands on two guiding principles [29]:

- **Do not Repeat Yourself**. This is a principle that aims to reduce the repetition of information that states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system". By not repeating the same information, the code is more maintainable, extensible, and less buggy.
- **Convention Over Configuration**. This is a web application development principle that aims to reduce time and effort taken by developers, because in most frameworks there is a need of writing pages of configuration while in Ruby on Rails these are already configured by default.

Ruby on Rails also uses a Model-View-Controller (MVC) architecture, which aims to separate business logic and application data from the presentation data to the user, as represented in Figure 3.10.

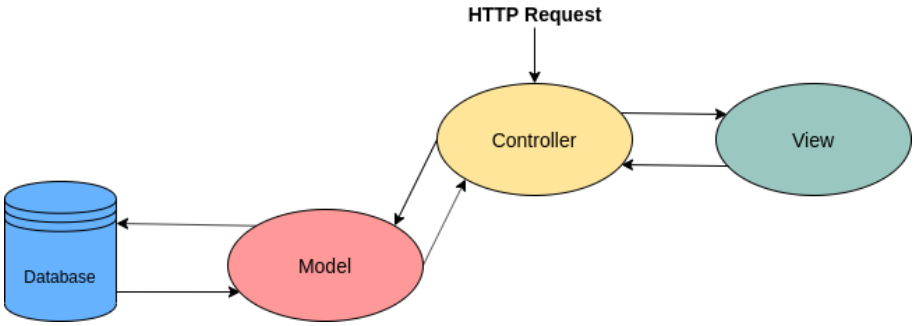



Figure 3.10: MVC model

Each one of the of the MVC components has the following objectives [49]:

- **Model**. Model component corresponds to all the data-related logic, meaning that all the interactions with the database will be done in this component.

- **View**. The View component is used for presenting the data of the model. In this case there will be no View, since the UI belongs to the mobile app,
- **Controller**. Controller works as a middleman between the model and the view components. It listens to all incoming requests and performs an appropriate response back to them, using the Model component.

Like most programming languages, Ruby leverages an extensive set of third-party libraries. These are released in the form of gems, which are packaged libraries that can be installed with a tool called RubyGems [21]. Such gems contain specific pieces of functionality, as well as any files or assets related to that functionality; therefore are in line with the "Convention Over Configuration" principle specified before. This principle also makes Ruby on Rails a favorable choice, since the framework abstracts and simplifies common repetitive tasks. The developer does not have to spend a lot of time configuring files to get setup since Rails comes with a set of conventions that help speed up development [71]. This could be beneficial considering the existence of a deadline and the fact that the author also had to develop the client-side.

### 3.3.3 Database

A database can be defined as an organised collection of data, whereas a database system (DBMS) is a software used to manage data [43]. For this project, the data needs to be divided into different tables that can relate to each other. For this reason, it requires a relational database management system (RDBMS). As described in [41], there are many possibilities regarding which RDBMS to use, such as MySQL, PostgreSQL, Oracle DB, and SQLite.

The use of PostgreSQL was a project restriction made by the HYP team, since it is the database they most frequently pair with Ruby on Rails.

### 3.3.4 Support Tools

Other than the tools mentioned before for the client, server-side, and database, the following tools were selected to assist the author throughout the project:

- **Visual Studio Code**. A source-code editor developed by Microsoft, that was used for the development of the Sample App, and will possibly be used for the mobile game development.
- **Overleaf**[1]. An online latex text editor used to write this paper.
- **Slack**. A messaging platform used to communicate with the HYP team.
- **Draw.io**[2]. A free online diagram editor that was used to create diagrams presented in the section of System Description.
- **GitLab**[3]. Online, open-source git repository used to share code with the HYP team.
- **TeamGantt**[4]. Website that allows building Gantt diagrams.

[1]See more: `https://www.overleaf.com/`
[2]See more: `http://draw.io/`
[3]See more: `https://gitlab.com/`
[4]See more: `https://gitlab.com/`

- **Trello**[5]. Website for project management used in the development phase, which allows organizing and keeping track of project tasks.

## 3.4 Risk Management

Risk management is the process of identifying, analysing and responding to risk factors during the lifetime of a project, to reduce the likelihood of an event occurring and/or the magnitude of its impact [89]. The first step is to identify the risks that might affect the project or its outcomes. Once the risks have been identified, these must be evaluated for their probability of occurrence. These risks can be simple to measure or impossible to know for sure, but it is essential to make the best predictions possible [88]. The probability of occurrence can be divided into low(<40%), medium (40% to 70%), and high (>70%). The next step is to evaluate the magnitude of their overall consequences and how they can affect the Threshold of Success (ToS). The ToS represents the boundary between success and failure of a project. In this project, it was defined that to reach the ToS, all "Must Have" requirements had to be accomplished by the end of the project. Lastly, and most importantly, a mitigation plan must be defined so that risks can be eliminated or at least its impact minimised.

The following risks were identified during the time of the project:

- **R_1**. Inexperience in client-side mobile frameworks can cause some difficulty in the adaptation to the project;
- **R_2**.Inexperience in Ruby on Rails can jeopardize the project;
- **R_3** Additional requirements may be identified during the subsequent phases;
- **R_4** Google takes longer than predicted to review the app, delaying its release on Play Store;
- **R_5** Testing phase reveals errors/bugs that will demand more time spent on development;

| ID | Probability | Impact | Mitigation Plan | Occurred |
|---|---|---|---|---|
| R_01 | High | High | Study those frameworks using online documentation, tutorials, and also develop a simple app | Yes |
| R_02 | Medium | High | Learn through tutorials and documentation | Yes |
| R_03 | Low | High | Possibility to discard "Could Have" requirements | No |
| R_04 | Medium | High | - | Yes |
| R_05 | High | Low | Dedicate more time to testing and software improvements | Yes |

Table 3.2: Risks

Risk 1 was the one with a higher probability of occurrence due to the author's inexperience in mobile development frameworks. Notwithstanding, the risk impact was minimised since

[5]See more: `https://www.teamgantt.com/`

the author managed to gain experience with React Native, mainly due to the development of the Sample App.

Despite the inexperience with language Ruby, the author had already worked with a similar language (Python), which made it easier to adapt to the language and the framework, and therefore the risk having a medium probability of occurrence.

Regarding Risk 4, at some point in the development phase the app needs to be released to the Play Store in testing mode, with the purpose of adequately trying-out some of the mobile game functionalities. At this stage, the app is made available for a selected public to test it as well. Before the app gets release, Google needs to review it to check if everything is as it should be. This process can sometimes take days and therefore jeopardize the timely delivery of the project.

Concerning Risk 5, the testing estimated duration may have been too optimistic (especially for functional testing). As a result, testing took the author longer than expected.

# Chapter 4

# System Description

This chapter begins by explaining the main concept of the mobile game, using a possible game scenario as an example. Afterward, the user stories are described, which together with the comparison of mobile games included in the previous section, helped to define the requirements. Then, the use cases and the prioritised functional and non-functional requirements for the mobile game are presented. Subsequently, the System Architecture is detailed, followed by the Entity-Relationship Diagram, the Navigation Diagram, and the Wireframes.

## 4.1 Game Concept

Before starting to explain the requirement identification process, it is essential to have a clearer idea of the game concept inspired in telepathy.

To better understand the game flow, the figure below (4.1) represents a possible scenario.

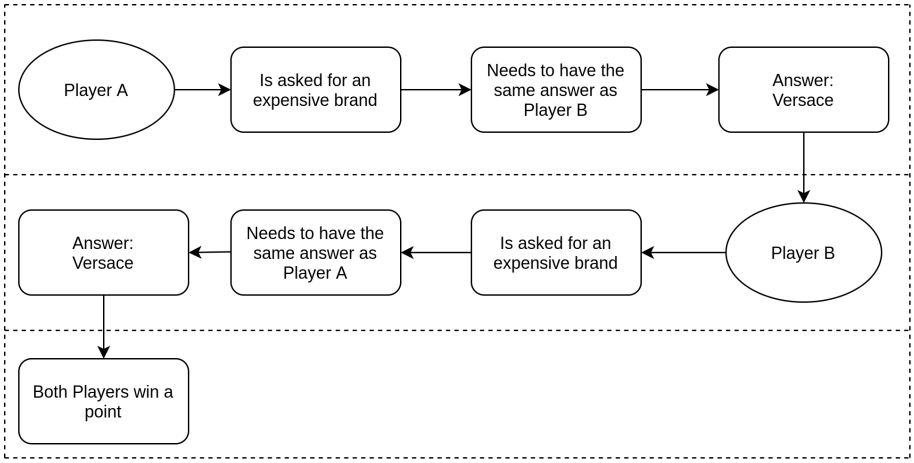



Figure 4.1: Game scenario example

In the scenario presented in Figure 4.1, Player A starts a game with Player B. Each player is using their own device. The mobile game asks Player A to name an expensive brand; subsequently, when asked the same question, Player B will attempt to provide the same answer. Once Player A submits their answer, Player B receives a notification informing that Player A answered a question and that it is their turn to play. Player B will be presented with the same question while being warned that the answer shall match the one

given before by Player A. If both players provide a matching answer, they both win a point. Afterward, Player B can start a new round, and the whole process repeats. As mentioned before, the idea is that a player answers a given question based on what they believe the other player is going to answer or has already answered.

The next section presents the user stories of the mobile game.

## 4.2 User Stories

A user story is a simple description of a feature told from the perspective of the person who desires a functionality, usually a user or customer of the system [38]. These are usually used in Agile methodology's such as Scrum, but they can be helpful for any methodology and are useful on-account-of it forces the team to put themselves in the user's shoes and start identifying requirements [70] [86]. They typically follow a simple template:

**As a** <type of user>, **I want to** <some goal> **so that** <some reason>

The user stories for the mobile game were identified during the requirements specification phase through multiple sessions of discussion between the author and the HYP team; the author would bring new ideas for each session. These sessions resulted in the following user stories:

**User stories about user accounts**

(a) **As a** unauthenticated user, **I want to** register **so that** I can create an account

(b) **As a** unauthenticated user, **I want to** login **so that** I can enter the game with my account

(c) **As a** unauthenticated user, **I want to** login with Facebook **so that** I can create an account faster

(d) **As a** unauthenticated user, **I want to** recover my password **so that** I can login again

(e) **As a** user, **I want to** logout **so that** I can leave my account

**User stories about user profile and information**

(a) **As a** user, **I want to** view my profile **so that** I can view my personal information

(b) **As a** user, **I want to** edit my profile **so that** I can change my profile picture

(c) **As a** user, **I want to** edit my profile **so that** I can change my username

(d) **As a** user, **I want to** edit my profile **so that** I can change my password

**User stories about the gameplay**

(a) **As a** user, **I want to** create a new game **so that** I can play the game with a player

(b) **As a** user, **I want to** want to invite friends **so that** I can play the game with them

(c) **As a** user, **I want to** want to play with a random user **so that** I can play the game

(d) **As a** user, **I want to** want to play with a friend **so that** we can have fun together

(e) **As a** user, **I want to** choose a game type **so that** I can answer a question within that game type

(f) **As a** user, **I want to** answer a question **so that** my friend tries to get it right in his turn

(g) **As a** user, **I want to** answer a question that my opponent already answered **so that** I can get it right

(h) **As a** user, **I want to** answer to questions correctly **so that** I can earn coins

(i) **As a** user, **I want to** answer to questions correctly **so that** I increase my right answer streak with a player

(j) **As a** user, **I want to** change a question **so that** I can get another that I like more

(k) **As a** user, **I want to** get help answering the questions **so that** I have a higher probability of get them right

(l) **As a** user, **I want to** have a wide variety of questions within each category **so that** they rarely repeat

(m) **As a** user, **I want to** watch advertisement videos **so that** I can win free coins

**User stories related with the store**

(a) **As a** user, **I want to** buy re-rolls with coins **so that** I can change questions

(b) **As a** user, **I want to** buy power-ups with coins **so that** I can increase the number of coins I win in a round

(c) **As a** user, **I want to** buy power-ups with coins **so that** I can get help answering the questions

(d) **As a** user, **I want to** buy coins with real money **so that** I can buy shop products

**User stories related with other features of the game**

(a) **As a** user, **I want to** see game results **so that** I can see how many questions me and my friend got right in a row

(b) **As a** user, **I want to** view other rounds statistics **so that** I can see what me and the other player answered

(c) **As a** user, **I want to** receive notifications **so that** I can know when it is my turn to play

(d) **As a** user, **I want to** turn off notifications **so that** I don't receive notifications about the game

(e) **As a** user, **I want to** receives daily bonus **so that** I can receive more coins

(f) **As a** user, **I want to** see a tutorial **so that** I know how to play the game

## 4.3 Use Cases

Use cases are used in system analysis to identify, clarify and organise system requirements. A use case describes an interaction between the system and an actor with the objective of reaching a certain goal [67]. Use cases can be represented through many forms. For each one of the uses cases, it was decided to use a format that has the following fields:

- **Name**. A unique identifier that represents the purpose of the use case
- **Actor**. Actor involved in the use case
- **Description**. Goal to be achieved by use case and sources for requirement
- **Pre-Conditions**- State of the system before the use case scenario happens
- **Post-Conditions**. State of the system after the use case scenario happens
- **Basic Flow**. Steps that the actor follows to make sure that the purpose of the use case is met
- **Alternate Course**. Represent alternate or undesirable paths to the user

To write this type of use cases, the team needs to have a clear idea of how the mobile game is going to work. Usability was taken into consideration during the writing of use cases, since these already describe how a user should interact with the mobile app to complete the different tasks. For this reason, the team had to think in advance about features such as the placement of the different components (e.g., "From where can I access the shop?"), and the number of clicks until the completion of a task.

Even though all use cases used in this project are outlined in Appendix C, illustrated below (Tables 4.1, 4.2 and 4.3) are three of the most important ones, regarding registration, creating a new game with a random player, and guessing the other player answer.

| | |
|---|---|
| **Name** | Registration |
| **Actor** | User |
| **Description** | The user has to register so they can enter the mobile app |
| **Pre Conditions** | 1. The e-mail that the user uses is not registered in the database. |
| **Basic Flow** | 1. Navigate to Registration screen<br>2. Fill name<br>3. Fill email<br>4. Fill password<br>5. Click in register button |
| **Post-Conditions** | User account is created and registered in database successfully |
| **Alternate Course** | 3. Email not in the right format<br>3.1. Message error appears<br>3.2 User stays in register screen<br>4. Password too short or long<br>4.1 Message error appears<br>4.2 User stays in register screen<br>5. Server error message<br>5.1 E-mail already exist in database<br>5.2 User stays in register screen |

Table 4.1: User registration use case

| | |
|---|---|
| **Name** | Create new game with a random player |
| **Actor** | User |
| **Description** | A user can create a new game with a random player |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Choose a game type<br>2. Select random player option<br>3. Clicks in button "Create Game" |
| **Post-Conditions** | A new game with random player is created and database updated successfully. |
| **Alternate Course** | a) 4. Server error message<br>4.1 No players available<br>b) 4. Server error message<br>4.1 Cannot create a game |

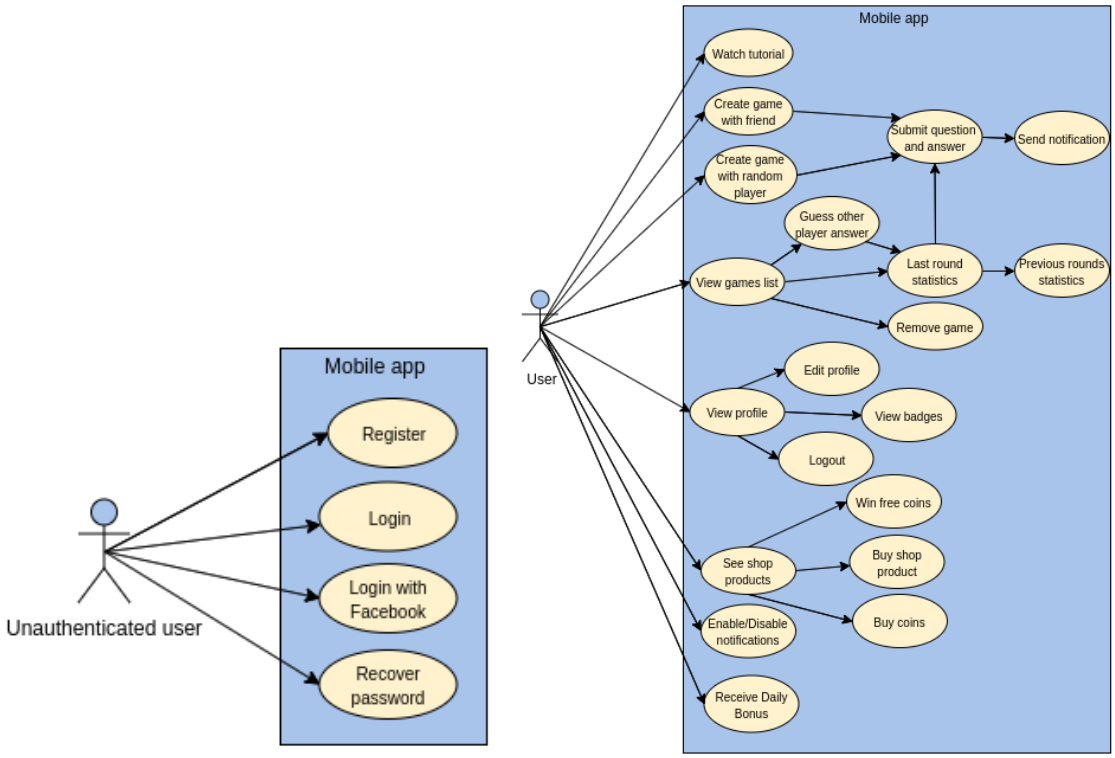
Table 4.2: Create new game with a random player use case

| | |
|---|---|
| **Name** | Guess other player answer |
| **Actor** | User |
| **Description** | The user chooses a game from their list of games that has a ongoing round. Then they answer the question that the other player already answered |
| **Pre Conditions** | 1. The user is authenticated<br>2. The user is in the Home Page<br>3. The user has at least one ongoing game in which it is his turn to guess the other player answer |
| **Basic Flow** | 1. Chooses a game from the games list<br>2. Fills input field with an answer<br>3. Submit answer |
| **Post-Conditions** | User submits the answer and database is updated successfully. |
| **Alternate Courses** | 2. User uses Power-Up to help him answer<br>2.1 User click in power-up helper<br>2.2 First letter of the other play answer appears<br>2.3 User answers the question<br>2.4 User submits answer<br>3. Server error message<br>3.1 Cannot submit answer |

Table 4.3: Guess other player answer use case

### 4.3.1 UML Use Case Diagrams

Use case diagrams are normally used to describe a set of actions that some system or systems should or can perform in collaboration with one or more external users of the system (actors) [34]. In this case, diagrams were used with the purpose of ensuring a better overview of the use cases and their corresponding actors.




(a) Unauthenticated users (b) Users

Figure 4.2: Diagram of user use cases

Figure 4.2 presents two different types of users: unauthenticated users and authenticated users. Unauthenticated users have the usual registration, authentication and password recovery features available, while authenticated users have all game features available, as specified in Figure 4.1.

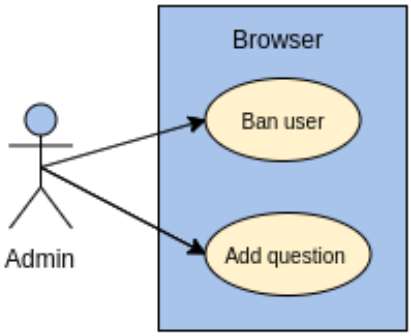



Figure 4.3: Diagram of admin use cases

Admins will have an admin console (backoffice) in which they will be able to perform CRUD (create, read, update and delete) operations in the various entities. However, the most important ones are the possibility of banning users and add new questions to a specific game type.

## 4.4 Functional Requirements

The purpose of this section is to specify the functional requirements, that are based on the user cases described before. Functional requirements were prioritised in a meeting with the HYP team in accordance with their importance for the project. For this prioritisation the MoSCoW method of prioritization was used, which is divided in [1]:

- **Must Have**. Requirement is crucial for the project. Cannot deliver a viable solution without it.
- **Should Have**. Requirement is important but not vital.
- **Could Have**. Requirement is wanted or desirable but less important.

### User Authentication

| ID | Description | Priority |
|---|---|---|
| FR_01 | Register | Must Have |
| FR_02 | Login | Must Have |
| FR_03 | Login with Facebook | Must Have |
| FR_04 | Recover password | Must Have |
| FR_05 | Logout | Must Have |

Table 4.4: User authentication functional requirements

### User Profile

| ID | Description | Priority |
|---|---|---|
| FR_06 | View profile | Must Have |
| FR_07 | Edit profile | Must Have |
| FR_08 | View badges | Must Have |

Table 4.5: User profile functional requirements

### Game related

| ID | Description | Priority |
|---|---|---|
| FR_09 | Create game with friend | Must Have |
| FR_10 | Create game with random player | Must Have |
| FR_11 | Submit question and answer | Must Have |
| FR_12 | Guess other player answer | Must Have |
| FR_13 | Last round statistics | Must Have |
| FR_14 | Previous rounds statistics | Should Have |
| FR_15 | View game list | Must Have |
| FR_16 | Remove game | Should Have |

Table 4.6: Game related functional requirements

### Shop

| ID | Description | Priority |
|---|---|---|
| FR_17 | View products in shop | Must Have |
| FR_18 | Buy shop product | Must Have |
| FR_19 | Win free coins | Should Have |
| FR_20 | Buy coins | Must Have |

Table 4.7: Shop functional requirements

### Others app functionalities

| ID | Description | Priority |
|---|---|---|
| FR_21 | Send notifications | Must Have |
| FR_22 | Enable/disable notifications | Must Have |
| FR_23 | Game tutorial | Could Have |
| FR_24 | Receive daily bonus | Could Have |

Table 4.8: Other functionalities functional requirements

### Admin

| ID | Description | Priority |
|---|---|---|
| FR_25 | Ban user | Should Have |
| FR_25 | Add question | Must Have |

Table 4.9: Admin functional requirements

## 4.5 Non-functional Requirements

While functional requirements describe functions that a software should perform, non-functional requirements define the quality attribute of a software system and how it should work [39]. All of these are essential for a system to work correctly and therefore are prioritised as Must Have.

### 4.5.1 Usability

As mentioned earlier, usability is a key attribute in every mobile application, since a bad user experience can lead the user to never use an application again. It should be easy for the users to achieve goals and to become familiar with the mobile application. This attribute was taken into consideration during the creation of the use cases and also during the wireframes design phase.

| **ID** | NFR_01 |
|---|---|
| **Stimulus Source** | User |
| **Stimulus** | User trying to access a specific functionality |
| **Enviroment** | Normal/Fully functional |
| **Artifact** | Mobile App |
| **Response** | Functionality can be accessed in an efficient and effective way |
| **Metrics** | Time spent and number of clicks until completing an action |

Table 4.10: Non-functional requirement - Usability

### 4.5.2 Scalability

Scalability is the ability of a system to handle increased workload [66]. It is an essential attribute to ensure the normal functioning of the mobile game when handling a high amount of requests. The server is going to be hosted in Amazon Web Services (AWS). AWS is a web service that provides secure, resizable compute capacity in the cloud [8]. AWS has a service that allows making use of horizontal scaling: AWS elastic beanstalk. This service permits creating an Elastic Load Balancing load balancer for the environment, which distributes traffic among the different server instances depending on the current traffic [22]. This service helps making sure that users receive their responses in acceptable times, even when the server is dealing with an high amount of requests. The System scalability was tested later on during the testing phase, through load testing (request simulation).

| **ID** | NFR_02 |
|---|---|
| **Stimulus Source** | User |
| **Stimulus** | High amount of server requests |
| **Enviroment** | Heavy load of users |
| **Artifact** | Server |
| **Response** | Server responds to requests successfully without much latency (<= 3 seconds) |
| **Metrics** | Response time depending on number of request |

Table 4.11: Non-functional requirement - Scalability

### 4.5.3 Availability

From the user point of view, availability over a specified time interval is the percentage of that interval during which the system is available for normal use [50]. Any downtime can be critical for the success of the game and that is why availability is an important attribute for the proper functioning of an application and user satisfaction. As mentioned in their official documentation, AWS guarantees a 99.9% availability time per month of their services [5].

| | |
|---|---|
| **ID** | NFR_03 |
| **Stimulus Source** | User |
| **Stimulus** | Missing response from server |
| **Enviroment** | Failure |
| **Artifact** | Server |
| **Response** | Server must be available 99.9 % of the time |
| **Metrics** | Percentage of downtime per month |

Table 4.12: Non-functional requirement - Availability

### 4.5.4 Security

Security is important for systems that store user data, since no one wants other people to be able to access their information.

| | |
|---|---|
| **ID** | NFR_04 |
| **Stimulus Source** | User |
| **Stimulus** | User tries to register account with acceptable information |
| **Enviroment** | Normal/Fully functional |
| **Artifact** | System |
| **Response** | Server encrypts user password and stores it in the database encrypted |

Table 4.13: Non-functional requirement - Security: Password Encryption

| | |
|---|---|
| **ID** | NFR_05 |
| **Stimulus Source** | Unauthorized User |
| **Stimulus** | User attempts to access or modify information that does not belong to them |
| **Enviroment** | Normal/Fully functional |
| **Artifact** | System |
| **Response** | Server does not allow user to access information and sends error message |

Table 4.14: Non-functional requirement - Security: Unauthorized Access

| **ID** | NFR_06 |
|---|---|
| **Stimulus Source** | User |
| **Stimulus** | User tries to fake a purchase event when buying coins |
| **Enviroment** | Normal/Fully functional |
| **Artifact** | System |
| **Response** | Server does not allow user to buy coins and sends error message |

Table 4.15: Non-functional requirement - Security: Fraudulent purchases

## 4.6 Constraints

Tables 4.16, 4.17 and 4.18 present the main technical constraints imposed by HYP that had to be respected during the development of the project.

| **ID** | TC_01 |
|---|---|
| **Source** | HYP |
| **Title** | Back-end in Ruby on Rails |
| **Description** | The server side must be developed in Ruby on Rails since some HYP software engineers already have experience in this framework |

Table 4.16: Technical constraint - Ruby on Rails

| **ID** | TC_02 |
|---|---|
| **Source** | HYP |
| **Title** | Database in PostgreSQL |
| **Description** | The database that will store all the application necessary information will be in PostgreSQL since it is the databse that the HYP team frequently pair with Ruby on Rails |

Table 4.17: Technical constraint - PostgreSQL

| | |
|---|---|
| **ID** | TC_03 |
| **Source** | HYP |
| **Title** | Amazon Web Services |
| **Description** | The web server is going to be store in Amazon Web Services since its the cloud that HYP team normally use to store most of their applications |

Table 4.18: Technical constraint - AWS

## 4.7 System Architecture

The System Architecture is presented in Figure 4.4. This architecture is composed by the mobile app, developed in React Native, that makes Representational State Transfer (REST) requests to the server developed in Ruby on Rails. The server is composed of an Application Programming Interface (API) that uses the Controller and Model components from the MVC model to communicate with the mobile game, as mentioned earlier. The admin console will use the complete MVC model that can be accessed through a specific Uniform Resource Locator (URL) and will also make REST requests to the server.

The server retrieves information from the PostgreSQL database, using SQL queries. The server is separated from the database because of the possibility of existing many server instances, in case the system reaches an unbearable number of requests.

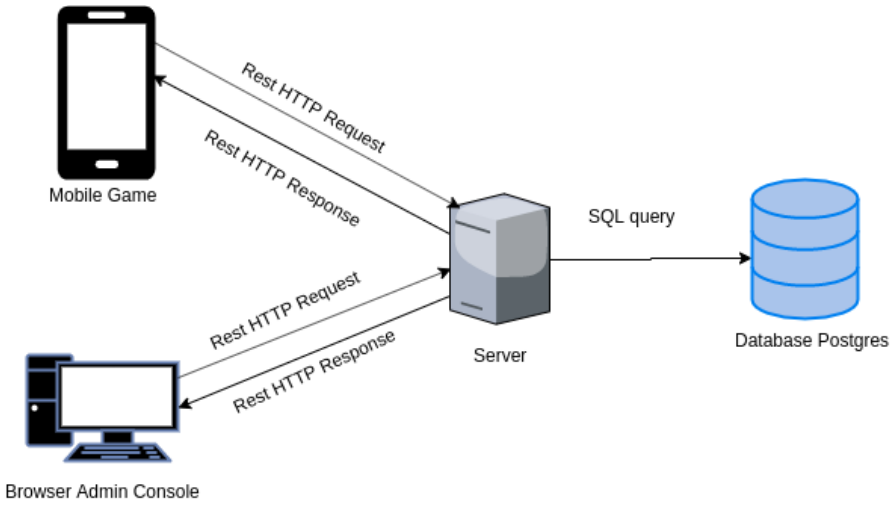



Figure 4.4: System architecture

The Containers diagram for the project is presented in Figure 4.5. A Containers diagram zooms into the software system, showing the containers (applications, data stores, microservices, etc.) that make up that software system, as well as their respective responsibilities [6].

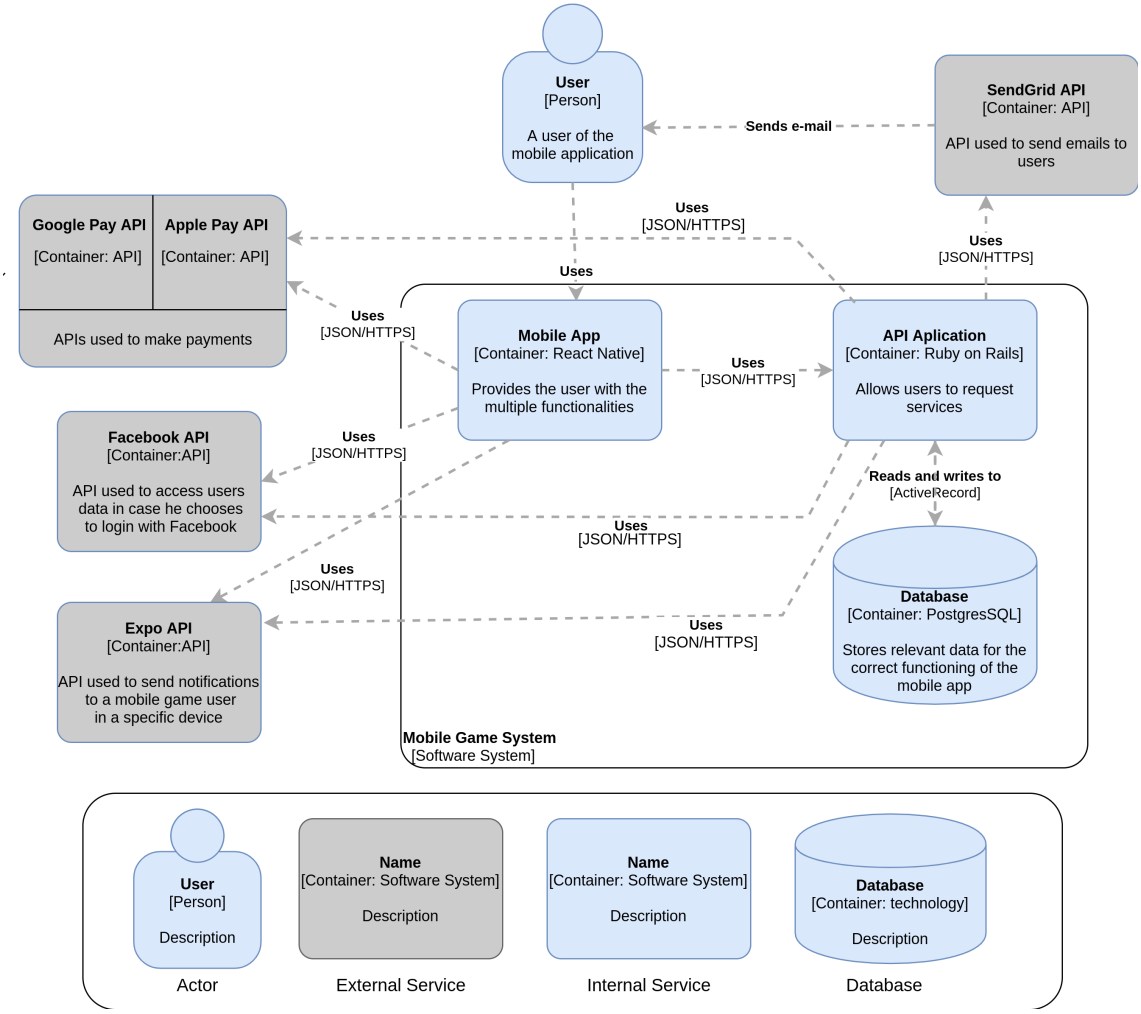



Figure 4.5: Containers diagram

As illustrated in Figure 4.5, other than the mobile game system already described in Figure 4.4, composed by the mobile app, the API, and the database, additional software systems are going to be integrated, specifically:

- **SendGrid API** to send e-mails in case users want to recover their password.
- **Google Pay API and Apple Pay API** for payment purposes, in case a user wants to buy coins in the game.
- **Facebook API** to retrieve users Facebook information in case they choose to login with Facebook.
- **Expo API** to send notifications to a user device when another player answers a question.

## 4.8 Entity-Relationship Diagram

An Entity-Relationship Diagram (ER) is a type of flowchart that illustrates how "entities" relate to each other within a system [42]. The system ER diagram is presented in Figure 4.6, regarding the information to be stored in the database.

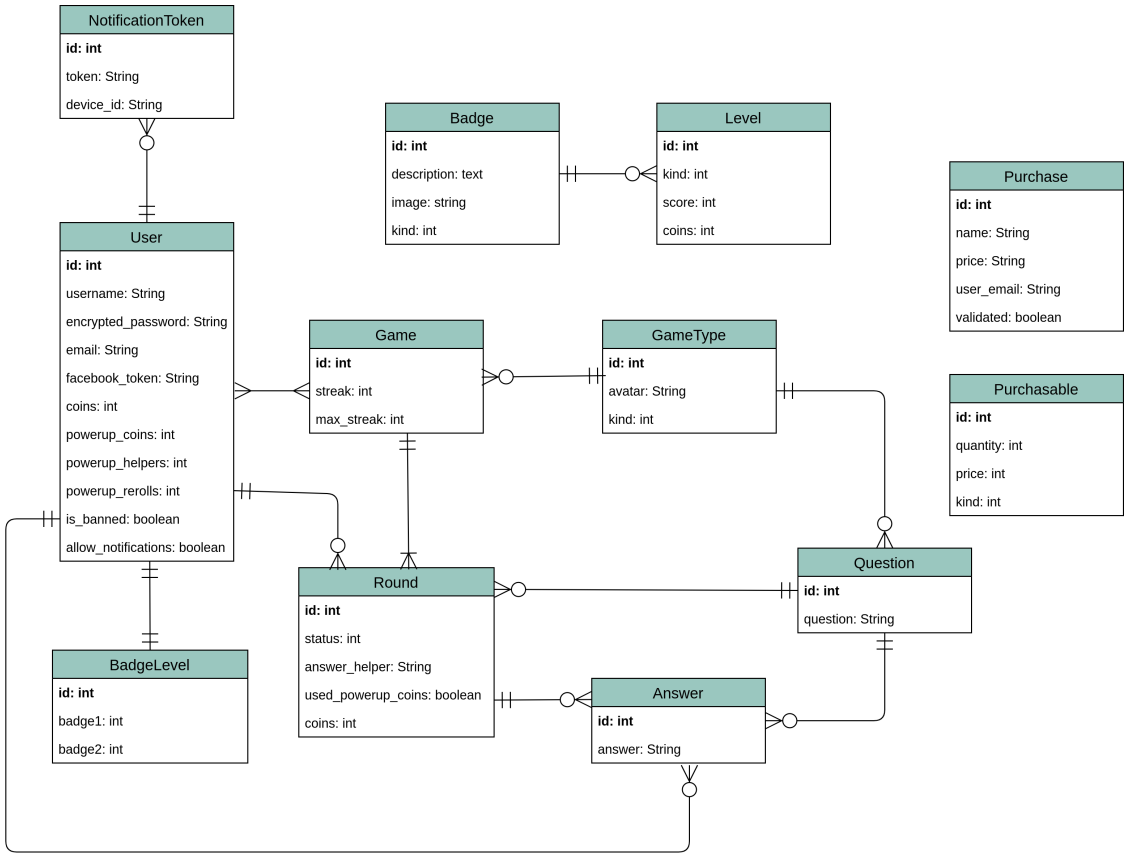



Figure 4.6: Entity-relationship diagram

All entities have a primary key id (identifier). Table User has all the necessary information for the user, such as username, password, email, facebook_token (if necessary), avatar (profile image), the coins they currently have (and can spend in the store), the total coins ever spent, and the different powerups (items to use in-game). One user can have many games (relationship many-to-many because a Game has two users), many rounds in which is their turn to play, many answers, has a relationship one-to-one with table BadgeLevel because a user has a certain level in each one of the existent badges, and can have many notifications tokens which has token and the device_id for the server to be able to send notifications to this user.

On the table Game, the streak (current number of rounds won by both players in a row in a game), and max streak (highest streak of rounds won) are saved. A game is played by two users (table User), is composed of many rounds (table Round) and has an associated game type (many-to-many because a game type can belong to many different games).

On the table Round, the number of coins associated with that round in case both players win, the used_powerup_coins that tells if the powerup coins was used, the answer_helper that represents the string in case a player uses powerup helpers, and the status representing which user has to play next, are saved.

Table GameType is used because users can play different kinds of games. In table GameType, the kind of the category and the related avatar are saved. A game type can belong to many games and have many associated questions.

On the table Question, the text that represents that question is saved. A question belongs to one game type and can be in many different rounds.

On the table Answer, the text that represents an answer is saved. An answer belongs to a user and to a round.

On the table Badge, a description, an image and a kind of a specific badge are saved. Each one of the badges can have many levels (relationship one-to-many). Each level has a kind (Bronze, Silver ...), a score a user needs to reach to be awarded that level, and the coins the user wins in case they reach that level.

On the table Purchasable, are saved the quantity, the price (in coins) and a kind (reroll, helper, or powerup coins) of items that a user can buy in the shop.

Finally, the table Purchase represents a purchase that a given user did with real money to buy coins to use in the store, and has a status field to see if the transaction is already completed.

## 4.9 Navigation Diagram and Wireframes

The navigation diagram of the mobile application is presented in Figure 4.7, which shows the application screen flow from the player's perspective. Afterwards, the wireframes of the mobile application are presented. Wireframes for the mobile app are a simplified visual concept of the future app, without design, that helps to understand how the app works [55].

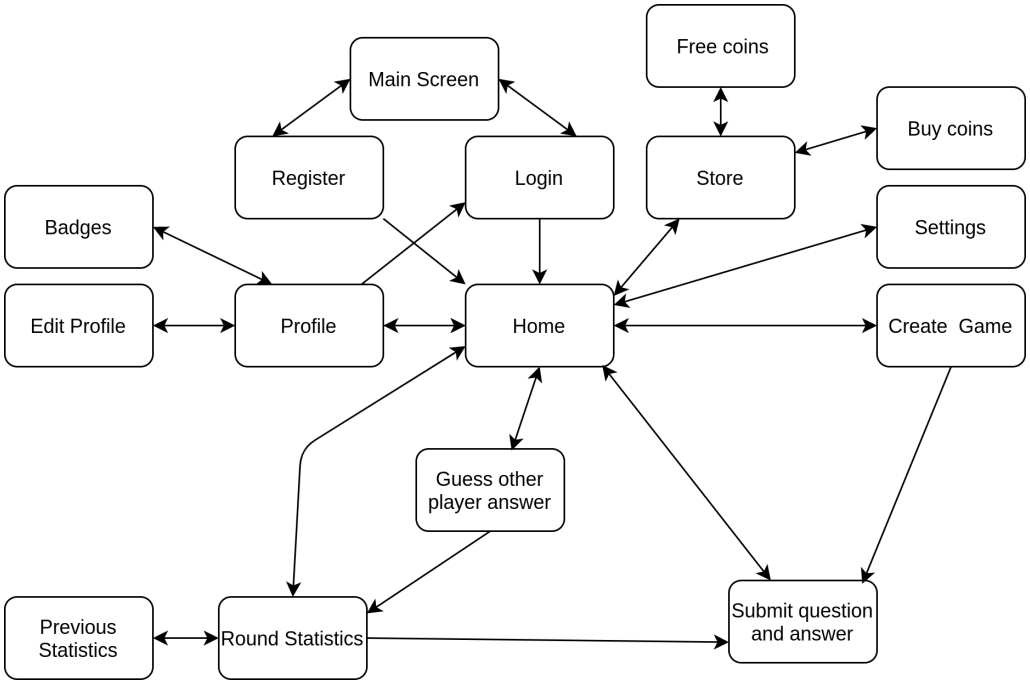
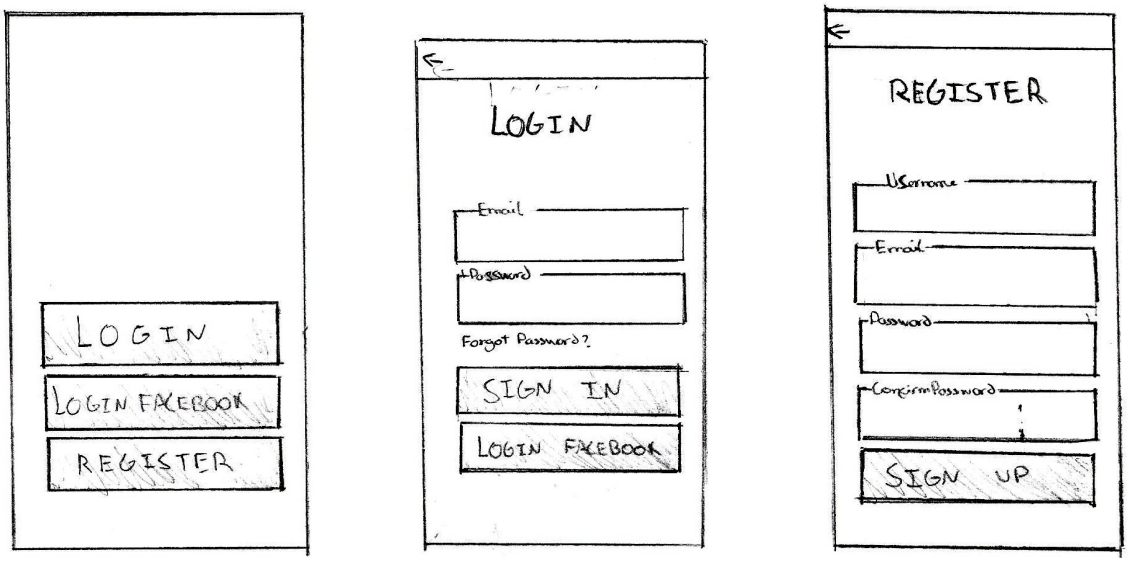



Figure 4.7: Navigation diagram

When a user enters the mobile application for the first time, they encounter the main screen (Figure 4.8 (a)). On this screen, the user can choose whether to login, login with a Facebook account, or register. If they choose the first option, they navigate to the login screen, where they have to fill the input fields with the correct information. In case the user forgets their password, they can recover it by clicking in "Forget Password?". If the user wants to register in the application, they can navigate to the register page (Figure 4.8 (c)) and fill the input fields. In both cases (login and register), if the information is valid,

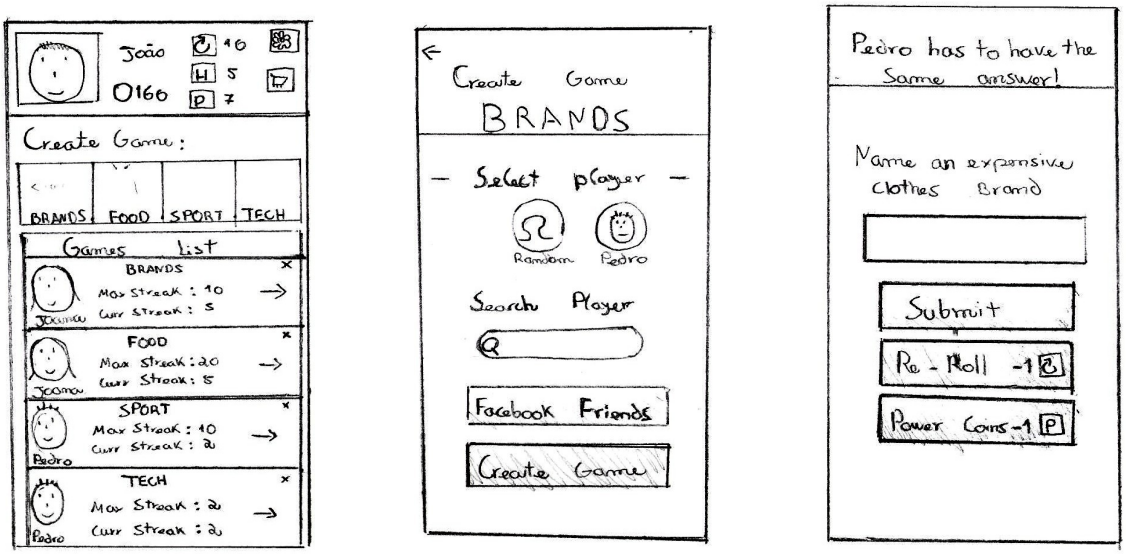
the user navigates to the home page (Figure 4.9(a)).




(a) Main screen (b) Login screen (c) Register screen

Figure 4.8: Mobile game wireframes 1

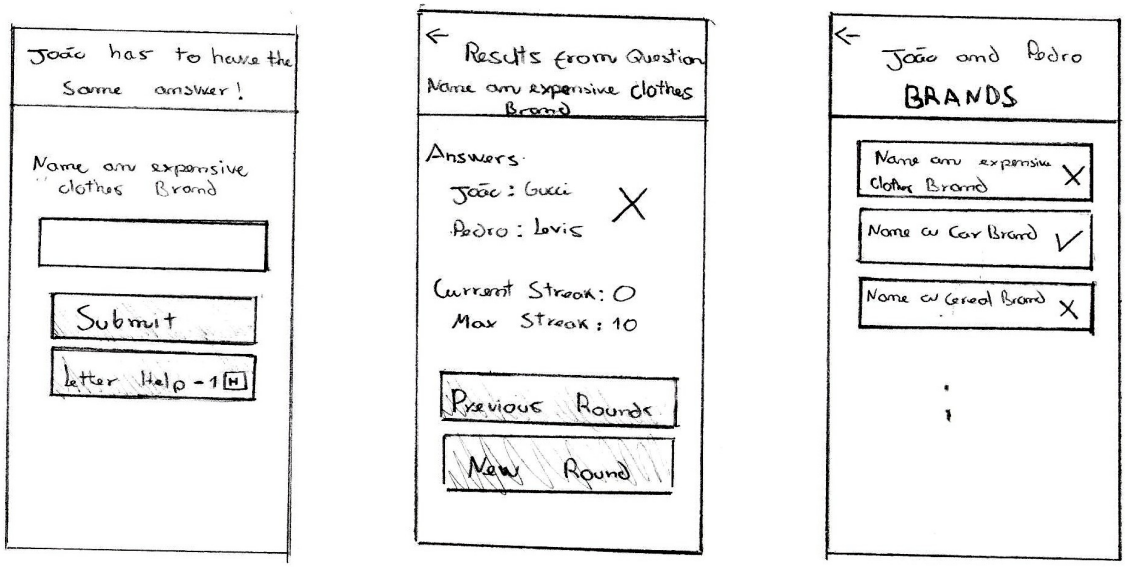
The user can access most of the application functionalities in the Home Page. They can access their profile after clicking in their avatar (Figure 4.10(a)), the settings screen (Figure 4.12 (c)), the game shop (Figure 4.12(a)), view the game list (in which they can continue their respective games), and also create a new game. A new game can be created by choosing a game type on the home screen. Afterwards, the user navigates to the create game screen (Figure 4.9 (b)), in which they can choose from playing with a random player, a Facebook friend, or search for a username. After selecting a player, the user clicks "Create Game", and a round begins. For the user that created the game, a question appears, and they will have to answer depending on the kind of round (in Figure 4.8(c), both players have to get the same answer). Before submitting their answer, the user can re-roll the question (change it) or increase the number of coins to be won in case both get it right. After submitting, a notification is sent to the other player, and the user navigates back to the Home Page.




(a) Home screen (b) Create game (c) Submit question and answer

Figure 4.9: Mobile game wireframes 2

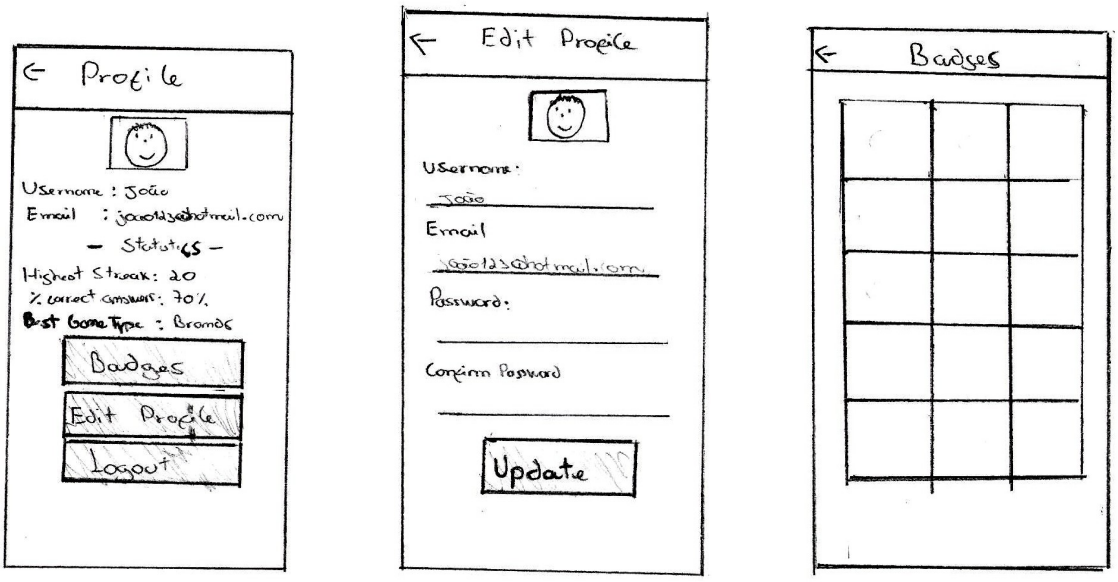
Once a user submits their question and answer, the other player can access it by clicking in the respective game in the Home screen. Subsequently, the same question appears, and the player has to answer correctly on account of what the other player may have answered (Figure 4.10(a)). Before submitting their answer, they can click in "Letter Help" (in case they have them available), and the first letter of the correct answer will appear in order to assist the player. After submitting, the results of that round can be seen (Figure 4.10(b)), specifically the answers given by both players and the current and maximum correct answer streak within that game type, with that player. In this screen, they can click in "Previous rounds" to see other rounds results and statistics (Figure 4.10(c)), as well as start a new round by clicking in "New Round", whereas the process repeats.




(a) Answer question (b) Last round statistics (c) Previous round statistics

Figure 4.10: Mobile game wireframes 3

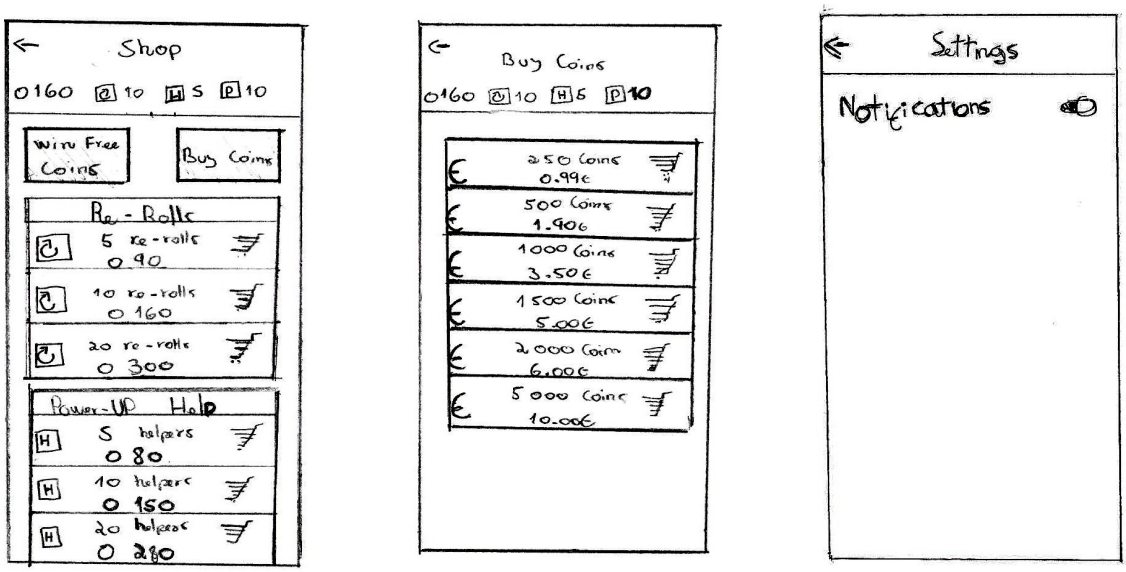
As said before, the user can navigate to their profile through the home screen (Figure 4.11(a)), in which they can view their information. In this screen, the user can logout, navigate to the edit profile screen, and go to the badges screen. In the edit profile screen (Figure 4.11(b)), the user can change their personal information, and in the badges screen (Figure 4.11(c)) they can view their badges. Each badge represents goals that the user can achieve, that will reward them with coins.




(a) Profile (b) Edit profile (c) Badges

Figure 4.11: Mobile game wireframes 4

In the shop screen (Figure 4.12(a)), which can be accessed through the home screen, the user can view the different items they can buy and their respective cost (in-game coins). They can buy three different types of items: re-rolls, power-up help, and power-up coins (their goal already specified earlier). By clicking in "Win free coins", an advertisement video appears, and if the user finishes watching it, they will be awarded coins. If the user wants to buy coins, they can click in the button "Buy coins" and navigate to the respective screen (Figure 4.12(b)). On this page, the user can spend real money to buy coins so that they can spend it on shop items.




(a) Shop (b) Buy coins (c) Buy coins

Figure 4.12: Mobile game wireframes 5

This chapter provided a description of essential elements for the development of a new asynchronous mobile game using telepathy as a theme. Chapter 5 details how it was accomplished.

This page is intentionally left blank.

# Chapter 5

# Development

This chapter details the implementation phase of this project. It starts by explaining the development process and the organizational standards followed by the author to accomplish the project goals. Subsequently, to ensure enough understanding of some essential implementation details, a description of the server-side and client-side project structure is provided. Finally, the requirements and their implementation details fulfilled during the development phase are explained.

## 5.1 Process and Organization

As every other software project, being organized is a critical element that can contribute to the likelihood of its success. For this purpose, a Trello Board was set up at the beginning of the development phase. A Trello Board is a tool designed for organizing projects of any size, by dividing the tasks in Trello Cards. In each one of these cards, comments may be added, file attachments uploaded, checklists created, as well as labels and due dates, and more [33]. These can be especially useful when working in a team because each member can keep track of other people's work.

Figure 5.1 is presentes the Trello Board of this project in its final stages of development:

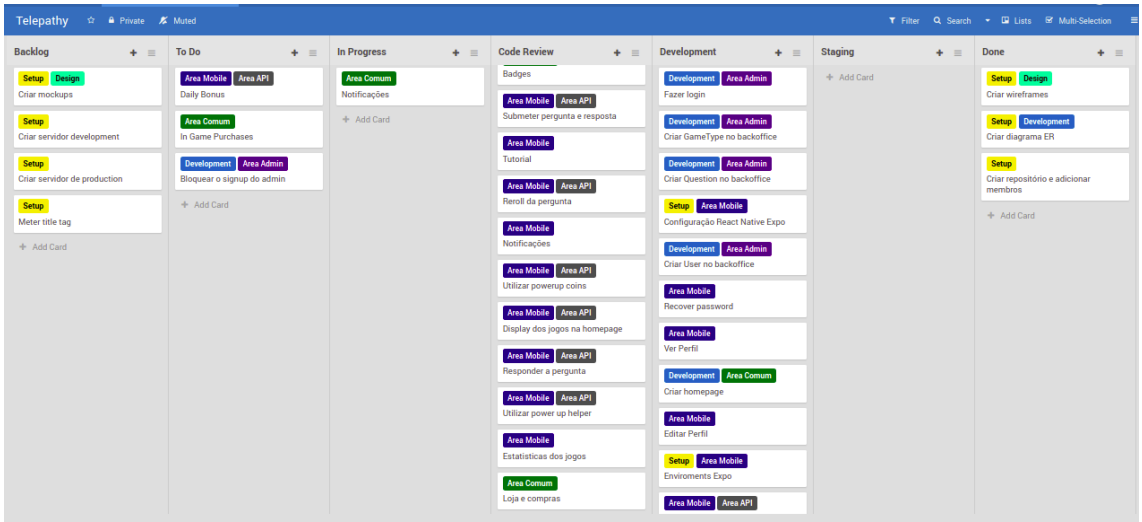



Figure 5.1: Project trello board

The Trello Board is divided in 7 columns:

- **Backlog**, list of tasks that need to be dealt with.
- **To Do**, list of tasks to start in a near future.
- **In Progress**, list of tasks being handled in the present.
- **Code Review**, list of tasks completed and awaiting review to be merged into the development branch.
- **Development**, list of tasks already reviewed and merged into the development branch.
- **Staging**, list of tasks pending on testing to be considered as fully completed.
- **Done**, list of fully completed tasks.

Tasks are picked from the backlog taking on account the following points:

- **Their importance for the objectives of the project**. For example, if a task represents a "Must Have" requirement, it has a higher priority comparing with a "Could Have" requirement.
- **The dependency of tasks**. Some tasks depend on others to be completed. For example, for a user to be able to submit an answer to a given question, a game needs to be created first. In this case, the task "Create game" must precede "Submit Answer".
- **The ease of the task implementation**. Some tasks are easier to develop than others. In the author's point of view, developing easier tasks first is beneficial for his learning curve.

At the beginning of the development phase, two repositories at GitLab were created: one for the mobile app and another for the server-side. Every time the author completed a task, he created a merge request to the "development" branch in its supposed project repository. Then, a senior developer would review the code in the merge request. If the reviewer understood everything to be correct, the merge request would be accepted, adding the code to the "development" branch. If they identified errors, imperfections, or a need for code improvements, they would comment on the respective code lines so that the developer could address them as soon as possible. The "development" branch contains the code that a senior engineer from HYP reviewed, but has not been fully tested.

When the developer makes a merge request to the "development" branch, he proceeds to update the Trello Board, passing the respective task(s) card(s) to the "Code Review" column. When the reviewer accepts the merge request, the author moves it to the "Development" column. This way, the developer can always keep track of their tasks.

The reviewing code method is an effective quality assurance technique with the main goals of discovering direct quality problems in the code [57].

## 5.2 Project Structure

As mentioned in section 3.3.1, writing maintainable code is very important in any project nowadays, especially when working on a project with a team. Maintainable code is code easily modifiable, extendable, and readable [69]. A developer does not know if they will need to go back to a past project, or if someone else will be a part of it in the future. For this reason, it is crucial to set several rules and respect them accordingly; otherwise, it can cause substantial delays (when fixing bugs, for example) and unnecessary code refractors, among others. In light of the above, the author attempted to follow the frameworks and company standards.

The following subsections present the file structure of both the server and the client-side to understand some essential implementation details and the division of the files and folders in each one of them.

### 5.2.1 Server Side

The server-side is mainly divided into two parts: the user area and the admin area. To better understand this division and how the project is organized, the figure below represents the server-side file structure joined by a description of three key folders: app, db and config.

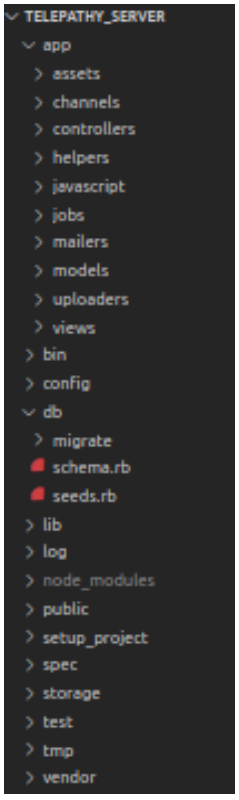



Figure 5.2: Server file structure.

Most of the work was conducted in the first folder ("app"), particularly in its subfolders: controllers, models and views.

Controllers are responsible for routing external requests to internal actions. Even though it is not visible in Figure 5.2, the controller's folder contains distinct controllers for the admin and the mobile app.

The user API endpoints are presented in Appendix D. Each of these includes a name, a description and the parameters received. It is also important to mention that, for a user to be able to make use of these endpoints, they must send on the header of the request their uid (email), their client (unique identifier) and their access token (unique identifier), which are used as a security mechanism to identify the users.

The models folder contains the models' files and each one represents an entity from the ER diagram in section 4.8, Figure 4.6 (e.g., user, game). Inside each file are written the associations between the distinct entities and the necessary validations. Also, it is where the CRUD operations are performed.

The views folder contains mainly the views for the admin console written in the format "html.erb".

The db folder contains all the database information: the schema, which lists all tables and columns on the database, and each migration that introduced changes to the schema.

Finally, the config folder includes the Ruby Gems (libraries), the routes that make Rails able to connect requests to a controller action and configurations for the server and its environments (development, staging, and production).

### 5.2.2 Client Side

This subsection presents the client-side implementation details from the project developed using the React Native framework. Figure 5.3 shows the client-side project structure.

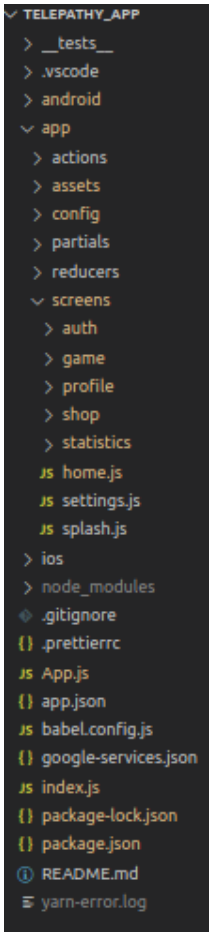



Figure 5.3: Client side file structure.

There are three main folders in this React Native project: the android folder, which has the native code (written in java) needed to run the app in an android platform; the app folder, where most of the code is written (in javascript); and ios folder which has the native code (written in Swift), as needed to run the app in an ios platform. At the bottom of the file structure are other important files, such as the "app.json," that has essential variables for the project (such as the app name, display name, among others). The "package.json" shows the dependencies/libraries installed and used in the project, and the "index.js" represents the starting point for the project to run.

Inside the app folder, each folder has different goals. Inside the screens folder are all the application layouts, which are also divided into different folders depending on the related requirements (authentication, shop...). In the assets folder is where all the app styles are defined, which in turn are divided into different files depending on the related component (buttons, images, texts ...). In the actions folder are the files with the functions needed to send the desired requests to the server API. These functions are called inside the layouts files when a specific API request is required. In the config folder are the configurations and global variables used in the project.

The reducers folder is related to an important library that was used on this project: the redux. Redux is a state management tool, by which the state of the application can be kept in store (even when the app is closed) and be accessed from all components. For example, when navigating to the profile screen, a user can see their username, email, and avatar. To get these variables, no server requests are required because they are already saved in the redux store since the player logged in. However, for example, if a user changes their username in the "Edit Profile" screen, the redux store needs to be updated after the server responded. The files inside the reducers folder have the purpose of changing the redux store when an update is required.

## 5.3 Requirements

This section elaborates on the requirements satisfied during the development of the project, which were all accomplished. First, a Gantt Chart with the task completion time outcome is introduced to show how the work was divided during the development phase. Then, in the following subsections, are described the implementation details of the requirements taking into account their context to the project, starting by the admin functionalities implemented and then going into more detail for the implemented user's features. The requirements presented are joined by the representative layouts.

### 5.3.1 Tasks Completion Outcome

Figure 5.4 shows a Gantt Chart representing the division of tasks during the development phase.

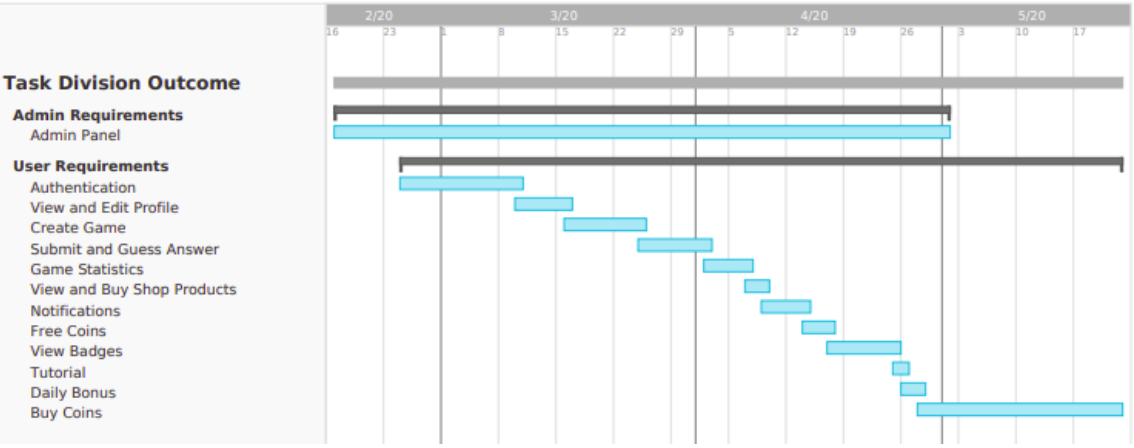



Figure 5.4: Tasks completion outcome

After the Ruby on Rails adaptation phase, which lasted nearly two weeks, it was decided that the first thing to do should be configuring the server-side project, which was done with the help of a senior engineer at HYP. After that, the author started by implementing some functionalities in the admin panel.

The functionalities of the admin panel were developed depending on its utility for the client-side. For example, before starting developing authentication for the client-side, the author made sure that it was possible to perform operations on the user model in the admin panel because of the usefulness they could bring for those specific tasks.

### 5.3.2 Admin Panel

An admin panel can be practical since it allows for accessing and manipulating data within the user interface of the website. For that reason, it allows admins to perform desired CRUD operations in the database without the use of other more complex methods. When the app goes live, the admin panel will be the primary resource for changing the database.

This subsection focuses on showing some functionalities available for the administrators, which are accessible after an administrator logs in with acceptable credentials.

#### Users

Figure 5.5 illustrates the Admin Panel for the user's section. Admins can perform multiple operations in registered users, such as view their information, edit their information (which can be useful for testing or if some transaction fails), delete a user, and ban a user.

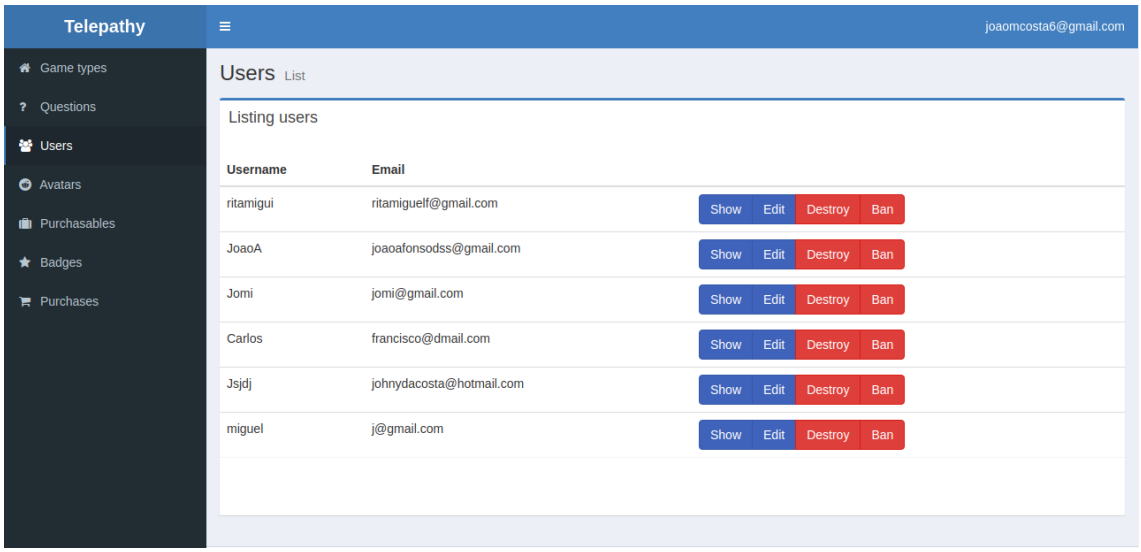



Figure 5.5: User admin panel

### Game Types

As specified in the requirements, a user must first choose a game type to start a game. Within this entity (game type), an admin can make all the CRUD operations. Figure 5.6 presentes the layout that allows creating a new game type, requiring the selection of an avatar and a kind (e.g. Brands).

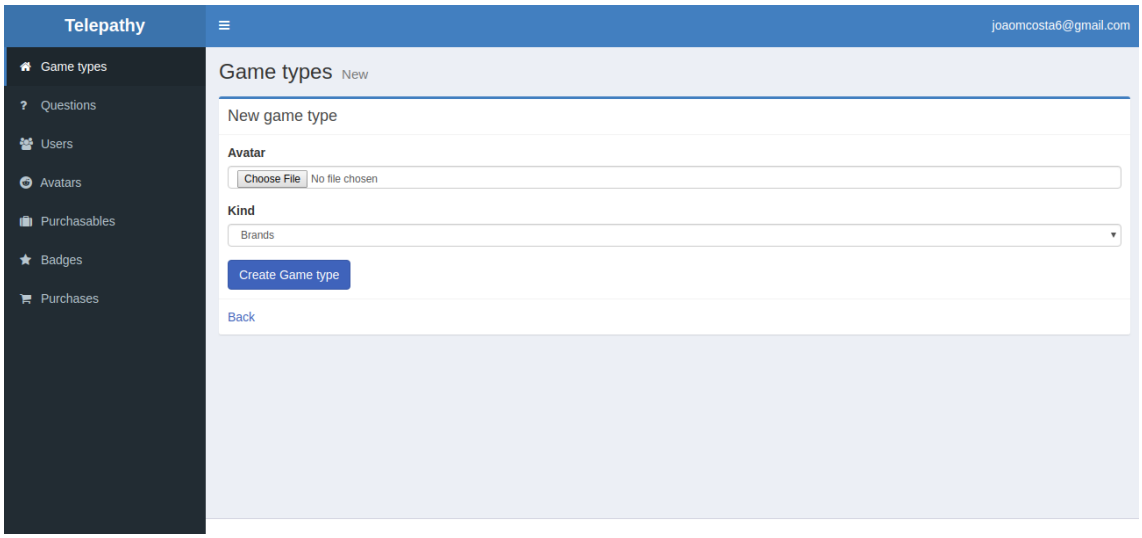



Figure 5.6: Create game type admin panel

### Questions

An admin must have the ability to add questions to a specific game type. To make that feasible, a "Questions" section was added to the admin panel, where they can make all the CRUD operations. Figure 5.7 presentes the index page of the Questions. As mentioned before, a question belongs to a game type and has a text field, which is the question itself.

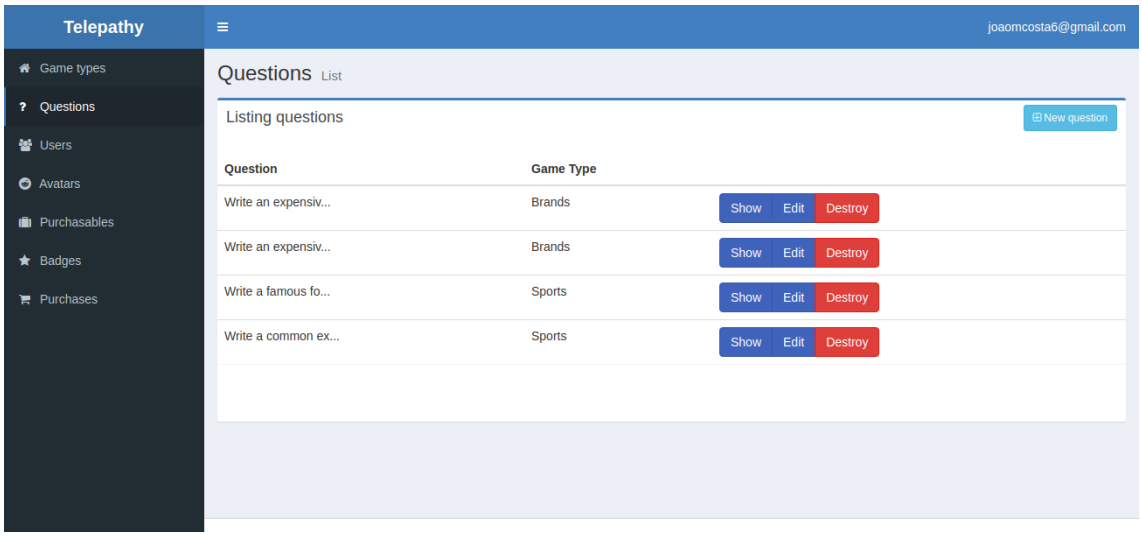



Figure 5.7: Questions admin panel

### 5.3.3 User

This section presents the implemented functionalities on account of the user requirements identified in section 4.4, and others deemed essential for the application to work correctly. The functionalities are presented by the order of their development, as pointed out in Figure 5.4. In each functionality is explained the development process that resulted in the requirement completion, the problems encountered (if any), and the resulting mobile game layouts. Once again, it is important to refer that the final design is not yet integrated in the mobile game; HYP's designers will complete it at a later stage since it was not considered essential for the core objective of the current project, which was to develop an MVP. However, the author made and effort to make the components legible, to be able to reach as many conclusions as possible in the usability tests.

#### Authentication

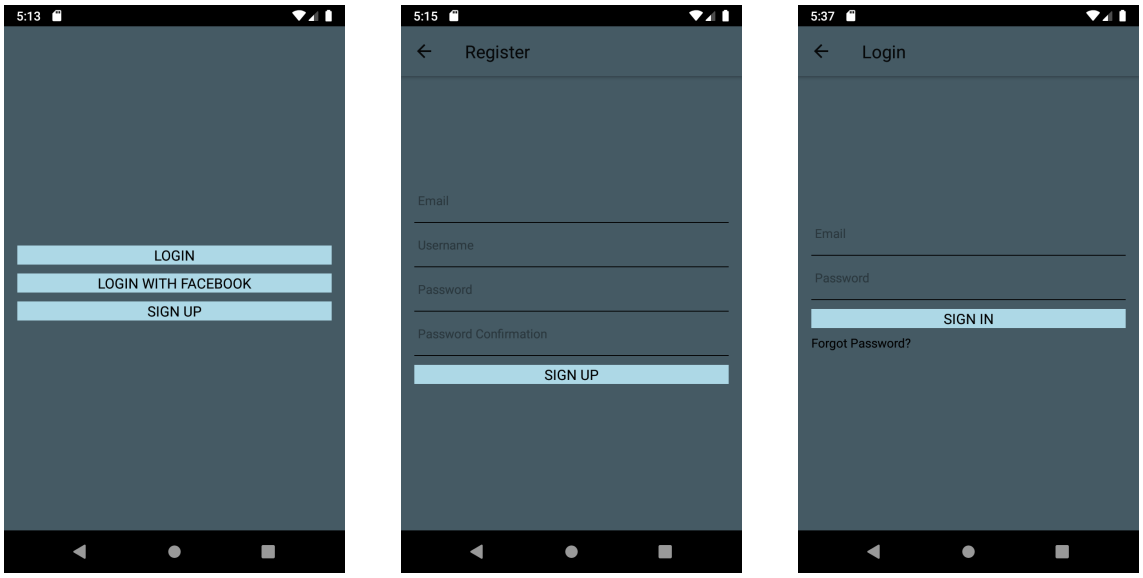
After configuring the React Native project, the first step was to implement user authentication. The following requirements are included in user authentication: register, login, Facebook login, password recovery, and log out. These are identified as "Must Have" requirements since user identification is necessary for users to access their information and keep the information secure.

Register, login, and logout authentication requirements make use of endpoints that are generated from a Ruby Gem called "Devise Token Auth" [1]. This gem, when installed, automatically adds fields to the user model (such as email, password, among others), adds many validations (e.g., checking if a user email is unique, when trying to register), provides methods (endpoints) for these authentication requirements, refreshes the access tokens on each request, and expires them in a short time, making the app more secure.

Figure 5.8 presents three layouts: the main screen, the register screen, and the login screen. The main screen is the first screen that users see when they first open the app to be able to authenticate. In the register screen, users have to insert the necessary information to register in the app. This information must be valid; otherwise, the app can throw

[1] See more: `https://github.com/lynndylanhurley/devise_token_auth`

different errors such as: "Username already taken", "Password too short", "Password doesn't match Password Confirmations", among others. In the login screen, users must type the information of an already existing account to be able to enter the app; otherwise, the following error is shown: "Invalid Credentials".

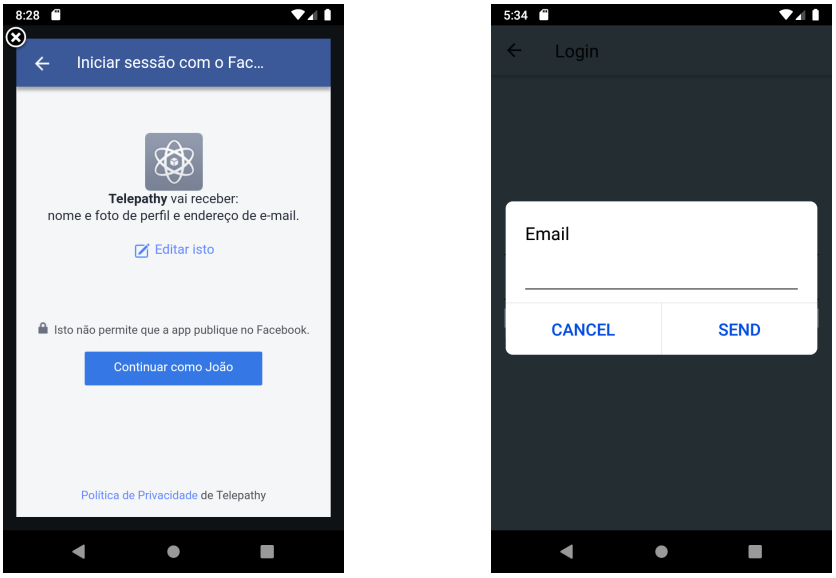



(a) Main screen (b) Register (c) Login

Figure 5.8: Authentication layouts

To use the Facebook API for user login and register, the author had to first register the app on Facebook developer's website [2]. After registering the app, an id was made available to use on the client-side, to allow the app to connect with the Facebook API. When a user uses the app's Facebook login for the first time, Facebook asks the user for permission to share their information with the app (Figure 5.8(a)). This information includes the user's email and name since these are necessary fields to create a new user account in the project server. After accepting, the Facebook API will return a token. Subsequently, that token is sent to the project server-side to confirm that the token indeed corresponds to a Facebook account. After confirmation, the user's Facebook email and name are retrieved (using the token). If the user does not yet have an account, a new one is created using the user's Facebook email and name (username generated based on name). Finally, a JSON response is sent to the mobile app with the freshly created information. This information is saved in the redux store and the user is redirected to the Home Page, and can then make use of the other mobile game functionalities.

[2]See more: `https://developers.facebook.com/`




(a) Facebook login (b) Recover password

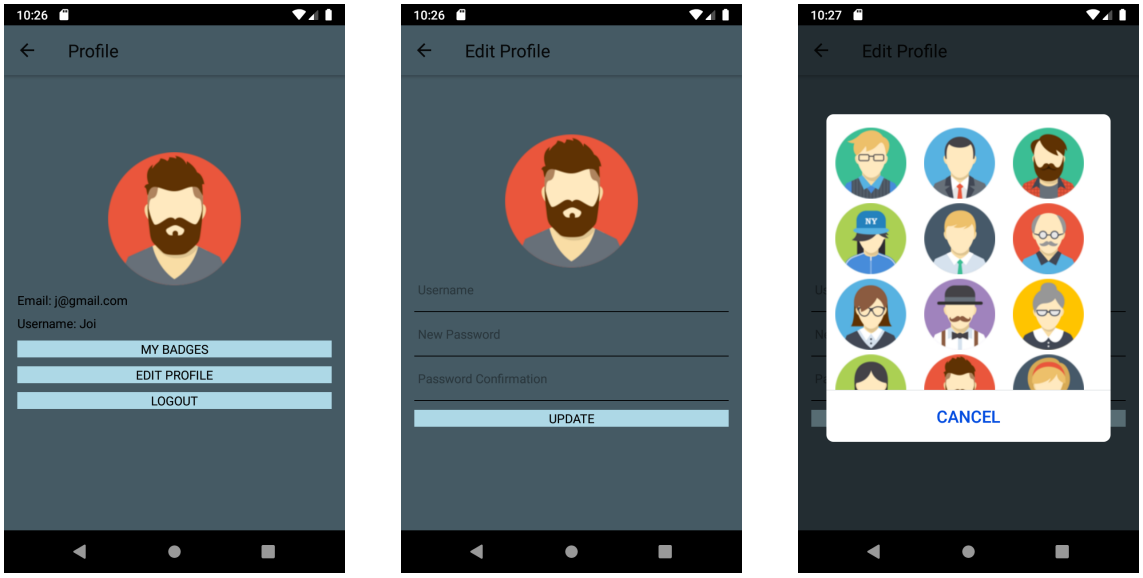
Figure 5.9: Facebook and recover password authentication layouts

Clicking "Forgot Password?" in the login screen allows to recover the password (Figure 5.8(c)). After clicking, a pop up appears in which the user can insert the email, as represented in Figure 5.9(b).

After inserting the e-mail and clicking "Send", a request is sent to the server, including the e-mail in the body of the request. On the server-side, if there is a user with the same e-mail as the one inserted, a new random password is generated and saved in the database. Then, an e-mail is sent to that user e-mail containing the new generated password to login. If the user wants to change this password, they can do it in the "Edit Profile" screen, which will be presented afterward.

### View and Edit Profile

After developing authentication requirements, the author started developing the user profile. As presented in Figure 5.10(a), the profile screen shows the user some of their personal information, which includes their username, email, and avatar. After clicking in “Edit Profile,” the user navigates to the screen presented in Figure 5.10(b), where they can change their username, password and avatar (Figure 5.10(c) if the server accepts their input. If the user inserts invalid information, they will receive a server error.

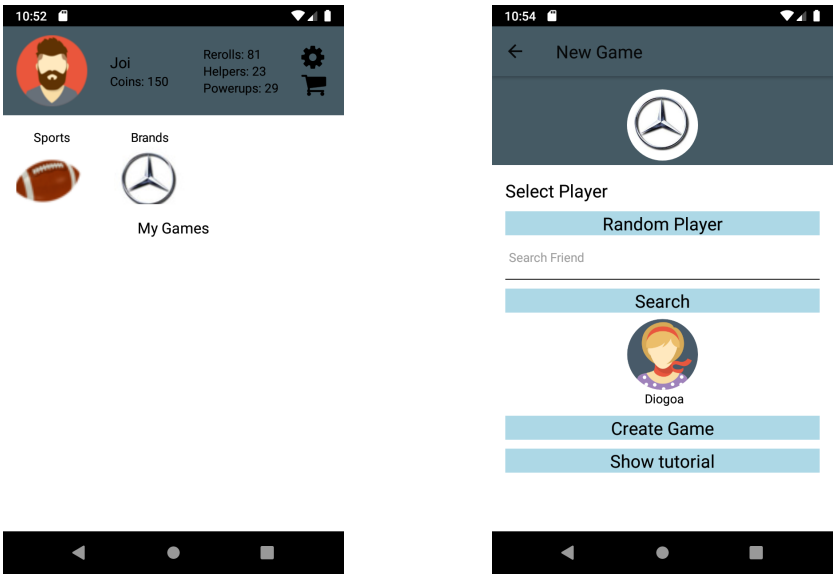



(a) Profile (b) Edit profile (c) Choose avatar

Figure 5.10: Profile layouts

### Create Game, View Games List and Remove Game

To get the "Create Game" requirement completed, firstly were implemented the models, controllers, and views regarding the game types and the user games on the server-side (since a user needs to choose a game type before creating a game). After adding game types in the admin panel and implementing what was necessary on the client-side (such as layouts, requests, and reducers), choosing a game type became available in the Home screen (Figure 5.11(a)).




(a) Home screen (b) Create game

Figure 5.11: Home and create game layouts

After selecting the desired game type in the Home Screen (e.g., Brands), the user navigates

to the "Create Game" screen (Figure 5.11(b)). To create a game, the user has two options: play with a random user or search for a specific one. If they choose the first option, they should click on the button "Random Player." After doing so, a request is sent to a specific server endpoint, which responds with a random player information (id, avatar url, and username). This player is then displayed on the current screen of the mobile app, using the received information (Figure 5.11(b)). The user can continuously click on "Random Player" until they get the desired player. If a user wants to search for a specific player, they can insert their email or username in the search input field. After inserting it and clicking "Search," the server responds with the users matching that input (maximum 5), which are displayed in the mobile app. The user can then choose one of them to play with (as displayed in Figure 5.11(c)). Then, they must click in "Create Game," and a new game is created between the two users within the chosen game type.

After creating the game, the user navigates to the "Answer Question" screen (which is going to be presented in the next subsection). However, the user can go back to the Home screen, where they can see the freshly created game (Figure 5.12).

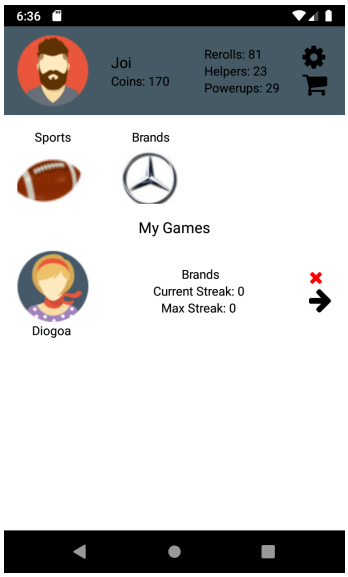



Figure 5.12: Home screen after created game

The user can also remove a game with another player by clicking in the remove button at the top far right of that game. After clicking, the user is asked if they want to remove that game. If they answer "Yes", the game is removed from the database, and neither of the players will be able to see that game on their game's list.

### Submit and Guess Answer

After clicking the "Create Game" button, the server creates a game between both users, picks a random question within the chosen game type, and sends it to the user that created the game. The mobile game saves the received information in the redux store, and the user navigates to the "Answer Question" screen (Figure 5.13)

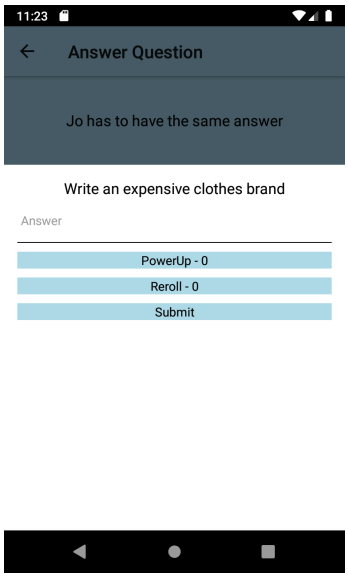



Figure 5.13: Answer Question

Before submitting the answer, the user can re-roll the question as many times as they want and/or increase the number of coins to win in that specific round (as long as they own enough re-rolls/powerup coins). After clicking on the "Reroll" button, a request is sent to the server and the last responds with a new question within the selected game type, which replaces the previous. If the user decides to click in the "PowerUp" button, both players can win an additional 30 coins if they win the round. When the player submits their answer, they navigate to the Home screen and need to wait for the other user to play.

The other player is notified that it is their turn to play. If they open the mobile game (or are already in it), the game will be displayed in their game list. Clicking on the game, the user navigates to the "Guess Answer" screen (Figure 5.14).

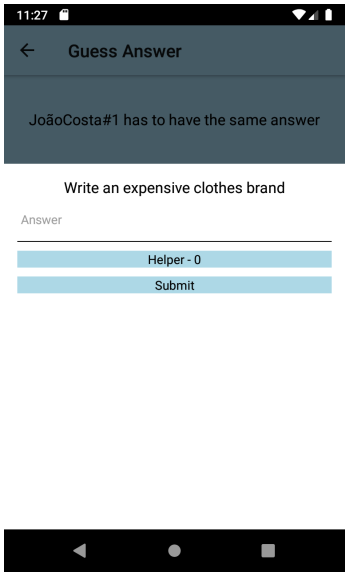


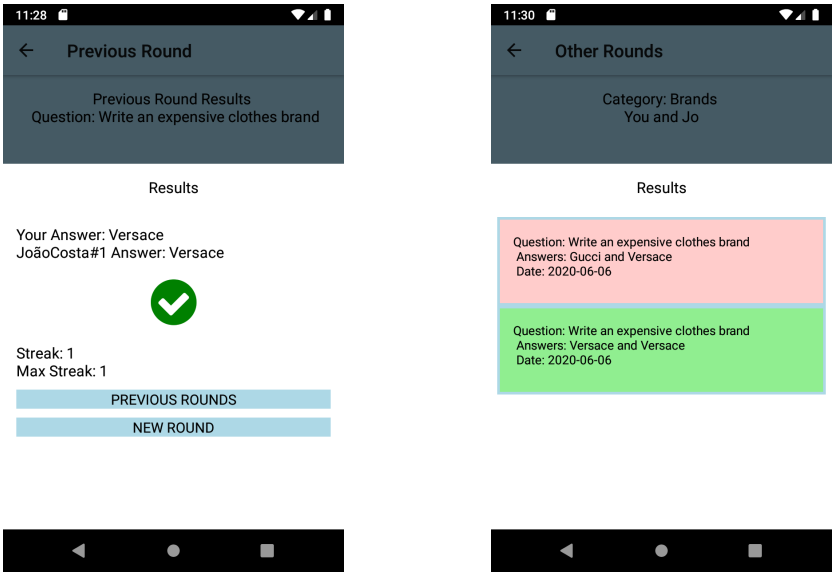

Figure 5.14: Guess Answer layout

On this screen, the user has to guess the other player's answer, as mentioned in the System Description chapter. If a player owns power-up helpers, they can continually click on the button "Helper" (for as long as these are available) to obtain letters included in the word(s)

of the other player's response, which will appear in the answer input field. Every time a player uses a power-up helper, each additional letter must be saved in the server database. Otherwise, if that user decided to log out when it is their turn to guess the answer, they would lose the current state of the answer.

### Game Statistics

After submitting the answer, both player's answers are compared in the server. These answers do not need to be exactly the same to win that round. To compare both strings is used a Ruby Gem that allows using the *Cosine similarity metric* [3] on the server-side. This algorithm allows measuring how similar two strings are. If the similarity between both strings is higher than a certain value (0.9 in a range of 0 to 1), the round is defined as won, both players win 20 coins (if powerup coins were used, they win 50 coins), the streak between the players in the current round increases, and the max streak is increased if it overtook the current max streak. If their answers do not match, the round is defined as lost, neither of the players wins coins, and the current streak goes to 0. This information is then sent back to the mobile game, it is saved in the redux store, and the user navigates to the "Previous Round" screen. The other player can also view the last round statistics if they click on that game in their games list. As represented in Figure 5.15(a), this screen shows both player's answers, displays if they won that round, and the current and max streak reached within that game type.




(a) Previous round (b) Other rounds

Figure 5.15: Previous round and other rounds layouts

By clicking in the button "Previous Rounds"(in Figure 5.15(a)), the user navigates to a new screen which shows a list of the previous round results (Figure 5.15(b)). Each of the finished rounds shows the question, the given answers, the date in which the round ended, and whether it was won (green if they won, red otherwise). If the player clicks in "Previous Round" button, a new round is created, a new question is generated within the selected game type, and the previous process can be repeated.

[3] See more: `https://github.com/mhutter/string-similarity`

### View and Buy Shop Products

The functionalities that were developed next were viewing and buying shop products ("Must Have" requirements). After creating the necessary models, controllers, and views related to the shop products in the server-side, the author added some shop products in the admin console. Each one of these has a kind (reroll, powerup or helper), a price (in coins), and a quantity.

After implementing the necessary code on the client-side, a user could already navigate to the Shop screen through the Home screen. As shown in Figure 5.16, this screen displays all the possible items a user can purchase.

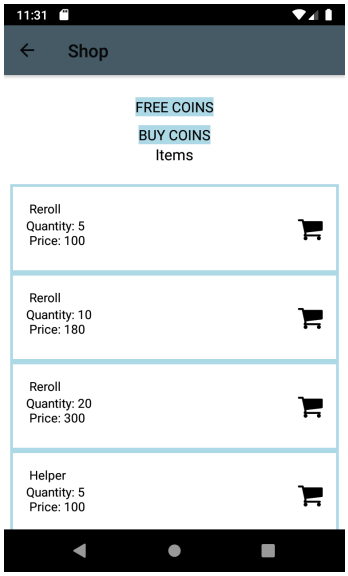



Figure 5.16: Buy items

If a user decides to buy an item, they must click on the shop icon at the far right of the desired item. Afterward, a request is sent to the server containing the product id in the body; if the user has enough coins to buy that specific item, they receive the item quantity, and a success response is sent back to the client-side. In case the user does not have enough coins, the server renders an error message which is then displayed in the mobile app.

### Notifications

As noted in the State of the Art chapter, notifications can be one of the most critical functionalities in asynchronous multiplayer online mobile games, since they allow users to obtain immediate information related to the mobile game. In this case, notifications let users know when it is their turn to play and when a round finishes.

The libraries selected for notifications were the expo push notifications library for the client-side and the expo-server-sdk-ruby for the server-side [4]. When the user logins in the mobile app, a request is sent to the expo API, which returns a token. This token is a unique identifier that allows sending notifications to this user, on that particular device, if the mobile app is installed. A user can have multiple notification tokens since they can be logged in on multiple devices. This token is then sent to the project server, and it is saved in the database for that specific user.

[4]https://docs.expo.io/guides/push-notifications/

Figure 5.17 presents a diagram that allows understanding how notifications are sent.

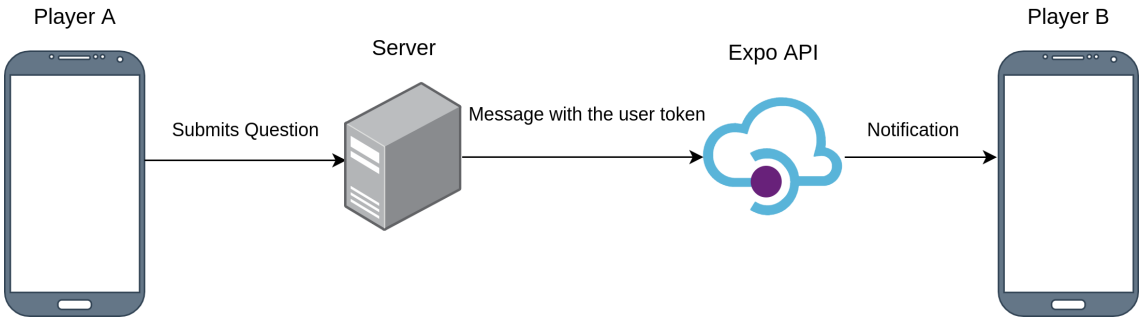
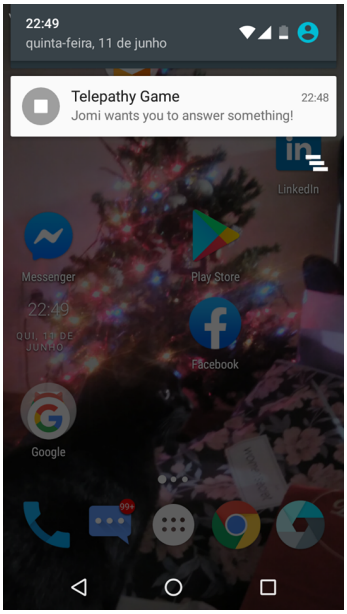



Figure 5.17: Guess Answer layout

In a scenario where player A is playing with player B, after player A submits a question and answer, a request is sent to the server. On the server-side, adding to all the operations involved after submitting an answer, a function is called that handles sending the notification to Player B. This function receives the desired message and all of Player B's notification tokens. Then, for each of the Player B's notification tokens, the expo API will handle sending the message to the required devices. Figure 5.18 presents the notification that Player B receives after Player A submits a question and answer.




(a) User receiving notifications


(b) Activate/deactivate notifications

Figure 5.18: Notification related layouts

It is also important to point that, after a user logouts of the mobile app, it is necessary to remove the notification token that belongs to that device from the database. Otherwise, they would still receive notifications even though they logged out.

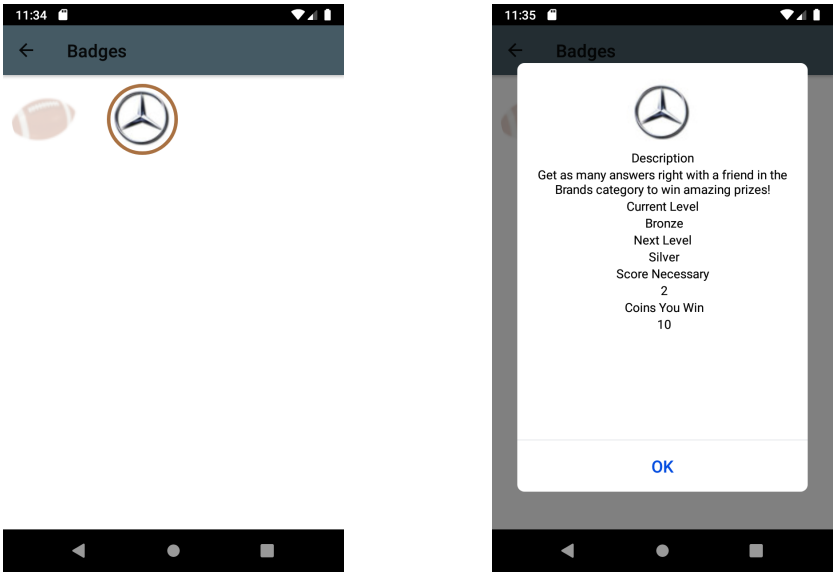
If the user desires, they can deactivate/activate notifications by navigating to the Settings screen (Figure 5.18(b)).

### View Badges

Adding badges (or achievements) to the mobile game was identified as a "Must Have" requirement since that is an important feature to increase user engagement and goal commitment, which can result in using the mobile game more actively. In addition, the visual elements of the badge itself and the included descriptions regarding the goal and how to unlock a badge can give rise to intrinsic user motivations [73].

Two different badges were created: one regarding the sports category and another regarding the brands category. Each type of badge has different levels (e.g., bronze, silver); a user must reach a certain streak with another player within a category to win a new level. For example, for a user to reach bronze in the brands' badge, they must win two rounds in a row (streak) in that specific category. If they do so, they earn 20 coins.

A user can access their badges by navigating to their profile and clicking "My Badges." After doing so, a request is sent to the server, and the last responds with the information of all badges and also the current user levels on those badges. Afterward, the user can see the badges (Figure 5.19(a))




(a) Badges (b) Badge information

Figure 5.19: Badges and badge information layouts

Different border colors are used to distinguish the different user levels for each one of the badges, since it is a common approach used by other mobile games (e.g., brown border for bronze). If the user clicks in one of the badges, a pop up appears with the information related with that badge (Figure 5.19(b)), which includes the badge description, the current user level, the next level, the score they need to reach next level, and the coins they earn in case they reach it.

### Free Coins

As mentioned before in the System Description chapter, a user can watch an advertisement to win 20 coins. For this functionality, a decision was needed regarding which advertisement network to use. An advertising network is a company that connects advertisers to

applications that want to host advertisements and make a profit from them [2]. To make this decision, various factors were taken into consideration:

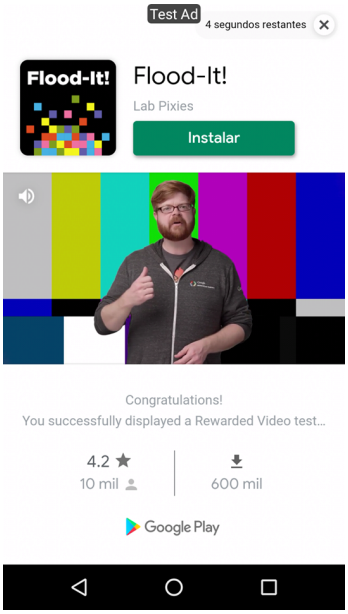
- **Libraries available and documentation**. Not all advertisement networks have libraries available to use with React Native. Additionally, even if there is a library available, several networks contain poor documentation and/or do not have an active maintenance;
- **Size of advertisement network**. Bigger ad networks have more options when trying to match an ad based on an app content. Better targeting can increase the user engagement and result in a higher number of clicks [74];
- **RPM**. RPM or revenue per thousand impressions are the estimated earnings that can be obtained for every 1000 impressions received (for every 1000 ads displayed) [72];
- **Fill Rate**. Fill rate is the percentage of ad requests that get filled by the ad networks. A fill rate of 0% means, in this case, that every time a user tries to watch the video to win 20 coins, no ad is displayed, since the mobile app gets no response from the ad network [92].

The final decision was to use Google's advertisement network: AdMob. Other than being one of the most popular ad networks and having a library with good documentation [5], AdMob has a feature named AdMob Mediation, that allows apps to be server ads from multiple sources, including the AdMob Network, third-party ad networks, and AdMob campaigns. This feature helps maximize the fill rate and increase monetization because it allows sending ad requests to multiple networks and fins the best ad network to serve ads and to make a profit [24].

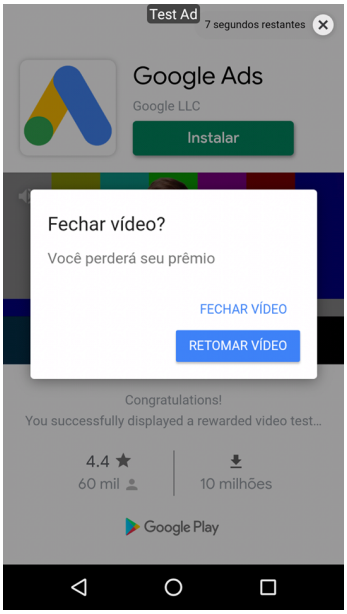
After registering the app in the AdMob website [6], the author was able to choose the type of ad that he wanted to include in the mobile app (in the AdMob website). "Rewarded video ads" were used, since the goal is for the user to be rewarded after seeing an ad video. "Rewarded video ads" are full-screen video ads that users can watch in exchange for in-app rewards (coins in this case) .

A user can watch an advertisement by navigating to the shop and clicking in the button "Free Coins" (Figure 5.20(a)). Afterward, a random video ad is displayed. If the user watches the add until the end, they are awarded 20 coins. Otherwise, if the user decides to close the ad, an alert appears warning the user that if they close the ad, they will not be awarded the prize (Figure 5.20(b)).

[5] https://docs.expo.io/versions/latest/sdk/admob/
[6] https://admob.google.com/home/




(a) Ad display



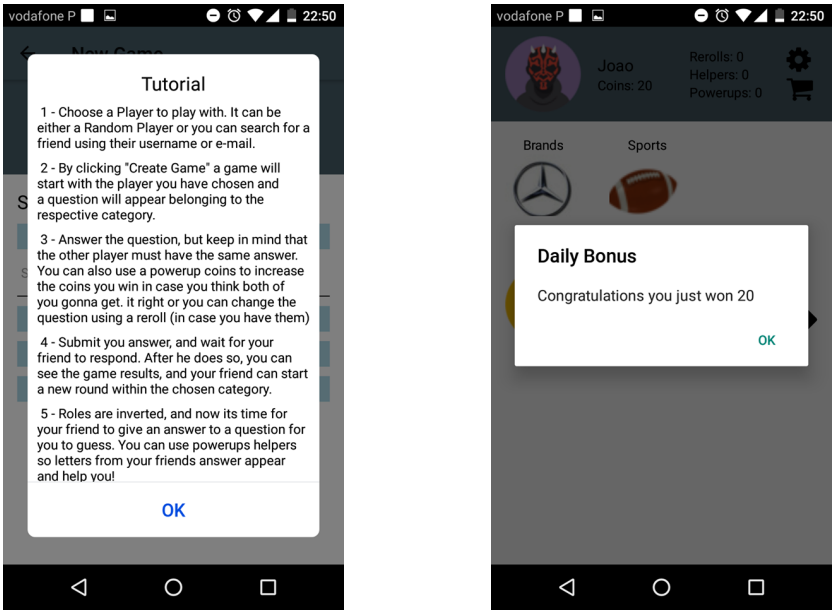

(b) Ad closed

Figure 5.20: Advertisment layouts

### Tutorial and Daily Bonus

After finishing the "Free Coins" requirement ("Should Have"), the author implemented two "Could Have" requirements: the tutorial and the daily bonus. The tutorial, which is beneficial for users that do not understand immediately how the game works, only required implementation on the client-side. To view the tutorial, a player must navigate to the "Create Game" screen and click in "View Tutorial" (Figure 5.21(a)). After that, a pop up appears displaying a set of instructions meant to give the player a deeper understanding of the game. This tutorial is also displayed when a user registers in the mobile game, and navigates to the home screen for the first time.

The daily bonus consists of a user winning 20 coins every time they make use of the app since the day before. Having a daily bonus can be beneficial for the company since it can exalt user motivation to use the app more often. Every time a user is about to receive the daily bonus, they receive an alert on the mobile app (Figure 5.21(b)).




(a) Tutorial (b) Daily bonus

Figure 5.21: Tutorial and daily bonus layouts

### Buy Coins

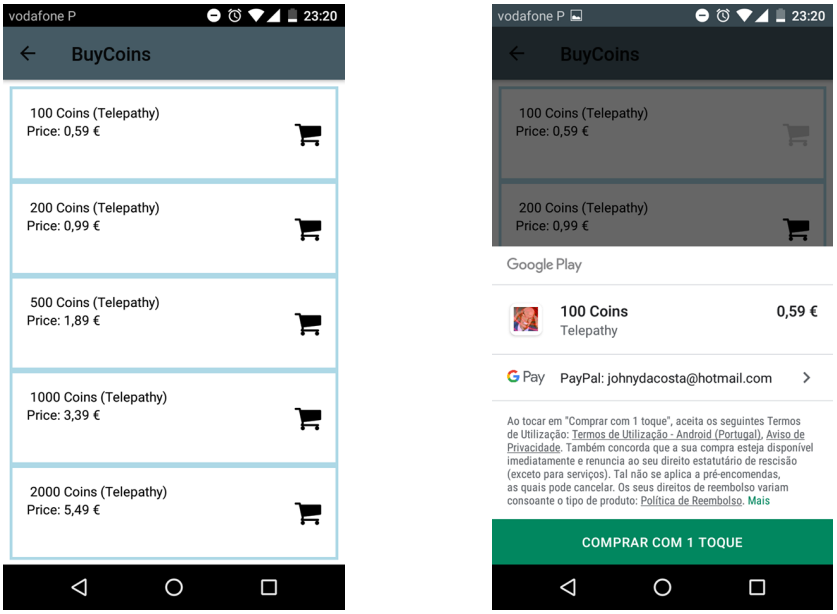
The "Buy Coins" requirement was the one that took the author the longest to complete (comparing with all other requirements). For an android user to buy coins using the Google Pay system, the mobile app needs to be released in Play Store. For this purpose, the author needed to publish the app in the Google Play Console website [7] in at least a closed test track (which means that only selected people can download the app from the Play Store). The app is not immediately available after publishing it in google play console because it first needs to be reviewed by Google. As mentioned in the Google support website, the review time usually takes a few hours for the app to be live in a testing track [8]. Due to the current pandemic situation (COVID-19), google issued a warning informing that the review times could take longer than seven days [27].

The concern here was not only that it could take longer for the app to be published, but also the fact that the author implemented this functionality without the possibility of testing it (to test in-app purchases using Google Pay, the app needs to be published in Play Store). In a first attempt, the app took six days to get released and it did not work adequately. For this reason, the author had to find what was causing the error and publish a new version of the app to Google Developers Console, which entailed a new review period. Fortunately, review times for updates are shorter than for the first release. The app required four updates to make the "buy coins" requirement completed; each update took less than two days to be reviewed.

To buy coins, the user must navigate to the shop and click in "Buy Coins". After this, they navigate to the "Buy Coins" screen (Figure 5.23(a)) in which a list of possible products for purchase is presented. Each one of the options has the number of coins and their price. These products were all inserted in the Google Developer Console, in the app section.

[7] https://play.google.com/apps/publish
[8] https://support.google.com/googleplay/android-developer/answer/3131213?hl=en

If a user decides to buy coins, they must click in the shop icon at the right of the desired product. After doing so, the user can proceed with the payment using their google account (Figure 5.23(b)).




(a) Buy coins (b) Payment

Figure 5.22: Buy coins layouts

After purchasing a product, a request is sent to a server endpoint containing the purchase receipt data, to validate the purchase using receipt validation. Receipt validation is a security mechanism whereby the payment platform validates that an in-app purchase indeed occurred as reported, to fight against fraudulent revenue events [31]. In the server side of the project, a request containing the receipt data is sent to the payment platform to make sure the purchase indeed occurred. The user receives their coins after this confirmation.

## 5.4 Security

### 5.4.1 Mechanisms Utilized

This section identifies the mechanisms used to validate the security quality attributes specified in section 4.5.4, some of which already mentioned in the previous section. In order to protect the system against unauthorized access and fraudulent requests, the following procedures were used:

- **Use of Secure Sockets Layer (SSL)**. SSL is a cryptographic protocol to provide security over internet communications, with the benefit of protecting against man-in-the-middle (MitM) attacks that are trying to intercept communications between the server and the client side [53]. This allows to secure any data transmitted between server and the mobile app.
- **Access tokens changing on each server request and expire in short time**. In order for a user to make successful requests to the server, they need to send their last tokens in the header of the request; otherwise, the request will fail. Every time

a user makes server requests, a new access token is generated for that user and sent back to the mobile app to use in the next request, making it harder for attackers to access the user's information.

- **Store encrypted password in database**. The user password is the first line of defense against unauthorized access, and it is very important keeping it secure. User passwords are stored hashed in the database, to protect against attackers that somehow gained access to the database (e.g. through SQL injection);
- **Use of receipt validation**. As specified before, the use of receipt validation is a good mechanism to protect the system against fraudulent revenue events.

### 5.4.2 GDPR

Despite the fact that the mobile game was not yet release for public, it is crucial that the app respects the GDPR law when it does get released. As mentioned in the official documentation [17], General Data Protection Regulation (GDPR) "is the toughest privacy and security law in the world. Though it was drafted and passed by the European Union (EU), it imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the EU. The regulation was put into effect on May 25, 2018. The GDPR will levy harsh fines against those who violate its privacy and security standards, with penalties reaching into the tens of millions of euros."

The mobile game under development collects and utilizes users' personal data. Therefore, there are rules that need to be respected by the company [16], namely:

- **Asking For User Consent**. Every app must request a user to collect, use, and transfer personal data. In almost every case, consent request is given in the step of creating an account (normally through a checkbox).
- **The Right to Access Data**. If a user requests their data, the company needs to honor that request and provide the user information within one month or two months (two months in case the requested data is too large to fulfill in one month).
- **The Right of Restriction of Processing**. If a user asks the company to stop processing their data, they must comply immediately.
- **The Right to Rectification**. If a user finds their data to be inaccurate or incomplete, they have the right to have it changed.
- **The Right to Data Portability**. Users have the right to transmit their data to another mobile app.
- **Right to Be Informed**. It is the right of users to be informed about who is collecting their data and for what purposes.
- **Right to Erasure**. Users have the right to erase their data, in case they want to.

These aspects need to be respected for the commercial version of the application.

The next chapter focuses in the testing phase of the project.

This page is intentionally left blank.

# Chapter 6

# Testing

Software testing is defined as the activity of checking if the current software results match the expected results, to ensure that the software system is defect free and that a quality product is delivered to the customers [51] [54].

This chapter focuses on the testing techniques and tools employed by the author. First, functional testing is presented, which includes unit and integration testing in the server-side and end-to-end testing the client-side. Afterward, some conclusions are drawn about the load testing, which shows the different response times depending on the number of requests using the current cloud characteristics. Finally, the usability testing and their results are presented.

## 6.1 Functional Testing

Functional testing has the purpose of validating the software system regarding the functional requirements/specifications [45]. This chapter describes the approach followed to validate the functional requirements of the system.

### 6.1.1 Unit and Integration Testing

Unit testing is where individual units/components of the software are tested to make sure they perform as desired [62]. Unit testing was executed manually during the development phase, and also at the end, mainly through rails console. Rails console (command line) allows the developer to interact with the application and can be useful since it can be used to debug and make experiments within the application's data [28].

Afterward, the author started the integration testing. Integration testing is where different modules/components are integrated to make sure that they work as expected [47]. Integration tests where performed in the different actions (endpoints) in each one of the user controllers in the server-side of the project, to secure that when a user sends a server request, they receive the expected response.

The tool used to write the integration tests was RSpec [1]. RSpec is a unit/integration test framework for the Ruby programming language, which focuses on the "behavior" of the application being tested. All the test cases made with RSpec were white-box tests since

[1] https://github.com/rspec/rspec-rails

the author knows the internal structure and implementation details of the code.

To understand how these type of tests work, Figure 6.1 presents a simple integration test which tests one of the server endpoints. The endpoint tested is called every time a user finishes watching an ad.

```
21  describe "PUT #reward_free_coins" do
22    it "returns a status message" do
23      before_coins = User.first.coins
24      random_coins = rand(1..100)
25      put :reward_free_coins, params: {coins: random_coins}, as: :json
26      json = JSON.parse(response.body)
27      expect(json['success']).to eq(true)
28      expect(json['data']).to eq(before_coins+random_coins)
29      expect(User.first.coins).to eq(before_coins+random_coins)
30    end
31  end
```

Figure 6.1: RSpec testing

After a user finishes watching an ad, a request is sent to the "reward_free_coins" endpoint to increase the user coins. The test case illustrated in Figure 6.1 evaluates if the server behaves as expected when that specific request is made. The code on line 23 saves the user coins in a variable before the request; the following line generates a random number of coins that are used to simulate the request on line 25. After this, lines 27, 28, and 29 verifies if the request executed as expected. Line 27 checks if the response sent to the user was successful (without errors). Line 28 checks if the right amount of coins was sent in the body of the response, and line 29 checks if the right amount of coins was saved in the database regarding that specific user.

To check the output of the tests, it is necessary to run the file where the test belongs with RSpec, and it will output the results (Figure 6.2).

```
joaocosta9@joaocosta9 ~/telepathy_server testing rspec spec/controllers/api/v1/user_controller_spec.rb
..........

Finished in 0.48104 seconds (files took 2.78 seconds to load)
10 examples, 0 failures
```

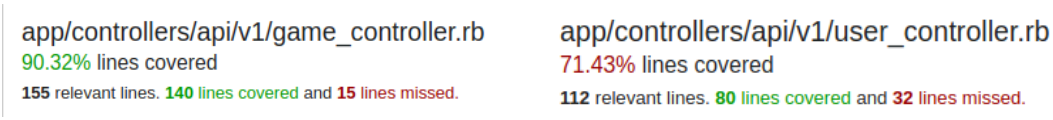
Figure 6.2: RSpec output

If some value is not as expected, RSpec will output that value and provide information regarding what went wrong. The test demonstrated in Figure 6.2 is a simple one since the endpoint being tested does not have many possible paths. However, when performing tests of higher complexity like the update endpoint - required when a user wants to update their username, password or avatar -, there can be a lot more paths since the test can throw multiple error messages such as "Username has already been taken" or/and "Password is too short," among others. In this case, it is essential to check if the user is receiving the expected error messages on the mobile game, depending on their inputs.

To verify if all the paths for the user API were covered, a code coverage analysis tool for Ruby was used: SimpleCov [2]. SimpleCov allows the tester to check the percentage of code covered for specific files and also points out the lines of code that were not covered during the tests. Figure 6.3 displays the percentage of code covered in the files "user_controller.rb" and "game_controller.rb", which are the controllers that handle the

[2]https://github.com/colszowka/simplecov

majority of the user requests.




(a) Game controller (b) User controller

Figure 6.3: Code coverage

The majority of the code in the game controller was covered with exception of some lines of code that are triggered when some internal server error happens (e.g., failing to save something in the database). For the user controller, 71,43% of the code was covered since the Facebook login and the purchase validation endpoints need real values to cover some of their paths, and therefore were not covered during automated testing; the Facebook login endpoint needs a real token related to a real Facebook account, and the purchase validation needs the JSON of a receipt after a user makes a real purchase in the app. Therefore, these two endpoint were validated through manual testing.

All the test cases performed during integration testing are described in Appendix E. Each one of the tables relates to the different test cases made for a specific server endpoint. Each test case has an id, a description, and most importantly the expectations (what is indeed being tested). Even though some of the tests have not been successful in a first attempted, eventually all tests passed.

### 6.1.2 End-To-end Testing

After making sure that every server endpoint was behaving as expected in each possible request, it was time to test if the mobile app was acting accordingly. For this purpose, end-to-end testing technique was used. End-to-end testing is a technique that tests the entire software product from beginning to end to guarantee that the application flow behaves as anticipated, by ensuring that all integrated pieces work together as expected. Its main purpose is to test the user's experience by simulating real scenarios and validating the system and its components for integration and data integrity [44]. To achieve this purpose, a end-to-end React Native library was used: Detox [3]. Detox is an end-to-end grey-box tool for the UI testing of React Native apps [10]. With this tool, the tester can write code that is able to simulate multiple user actions in a simulator, with the purpose of checking if the outcome of these actions are correct. Figure 6.4 presents the code of one of the test cases regarding the user login.

[3]https://github.com/wix/Detox

```
1  describe("Login", () => {
2    beforeEach(async () => {
3      await device.reloadReactNative();
4    });
5
6    it("should login and go to home screen", async () => {
7      await expect(element(by.id("main_screen"))).toBeVisible();
8      await element(by.id("login")).tap();
9      await element(by.id("email_input")).typeText("j@gmail.com");
10     await element(by.id("password_input")).typeText("benfica9");
11     await element(by.id("login_tap")).tap();
12     await element(by.id("login_tap")).tap();
13     await expect(element(by.id("home_screen"))).toBeVisible();
14   });
15 });
16
```

Figure 6.4: Detox test

The code snippet presented above simulates the process of a successful login since a user enters the app for the first time. Each of the different lines from line 7 to 13, represent different actions. For example, line 9 finds the component that has the id "email_input" on the layouts (which is the email input field for login), and types "j@ggmail.com". In this test case, after inserting valid login information and pressing the login button (line 12), is expected for the user to navigate to the home screen (line 13). This test case is not enough to validate that the login is working correctly in the mobile app, since there is also the scenario of a user inserting incorrect login information. In this case, it is expected that the user stays in the login screen and receives an error message.

To check the output of the tests, it is necessary to run the file where the test belongs with Detox, and it will output the results. Just like the integration testing with RSpec, if some value is not the expected one, Detox will display information about what went wrong.

End-to-end testing was done taking on account all possible user actions within the app and checking if the components of the app behave as expected for each test case.

## 6.2 Load Testing

The author proceeded to load testing after finishing the functional testing. Load testing is a type of non-functional testing with the objective of understanding the behavior of the application under a specific load to determine the system's behavior under both normal and at peak conditions [48].

In the final stages of development, the server side code was deployed to the AWS cloud to ensure that users could test the app freely as soon as it was released in the app store in testing mode. The AWS EC2 instance matches the lowest specification instance model (1 vCPU, 512 MB RAM and 20 GB of SSD) [3], since it is enough for at least the testing phase of the project.

Even though the app will not be released to the public during the duration of this project, it is important to study how much traffic the cloud can handle when dealing with a certain number of users and server requests. Such analysis can help determine if a clouding upgrade is needed when the app gets released, keeping in mind that the response time should not exceed three seconds, as stated in the scalability scenario in section 4.5.2.

The tool used for benchmarking was ab apache [4]. The tester can use this tool to simulate a chosen number of requests to the server at the same time, and see conclusive data such as: the number of failed requests, the requests per second that the server is capable of serving, the average response time, among others.

The tests were performed to one of the API endpoints that requires a higher computation effort (user_index), querying the database. Table 6.1 presents the test outputs, when dealing with a certain number of simultaneous requests (concurrency).

| **Requests** | 20 | 80 | 200 | 350 | 400 | 450 | 550 |
|---|---|---|---|---|---|---|---|
| **Concurrency** | 1 | 4 | 10 | 20 | 25 | 30 | 40 |
| **Requests per second** | 3.92 | 10.98 | 14.6 | 14.98 | 14.74 | 14.38 | 14.76 |
| **Average latency (ms)** | 254.9 | 364.2 | 684.9 | 1334.8 | 1696.2 | 2086.9 | 2710.3 |
| **Max latency (ms)** | 271 | 478 | 881 | 2057 | 2494 | 3155 | 4780 |

Table 6.1: Load testing to endpoint test outputs

The maximum number of requests that the server can serve (third line of Table 6.1) never overtakes the 15 requests per second.

As table 6.1 demonstrates, the maximum latency (3 seconds) is reached when the server is dealing with 30 simultaneous requests at the same time. In the author's belief, for the server to be dealing with 30 concurrent requests at any time, the app would need to have a considerable amount of users. Nevertheless, the number of users required to reach this value is unpredictable since users will interact with the app differently (some will use it more often, others less). Therefore, in the author's point of view, the current AWS cloud characteristics may be enough for the initial app deploy but require request monitorization techniques. For example, ensuring close control of the number of simultaneous requests so that admins get notified when these rise unexpectedly. In a scenario where the server starts receiving nearly 30 simultaneous requests, the cloud should be upgraded through vertical scaling (upgrading hardware) and/or vertical scaling (by increasing the number of instances).

## 6.3 Usability Testing

After ensuring that the app was as much bug-free as possible through the functional testing, a usability test plan was prepared. Even though the app's design can make it impossible to draw definite conclusions, the team believes that conducting usability testing at this stage of the project can provide useful user feedback, positively contributing for the future stages of development.

### 6.3.1 Test Participants

The usability test participants were selected taking into account if they matched the target audience for the mobile game. In this case, according to the estimate made by the author and the HYP team, the target audience is people that usually play mobile games with ages ranged between fifteen and thirty years old.

Sources such as [19] and [37], understand that the ideal number of testers should be five or six since it is deemed to be enough to get close to user testing maximum benefit-cost

[4] https://httpd.apache.org/docs/2.4/programs/ab.html

ratio. Therefore, the author gathered five participants to perform usability tests with a physical presence. Four of those participants are software engineering students that often play mobile games, and the remaining person is a non-engineering student that occasionally plays mobile games.

### 6.3.2 Test Procedure

The tests were performed in an undisturbed environment to ensure that there would be no distractions and that all interactions would only take place between the author and the tester. The test started with an explanation of the scope of the mobile game and the objective of the test. Then the participant was asked to sign a consent form that, in summary, attests to its voluntary nature and reinforces the commitment to raise or provide information regarding any concerns that may arise during the test.

Each participant was asked to perform fourteen different tasks:

1. Register in the application;
2. Edit you username and avatar;
3. Explore one of the badges;
4. Buy 10 re-rolls;
5. Try to buy 100 coins, but do not finish the transaction;
6. Win free coins;
7. Deactivate notifications and then activate them again;
8. Create a game in the "Brands" category with a random player;
9. Create a game in the "Sports" category with player "JoaoCosta#1";
10. Go to the last created game, re-roll the question, and answer the question;
11. Upon receiving a notification, check the round results and then check the previous rounds results;
12. Upon receiving another notification, go to the game identified by the notification, use a powerup helper and answer the question;
13. Remove the last game;
14. Logout from the application.

Each task was performed by the order identified above. For each task, the author wrote the number of clicks while also taking notes regarding the participant's interaction with the app.

After performing all the tasks, each tester filled an online satisfaction questionnaire concerning their user experience while performing the above tasks.

Finally, the author asked the testers for improvement suggestions while they were freely exploring the mobile app.

### 6.3.3 Test Results

#### Tasks

As mentioned in the previous section, the measuring metric used was the number of clicks. Table 6.2 shows the usability testing expected number of clicks for each task.

| **Task** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Clicks** | 6 | 6 | 4 | 2 | 3 | 2 | 3 | 3 | 3 | 4 | 2 | 4 | 2 | 2 |

Table 6.2: Usability testing metrics expectation

The table below shows the performance of each one of the users in the usability tests.

| Task / Participant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| User 1 | 6 | 6 | 4 | 2 | 3 | 2 | 3 | 3 | 4 | 6 | 4 | 4 | 2 | 2 |
| User 2 | 6 | 6 | 4 | 4 | 3 | 2 | 3 | 3 | 3 | 4 | 3 | 4 | 2 | 2 |
| User 3 | 7 | 8 | 10 | 3 | 3 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| User 4 | 7 | 7 | 4 | 3 | 4 | 2 | 4 | 3 | 4 | 4 | 4 | 4 | 2 | 3 |
| User 5 | 6 | 6 | 4 | 2 | 3 | 2 | 3 | 3 | 4 | 5 | 4 | 4 | 2 | 2 |

Table 6.3: Number of clicks per user in each task

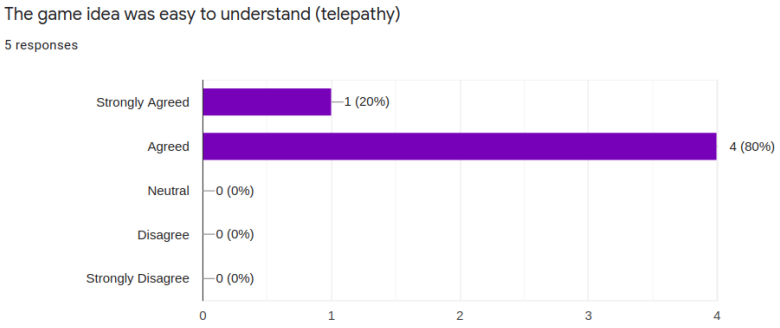
All the tasks were completed successfully by each one of the participants. The majority of the tasks presented satisfactory results. The most discrepant values are marked with pink in the table above.

As illustrated in table 6.3, users had more difficulties in tasks 2, 3, and 11. For user 3, task 2 took longer to complete, as well as two additional clicks, because the user thought that updating the username and avatar required inserting their password and password confirmation. In task 3, user 3 was unable to find their badges in a first attempt (even though the "View Badges" button was displayed previously while going through their profile in task 2), as they first went to the settings screen and then to the shop.

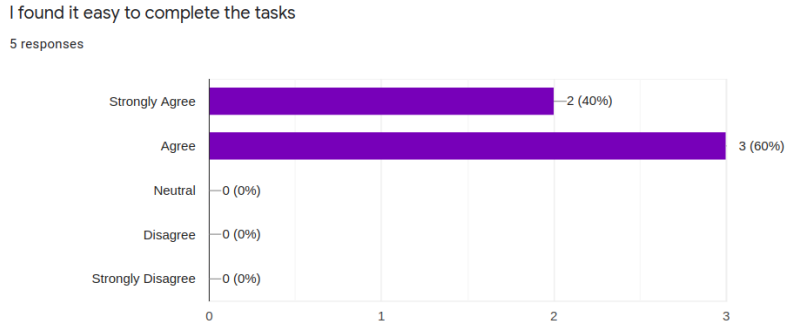
Task 11 was where the test revealed a weaker performance. Upon receiving a game notification, the users though that, after clicking it, they would immediately navigate to that game; however, when clicking in the notification, the users navigate to the home screen, where they should then click in the respective game in their game's list. This brought confusion to some of the test participants, therefore resulting in more clicks and time wasted.

#### Pos-Questionnaires

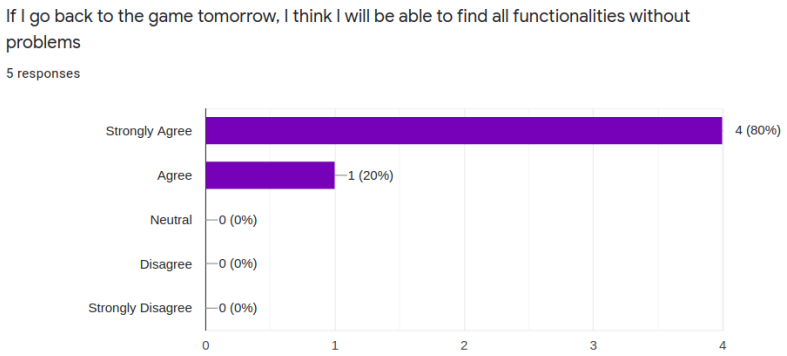
Figures below present some of the most relevant questions from the online questionnaire that the usability tests participants needed to answer.



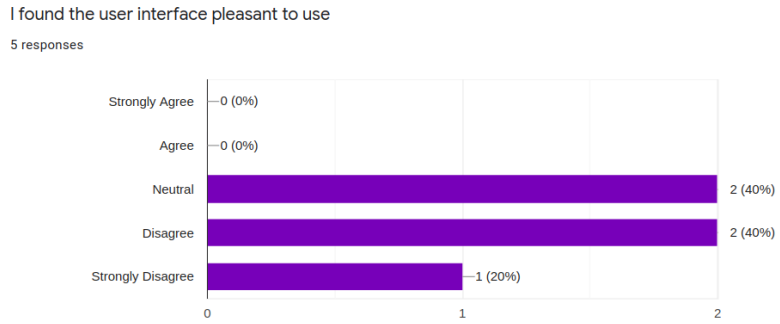

(a) Question one




(b) Question two




(c) Question three




(d) Question four

Figure 6.5: Pos-Questionnarie questions

Overall, after observing the answers above displayed, the users understood the concept of the game when they started playing and also found it easy to complete the tasks. Even though they had a certain ease completing the tasks, they did not find the user interface much pleasant to use, believing that it should be more engaging.

**Recommendations**

At the end of the tasks and questionnaire, the participants provided the following recommendation:

- User 1 thinks that it would make more sense to have the logout button in the settings screen.
- Users 1 and 2 shared that they felt difficulties understanding the purpose of the powerups, while playing. Therefore, in their understanding, providing a way to describe the powerups while playing would be a good idea.
- User 2 believes that after clicking the logout button, the game should ask for confirmation before leaving the account.
- User 2 is of the opinion that clicking in the back button in the home screen should result in leaving the app.
- User 2 and 5 believe that the tutorial should be more engaging, and not only text.
- User 3 thinks that badges should be accessed from the home screen and not from the profile.
- All users believe that after clicking the notification, the user should navigate to the respective game.
- Three users believe that the last round results should be displayed in a more amusing way (e.g., through an animation).
- Two users think that the buttons and input field in the create game screen should be displayed in a different way, since it is confusing.
- All users believe that the overall user interface look should be improved.

### 6.3.4 Test Conclusions

Overall, the participants were able to perform the tasks efficiently and effectively, with some exceptions. Regarding task 11, for example, it became apparent that it would be preferable for the user to navigate to the respective game after clicking the notification, since there was some hesitation on the participants side.

The participants understood the game objective, and four of them enjoyed the game thematics. However, all of them shared the understanding that more work should be put on the user interface look. This feedback did not come as a surprise considering that, as stated before, the look of the user interface was not a priority in this phase of the project.

In summary, the users were able to perform all tasks successfully and, given the fact that the project result is, so far, an MVP, it was possible to obtain some relevant indications for the future commercialization of the solution.

This page is intentionally left blank.

# Chapter 7

# Conclusion

The project proposal put forward by the HYP team required the development of an asynchronous online multiplayer mobile game focused on telepathy as a theme. In order for that project to become a reality, the author had to go through a whole engineering process involving several different stages. The selected methodology was iterative Waterfall. The first semester was divided into a feasibility study, requirements analysis and specification, and design, whereas the second was divided into development and testing.

The project started with the study of the mobile gaming market, followed by a competition analysis of mobile games sharing the features of being asynchronous, multiplayer based, and/or inspired by the telepathy theme. The purpose was to provide the author with insights about the characteristics of these specific types of games and therefore afford guidance in the requirements identification.

The author's inexperience in mobile development frameworks was perceived as an impactful risk that could jeopardize the timely development of the mobile game. To minimize the risk and associated impact, the author persisted in gaining experience in mobile development throughout October and November (2019), using online documentation and developing a simple app (Sample App) employing Flutter and React Native frameworks. At the end of that period, it was decided that React Native would be the framework to use for the development of the mobile game.

Throughout the following phase, the goal was to identify the functional and non-functional requirements for the project. For that purpose, user stories were firstly used to help define requirements, based on the most inherent features according to the State of the Art, on the HYP team's expectations (shared during the initial meetings), as well as on multiple sessions of discussion between the author and the team. After completing the user stories, the system use cases were assembled and then prioritized in a meeting with the HYP team, in accordance with their importance to the project. Afterward, the author identified the quality attributes deemed essential for the system to work correctly.

Subsequently, the system architecture was pointed out to define the structure and behavior of the system and their respective interactions with other systems, followed by the creation of a visual representation of the system's data through an entity-relationship diagram. Afterward, a navigation diagram was developed in order to understand the application screen flow, and low-fidelity wireframes designed, displaying how the planned user interface provides the functionalities.

In the second semester, after a two-week Ruby on Rails learning period, the author started developing the mobile game. The development phase led to the fulfillment of all the require-

ments which were completed in the planned time and resulted in the proposed Minimum Viable Product. Considering that all "Must Have" requirements were successfully completed, as well as the remaining requirements ("Could Have" and "Should Have"), the ToS was achieved.

To truly validate the functional requirements of the mobile game, the testing phase started with functional testing, where three types of testing techniques were used: unit, integration, and end-to-end testing. Even though the system revealed some bugs/errors at first, these were successfully resolved in due time.

As for the security quality attribute, different security mechanisms were employed, namely: using SSL, encrypting the password in the database, introducing user access-tokens that change in each request, and receipt validation. Regarding availability, the current server hosted in Amazon's Web Services promises an availability of 99.99 %. Concerning scalability and performance, the load testing showed that for this mobile game, the current cloud characteristics are enough for the initial mobile game stages, but require continuous monitoring.

Usability tests were conducted to detect possible problems with the mobile game and to provide useful user feedback, which can positively contribute to future stages of the development. Even though the mobile game user interface was not a priority, the author made an effort for the mobile game components to be as comprehensible as possible. Overall, the usability test participants efficiently completed the tasks assigned and provided recommendations to be considered for the future.

The most significant challenges faced by the author arose from having to learn two new frameworks and programming languages from scratch and from the need to provide insightful contributes to the mobile game requirements.

Regarding future work, the conclusions and user recommendations resulting from the usability tests should be evaluated before making changes to the app interface. App's design and accessibility are two essential features to make a good user experience that, other than the functionalities themselves, will contribute to the app's success. In the author's opinion, one of the biggest challenges will lie in finding exciting and engaging content (questions) for each game type. As mentioned in the first chapter of this paper, creativity, diversity, and market orientation will be essential for the development of the ideas to be translated to the game, to keep the players entertained and engaged. Given the multiplicity of game types, it can prove challenging for the team to fashion an endless number of questions and quizzes for each game type. In parallel, poor choices on the content may negatively impact the success of the app. As such, the author believes that it would be an interesting option to insert a functionality in the app by which users can put forward or suggest new questions or quizzes to add to specific game types; if a proposal succeeded, they could be awarded a prize (e.g., coins).

The internship at HYP was a fruitful experience. The author had the opportunity to acquire expertise not only in mobile development but also in Software Engineering since he was under the guidance of experienced developers who encourage and stimulate discussion and the use of software-oriented processes.

# References


[1] 10 moscow prioritisation. https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation. Accessed: 05/12/2019.

[2] Ad network. https://www.muvi.com/wiki/ad-network.html. Accessed: 13/05/2020.

[3] Amazon ec2 pricing. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed: 09/06/2020.

[4] Android v ios market share 2019. https://deviceatlas.com/blog/android-v-ios-market-share. Accessed: 09/10/2019.

[5] Aws systems manager announces 99.9% service level agreement. https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/. Accessed: 14/06/2020.

[6] The c4 model for software architecture. https://www.infoq.com/articles/C4-architecture-model/. Accessed: 20/12/2019.

[7] Case: 7 ux considerations when designing lens hawk. https://baymard.com/blog/ux-considerations-designing-lenshawk. Accessed: 22/06/2020.

[8] Cloud computing with aws). https://aws.amazon.com/what-is-aws/. Accessed: 14/06/2020.

[9] Dart. https://dart.dev//. Accessed: 18/10/2019.

[10] Detox: Superfast e2e react native ui testing. https://bitbar.com/blog/detox-superfast-e2e-react-native-ui-testing/. Accessed: 09/06/2020.

[11] Draw something historical timeline: February 6, 2012 launch to march 21, 2012 $183m sale. http://www.stevepoland.com/draw-something-historical-timeline-february-6-2012-launch-to-march-21-2012-183m-sal Accessed: 19/10/2019.

[12] Ergonomics of human-system interaction — part 210: Human-centred design for interactive systems. https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en, note = Accessed: 24/06/2020.

[13] Flutter : From zero to comfortable. https://proandroiddev.com/flutter-from-zero-to-comfortable-6b1d6b2d20e. Accessed: 15/11/2019.

[14] Flutter hot reload. https://flutter.dev/docs/development/tools/hot-reload. Accessed: 19/12/2019.

[15] Flutter technical overview. https://flutter.dev/docs/resources/technical-overview/. Accessed: 17/10/2019.

[16] Gdpr compliance for apps. https://www.privacypolicies.com/blog/gdpr-compliance-apps/#The_Right_To_Access_Data. Accessed: 22/06/2020.

[17] General data protection regulation (gdpr). https://gdpr.eu/tag/gdpr/. Accessed: 22/06/2020.

[18] How do free apps make money on android and ios in 2019. https://thinkmobiles.com/blog/how-do-free-apps-make-money/. Accessed: 12/10/2019.

[19] How many test users in a usability study? https://www.nngroup.com/articles/how-many-test-users/. Accessed: 19/06/2020.

[20] Hyp. https://hyp.pt/about. Accessed: 10/10/2019.

[21] Libraries. https://www.ruby-lang.org/en/libraries/. Accessed: 09/06/2020.

[22] Load balancer for your elastic beanstalk environment. https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.elb.html. Accessed: 14/06/2020.

[23] A look behind the 'words with friends' iphone gaming phenomenon. https://techcrunch.com/2010/06/10/a-look-behind-the-words-with-friends-iphone-gaming-phenomenon/. Accessed: 19/10/2019.

[24] Mediation. https://developers.google.com/admob/android/mediate. Accessed: 14/06/2020.

[25] Mobile technology, its importance, present and future trends. https://www.finextra.com/blogposting/14000/mobile-technology-its-importance-present-and-future-trends. Accessed: 23/12/2019.

[26] Noumi. https://play.google.com/store/apps/details?id=es.treebit.noumi&hl=en. Accessed: 28/12/2019.

[27] Publish an app. https://support.google.com/googleplay/android-developer/answer/6334282?hl=en.

[28] The rails command line. https://guides.rubyonrails.org/command_line.html. Accessed: 18/06/2020.

[29] Rails getting started. https://guides.rubyonrails.org/getting_started.html#what-is-rails-questionmark. Accessed: 19/12/2019.

[30] React native. https://facebook.github.io/react-native/. Accessed: 15/10/2019.

[31] Receipt validation. https://www.appsflyer.com/mobile-fraud-glossary/receipt-validation/.

[32] Sdlc - waterfall model. https://www.tutorialspoint.com/sdlc/pdf/sdlc_waterfall_model.pdf. Accessed: 10/11/2019.

[33] Trello board. https://trello.com/en/tour. Accessed: 08/05/2020.

[34] Uml use case diagrams. https://www.uml-diagrams.org/use-case-diagrams.html. Accessed: 10/12/2019.

[35] Usability testing. https://www.usability.gov/how-to-and-tools/methods/usability-testing.html. Accessed: 11/10/2019.

[36] Usability testing. http://webservices.itcs.umich.edu/drupal/wwwsig/sites/webservices.itcs.umich.edu.drupal.wwwsig/files/Usability-Testing-Basics.pdf. Accessed: 13/10/2019.

[37] Usability testing – how many users do you need? https://www.uxdesigninstitute.com/blog/usability-test-how-many-users/. Accessed: 19/06/2020.

[38] User stories. https://www.mountaingoatsoftware.com/agile/user-stories. Accessed: 09/12/2019.

[39] What are the non-functional requirements? https://reqtest.com/requirements-blog/what-are-non-functional-requirements/. Accessed: 14/12/2019.

[40] What is a gantt chart? https://www.apm.org.uk/resources/find-a-resource/gantt-chart/. Accessed: 19/12/2019.

[41] What is a relational database management system? https://www.codecademy.com/articles/what-is-rdbms-sql. Accessed: 04/01/2020.

[42] What is an entity relationship diagram (erd)? https://www.lucidchart.com/pages/er-diagrams#section_0. Accessed: 04/01/2020.

[43] What is database ? https://www.geeksforgeeks.org/what-is-database/. Accessed: 04/01/2020.

[44] What is end-to-end (e2e) testing? all you need to know. https://www.katalon.com/resources-center/blog/end-to-end-e2e-testing/. Accessed: 09/06/2020.

[45] What is functional testing? types examples (complete tutorial). https://www.guru99.com/functional-testing.html. Accessed: 14/06/2020.

[46] What is games 'user experience' (ux) and how does it help? https://usabilitygeek.com/the-difference-between-usability-and-user-experience/. Accessed: 24/06/2020.

[47] What is integration testing (tutorial with integration testing example). https://www.softwaretestinghelp.com/what-is-integration-testing/. Accessed: 17/06/2020.

[48] What is load testing in software testing? examples, how to do, importance, differences). http://tryqa.com/what-is-load-testing-in-software/. Accessed: 18/06/2020.

[49] What is mvc architecture? https://www.w3schools.in/mvc-architecture/. Accessed: 19/12/2019.

[50] What is server availability. https://www.igi-global.com/dictionary/performance-analysis-models-web-traffic/26559. Accessed: 12/12/2019.

[51] What is software testing? introduction, definition, basics types. https://www.guru99.com/software-testing-introduction-importance.html. Accessed: 02/06/2020.

[52] When to choose waterfall project management over agile. https://www.smartsheet.com/when-choose-waterfall-project-management-over-agile. Accessed: 02/01/2019.

[53] Why all sites now require ssl (https). https://www.granite5.com/insights/use-https-vs-http-benefits-switching/. Accessed: 14/06/2020.

[54] Why is software testing necessary? http://tryqa.com/why-is-testing-necessary/#:~:text=Software%20testing%20is%20very%20important,the%20implementation%20of%20the%20software. Accessed: 02/06/2020.

[55] Why is wireframing your mobile app so important? https://www.upwork.com/hiring/for-clients/importance-of-wireframing-mobile-apps/. Accessed: 06/01/2020.

[56] Always on the move. 2014. Accessed: 07/10/2019.

[57] The choice of code review process: A survey on the state of the practice. 2017. Accessed: 27/05/2020.

[58] Xamarin vs react native vs ionic vs nativescript: Cross-platform mobile frameworks comparison. https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-compar 2018. Accessed: 20/10/2019.

[59] Iterative waterfall model | software engineering. http://www.geektonight.com/iterative-waterfall-model-software-engineering/, 2019. Accessed: 20/10/2019.

[60] React native vs. ionic vs. flutter: Comparison of top cross-platform app development tools. https://codeburst.io/react-native-vs-ionic-vs-flutter-comparison-of-top-cross-platform-app-development-tools- 2019. Accessed: 20/10/2019.

[61] Where do cross-platform app frameworks stand in 2020? https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/, 2019. Accessed: 20/10/2019.

[62] Doroto Huiziga Adam Kolawa. *Automated Defect Prevention: Best Practices in Software Management*. Wiley, 2007.

[63] altexsoft. The good and the bad of ionic mobile development. //www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/, 2019. Accessed: 19/12/2019.

[64] Ian Bogost. Asynchronous multiplay, 2004. Accessed: 12/11/2019.

[65] Jorge C. S. Cardoso. Usability testing, 2018. Accessed: 12/10/2019.

[66] John B. Goodenough Charles B. Weinstock. On system scalability. 2006. Accessed: 02/01/2019.

[67] Alistar Cockburn. *Writing Efective Use Cases*. Wiley, 2000. Accessed: 05/12/2019.

[68] Existek — Software Development Company. Hybrid vs native app: Which one to choose for your business? :https://medium.com/existek/hybrid-vs-native-app-which-one-to-choose-for-your-business-e51542554078, 2019. Accessed: 19/12/2019.

[69] Tony Davis. What is "maintainable code"? https://www.red-gate.com/simple-talk/blogs/what-is-maintainable-code/, 2009. Accessed: 13/05/2020.

[70] MURIEL DOMINGO. User stories: As a [ux designer] i want to [embrace agile] so that [i can make my projects user-centered]. https://www.interaction-design.org/literature/article/user-stories-as-a-ux-designer-i-want-to-embrace-agile-so-that-i-can-make-my-project 2019. Accessed: 20/12/2019.

[71] Matthew Ford. Ruby on rails: What it is and why you should use it for your web application. https://bitzesty.com/2014/03/03/ruby-on-rails-what-it-is-and-why-we-use-it-for-web-applications/. Accessed: 21/06/2020.

[72] Kean Graham. Rpm vs cpm. https://www.monetizemore.com/blog/rpm-vs-cpm/, 2018. Accessed: 23/05/2020.

[73] Juho Hamari. Do badges increase user activity? a field experiment on the effects of gamification. 2015. Accessed: 23/05/2020.

[74] VISHVESHWAR JATAIN. 5 things you need to consider when choosing an ad network. https://www.adpushup.com/blog/5-things-you-need-to-consider-when-choosing-an-ad-network/, 2015. Accessed: 23/05/2020.

[75] Amit Khirale. Comparison between flutter vs react native for mobile app development. https://www.angularminds.com/blog/article/comparison-between-flutter-vs-react-native-for-mobile-app-development.html, 2018. Accessed: 22/11/2019.

[76] Justin Mifsud. The difference (and relationship) between usability and user experience. https://usabilitygeek.com/the-difference-between-usability-and-user-experience/. Accessed: 22/06/2020.

[77] newzoo. 2017 global games market report. 2017. Accessed: 10/11/2019.

[78] Jakob Nielsen. Usability 101: Introduction to usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/, 2012. Accessed: 11/10/2019.

[79] International Organization of Standardization. Guidance on usability. 1998. Accessed: 10/10/2019.

[80] Tim Parsons. When to use waterfall vs. agile. https://www.macadamian.com/learn/when-to-use-waterfall-vs-agile/, 2019. Accessed: 09/06/2020.

[81] Adam Pedley. How react native works. http://www.discoversdk.com/blog/how-react-native-works, 2017. Accessed: 22/10/2019.

[82] Adam Pedley. How flutter works. https://buildflutter.com/how-flutter-works/, 2018. Accessed: 20/10/2019.

[83] Javier Ramos. Front end web development guide: Web and mobile. https://medium.com/@javier.ramos1/front-end-web-development-guide-web-and-mobile-67c47d674e0f. Accessed: 13/10/2019.

[84] Nayan B. Ruparelia. Software development lifecycle models. 2010. Accessed: 10/11/2019.

[85] Paul Sawers. The story behind ruzzle, the ridiculously popular mobile game gaining 2 million new users per week. https://thenextweb.com/apps/2013/01/10/zynga-mark-2-meet-mag-interactive-the-developers-behind-ridiculously-popular-social-word 2013. Accessed: 24/10/2019.

[86] Alexander Sergeev. User stories and waterfall. https://hygger.io/blog/user-stories-and-waterfall/, 2016. Accessed: 09/12/2019.

[87] Ricardo Silva. Design of a tactical turn-based game for mobile devices, 2016. Accessed: 12/11/2019.

[88] John Snow. Developing a risk management plan, 2010. Accessed: 27/11/2019.

[89] Michael Stanleigh. Risk management... the what, why, and how. https://bia.ca/risk-management-the-what-why-and-how/, 2011. Accessed: 25/11/2019.

[90] Edmonton Sun. 'trivia crack' the top download. https://edmontonsun.com/2014/12/16/trivia-crack-the-top-download/wcm/1f7637d3-7034-458d-9553-19f52892a8f4, 2014. Accessed: 10/10/2019.

[91] Tom Wijman. The global games market will generate $152.1 billion in 2019 as the u.s. overtakes china as the biggest market. 2019. Accessed: 07/10/2019.

[92] Melissa Zeloof. What is fill rate and how can app developers increase it? https://www.ironsrc.com/blog/what-is-fill-rate/, 2018. Accessed: 23/05/2020.

# Appendices

This page is intentionally left blank.

# Appendix A

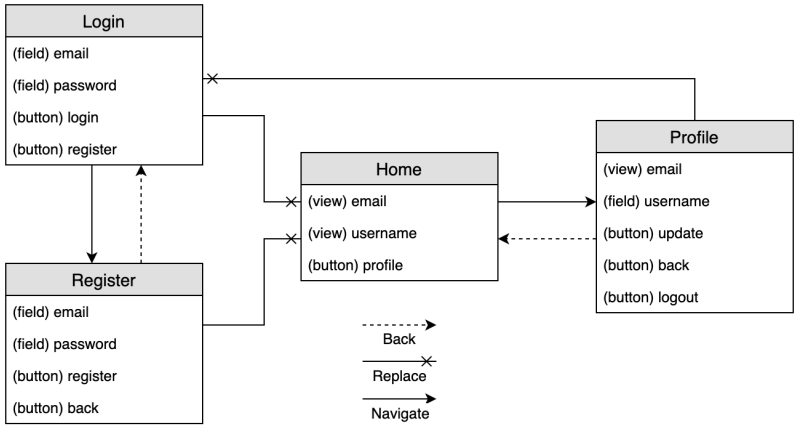
# Sample app

The objective of building this app is to help understand its development complexity on both **React Native** and **Flutter**. Implementing some core concepts in both frameworks will give us some insight to help you choose the right development framework for your app.

## What is the Sample app

Sample app is a MVP app that has the core functionalities present in most apps: authentication, register, a home and profile page. After you decide what framework best suits your needs, the code you created can be used later for your app.

This app should start on the Login screen. From there, you can choose to login or register. The flow of the app is represented by the following flow diagram.




Notice that there is a difference between navigations: **Navigate** and **Replace**.

> When you **navigate** to a new screen, you are pushing the new screen to the "navigation stack", keeping the old screen. This allows you to go back to the old screen.
>
> When **replacing**, you are removing the old screen and adding the new one. This prevents you to go back to the old screen.
>
> This distinction is important because, for example, you don't want the user to be able to go back to the login while authenticated.

For each screen, you also have some **fields**, **buttons** and **views**. Fields are text fields to change information, views are just some way to display information and buttons are links to other screens.

## Requirements

> Keep in mind that the app does not need to be built exactly as described. The important thing is to try different features and compare how hard is to implement them in **Flutter**/**React Native**.

In order to comparare of both frameworks, you should test the following:

- Navigation (already included in flutter, external library in react native)
- Redux (instead of saving the information on the "state", save it in redux store)
- API calls (you can use the api calls form the Lalaoke Gitlab repository for authentication/register/edit profile)
- Native components (buttons, views, images, whatever you want to test)
- Styles (change the style of some components)
- Multiple environments (development, staging, production) (use something like react-native-config)

# Appendix B

```dart
import 'package:flutter/material.dart';

class Buttons extends StatelessWidget {
  String textfield;
  double fontsize;
  Color color;
  Function onTap;
  double width;
  double height;

  Buttons({this.textfield, this.color, this.onTap, this.width}); //constructor

  Buttons.big({this.textfield, this.onTap, this.color}) { //constructor for a big button
    this.width = double.infinity;
    this.height = 30;
    this.fontsize = 20;
  }

  Buttons.small({this.textfield, this.onTap, this.color}) { //constructor for small button
    this.width = 100;
    this.height = 30;
    this.fontsize = 20;
  }

  Widget build(BuildContext context) { //build button with given variables
    return Padding(
        padding: EdgeInsets.only(bottom: 10),
        child: SizedBox(
            width: this.width,
            height: this.height,
            child: RaisedButton(
              color: this.color,
              onPressed: this.onTap,
              child: Text(
                this.textfield,
                style: TextStyle(
                  color: Colors.black,
                  fontSize: this.fontsize,
                ), // TextStyle
              ), // Text
            ))); // RaisedButton // SizedBox // Padding
  }
}
```

Figure 1: Flutter button

# Appendix C

| | |
|---|---|
| **Name** | Registration |
| **Actor** | User |
| **Description** | The user has to register so they can enter the mobile app |
| **Pre Conditions** | 1. The e-mail that the user uses is not registered in the database. |
| **Basic Flow** | 1. Navigate to Registration screen<br>2. Fill name<br>3. Fill email<br>4. Fill password<br>5. Click in register button |
| **Post-Conditions** | User account is created and registered in database successfully |
| **Alternate Course** | 3. Email not in the right format<br>3.1. Message error appears<br>3.2 User stays in registers screen<br>4. Password too short or long<br>4.1 Message error appears<br>4.2 User stays in register screen<br>5. Server error message<br>5.1 E-mail already exist in database<br>5.2 User stays in register screen |

Table 1: User registration

| | |
|---|---|
| **Name** | Login |
| **Actor** | User |
| **Description** | The user needs to login so they can enter the mobile app |
| **Pre Conditions** | 1. User is already registered in the database<br>2. User not authenticated |
| **Basic Flow** | 1. Navigate to Login screen<br>2. Fill email<br>3. Fill password<br>4. Click in Sign In button |
| **Post-Conditions** | User enters the mobile app successfully |
| **Alternate Course** | 4. Server error message<br>4.1 Wrong email and password combination<br>4.2 User stays in login screen |

Table 2: User authentication

| | |
|---|---|
| **Name** | Login with Facebook |
| **Actor** | User |
| **Description** | The user can login with their Facebook account |
| **Pre Conditions** | 1. User not authenticated |
| **Basic Flow** | 1. The user click in the button "Login with Facebook" |
| **Post-Conditions** | User login with Facebook is successful |
| **Alternate Course** | 1. Server error message<br>1.1. Facebook service fails<br>1.2. User stays in login screen |

Table 3: Facebook authentication

| | |
|---|---|
| **Name** | Recover password |
| **Actor** | User |
| **Description** | In case the user forgets their password they can recover it |
| **Pre Conditions** | 1. The user is not authenticated. |
| **Basic Flow** | 1. Navigate to the Login page<br>2. Click in "Recover Password"<br>3. Fills email<br>4. Click in recover button |
| **Post-Conditions** | User recovers password successfully. |
| **Alternate Course** | 3. Email not in the right format<br>3.1 Message error appears<br>3.2 User stays in login screen<br>4. Server error message<br>4.1 Email does not exist in database<br>4.2 User stays in login screen |

Table 4: Recover password

| | |
|---|---|
| **Name** | Logout |
| **Actor** | User |
| **Description** | User can leave their account |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Navigates to the Profile screen<br>2. Clicks in "Logout" |
| **Post-Conditions** | User logouts successfully |
| **Alternate Course** | 2. Server error message<br>2.1 Can not perform logout<br>2.2 User stays in profile page |

Table 5: Logout

| | |
|---|---|
| **Name** | View Profile |
| **Actor** | User |
| **Description** | User can view their profile information |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Clicks in the profile button |
| **Post-Conditions** | User navigates to their profile and views their information successfully. |
| **Alternate Course** | None |

Table 6: View profile

| | |
|---|---|
| **Name** | View badges |
| **Actor** | User |
| **Description** | User can see their badges |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Clicks in the profile button<br>2. Clicks in "View Badges" |
| **Post-Conditions** | User navigates to the badges screen and views their badges successfully. |
| **Alternate Course** | 2. Server error message<br>2.1 Cannot navigate to badges screen |

Table 7: View badges

| | |
|---|---|
| **Name** | Edit profile |
| **Actor** | User |
| **Description** | A user can edit their profile |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Clicks in the profile button<br>2. Clicks in the edit Profile button<br>3. Can change information: username, password, avatar, description<br>4. Clicks in the update button |
| **Post-Conditions** | User edits their profile information and database is updated successfully. |
| **Alternate Course** | 4. Password too short or long<br>4.1 Error message<br>4. Server error message<br>4.1 Cannot update profile<br>4.2 User stays in edit profile screen |

Table 8: Edit profile

| | |
|---|---|
| **Name** | Create new game with a friend |
| **Actor** | User |
| **Description** | A user can create a new game with a friend |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Choose a game type<br>2. Search for a player name or choose a Facebook friend<br>3. Click in the player name<br>4. Click in button "Create Game"<br>5. Navigates to question screen |
| **Post-Conditions** | A new game with friend is created and database updated successfully. |
| **Alternate Course** | 2. Inserted player name does not exist<br>5. Server error message<br>5.1 Can not create a game<br>5.2 User stays in create game screen |

Table 9: Create game with a friend

| | |
|---|---|
| **Name** | Create new game with a random player |
| **Actor** | User |
| **Description** | A user can create a new game with a random player |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Choose a game type<br>2. Select random player option<br>3. Clicks in button "Create Game" |
| **Post-Conditions** | A new game with random player is created and database updated successfully. |
| **Alternate Course** | 3. Server error message<br>3.1 No players available<br>3. Server error message<br>3.1 Cannot create a game |

Table 10: Create new game with a random player

| | |
|---|---|
| **Name** | Submit question and answer |
| **Actor** | User |
| **Description** | The user after choosing a game from their list of games, they can answer a question so that the other player can answer it next |
| **Pre Conditions** | 1. The user is authenticated<br>2. The user is in the Home Page<br>3. The user has at least a game in which its their turn to submit a answer and question about themselves |
| **Basic Flow** | 1. Clicks in a game in the games list<br>2. Views last round results and clicks in play new round<br>3. Fills input field with an answer<br>4. Submit answer |
| **Post-Conditions** | User submits questions and answer, and database is updated successfully. |
| **Alternate Course** | 3. User can:<br>3.1 Use a Re-rolls and/or use a power-up coins<br>3.2 Answer the question<br>4. Server error message<br>4.1 Could not submit answer |

Table 11: Submit question and answer

| | |
|---|---|
| **Name** | Guess other player answer |
| **Actor** | User |
| **Description** | The user chooses a game from their list of games that has a ongoing round. Then they answer the question that the other player already answered |
| **Pre Conditions** | 1. The user is authenticated<br>2. The user is in the Home Page<br>3. The user has at least one ongoing game in which it is their turn to guess the other player answer |
| **Basic Flow** | 1. Chooses a game from the games list<br>2. Fills input field with an answer<br>3. Submit answer |
| **Post-Conditions** | User submits the answer and database is updated successfully. |
| **Alternate Courses** | 2. User uses Power-Up to help them answer<br>2.1 User click in Power-Up helper<br>2.2 First letter of the other play answer appears<br>2.3 User answers the question<br>2.4 User submits answer<br>3. Server error message<br>3.1 Cannot submit answer |

Table 12: Guess other player answer

| | |
|---|---|
| **Name** | See last round statistics |
| **Actor** | User |
| **Description** | The user may view last round statistics at the end of a round |
| **Pre Conditions** | 1. The user is authenticated<br>2. The user is in the Home Page<br>3. User has ongoing games, where they are waiting for the other player to play |
| **Basic Flow** | 1. Clicks in a game from their list of games |
| **Post-Conditions** | User sees last round statistics successfully. |
| **Alternate Course** | 1. There are no played rounds in that game<br>1.1 Game is unclickable<br>1. Server error message<br>1.1 Cannot navigate to last round statistics |

Table 13: See last round statistics

| | |
|---|---|
| **Name** | See previous rounds statistics |
| **Actor** | User |
| **Description** | The user may view previous round statistics |
| **Pre Conditions** | 1. The user is authenticated<br>2. The user is in the Home Page<br>3. The user has ongoing games, where they are waiting for the other player to play |
| **Basic Flow** | 1. Click in an game from their list of games<br>3. Click in the button of previous statistics |
| **Post-Conditions** | User sees game statistics successfully. |
| **Alternate Course** | 1. There are no played rounds in that game<br>1.1 Game is unclickable<br>1. Server error message<br>1.1 Cannot navigate to last round statistics<br>3. Server error message<br>3.1 Cannot navigate to previous rounds statistics |

Table 14: See previous rounds statistics

| | |
|---|---|
| **Name** | See shop products |
| **Actor** | User |
| **Description** | In case the user wants to see game products, they can go to the shop |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Clicks in the shop button |
| **Post-Conditions** | User enters the shop successfully. |
| **Alternate Course** | 1. Server error message<br>1.1 Cannot navigate to shop |

Table 15: See shop products

| | |
|---|---|
| **Name** | Buy product in shop |
| **Actor** | User |
| **Description** | In case the user wants to buy re-rolls, power-up coins or power-ups helpers they can go to the shop and buy them |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Click in the shop button<br>2. Choose a item<br>3. Click in the buy button |
| **Post-Conditions** | User buys a shop item and database is updated successfully. |
| **Alternate Course** | 1. Server error message<br>1.1 Cannot navigate to shop<br>3. User doesn't have enough coins<br>3.1 User receives error message<br>3.2 User stays in shop screen |

Table 16: Buy product in shop

| | |
|---|---|
| **Name** | Buy coins |
| **Actor** | User |
| **Description** | In case the user wants more coins they can buy them with real money |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the shop screen |
| **Basic Flow** | 1. Click in buy coins button<br>2. Choose one of the available options<br>3. Click in the buy button |
| **Post-Conditions** | User buys coins and database is updated successfully. |
| **Alternate Course** | 3. Server message error<br>3.1 User doesn't have enough money<br>3.2 User stays in buy coins screen |

Table 17: Buy coins

| | |
|---|---|
| **Name** | Win free coins |
| **Actor** | User |
| **Description** | In case the user wants more coins they can watch an advertisement video |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Click in the store button<br>2. Click in "Win free coins"<br>3. View advertisement video<br>4. Coins are added to wallet |
| **Post-Conditions** | User buys coins successfully. |
| **Alternate Course** | 3. Server message error<br>3.1 Advertisement does not load<br>3.1 User stays in store screen |

Table 18: Win free coins

| | |
|---|---|
| **Name** | View games list |
| **Actor** | User |
| **Description** | The user can view their game list in the homepage |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Views their game list |
| **Post-Conditions** | User views their game list successfully. |
| **Alternate Course** | 1. There are no games in the games list<br>1.1 User is not part of any game |

Table 19: View games list

| | |
|---|---|
| **Name** | Delete game |
| **Actor** | User |
| **Description** | The user can delete a game from their game list |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Choose a game from games list<br>2. Click in delete game |
| **Post-Conditions** | User views deletes a game and database is updated successfully. |
| **Alternate Course** | 1. There are no games in the games list<br>1.1 User is not part of any game<br>2. Server error message<br>2.1 Unable to delete game |

Table 20: Delete game

| | |
|---|---|
| **Name** | Receive daily bonus |
| **Actor** | User |
| **Description** | The user receives daily bonus for every day in a row they enter the mobile game |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user entered the game in the previous day |
| **Basic Flow** | 1. Open the mobile game |
| **Post-Conditions** | User receives daily reward. |
| **Alternate Course** | 1. The user doesn't receive the daily bonus<br>1.1 User did not enter the game the day before |

Table 21: Receive daily bonus

| | |
|---|---|
| **Name** | Send notification |
| **Actor** | User |
| **Description** | In case the user submits a question and answer, a notification is sent to the other player that they can play |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user has at least game<br>3. The user submits a question and answer |
| **Basic Flow** | 1. Send notification |
| **Post-Conditions** | User sends notification successfully. |
| **Alternate Course** | None |

Table 22: Send notifications

| | |
|---|---|
| **Name** | Disable notifications |
| **Actor** | User |
| **Description** | In case the user doesn't want to receive game notifications they can disable them |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page |
| **Basic Flow** | 1. Click in the Settings page<br>2. Unchecks option "Receive notifications" |
| **Post-Conditions** | User disables notifications successfully. |
| **Alternate Course** | 2. Server error message<br>2.1 Cannot to disable notifications |

Table 23: Disable notifications

| | |
|---|---|
| **Name** | Enable Notifications |
| **Actor** | User |
| **Description** | In case the user wants to receive game notifications they can be re-enable them |
| **Pre Conditions** | 1. The user is authenticated.<br>2. The user is in the Home Page<br>3. The user previously disabled notifications |
| **Basic Flow** | 1. Click in the Settings page<br>2. Checks option "Receive notifications" |
| **Post-Conditions** | User enables notifications successfully. |
| **Alternate Course** | 2. Server error message<br>2.1 Cannot enable notifications |

Table 24: Enable notifications

| | |
|---|---|
| **Name** | Watch tutorial |
| **Actor** | User |
| **Description** | When the user authenticates for the first time a tutorial should appear so they can know how to play the game |
| **Pre Conditions** | 1. The user has an account<br>2. The user never logged in |
| **Basic Flow** | 1. Logins with a valid account<br>2. The system provides a tutorial that will appear right after the first login |
| **Post-Conditions** | User sees the tutorial successfully. |
| **Alternate Course** | 1. Receives server error<br>1.1 Invalid combination email/password |

Table 25: Tutorial

| | |
|---|---|
| **Name** | Add question |
| **Actor** | Admin |
| **Description** | An admin can add new questions to the database |
| **Pre Conditions** | 1. The admin is logged in in the backoffice |
| **Basic Flow** | 1. Click in "Add Question"<br>2. Choose game type<br>3. Insert question<br>4. Click in "Add" |
| **Post-Conditions** | Admin adds question and databse is updated successfully. |
| **Alternate Course** | 4. Server error message<br>4.1. Cannot add question |

Table 26: Add question

| | |
|---|---|
| **Name** | Ban user |
| **Actor** | Admin |
| **Description** | An admin can ban users from the database |
| **Pre Conditions** | 1. The admin is logged in in the backoffice |
| **Basic Flow** | 1. Click in "Ban player"<br>2. Choose player<br>3. Click "Ban" |
| **Post-Conditions** | Admin bans user and database is updated successfully. |
| **Alternate Course** | 2. No users in the database |

Table 27: Ban user

# Appendix D

## Devise Endpoints Description

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| auth | After a user inserts their information in the register screen and clicks to register, a request is sent to this endpoint which creates a user account if the information inserted by the user is valid | • email<br>• username<br>• password<br>• password confirmation |
| auth_sign_in | After a user inserts they login information and click login, a request is sent to this endpoint which checks if the information inserted corresponds to a user account | • email<br>• password |
| auth_sign_out | After a user tries to logout, a request is sent to this endpoint which will invalidate the user's authentication token | None |
| validate_tokens | When a user enters the app, a request is sent to this endpoint with the current user tokens stored in the app, to check if the tokens are valid. If they are, the user navigates to the Home screen. Otherwise they navigate to the Main screen | None |

Table 28: Description of the devise endpoints

### User Controller Endpoints Description

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| index | After a user successfully logins in to the app, the app sends a request to this endpoint so the user receives all the information they need, including their games and personal information | None |
| reward_free_coins | After a user finishes watching an ad, the app sends a request to this endpoint so he can receive their coins | • number of coins |
| save_notification_token | After a user successfully logins in to the app, the app retrieves a token used to send notification to that particular device. Then, sends a request to this endpoint, that saves that token for that specific device. | • notification token<br>• device id |
| change_allow_notification | When a user activates/deactivates notifications, the app sends a request to this endpoint, that saves that value | • value (true or false) |
| validate_purchase | After a user completes a real purchase, a request is sent to this endpoint to verify if the transaction is trustworthy. | • purchase receipt |
| random_player_game | When a user chooses to play with a random player, the app sends a request to this endpoint, that responds with a random player information | None |
| search_user | When a user tries to search for a player to play a game with, the app sends a request to this endpoint which searches for players matching the user input, and sends back a response with the user found (if any). | • user input |
| buy_shop_item | When a user tries to purchase a store item, the app sends a request to this endpoint, which finishes the transaction | • item id |

Table 29: Description of the user controller endpoints

| **Endpoint Name** | **Description** | **Received Parameters** |
|---|---|---|
| forgot_password | In case a user forgets their password and insert their email to recover it, a request is sent to this endpoint, that generates a new password for the user and sends them an email containing that password | • email |
| update | After a user clicks the button to update their profile, a request is sent to this server endpoint which updates the user information depending on the parameters sent | • username<br>• password<br>• password confirmation<br>• avatar |
| facebook_login | The app sends a request to this endpoint when a user tries to login with Facebook. This endpoint creates a new user account in case the user tries to loggin with Facebook for the first time,. | • facebook token |

Table 30: Description of the user controller endpoints - part 2

### Game Controller Endpoints Description

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| create_game | If a user clicks to create a game, a request is sent to this endpoint which has the main purpose of creating a game between the current user and a opponent | • opponent id<br>• game type id |
| reroll_question | After a user clicks to reroll a question in a game, a request is sent to this server endpoint which changes the that round question to another from the same game type (if the user owns powerup rerolls) | • round id |
| submit_answer | The app sends a request to this endpoint when a user submits their answer to a specific question. This endpoint has the main purpose of adding that answer to the current round. | • round id<br>• answer |
| guess_answer | The app sends a request to this endpoint when a user submits their answer to a specific question that was already answered by the other player. This endpoint has the main purposes of adding the answer, checking if the round was won, among others. | • round id<br>• answer |
| new_round | After finishing a round, one of the users can start a new one. When they click the button to start a new round, a request is sent to this endpoint that creates a new round between the users with a random question from the current game type | • game id<br>• answer |
| use_helper | If a user decides to use a powerup helper, the app requests this endpoint that adds letters included in the word(s) of the other player's answer to a variable and then sends that variable current state back to the app (if the user owns powerup helpers) | • round id |
| use_powerup_coins | If a user decides to use a powerup coins, the app requests this endpoint that marks powerup coins as used for that round (if the user owns powerup coins) | • round id |
| use_powerup_coins | If a user decides to use a powerup coins, the app requests this endpoint that marks powerup coins as used for that round (if the user owns powerup coins) | • round id |
| delete_game | If a user decides to delete a game, the app requests this endpoint that removes that game from the database | • game id |

Table 31: Description of the game controller endpoints

### Other Controllers Endpoints Description

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| index | When a user want to changer their avatar, a request is sent to this endpoint which return all avatars | None |

Table 32: Description of the avatars controller endpoint

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| index | When a user enters the Badges screen, a request is sent to this endpoint which return all badges | None |

Table 33: Description of the badges controller endpoint

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| index | When a user enters the Home screen, a request is sent to this endpoint which return all game types | None |

Table 34: Description of the game types controller endpoint

| Endpoint Name | Description | Received Parameters |
|---|---|---|
| index | When a user enters the Store screen, a request is sent to this endpoint which return all purchasables | None |

Table 35: Description of the purchasables controller endpoint

# Appendix E

### Game Controller Test Cases

| ID | Description | Expectations |
|---|---|---|
| TestCase_01 | Game is created successfully | • Response status: 200<br>• Game was inserted in the database<br>• Created game has the chosen game type<br>• Created game question belongs to the chosen game type |
| TestCase_02 | Game already exists between both users for that game type | • Response status: 406<br>• Response message contains error: "You already have a game with this user within this game type"<br>• No game was created in the database |
| TestCase_03 | Opponent does not exist | • Response status: 404<br>• Response message contains error: "User not found"<br>• No game was created in the database |

Table 36: Test cases for the "create_game" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_04 | User has powerup rerolls | • Response status: 200<br>• Response contains the new question<br>• User loses 1 powerup reroll<br>• Round contains new question from the correct game type |
| TestCase_05 | User has no poweruop rerolls | • Response status: 406<br>• Response error message contains: "Sorry you have no re-rolls"<br>• User powerup rerolls stay the same<br>• Question stays the same |

Table 37: Test cases for the "reroll_question" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_06 | User submits question successfully in a round | • Response status: 200<br>• The saved round answer matches the one in the request<br>• The round only has one saved answer<br>• The next turn belongs to the player the user is playing with |
| TestCase_07 | User submits empty answer | • Response status: 406<br>• Response error message contains: "Sorry answer can't be empty"<br>• Question stays the same |
| TestCase_08 | User already submit an answer | • Response status: 406<br>• Response error message contains: "You already submitted your answer"<br>• Question stays the same |

Table 38: Test cases for the "submit_answer" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_09 | User submits right answer without using powerup coins | • Response status: 200<br>• The saved round answer matches the one in the request<br>• The answer was added to the round<br>• The Round was defined as won<br>• Both users won 20 coins<br>• Response includes the new round information and the right amount user coins<br>• The round has two answers<br>• Game streak increments |
| TestCase_10 | User submits right answer without using powerup coins | • Response status: 200<br>• The saved round answer matches the one in the request<br>• The answer was added to the round<br>• The Round was defined as won<br>• Both users won 20 coins<br>• Response includes the new round information and the right amount user coins<br>• The round has two answers<br>• Game streak increments |

Table 39: Test cases for the "guess_answer" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_11 | User submits wrong answer without using powerup coins | • Response status: 200<br>• The saved round answer matches the one in the request<br>• The answer was added to the round<br>• The Round was defined as loss<br>• Both users don't win coins<br>• Response includes the new round information and the right amount of user coins<br>• The round has two answers<br>• Game streak increments |
| TestCase_12 | User submits wrong answer using powerup coins | • Response status: 200<br>• The saved round answer matches the one in the request<br>• The answer was added to the round<br>• The Round was defined as loss<br>• Both users don't win coins<br>• Response includes the new round information and the right amount of user coins<br>• The round has two answers |
| TestCase_13 | User submits empty answer | • Response status: 406<br>• Response error message contains: "Sorry answer can't be empty"<br>• Question stays the same |
| TestCase_14 | User already submit an answer | • Response status: 406<br>• Response error message contains: "You already submitted your answer"<br>• Question stays the same |

Table 40: Test cases for the "guess_answer" endpoint - part 2

| ID | Description | Expectations |
|---|---|---|
| TestCase_15 | User creates new round successfully | • Response status: 200<br>• Round is added to the current game<br>• A new question belonging to the current game type is added to the round<br>• Response includes the round that was just created |
| TestCase_16 | User does not belong to round | • Response status: 406<br>• Response error message contains: "You do not belong to this game"<br>• No Round is created in the database |

Table 41: Test cases for the "new_round" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_17 | User uses helper successfully | • Response status: 200<br>• User loses 1 powerup helper<br>• Answer helper is updated in the database accordingly<br>• Response contains the new answer helper<br>• Round contains new question from the correct game type |
| TestCase_18 | User has no powerup helpers | • Response status: 406<br>• Answer helper is not updated<br>• User does not lose any powerup helpers<br>• Response error message contains: "Sorry you have no helpers" |

Table 42: Test cases for the "use_helper" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_19 | User uses powerup coins successfully | • Response status: 200<br>• User loses 1 powerup coins<br>• Round "used_powerup_coins" column is set to true in database<br>• Response contains the new answer helper<br>• Round contains new question from the correct game type |
| TestCase_20 | User has no powerup coins | • Response status: 406<br>• Powerup coins stays the same<br>• User does not lose any powerup coins<br>• Response error message contains: "Sorry you have no powerups" |
| TestCase_21 | User already used powerup coins | • Response status: 406<br>• Powerup coins stays the same<br>• User does not lose any powerup coins<br>• Response error message contains: "You already used powerups this round" |

Table 43: Test cases for the "use_powerup_coins" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_22 | Game is deleted successfully | • Response status: 200<br>• Game does not exist in the database<br>• Response includes all the user games exepect the deleted one<br>• Round contains new question from the correct game type |
| TestCase_23 | Game does not exists | • Response status: 404<br>• Response error message contains: "Game does not exist or was already removed" |

Table 44: Test cases for the "delete_game" endpoint

### User Controller Test Cases

| ID | Description | Expectations |
|---|---|---|
| TestCase_24 | User receives their information with no daily bonus | • Response status: 200<br>• User does not receive any coins from daily bonus<br>• Response includes user information and respective games information<br>• Response does not include daily bonus |
| TestCase_25 | User receives their information with daily bonus | • Response status: 200<br>• User receives daily bonus coins<br>• Response includes user information and respective games information<br>• Response includes daily bonus |

Table 45: Test cases for the "index" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_26 | Server creates new notification token for a device successfully | • Response status: 200<br>• User does not have a notification token for that device<br>• Server creates a new column with the device_id and notifications_token from the request<br>• Response includes the created notification token |
| TestCase_27 | Server updates notification token for a device successfully | • Response status: 200<br>• User has a notification token for that device<br>• Notification token gets replace by the new notification token from the request<br>• Response includes the updated notification token |

Table 46: Test cases for the "save_notifications_token" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_28 | User deactivates/activates notifications successfully | • Response status: 200<br>• Column "allow_notifications" belonging to a user changes to true if it was previously false, or to false if it was previously true<br>• Response includes the new value |

Table 47: Test cases for the "change_allow_notifications" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_29 | Request includes invalid receipt data | • Response status: 406<br>• User does not receive coins<br>• Purchase is saved in the database as "invalid"<br>• Response includes error "Something went wrong with your purchase" |

Table 48: Test cases for the "validate_purchase" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_30 | User finds a random player to play with | • Response status: 200<br>• Response includes a random user data |
| TestCase_31 | There are no player available | • Response status: 404<br>• Response includes error message "No users found" |

Table 49: Test cases for the "random_player_game" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_32 | User finds players by username/email successfully | • Response status: 200<br>• Database has users that match that input<br>• Response includes users that match the requested username/email |
| TestCase_33 | No users match the input | • Response status: 404<br>• Database has no users that match that input<br>• Response includes error message "No users found" |

Table 50: Test cases for the "search_users" endpoint

| ID | Description | Expectations |
|---|---|---|
| TestCase_34 | User buys items successfully | • Response status: 200<br>• User had enough coins to buy the items<br>• User coins after purchase are correct<br>• User received the item quantity |
| TestCase_35 | User has not enough coins | • Response status: 406<br>• Response includes error message "You don't have enough coins to purchase this item"<br>• User does not lose any coins<br>• User does not acquire any items |

Table 51: Test cases for the "buy_shop_item" endpoint