POLITECNICO
MILANO 1863

Design and Implementation of Mobile Applications
2022/2023

Spotify Music Quiz

Design Document

Antony Pascalino
Antonio Pettorruso

With the help of:

BENDING SPOONS

Table of contents

1 Introduction

1.1 Purpose

Spotify Music Quiz is a trivia music quiz game in which the players will face a series of music-based question related to their music tastes.
Differently from every other music quiz game, the questions are not randomly generated from the app based on the most famous music or some other random criterion. In *Spotify Music Quiz* the users are asked to log in with their personal Spotify® account once the app is open. In such way the app will be able to know the tastes of the player and will create questions based on the music tests of the player. The app will analyze the most listened song of the users, as well as their saved and created playlists, the authors they listen the most and the songs they have saved as their favorites.
The types of questions are the most varied: from guessing the release year of an album to guessing the author of the song you are currently listening to and so on.
The players can choose if facing all the types of questions in the *Classic mode* or play a specific mode, in which they will face only the single kind of question for that mode. However, the *Classic* mode is considered the main mode of the game.
The more the questions the players will guess, the higher their score will be.
It is also available a friends management system, where players can add other players and compare their score in each mode.
This document's aim is to explain and motivate the choices that were made during the design and implementation phase of the application. A high level but detailed explanation about the structure of the backend and of the business logic are provided, together with the layout of the various screens of the user interfaces. External services and plugins used are also mentioned; finally, a description of the tests carried out and a motivation for their relevance is available.

1.2 Features

1.2.1 Login with Spotify account

Once the app is launched the users are asked to sing in using their Spotify credentials. The login process levers the Spotify API for Authentication that will be discuss better later. Once the users complete the login process the app will obtain their personal information of their Spotify account which are:

- Display name
- E-mail
- Profile image
- Spotify ID

Such information will be used as identification information inside the app, both for front-end and back-end purposes, and cannot be modified by the users inside the app since they are strictly related to their Spotify account. If they will externally change some of the information of their Spotify account, those changes will be reflected inside the app on the next launch.

1.2.2 Play a game

The user, once selected the mode, can start a new game. Different questions will be proposed one after the other. For each question if the player gives the right answer, it gains a point and moves to the next question, if a wrong answer is given the game is over and the player can choose to try again or go back to the homepage.
Each question has a timer and if the user does not give an answer before the timer expires, it is considered as a wrong question and the game overs.
We can split the questions in two large group: multiple answer questions and textual answer questions. In the former ones, there is a text question generated by the app, which consists of a music related question, e.g. "Who is the author of the song Bohemian Rapsody?"; the user can choose between four answers but only one is the right one. In the latter ones, instead, the user will listen to a song and is required to write the title of the song or its author, based on what is asked by the current question. The matching with the correct answer is done implementing the Levenshtein Distance algorithm, in order to give the users a degree of errors they can make in writing the word. In this way it is avoided to have a wrong answer only because of a typo or a different case of the characters.
For each question a background song, related to the current question, is played.

1.2.3 Friends management

The application offers a screen in which current user's friends are shown ordered by their high score in the *Classic* mode. Users can add new friends using the functionality offered by the application itself. When adding a friend, a list of all the other registered users is shown, ordering them according to their names. Users can also filter the list writing the name of a particular user they would like to add.
It is noted that it would have been a different choice to synchronize the list of friends of the app with the list of Spotify friends. Unfortunately, Spotify APIs don't allow to retrieve the list of Spotify friends of a particular user, but only the number of friends, called followers on Spotify.

1.2.4 Modes

Other than the *Classic* mode, in which the player is asked different kind of questions, it have been added different modes of play. Each mode is related to a particular kind of question. Differently from the *Classic* mode, the user will face for the entire game always the same kind of question. Other than the *Classic* one, the modes that the app offers are:

- *Guess the song:* the player will listen to a song and is asked to write the title of the song.
- *Guess the singer:* the player will listen to a song and is asked to write the author of the song.
- *Recall the year:* the player is asked to select the correct year of release of a particular album.
- *Which album:* the player is asked to select the correct album in which a particular song have been released
- *Who is the author:* the player is asked to select to correct author of a particular song.
- *Author song:* the player is asked to select the correct song sung by a particular author.

The rules of the games are the same of the *Classic* mode and of course the questions are always related to the tastes of the user.

1.2.5 Leaderboards

Once flowing to a particular mode page, the users are shown a particular leaderboard. In this leaderboard is shown the list of their friends ordered by their high scores on that specific mode. Also, the current user high score is shown. If none of user's friends has ever played that mode, the leaderboard will be empty, and a message tell the user so. A link to add new friends is made available also in this page and if the user will add new friends that have a high score in that mode, it will be reflected right away on the leaderboard.

1.2.6 Authors score

When the users play any mode, each time they guess a correct answer they gain a point, and it is shown in real time during the game. Their sum will be the final score of the game once the player loses. More than that a different kind of points are stored: for each question related to a particular author, if the player guesses the correct answer, a point for that particular author is gained. All those points are then collected for each author and shown in a particular screen for the user. It will result in a leaderboard of the authors the user knows better, showing for each author the total number of questions that the users have ever guessed since their first game in the app.

2 Architecture

2.1 Overview

Spotify Music Quiz is a mobile application developed in Swift for iOS and iPadOS and thus it can run both on iPhone and iPad. Its architecture is quite simple: it relies on Firebase by Google as a safe and secure backend for storing the users' data and leverages the Spotify APIs to retrieve user infomration, i.e. personal data, and users tastes, i.e. saved playlist, most listenend songs and similar.

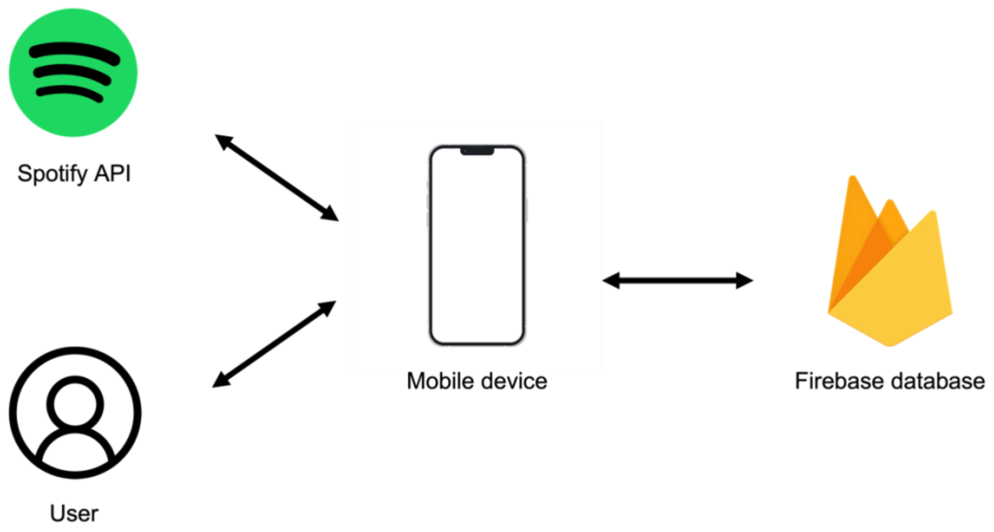


Figure 1: Architecture

It has been chosen to use the *Model-View-ViewModel* (MVVM) architectural pattern instead of the traditional Model-View-Controller (MVC) approach. MVVM offers a more reactive design pattern, which aligns well with the declarative nature of SwiftUI. With MVVM, the responsibilities are distributed among three distinct components: the Model, which represents the data and business logic; the View, responsible for rendering the user interface; and the ViewModel, acting as the intermediary between the Model and the View.

2.2 Model

For what concern the data management, the application relies on an online cloud database, Firebase Firestore.
The Database contains users' personal information – retrieved from their Spotify account – such as the name displayed on Spotify, the Spotify profile picture and the Spotify ID. More than that the database is used to store the list of friends for each user, the user high scores for each mode and their authors' score.

2.2.1 Data structure

The database structure is simple but efficient: there are two main collections, illustrated below:

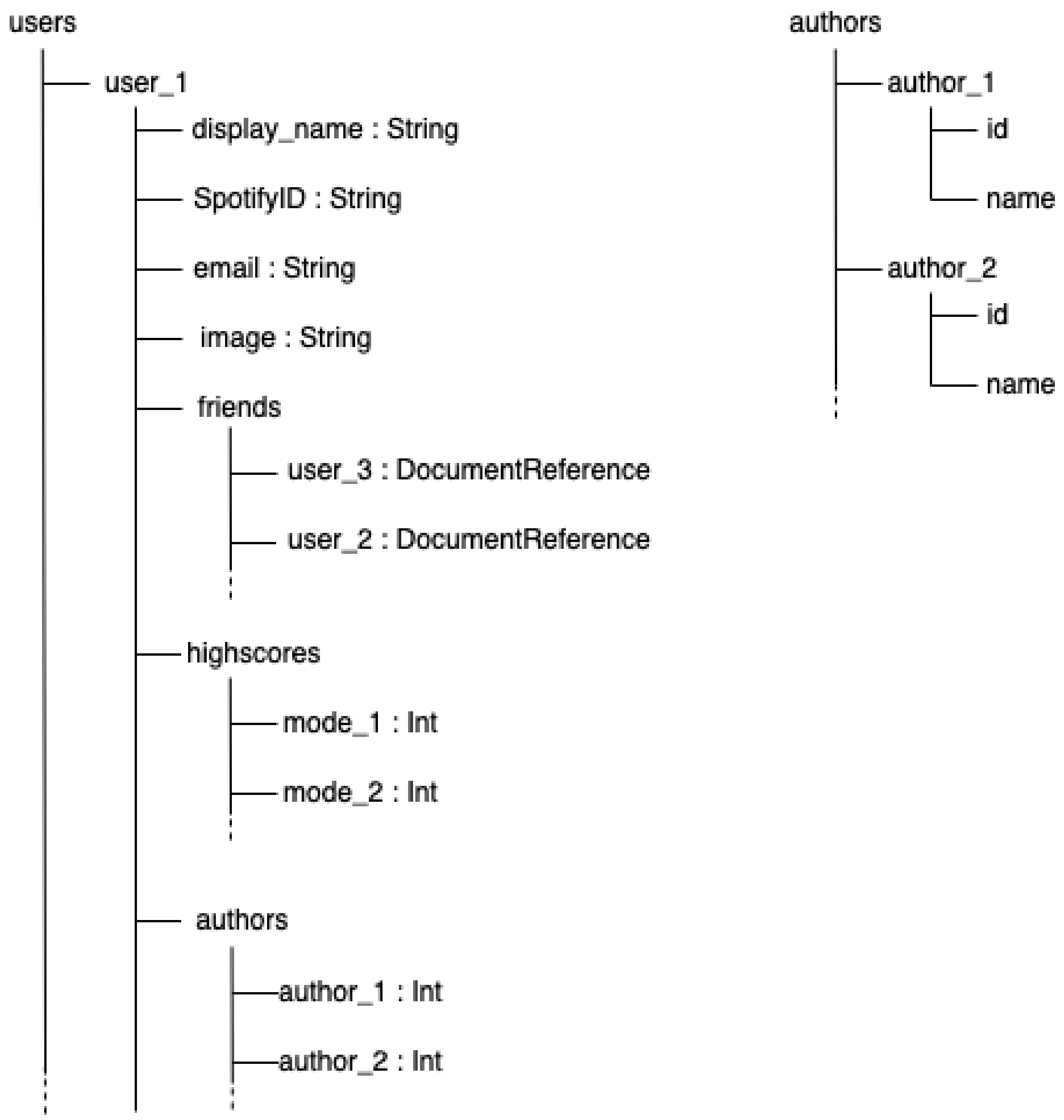


Figure 2: Database structure

Here a detailed description of the elements in the structure.

- **users:** it is a Collection which stores one Document for each user. Each Document has the same structure, and its fields are:
    - *display_name:* the nickname retrieved from the user Spotify account.
    - *SpotifyID:* the unique ID assigned by Spotify to each user.
    - *email:* the email address retrieved from the user Spotify account.
    - *image:* the URL to the profile picture retrieved from the user Spotify account.

    - *friends:* an array of DocumentReferences, each item is a reference to another user document.
    - *highscores:* a map in which keys are strings representing a particular Game Mode and values are integers representing the user's high score of the relative mode.
    - *authors:* a map in which keys are strings representing a particular author and values are the user's score of that relative author.

- **authors:** it is a Collection which stores one Document for each author. Each time a user guesses a correct answer relative to a particular author, a query is performed to check if that author is already present in the DB: if yes, no other action is performed, if not a new entry for the author is added. The two fields of each document are:
    - *id:* the unique ID assigned by Spotify to each author.
    - *name:* the stage name of the author on Spotify

2.2.2 Swift structures

In order to map the data retrieved from Firebase and the Spotify API into the project, custom data models i.e., Swift structures, have been designed. These Swift structures are designed to represent the data in a format that aligns with the needs of our application and facilitates easy manipulation and usage of the data within our codebase.
The mapping between the Firebase data structure of the database and the Swift structure has been simple, since we were the designer of both and thus the structures have been created with the same fields.
When mapping the data from the database or the Spotify API to Swift structures, instead, we often need to perform data transformations and mappings to convert the raw data into a format that our application can work with efficiently. This includes converting data types, handling optional values, and structuring the data in a way that aligns with our application's logic and requirements.
We avoid describing every single structure since in most of the cases they have simply the same fields of the JSON object retrieved from the call. Some of them have some methods to deal with data transformation. For example, a method `mapWithUser()`has been implemented to map the user profile structure from Spotify API to the User structure used around the project and on the Firebase datastore.

2.2.3 Data access

To interact with the database, two packages have been added to the project: *FirebaseFirestore* and *FirebaseFirestoreSwift.* They are package offered for Swift implementation of Firebase databases and are necessary for all the basic database operations.

The two libraries used in the project have been:

- **FirebaseFirestore**: library that allows to perform CRUD operations on the data (create, retrieve, update, delete). It is used inside the app for operations like add a new user with all the information from Spotify, add a friend reference to a user, update highscores and authors scores. Most of the methods are called asynchronously in order to obtain a smoother app in presence of heavy loadings from the database.
- **FirebaseFirestoreSwift:** library that allows to code and decode data in particular structure written in the Swift language. Thanks to this library when retrieving some data from the databases, its field are automatically mapped into the fields of the Swift structure. In the same way when is needed to create a new entry in the database it is possible to directly upload a Swift structure and the new Document will have the same fields of the structure.

  To obtain consistency is needed to keep coherent the Swift structure with the fields we want for each Firebase document.

2.2.4 UserManager

The UserManager class is the component that utilizes Firebase methods to interact with the application's database. This class acts as an interface between the app and the Firebase server, providing functions to store, retrieve, and update user data.

Some of the main tasks performed by the UserManager are maintaining a current user reference representing the currently logged-in user, implement methods to update the users and get user information saved on the database. Additional functions handle operations such as adding friends, updating author scores, and retrieving user friends.

Overall, the UserManager component acts as a crucial intermediary between the app and the Firebase backend, encapsulating the logic required to manage user data, communicate with the database, and perform various operations related to user profiles and interactions.

2.2.5 Business Logic

The whole business logic of the application can be divided in two main tasks: the handle of each single game and the creation and distribution of the questions. Two class have the relative task to handle such logic and its description is given below.

- **Question manager:** this component is implemented as a Swift class with several methods and variables. Its main task are the actual creation of the questions and their distribution for the game.
  To create the questions, the component retrieves from the *APICaller* (the component which retrieve information from the user Spotify account) all the information about the music related to the current user. Once obtained such information, several methods are called to create several kinds of question. For instance, the method `genWhoSingsQuestions()` once retrieved the music related to the current user, create all the question of the type *"Who sings the song…?"*.
  Once the questions have been created, they are shuffled and put into variables in order to be retrieved from the *GameManager* during the game.
  It has been noted that the questions are created in batch, 20 question each time. In this way the loading time of the question has been reduced and, since the game could be potentially infinite, it was needed to choose a batch dimension and 20 has been tested as good for the loading time without needing to suspend the game for a new loading.
  Moreover, the generation of the questions starts in a background thread once the user lands on the homepage and not when the user starts a game. In this way, if the user navigates arounds the app to other pages, the loading is performed in the meanwhile. Once the user starts a new game this reduces the loading time that can also be zero if enough time is elapsed and the generation of the questions is already completed.

- **Game Manager:** this method is meant to handle the flow of each game. Its method are the ones that manages each step, in response to each user input. Once the user starts a new game, the questions are retrieved form *QuestionManager* and put into an array. When the user gives an answer, the *GameManager* check if the current question is right or not. Based on that brings the user to the next question or to the game over page. If the user presses the pause button the *GameManager* stops the timer and resume it when the player presses play. Finally, when the game is over it resets all the parameters used during the game and reload all the new questions right away in the case if the user wants to play again and in this way the loading time is actually reduced.

2.3 View

Since has been decided to implement the View using SwiftUI framework, *Spotify Music Quiz* consists of several Views that, interacting together according to the business logic, build the correct view for the user and respond to the user's inputs.
In this section, a description of each View and the way they are organized for providing all the application's functionalities are explained.

2.3.1 Answer

This View is actually a component that is used when a multiple answers question is shown during the game. Each Answer is instantiated receiving as input a String *answer* which consists of the actual text of the answer. It also has a boolean *isCorrect* which is set when the Answer is created and is true if the answer is the correct one of the current question or false if not. When an answer is selected by the user, it calls some methods on the game manager in order to sequentially handle the flow of the game.

2.3.2 Game Controls

This View is actually a component that is used during the game. It is always visible during the entire game even if some of its buttons are enable/disable in particular moments of the game. The first button allows to go back to the homepage during the game, giving up on the current score; the second button allows to play/pause the game, if the game is paused the user cannot give any answer and the music in background stops. The third button, once an answer has been given, allows to go to the next question or to the game over page.

2.3.3 TimeBar

This View is actually a component that is used during the game. It is always visible during the entire game and, how the name suggests, consist of a time bar. At the begging of each question the time is reset to zero. Multiple answer questions have a timer of 20 seconds while the simpler multiple choices questions have 10 seconds of time. If the user does not give an answer before the time expired, the component calls the game manager to end the current game and the player loses.

2.3.4 ListImage

This View is actually a component that is used when a list of users is shown, if it is in the list of friends, the leaderboard of a game mode or the screen that allows to add new friends. It takes as input a String that is the URL to a specific user profile image. While loading a placeholder is set to a generic icon. The same icon is used if the users do not have a profile picture in their Spotify account.

2.3.5 GameMode
This View is actually a component that is used on the homepage to show a particular game mode. It is basically a button which leads to the screen of the relative game mode. It has been implemented as a single component on its own for a better structured project. Moreover, it is used multiple times in the homepage, so its implementation as single component avoids also code duplication.

2.3.6 LogOutAlert
This View is actually a component that is used in the home page when the user wants to log out from the app and leads back to the first screen of the app. It is simply an alert tha pops out asking the users if they really want to log out. On Yes button the log out happens on No the alert disappear. It has been chosen to implement a custom alert instead of the default alert offered by Swift for a better look and feels that is in this way coherent with the entire style of the app.

2.3.7 CustomAlert
This View is actually a component that is used during the game to ask the users if they really want to leave the game and give up. It is identical to the LogOutAlert described above but show a different message and leads to a different screen in case of Yes button is pressed. Thus, it has been chosen to implement it as a new alert.

2.3.8 AddFriendsView
This View allows to add new users to the list of friends of the current user. It shows the alphabetically ordered list of all the user that have an entry in the database, removing of course the current user. Each user in the list has a button that, once pressed, add the relative user to the current user list of friends on the database. The change is right away reflected in the app. If a particular user is already present in the current user list of friends, the previous button is disables. Also, a search bar is present in the bottom part of the view. Writing anything in the search bar will filter the list of users shown, leaving only those which contains the search word inserted by the user. The search is performed in insensitive case. For each row of a user, a *ListImage* view is instantiated.

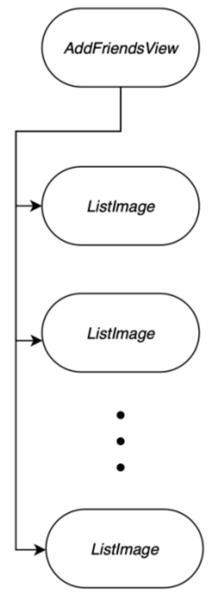


Figure 3: AddFriendsView

2.3.9 FriendsView

It shows the list of friends of the current user, ordered top-down from the friend with the higher high score to the one with the lowest one. The high score considered is the one of the *Classic* mode of game. If the current user has added no friends yet, a message is shown, inviting the user to add new friends using the specific button. Such button, once pressed, brings to the *AddFriendsView* described above. For each row of a user, a *ListImage* view is instantiated.

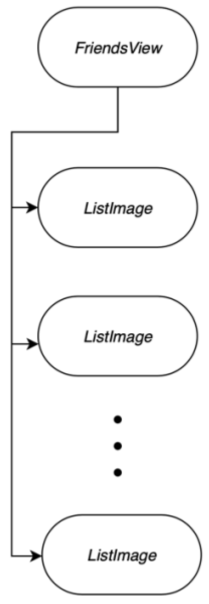


Figure 4: FriendsView

2.3.10 AuthorScoreView

It shows a list in which each row contains the name of an author and an integer. The integer represents the number of questions, related to that artist, that the current user has ever guessed since its first access in the application. The list is a representation of the authors that the current user knows better. The list is top-down ordered from the author the user knows better, with the highest score, to the author the user knows less, with the lowest score.

2.3.11 AuthView

It automatically opens a WebView, which leads to the Spotify login page. The requested web page is open giving as input the URL to the login page. Once the page is loaded the users are asked to insert their credentials of their Spotify account. If the credentials are correct, it receives a code. The code will then be sent to the Authentication manager component in order to exchange it for a token. The token will be stored in the current session of the app and used to call Spotify APIs.

2.3.12 ModeView

It is the screen related to a particular game mode of the app. Once the user chooses in the homepage which game mode wants to play, this view is shown. It has the leaderboard, in that specific mode, of the current user's friends. If no one of the current user's friend have never played this mode, the list will be empty, and a message informs the user of that. It also shows the high score of the current user in that specific mode. Moreover, a button which leads to *AddFriendsView,* is present useful in the case in which the user wants to add new friends and see the updated leaderboard. Finally, there is a button to start the game of that specific mode.

2.3.13 GameOverView

This View is shown once the players lose the game after they gave a wrong answer. It shows the score of the just finished game and a message is shown of that score is a new high score for the user. More than show the message, the method to update the user's high score in the database is called. Finally two buttons are present: one to go back to the homepage and one to play again in the same mode.

2.3.14 LoadingView

It is simply a view that appears when a long loading is happening. It is actually shown when the users login into the app, waiting for their personal data to load from Spotify API, and when the game is loading. The loading is quite a heavy process since it needs to load all the playlist, favorites authors, favorite songs of the current user, create all the needed question for the game and shuffle them. Even if the loading of the game happens in a background thread, if the user tries to start a game while the loading is not already completed, the view appears. Once the loading finishes the view automatically disappears.

2.3.15 ShazamLikeView

The view owes its name to the noticeable animation of the famous music recognizer app Shazam®, since it implements a very similar animation. This view is simply an animation that appears each time in the game there is an open text question, and the player has to write the correct answer. Once any answer is given by the user, the view changes and instead of the animation, the cover of the album related to the current listened song appears.

2.3.16 GuessTheSongView

This view, is one of the two alternative that can appear during the game, based on the current question type. This is the view that it shown when there is an open question in which the player is required to write the name of the title is listening to or its author. It consists of a *ShazamLikeView,* previously described, a Text and a TextField. The TextField is the view provided by SwiftUI that allows the user to input a text. Here the user is asked to write the answer that wants to give. Once the answer is given, the Text, previously invisible, appears showing the name of the song and the name of the author. Other than make the text appears, once the user gives an answer and commit it, the Game Manager is called to check the given answer and handle the next outcome of the game.

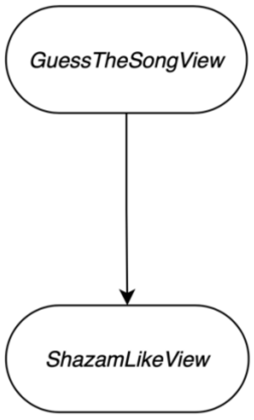


Figure 5: GuessTheSongView

2.3.17 LoginView

It is the first screen that appears when launching the app. It consists of the logo of the app and a button. The button invites the users to login using their Spotify credentials and once pressed brings to *AuthView,* in order to face the authentication process.

2.3.18 GameView

It is the main view that appears when a game start. It contains several components that all together offers all the functionality needed to play a game. On the bottom part of the screen there is the score of the current game, it is increased by 1 each time the player gives a correct answer. Then there is the actual text of the question, that is retrieved from the Game Manager. Below the question text, based on the current question type, there can be a set of four *Answer* component or the *GuessTheSongView*. In the case of the four answers, only one is correct and the others are wrong. After that the *TimeBar* component is present, giving the player an idea of the time remained to give the answer. Finally, the component *GameControls,* which allow the user to navigate to the next question, put the game in pause or give up the current game, how better explained before. The *CustomAlert* component is also present, but it is invisible unless the player presses the button to give up the game.

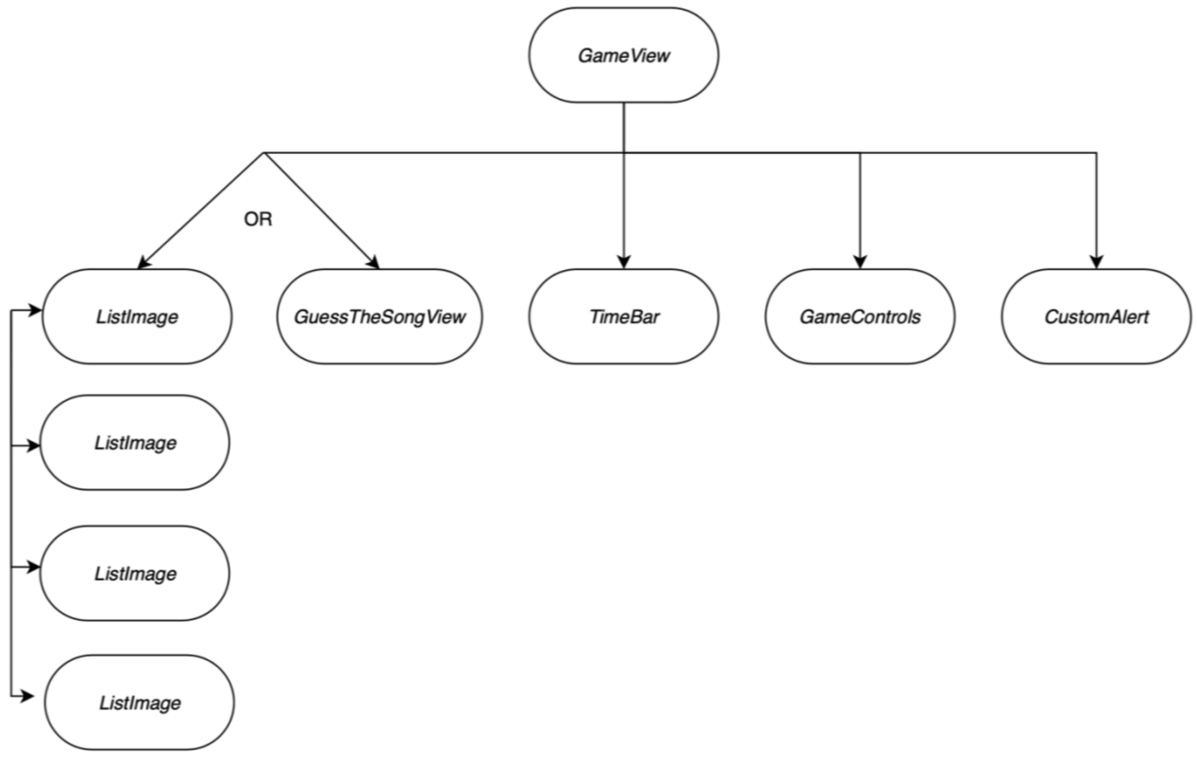


Figure 6: GameView

2.3.19 HomeView

The HomeView represents the homepage of the application. It is the first screen that appears to the user once the login phase successfully ends and when the user finishes a game and does not play another one right away. It shows the nickname of the current user and the profile picture, both retrieved from the Spotify account information. It is also shown the current user high score of the *Classic* mode. Then, there is a list of all the *GameMode* available in the app, that are shown on two rows through a horizontal ScrollView offered by SwiftUI. If the user clicks on any *GameMode,* this leads to the relative *ModeView* page. Finally in the right top corner, a button for the log out functionality is present. Once pressed the *LogOutAlert* appears asking the user if is sure to log out.

2.3.20 ContentView

This view works as a container for the main view of the app. It has a TabView offered by SwiftUI which contains three tabs: *HomeView, FriendsView* and *AuthorScoreView.* It shows three buttons in the bottom part of the screen and the user is able to switch between the three views described above, clicking on their relative buttons.

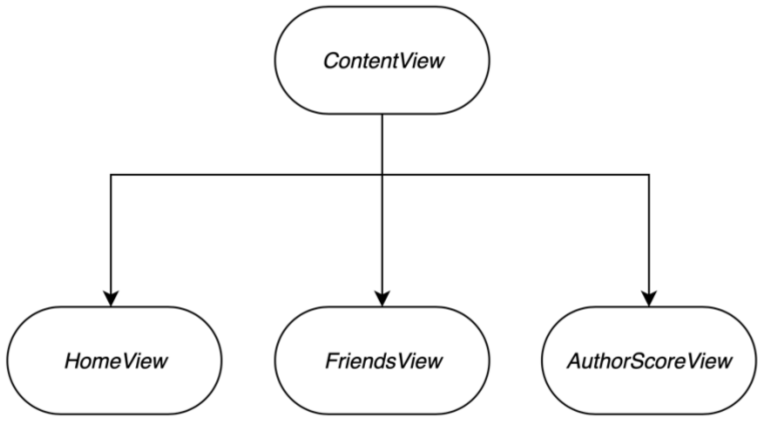


Figure 7: ContentView

2.4 ViewModel

The ViewModel acts as the intermediary between the Model and the View in the application. It provides the necessary data, behavior, and data binding to enable the View to render the user interface and respond to user interactions in a reactive and maintainable manner.

2.4.1 UserViewModel

The only data stored in the database by the app are user-based information. Even if there is a collection for authors, it has been created only with the purpose of keep track of users' authors scores. Given that, the only ViewModel component implemented for handling the data of the database is the *UserViewModel* class. It serves as a bridge between the user interface and the underlying *UserManager* component, providing data and behavior required by the views. Indeed, it simply call the methods of *UserMananger* component and, if needed, prepares and exposes the retrieved data in a way that is optimized for presentation and display purposes, ensuring the View remains focused on rendering and user interaction. Each View that needs to show user data has its own instance of *UserViewModel.*

2.5 Spotify API

In addition to integrating Firebase API as already discussed, it has been done a massive use of the Spotify API. They allowed to retrieve various user-related data, such as playlists, saved songs, and followed artists, enabling personalized and tailored content for each user. By integrating the Spotify API, we were able also to authenticate users securely, allowing them to log in to their Spotify accounts directly from our app. This authentication process grants us access to the user's Spotify data, ensuring that we can retrieve and display their playlists, saved songs, and artists that they follow. The Spotify authentication process has been used as main authentication process for the entire app since it allowed to securely and uniquely identify the user. In the table below will be listed all the endpoints of the Spotify API that have been used for the development of the app. In the table below, for each of them, is given a description of what they do and how they have been used.

`https://api.spotify.com/v1` is to be considered before each endpoint in the table, it has been omitted for simplicity.

Endpoint	**Use**
`/me/authorize`	Allows to authenticate the users using their Spotify credentials. It returns a token that will be used inside the app for calling other API
`/me/albums`	Gets a list of the albums saved in the current Spotify user's 'Your Music' library. It is used to create albums related question, e.g. “When was this album released?”
`/me/playlist`	Gets a list of the playlists owned by the current Spotify user. It is used for retrieve song that the currently user potentially knows. It is assumed that when users create a playlist, choosing some songs, they know them.

`/playlists/{playlist_id}/tracks`	Gets full details of the items of a playlist owned by a Spotify user. Together with the previous one allows to retrieve the track of a specific playlist.
`/me/tracks`	Gets a list of the songs saved in the current Spotify user's 'Your Music' library. It is used to create questions based on the saved tracks of the users, which they know.
`/me`	Gets detailed profile information about the current user (including the current user's username). It is used to retrieve users' information after they log in. Such information like, profile photo and username, are showed around the app like in the homepage.
`/me/following`	Gets the current user's followed artists. It is used to create artist related question like "Which is the most famous song of this artist?"
`/me/top/artists`	Gets the current user's top tracks based on calculated affinity. It is used complementary with the previous one in case some artists are often listened but not followed.
`/me/top/tracks`	Get the current user's top artists based on calculated affinity. It is used to get the tracks the user listens to the most according to Spotify recommender system.

	Questions are then created based on these tracks.
`/tracks/{id}`	Gets Spotify catalog information for a single track identified by its unique Spotify ID. It is used for question that are track-based like “In which album was this song released?”
`/recommendations`	Recommendations are generated based on the available information for a given seed entity and matched against similar artists and tracks. A list of tracks will be returned. It is used to create question based on song related to the user and to create the wrong answers. In this way the wrong answers are related to the correct one avoiding the question from being trivial.
`/artists/{id}/related-artists`	Gets Spotify catalog information about artists similar to a given artist. Similarity is based on analysis of the Spotify community's listening history. It is also used to generate not trivial wrong answers in the case the correct answer is an artist.
`/artists/{id}/top-tracks`	Get Spotify catalog information about an artist's top tracks by country. It is also used to create not trivial wrong answers in the case in which the correct answer is a year e.g. “In which year was this album released?”. In this way the wrong

	answers are release years of other albums of the same artist of the correct answer.

Table 1: Spotify API endpoints and their use

3 User Interface

3.1 UI Design

The UI design have been imagined thinking about the concept that the in the English language the word *Play* has a double meaning: it can refer to a song, “To play a song”, and can be referred to a game “To play a game”. Given that *Spotify Music Quiz* is a game based on music, it has been chosen to keep a correspondence between the two concepts. For example, the Play button commonly used to start a song here is used to start a new game. Similar matches between the two concepts are spread in the entire app. The design take inspiration from the official Spotify app, making the binding between *game* and *music* even stronger. Moreover, to use the app it is needed to already have a Spotify account, so is very likely that a user has ever used the Spotify official app. The Spotify-like design can thus be already familiar for the user.
To actually design the UI of the app has been used Figma software and the implementation of the UI has faithfully followed the original design.
As well as the official application of Spotify, the app does not make a difference between dark mode and light mode. Although SwiftUI implements by default the dark and light modes for every application, it has been forced to always appear in dark mode in order to be coherent with the design choice.

3.2 UI Implementation

To implement the User Interface has been chosen to use SwiftUI, instead of UIKit. SwiftUI is a modern declarative framework introduced by Apple, specifically designed for building user interfaces across Apple platforms. So, it was also very useful to adapt the app to the iPad screen. Moreover, choosing SwiftUI over UIKit, has made possible to take advantage of the latest advancements in Apple's UI development ecosystem.

3.3 Interface templates – iPhone

In this section, iPhone interface templates of all the screens of the applications are displayed and described, focusing on what happen when the user interact with the elements on the screen.

3.3.1 Login

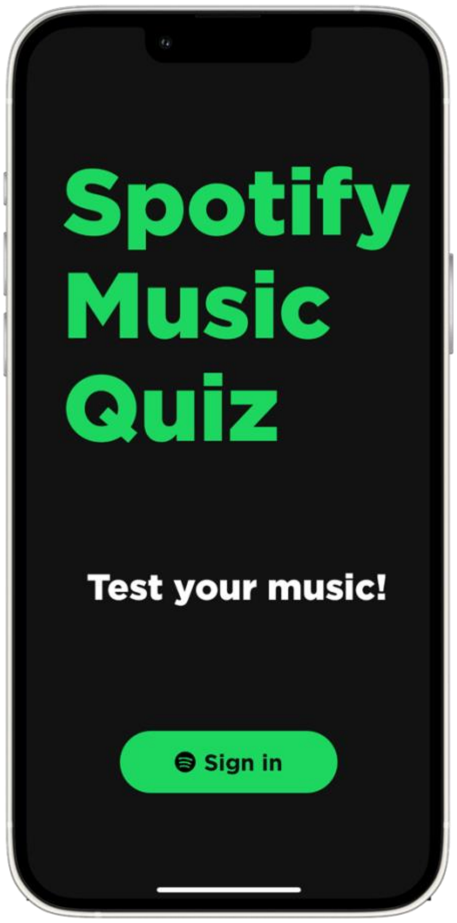


Figure 8: Login screen

This is the screen displayed when the application is opened for the first time or after a logout. The button allows to sign in using Spotify credentials. Once clicked the *Sign in* button, a *WebView* is open with the official Spotify log in page. It has not been reported since it has not been part of the design process of the app and it is the same for each application which implement Spotify Authentication.

3.2.2 Homepage

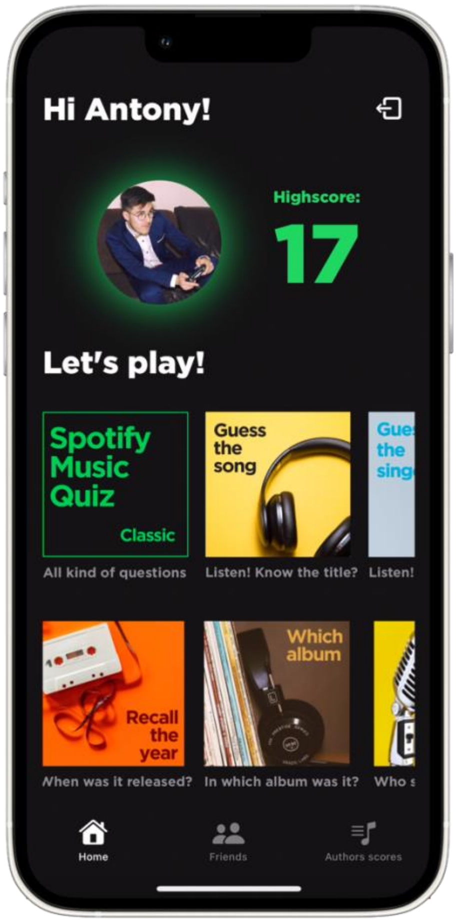
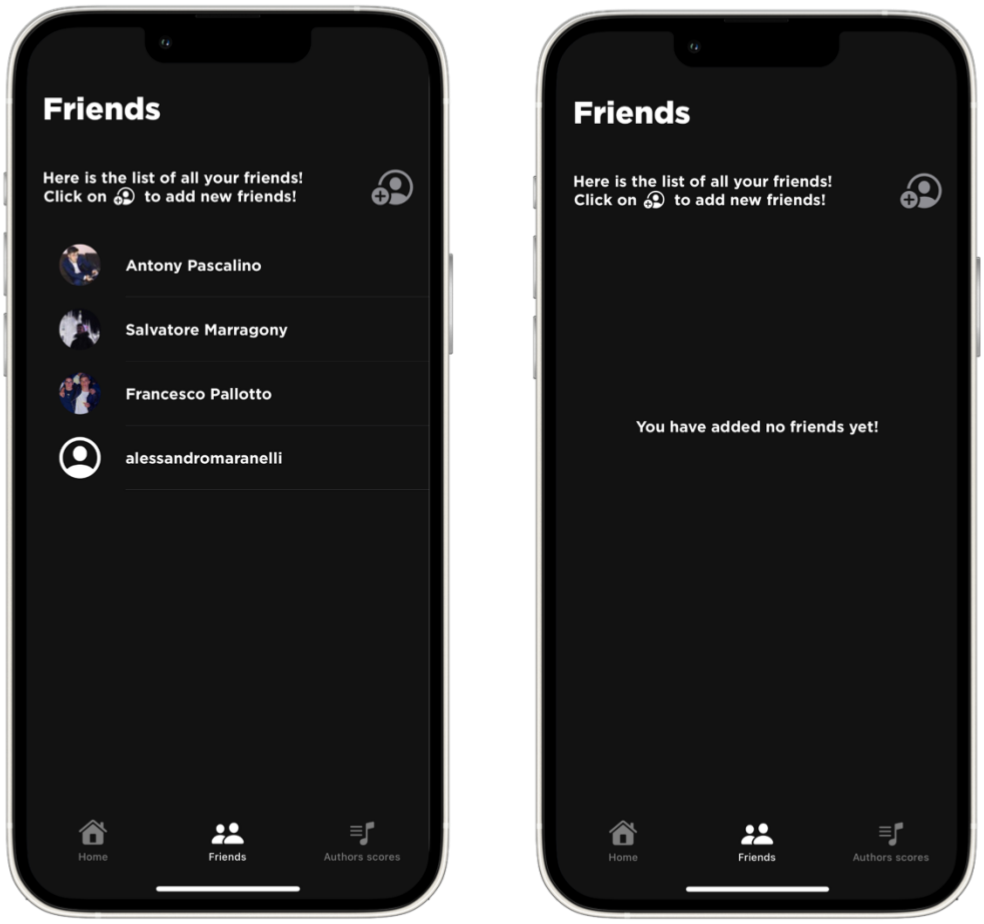


Figure 9: Homepage screen

Once the log in phase have been completed and the user correctly authenticated, the *Homepage* appears. On the top left corner, a text with the user nickname is showed. Next to it there is the log out button that once clicked makes appear the *LogOutAlert* to asks the user if wants really logout from the app. The Spotify profile picture of the user is showed next to the high score in the *Classic* game mode. Then two horizontal *ScrollView* allow the user to choose the Game mode that would like to play.
In the bottom part of the screen there is a *TabBar* that allows the user to jump to other pages, described next.

3.3.3 Friends


Friends

Here is the list of all your friends!
Click on to add new friends!

Antony Pascalino
Salvatore Marragony
Francesco Pallotto
alessandromaranelli

Friends

Here is the list of all your friends!
Click on to add new friends!

You have added no friends yet!

Figure 10: Friends screen

The second page reachable from the bottom *TabView,* is the *Friends* page. It simply shows the list of the current user friends, if the user already added any (left image). If the user has not added any friends yet a message informs of that (right image). In the top right corner, a button allows to add friends to the own list. Once clicked it brings to the *Add friends* page discussed below.

3.3.4 Add friends

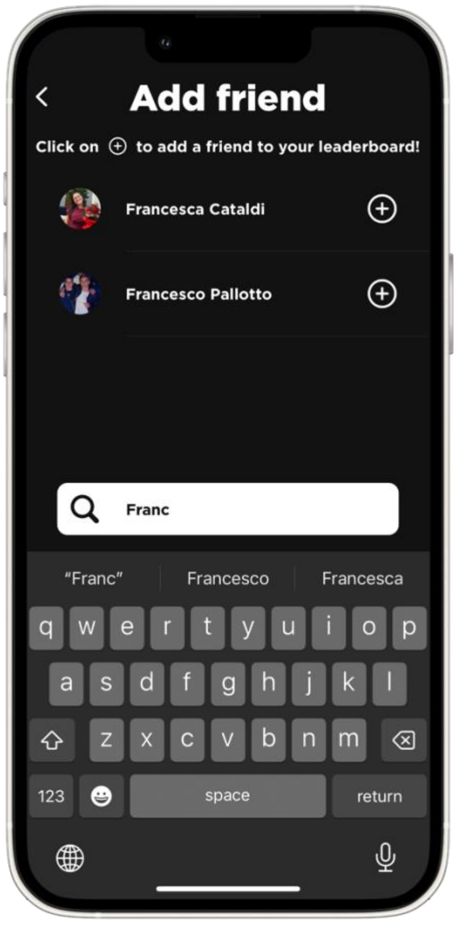
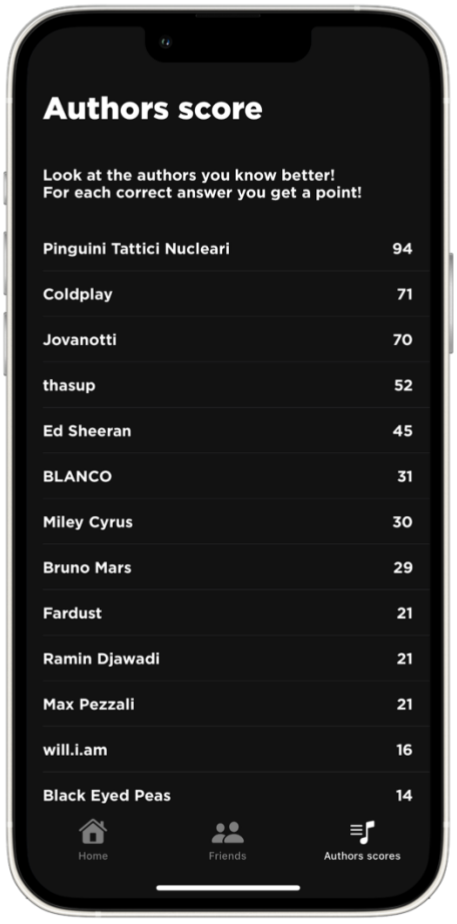


Figure 11: AddFriends screen

Across different pages of the app there is a button which allows to add new friend. Once clicked it bring to this *AddFriend* page. It shows a list of all the user that have ever logged into the *Spotify Music Quiz* app and that are saved on the database. Thanks to a button next to each user’s name, the current user can add new friends to the friends list. If a user in the list has been already added previously, such button will be disable and its appearance will be faded.
In the bottom part of the page, a search bar is available. It acts as a filter for the users list, so only the users that contain the search word in their name will be showed.

3.2.5 Authors score


Authors score

Look at the authors you know better!
For each correct answer you get a point!

Pinguini Tattici Nucleari	94
Coldplay	71
Jovanotti	70
thasup	52
Ed Sheeran	45
BLANCO	31
Miley Cyrus	30
Bruno Mars	29
Fardust	21
Ramin Djawadi	21
Max Pezzali	21
will.i.am	16
Black Eyed Peas	14

Figure 12: Authors score screen

The third and last page reachable from the bottom *TabBar* is the *Authors score* page. It shows the list of the authors that the player knows better, according to the correct answer given during the games for each author. Each time the user guesses a correct answer, gains a point for the author relative to the question. The screen is implemented using a vertical *ScrollView* and with a scroll gesture the user can browse all the authors for which has at least one point. If there is any author yet, a message invites the user to start a new game.

3.3.6 Game screen – Text question

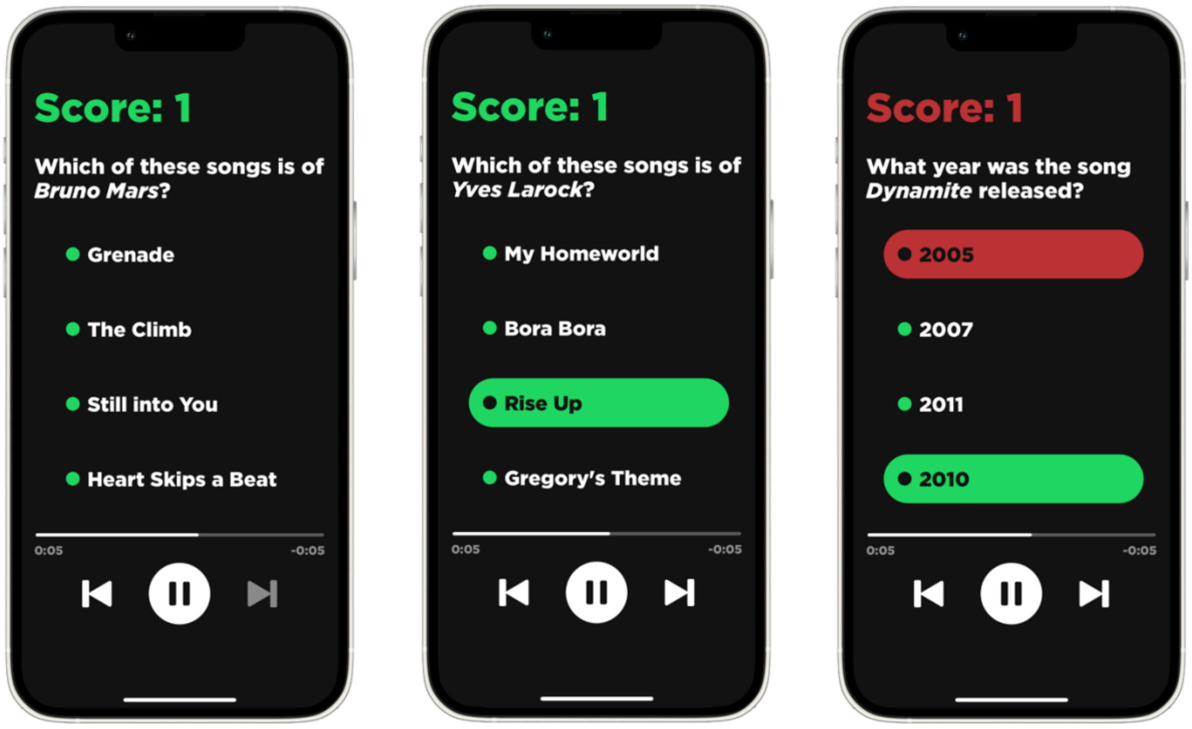


Figure 13: Game screen - Multiple choice question

The game screen appears once the loading is completed and the game actually started. It shows the current score of the game in the top part of the screen, below it the question text, the *GameControls* component in the bottom part and the *TimeBar* component over it. These elements are always present during each phase of the game, regardless of the specific type of question. The white line of the time bar flows towards the end, exactly like a music time bar in the Spotify app. On its left is put a timer that shows how may seconds are elapsed and, on its right, how many seconds remains to give an answer.
In this screen a multiple-choice question is showed. It consists of 4 *Answer* component of which only one of them represent the correct answer. If the user selects the correct answer, it becomes green. If the user select a wrong one, it become red and in the same instant the correct answer the becomes green. It is noted that also the color of the current score changes according to the correctness of the answer selected and that the Skip button is disabled before the user select an answer. Once it is enabled, it allows to go to the next question if the answer was guessed correctly, if not it leads to the *GameOver page.*

3.3.7 Game Screen – Listen question

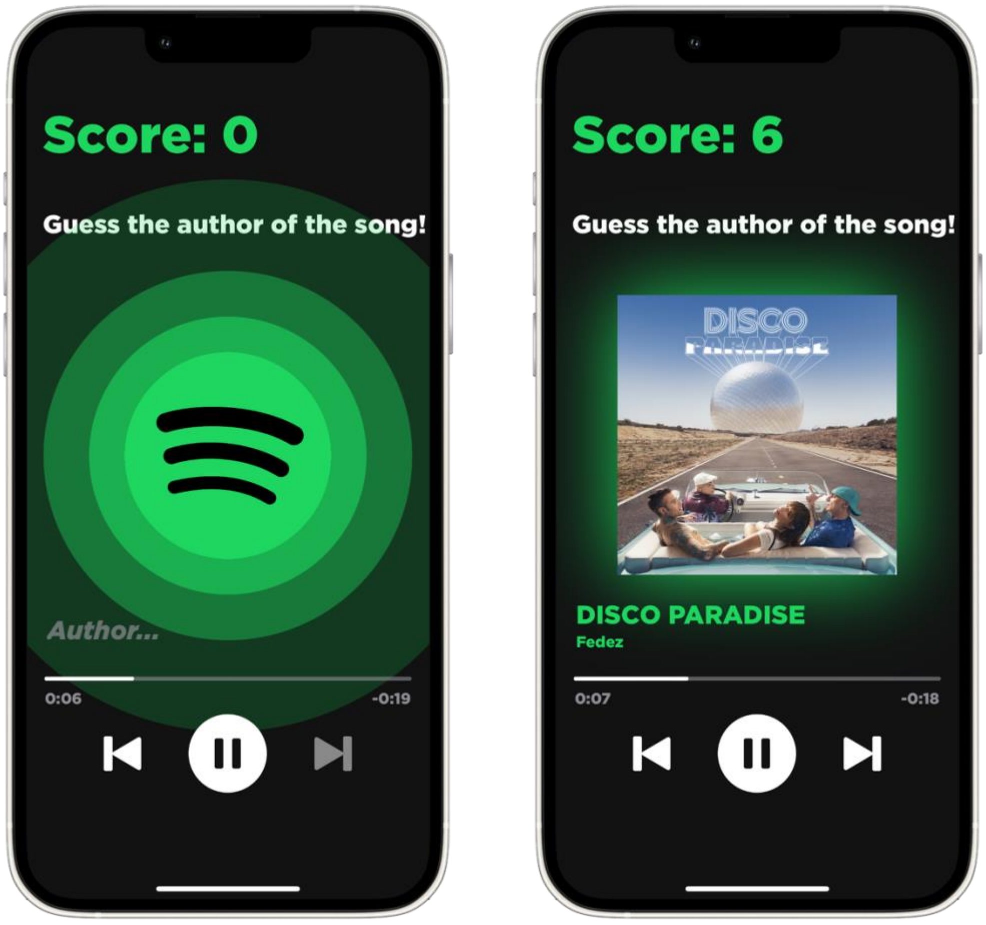


Figure 14: Game screen - Listen question

The other kinds of question that can be asked in a game are those in which the player is asked to listen to a song and guess the author or the title. Some elements remain the same of every other kind of question. The ones who change are the graphical ones and the answer. Graphically we have an animation that remember the one of the famous app Shazam in which we have some circles that enlarge and reduce their dimension (left image). Once an answer is given, the animation is replaced by the cover of the album of the song the player was listening (right image). Also, the song and author name appear once the answer is given. They will be green colored if the answer was true, red if was wrong.
To give the answer here, the player is asked to write the requested name with the keyboard. Once the user clicks on the text fields, it appears and allows the user to write the answer. Once the user press Enter in the keyboard the answer is given and checked.

3.3.8 Game Screen – Pause

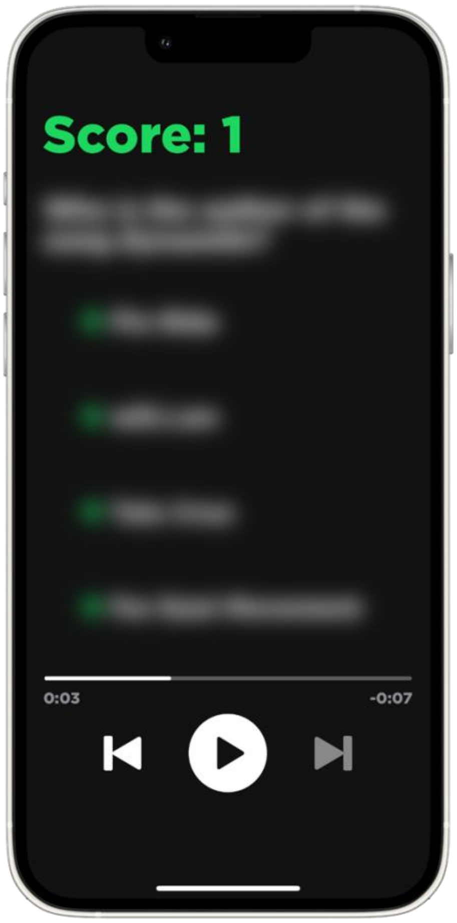


Figure 15: Game screen - pause

If, during the game, the user presses the pause button, the game actually stops. The timer stops running and the answers blur. While the game is in pause, if every song was playing, it stops and is not possible to click any answer. Once the user press play, the timer resumes, and the answers are again available to be clicked.

3.3.9 Alerts

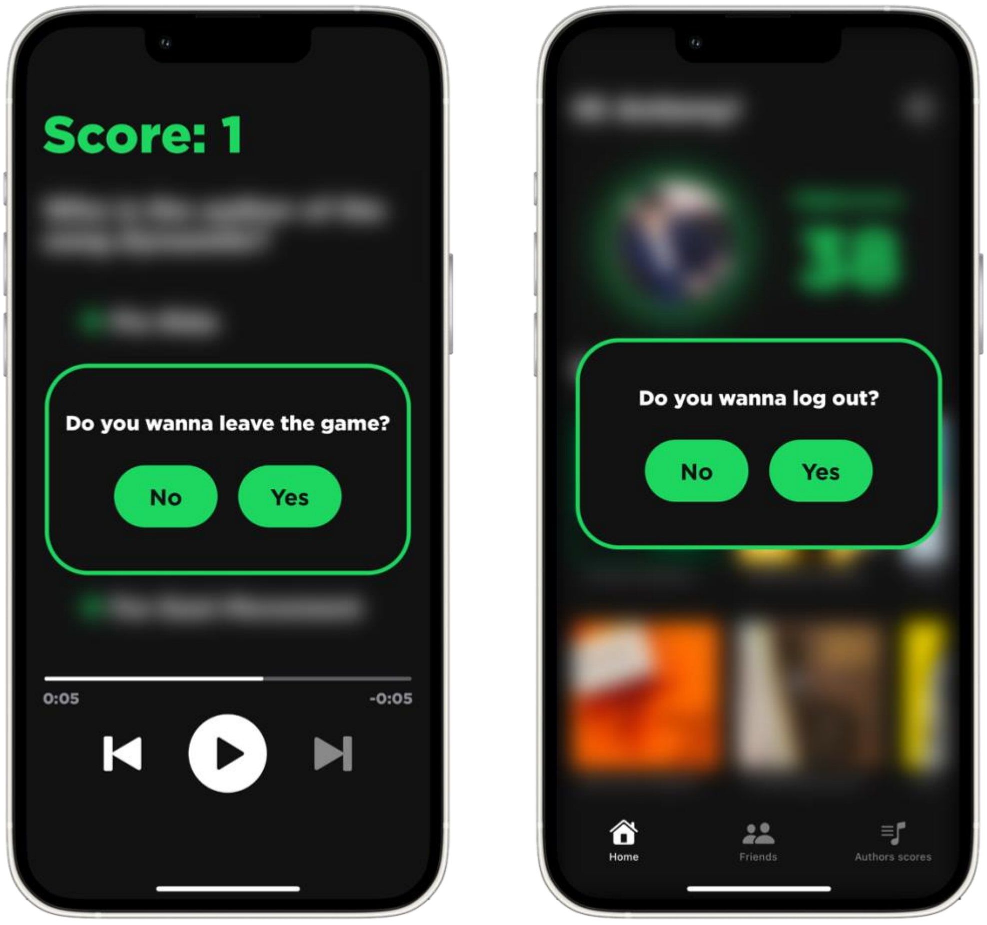


Figure 16: Custom alerts screens

Two custom alerts have been implemented. Both with the same purpose: asking the user if is really sure to go back, one for the log out, the other for leaving the game. If in the homepage the user clicks the button on the top right corner, the log out alert shows up. If during the game, the player clicks the backward button, the leave game alert appears. Both the alert, if the user selects No, disappear.

3.3.10 Game over

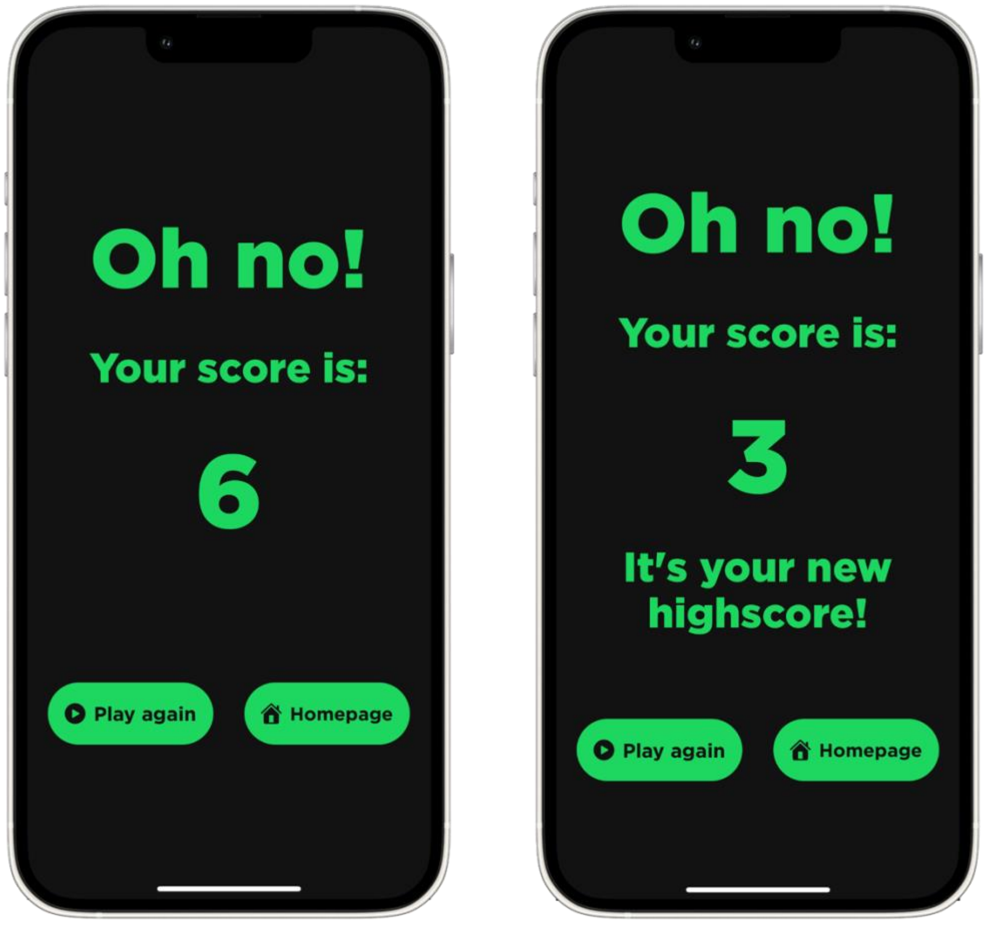
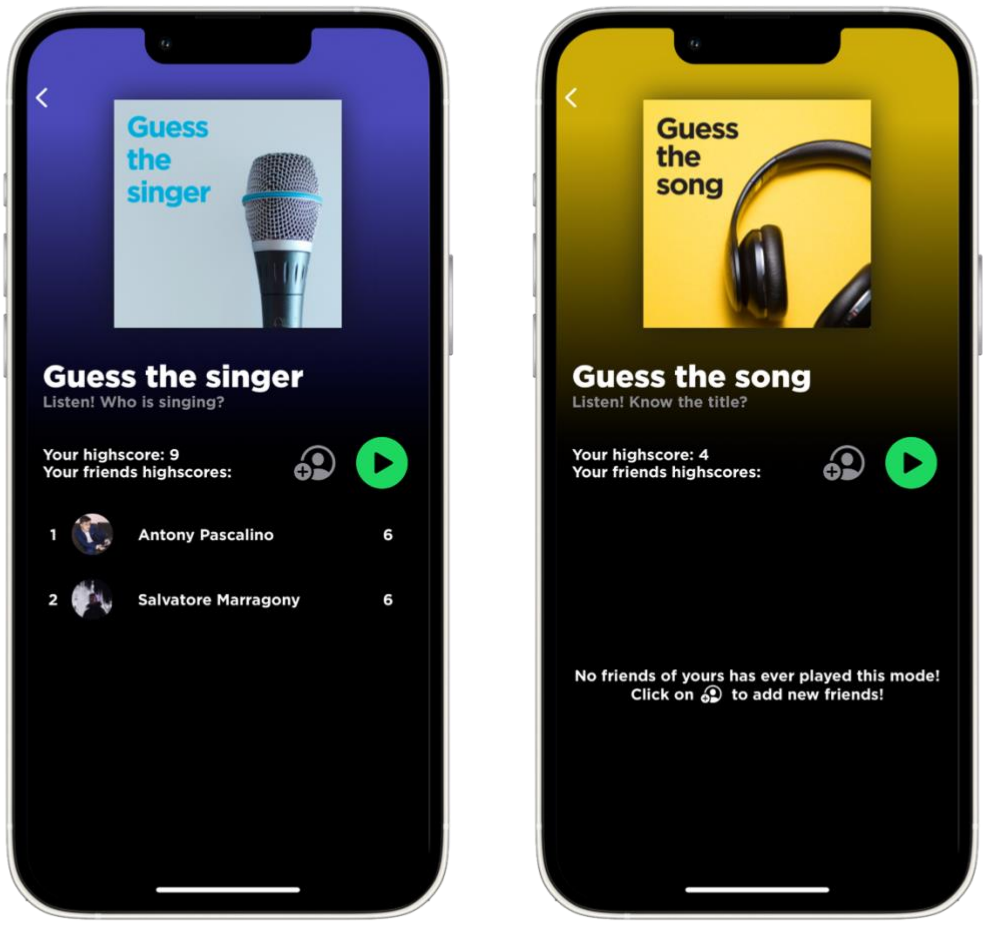


Figure 17: Game over screen

Once the user gives a wrong answer, the game over page appears. It shows the score performed by the player in the just over game and informs if the score is a new high score with a message. Two buttons are present: one allows to play a new game right away, the other to return to the home page.

3.3.11 Game Mode



Guess the singer
Listen! Who is singing?

Your highscore: 9
Your friends highscores:

1	Antony Pascalino	6
2	Salvatore Marragony	6

Guess the song
Listen! Know the title?

Your highscore: 4
Your friends highscores:

No friends of yours has ever played this mode!
Click on to add new friends!

Figure 18: Game mode screen

In the home page the users can select from a variety of game modes. Once they click on one of them, the relative game mode page is shown. Each game mode page has its own color and cover image. It shows also a list of the high score of the user’s friends in that specific game mode (left image). If no one of their friends have ever played that mode and has 0 as high score, the list will be empty, and a message informs the user of that (right image). Also the high score of the current player is written. To button are available in the page. The green play button allows to start a new game of that mode and the person-plus button allows to add new friends brining to the *Add friends* view previously discussed.

3.3.12 Loading

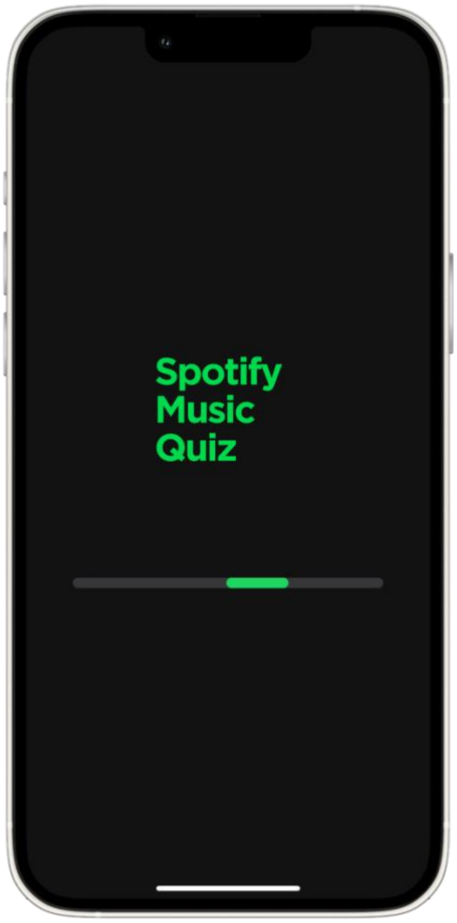


Figure 19: Loading screen

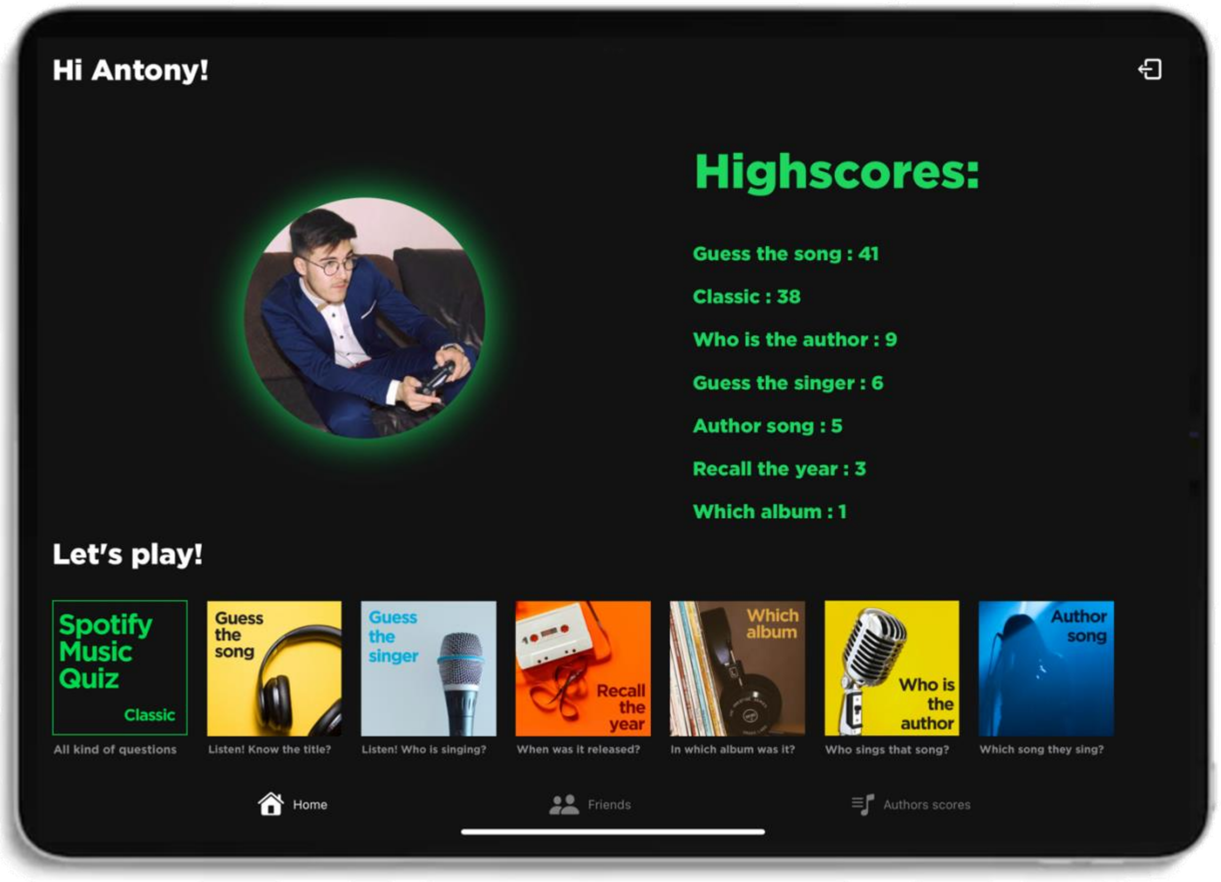
The loading screen appears whenever a loading is performed. Even if all the loadings happen on a background thread, it may happen that the user asks for the loading data before they are actually available. In those cases, the Loading screen appears. It happens for instance when the player wants to start a new game, but all the question has not been generated yet. The screen is composed by a simple loading bar and the logo of the *Spotify Music Quiz* app. They both are animated: the logo enlarges itself and reduces and the green loading bar goes from lefto to right and back.
Figure 13: Search Users Screen
This view is displayed right after clicking on the left-most icon of the bottom navigation bar. All the users, sorted by last access, are shown. The icons can be clicked for navigating to their profile page, otherwise the right bottom green icon can be clicked for checking the filtering options. The top search bar allows to search users by name.

3.4 Interface templates – iPad

For the iPad version of the application the design has been revisited. It has been adapted to the bigger screen of the iPad modifying the elements shown in the screen. Where it was possible, has been avoided the wrong “zoom” approach, in which the tablet version of the app is simply a scaled version of the smartphone one. Since the app usually don’t show so many information and is basically a game, some screen did not allow any “smart” revisitation of the UI. In those case the elements have been simply adapted to the bigger screen in a right way, always avoiding the bare scale effect. Often the official Spotify app for the iPad has been taken as inspiration to see how it was adapted to the tablet screen.

In this section the iPad interfaces are described, explaining how they have been adapted to a larger screen of a tablet. We omit the screens that are basically the same from the iPhone’s ones and focus instead on the different ones.

3.4.1 Homepage


Hi Antony!

Highscores:

Guess the song : 41

Classic : 38

Who is the author : 9

Guess the singer : 6

Author song : 5

Recall the year : 3

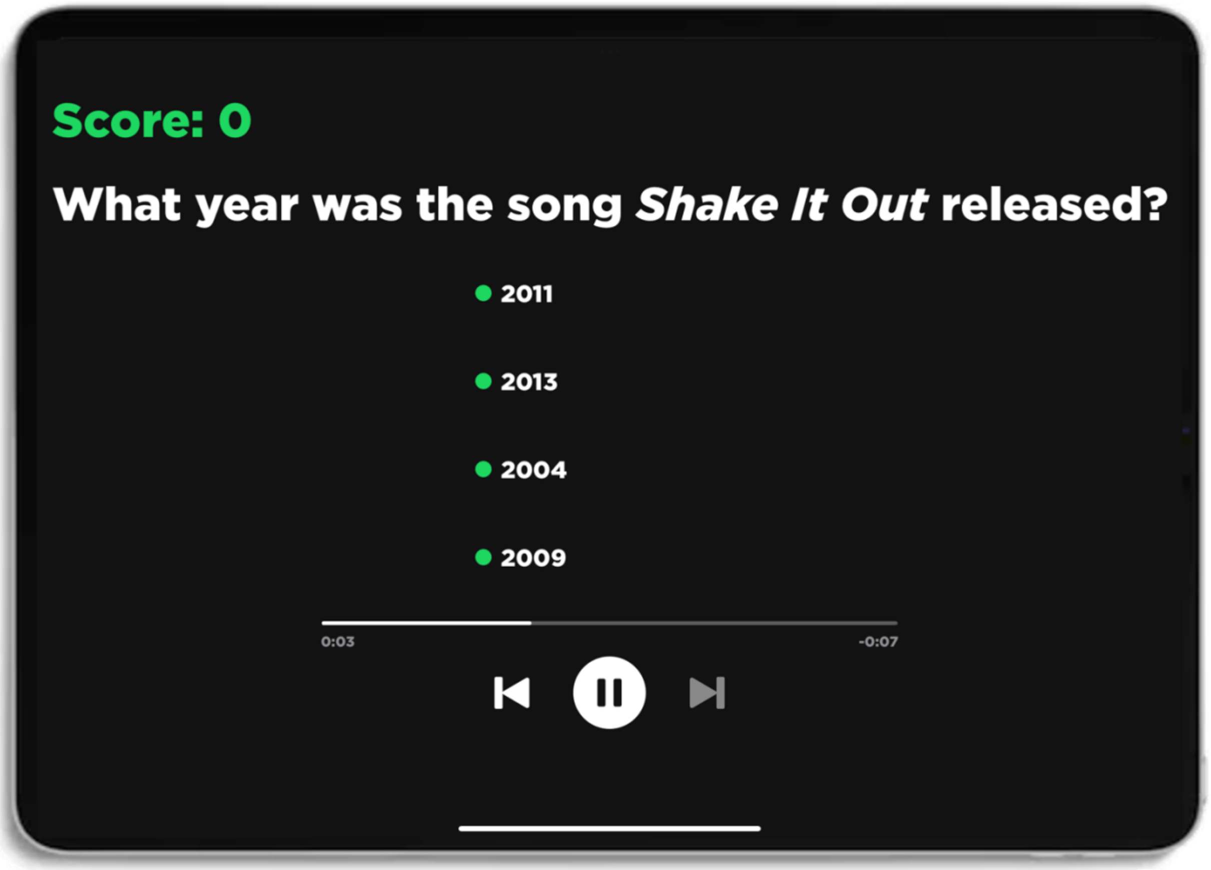
Which album : 1

Let's play!

Figure 20: Homepage screen - iPad version

The homepage has been adapted for the big screen of the iPad, with some adjustments. Thanks to the bigger screen of the iPad there is more space to show more information. Given that, instead to show only the high score of the *Classic* mode, it has been shown a list with all the high scores for each game mode.
More than that the profile picture size has been increased and the game *Modes* are now shown on a single row instead of two. In this way it is possible to see all the modes in single look and directly select the one the user wants without needing of scrolling.

3.4.2 Game – Text question


Score: 0

What year was the song *Shake It Out* released?

- 2011
- 2013
- 2004
- 2009

Figure 21: Game screen - Multiple choice question- iPad

This screen has been adapted following the Spotify official app in its iPad version. The time bar width has been increased, evaluating a tradeoff between the maximum width of the screen and its original width. Its greater width allows to have a better control on the time flow by the player. Thanks to the bigger screen size, the size of the text of the question has been increased for a greater readability since in the iPhone version sometime the question is a little bit stretched due to its length.

3.4.2 Game – Listen question

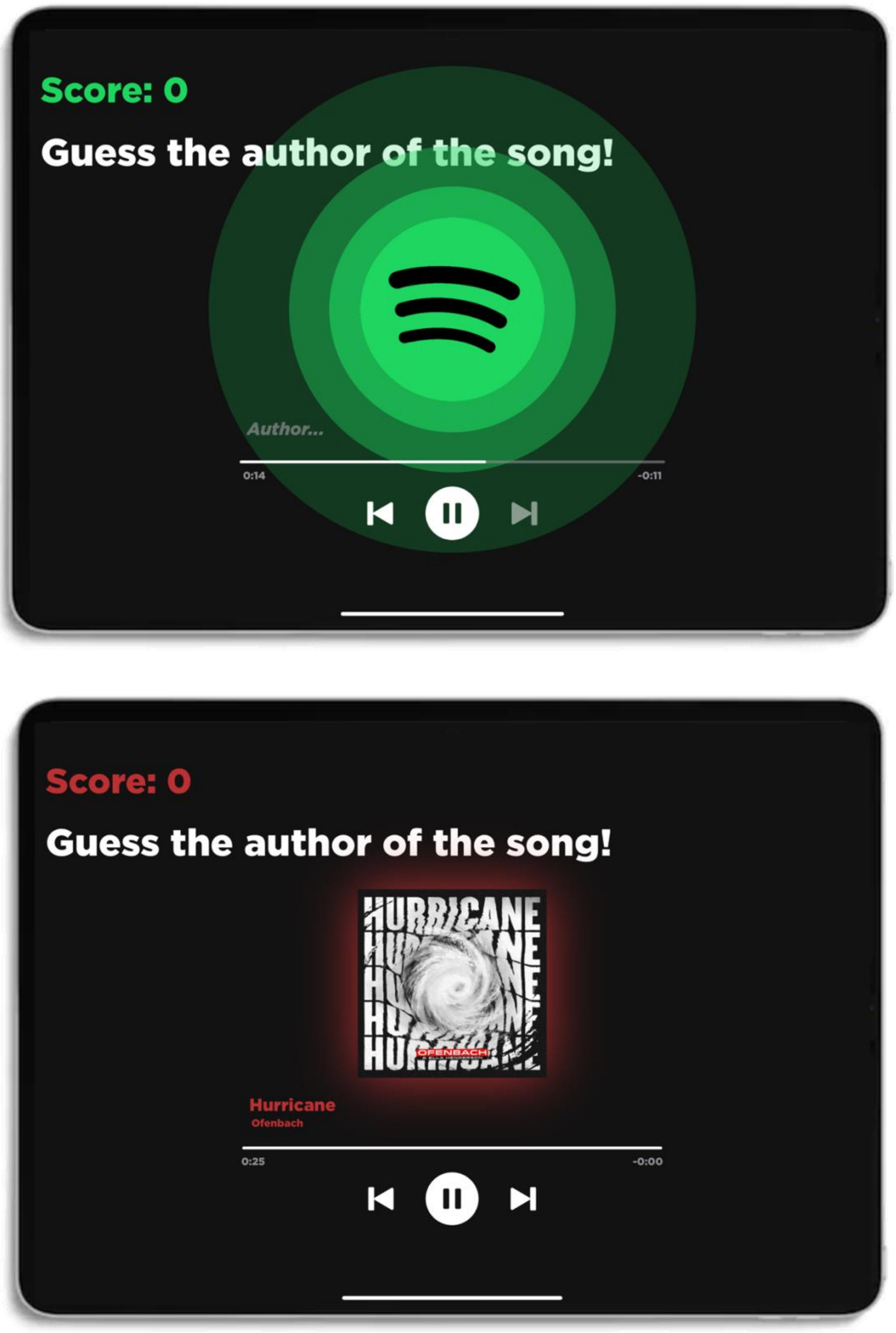


Figure 22: Game screen - Listen question - iPad

The listen question has been adapted as well. The question text and time bas adjustments previous discussed have been applied as well. Moreover, the animation

size has been increased. In this way the waves of the animation flow across the entire screen giving a sense of immersion in the music the player is listening to.
Also, the cover size of the album has been increased. Some covers have several details that are not so noticeable in a reduced screen size like the iPhone one. The bigger screen of the iPad allowed to overcome this problem.

4 Testing

4.1 Unit Testing

The unit testing has been an important part of the testing campaign since it allowed to test the business logic. These tests aimed to check the right functionality of the main functions of the application, in particular, the *gameManager* and *questionManager* classes since they encapsulate the main part of the app logic. These components often call methods of *APICaller* and *UserViewModel.* Since those methods perform call over the network, they have been "mocked" in order to return results without the need of a connection.
In the table below it is given a description of the tests performed, the problem faced, if any, and the solution applied in case of issues.

Test performed	**Problem found**	**Solution**
We tested the correct functioning of going to the next question and reloading new questions as soon as we got to the middle of the array of old ones.	Not perfect functioning due to threads, sometimes the question index was not correctly reset to 0	Use of DispatchQueue.async and a different execution order to allow all indexes to be set in time
Test that verified the correct generation of all questions together for the full mode	When there were just to generate all the questions, we encountered a problem. It did not enter the if any code, different from the others, was not passed.	We allowed all possible strings passed as queues that are not queues corresponding to any other mode to be valid to generate all questions for classic and full mode
Endpoint call to fetch user albums via Spotify API.	Some albums that came back didn't have tracks inside and therefore were unusable and weren't filtered	Added control on the presence of tracks also in albums

We wanted to test the correct functioning of the refresh token	Refresh token was not being refreshed because the boolean that controls this function was never set to true	Boolean set to true
We wanted to verify that the counting of correct answers was done correctly	When questions were regenerated then the count of correct answers was reset to zero	Code an increment of the exact answer count in `setNextQuestions()`
Check that in all the questions there were at least 3 questions per category	Any	/

Table 2: Unit tests performed, problems faced and solutions

4.2 UI Testing

Views have been widely tested one by one for making sure their functioning is the desired one. The *UserViewModel* class (that is our only ViewModel component) has been “mocked” for removing database queries and API calls. Instead, the correct behavior of the Views has been tested by injecting into them a pre-defined set of values (those that are normally retrieved from the database) and by asserting that the obtained layouts contain the expected set of Views.
The table below describes the most relevant tests performed, highlighting the problems they exposed. All the tests assert that the basic elements of the screen, were displayed correctly. The other assertions change depending on the specific screen being tested.

Test performed	**Problem found**	**Solution**
We tested various simulations of a game searching for some errors.	The test allowed to give several right answers in a faster way of a user. Given that the questions are loaded in batch, if the correct answers were given in a faster way, it caused an array out of bound since there were not enough time for generate the questions of the next batch	The batch size has been increased to 20 question per time avoiding any index out of bound and allowing fast loadings.
We tested the right functionality of the Friends page. We asserted that the list of friends was right showed and refreshed when interacting with the *UserViewModel*	It has been noted that if a new friend was added, the change was not reflected right away in the View	Once the method `addFriend()` of the *UserViewModel* was called, also its method `updateUser()` needed to be called
Has been performed a test on the right behavior of the	None, since the message was actually showed.	/

Authors scores and Friends list when the relative lists are empty. If the lists are empty the views should show a message informing the user.		
Each question has a background song retrieved from the Spotify preview relative to the specific question song. If the question song does not have a preview, the default song has to be played.	The songs actually did not start one after the other. If one was playing, another song did not load.	The *AVPlayer* needed to be stopped before starting to reproduce another question.
On the *GameOverView* the score of the game has to be showed together with a message if it is a new high score. If it is, the view also calls the method to update it on the server	The message did not appear, and the high score was not updated.	The method call to check if the score is high score was on a wrong view, so once the *GameOverView* appeared it was not called at all.

Table 3: UI Tests performed, problems and solutions

4.3 User feedback

The application has been used by a set of selected students for a limited amount of time. Their feedback on the usability of the application together with the intuitiveness of the user interface has been collected and considered for a proper re-arrangement of the screens' layout.

4.3.1 Feedback on UI

User's feedback was particularly useful for detecting and solving issues related to the User Interface. Some design choice made during the implementation phase turned out to be not so intuitive for a new user. Thanks to the feedback given by some students we have been able to improve the user interface making it more immediate and easier to use. In the table below we present a list of the interface issue that have been pointed out by the students and the relative adjustment made.

Interface issue	Adjustment
Users pointed out that the list of game modes in the homepage was not self-explicative. It was not actually clear in which kind of questions, a specific mode, would have consisted without starting a new game.	Under each game mode image in the homepage, has been added a description about the questions proposed in a specific mode.
In the Friends page, once the users log in for the first time, their friends list is actually empty and a button to add new ones was available in the right top corner. Some testers did not notice that button and seeing a black empty screen had the impression that something did not work.	When the friends list is empty, has been added a message which inform the user that has no friends yet and inviting to add new ones clicking the button on the top right corner.
In the Authors' scores page, like the previous problem, on users' first login the list was empty. Worse than previous problem, here there was no button to do anything, and testers did	It has been added a description above the screen explaining how those points are gained and if the list is empty, it is explained that the user has not guessed any question yet and invite them to

not know what to do to see something appears.	play a new game to score new authors' points.
In the game over page the score of the just finished game is showed. The testers pointed out that it was not said if the score was actually their new high score or not and to check it, they were forced to go back to the home page, open again the previous game mode page and look at the high score. Sometimes even without recall if it was a new high score or was actually the same as before.	Has been added a message that informs the user if the score of the just finished game was a high score or not.
Some testers found the game controls not intuitive. In particular the backward button was not self-explicative on its function and once clicked kicked out the player from the current game.	An alert has been added once the user presses the backward button asking if is sure to leave the game. In this way even if the player has doubts on its functionality it is caught by reading the alert message.
The pause button was actually self-explicative, but once pressed it only stopped the timer allowing the player to still read the question. Tester did make notice that it was too easy and sounds like cheating.	When the user pauses the game, the question actually blurs making it impossible to read the question and the answers. Moreover. the background music stops avoiding cheating on the listening question.
The forward button was quite explicative on its function. But for some tester it was not correct that when pressed before giving the answer it did nothing although it seemed active.	Before giving the answer, the forward button is greyed out giving the idea of being disabled. Once the user gives the answer it becomes white giving the idea of being active.

Table 4: Issues found by tester on UI and its adjustments

4.3.2 Feedback on the game

We asked the testers some feedbacks on the game system too in order to tune its difficulty and enjoyability as well as errors. These were actually really important as tests since we as developer were not able to well test the game difficulty. In the table below we describe the problems found and its solution.

Game issue	Solution
Testers noted that when reaching a high score in the game, some questions tended to reappear in the same game instance	We implemented a check on the questions when they are put in the questions array removing duplicates.
How explained in the document, the wrong answer is related to the right answer in order to not make the question trivial. Sometime some inconsistencies were found. e.g., in the question “Which of these songs is of this author?” some wrong question may be related to the right one but at the same time belonging to the same author creating for the player more than one correct answer.	When the wrong answers are generated a check on the author is done and if it is the same of the correct answer, it is removed. As well is done also with related album and artists.
Each question has a timer, and the question needs to be given before the time expired. Originally the timer was the same for each question, but some users found it insufficient in the listen question since they did not have the time to write the answer.	The timer is now different between the multiple-choice questions and the open text question. 10 seconds for the former and 20 seconds for the latter. Even the pause allows to have more time if needed.
For the enjoyability of the game some users made us notice that some questions they faced were about song they never listened to. It turned out those song was in playlist the saved. Sometimes those playlists were even	To generate the questions, we chose to consider only the playlist created by the user and not those simply added to their library. We assume that if users create a playlist they know the songs they added into it.

about relaxing music that they used to listening to while studying.	

Table 5: Issues found by testers on the game