# PROJECT_REPORT_ANA_LUIZA_CASTRO

*by* ANA LUIZA SILVA DE CASTRO

# Audio exchange application to tackle loneliness of older adults

**Ana Luiza Silva de Castro**

**13807717**

MSc Computer Science project report

Department of Computer Science and Information Systems,

Birkbeck College, University of London

**2022**

## SUMMARY

## Table of Contents

## TABLE OF FIGURES

## TABLE OF TABLES

## DEDICATION

To my grandmother,
whose kind stories and warm cakes
are in my most happy memories of childhood.

## ACKNOWLEDGMENTS

I would like to thank God, my husband, my family, and my supervisor for their support in the journey of writing this dissertation.

## 1 ABSTRACT

The number of older adults increases yearly, and they should account for more than 20% of the population by 2050. These people are more prone to feel alone for many reasons. For example, their children have already gotten their own families and responsibilities. COVID-19 increased even more in this condition because the elderly were the most fragile to the virus and had to follow stricter isolation rules. In this context, this project is related to developing an application to provide a means of communication between the elderly and younger generations through storytelling, to reduce the feelings of loneliness, anxiety, and sadness of older people. Many improvements can still be made to the application, and it is intended to tackle them shortly.

Supervisor: Professor Peter Wood.

# 2 INTRODUCTION

This report chapter describes the application's context and the motivation for its development.

## 2.1 Context

Loneliness is a familiar feeling for many older adults, and the number of seniors worldwide tends to increase yearly. Human beings are gregarious by nature. That is, it is part of human nature to participate in social groups, which is closely linked to the evolution of the species. In this way, the lack of social interaction can cause severe emotional problems, such as depression, and even lead to more tragic fates, such as suicide.

Some reasons for the decrease in the social contact of the elderly individual can be divided into three levels. The familiar one: the children have grown up and have their own families and responsibilities; the cultural one: there is little intergenerational contact, especially in western societies; and the social one: there is less responsibility from the community on older members.

In addition to the problems already mentioned, the COVID-19 pandemic has added yet another layer to the isolation of the elderly: health. Because they are the population most vulnerable to the virus, the elderly have had to submit to stricter isolation rules than the rest, aggravating the problem of loneliness among these individuals.

One of the possible solutions for this situation would be using digital technologies to facilitate contact with this group, which studies have shown to reduce stress and anxiety (Naeim et al., 2021). However, older people generally have little or no familiarity with new technologies. That is, they go through a process of double exclusion: not only in the real world but also in cyberspace.

Thus, a way to facilitate the interaction of the elderly public with new technologies can be through a design specifically made for the particular characteristics of this population. These characteristics can be physical: such as loss of visual acuity or decreased hearing capacity, and they can also be psychological: such as a greater fear of making mistakes when dealing with electronics and a feeling of frustration.

This report is divided into five parts: Introduction – which provides context; Specification – which contains information about requirements, personas, and user stories; Design – which explains the design methodology, the software architecture, and the user interface design; Implementation – which relates the tools and languages used, and each stage of the project implementation; and Testing and Evaluation, that clarifies decisions taken about the testing plan, the achievements and drawbacks of the process of developing the application. The complete background research for the project can be accessed in Castro (2022).

## 2.2 Motivation

It is in the scenario described above that the Sabiá app was designed to increase the well-being of the population in question. There are three primary motivations for the creation of Sabiá.

The first is personal: I am a Brazilian immigrant in the United Kingdom, where I came to study for this MSc in Computer Science. I left all my family in Brazil: dog, sister, father, mother, and grandmother. My grandmother and I are very close, and I know she misses me as I miss her. As my grandmother is of age and entirely technologically illiterate, I have had difficulty communicating with her, which is the primary motivation for creating this application.

The second is social: Thinking about this issue, I realized that it was not just my grandmother who felt this way. And I decided to research more in-depth about loneliness in old age, from which the contextualization provided above comes. Also, the issue of COVID-19 was a remarkable experience for all the people who went through such a drastic global event, separating families worldwide.

The third is political: The president of Brazil at the time of writing this text is Jair Bolsonaro, an extreme right-wing politician who has ruled the country since 2019. The extreme right has taken possession of national symbols such as the flag, the colors green and yellow and the football team shirt. But there was still a national symbol that remained intact to this illegitimate appropriation: the Sabiá. Sabiá, in Tupi, a Brazilian indigenous language, means the one who prays a lot and is the bird symbol of my country.

A famous poem puts this bird as one of the main nostalgias of a Brazilian in foreign lands. For clarification, the poem makes more sense in Portuguese because it rhymes.

*Exile song*

My land has palm trees
Where the Sabiá sings,
The birds that chirp here,
They don't chirp like there.

Our sky has more stars,
Our floodplains have more flowers,
Our woods have more life,
Our life more loves.

In brooding, alone, at night,
More pleasure I find there;
My land has palm trees,
Where the Sabiá sings.

My land has primes,
What such I can't find here;
In brooding – alone, at night –
More pleasure I find there;
My land has palm trees,
Where the Sabiá sings.

God doesn't allow me to die,
Without my going back there;
Without enjoying the primes
That I can't find around here;
Without ever seeing the palm trees,
Where the Sabiá sings.

*Gonçalves Dias*

In short, Sabiá is a technical creation but also a social, political, and personal one. And as a developer, I hope it can represent hope for a more humane world, a more democratic Brazil and more integrated elderly individuals, including my grandmother.

# 3 SPECIFICATION

This section of the dissertation addresses the functional and non-functional requirements with personas and user stories.

## 3.1 Non-functional Requirements

The non-functional requirements are described in Table 1. For the full list of non-functional requirements with the final evaluation, please consult Table 13 in appendix 9.1.1.

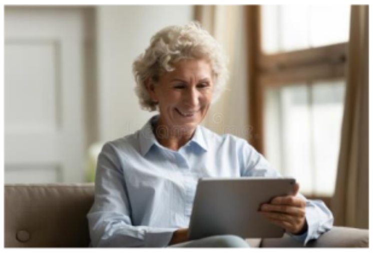
*Table 1 – Non-functional requirements.*

| **Requirement ID** | **Requirement Statement** | **Category (Must-have / Nice-to-have)** |
|---|---|---|
| NFR01 Time | The app, the documentation and the report shall be ready by 19/09/22. | Must |
| NFR02 Usability | User interface: shall be user-friendly. Users shall be able to navigate the app without external help. | Must |
| NFR03 Security | - Login;<br>- Password. | Must |
| NFR04 Documentation | - System documentation;<br>- Training material (manual). | Must |

## 3.2 Personas

The personas are as described in Table 2:

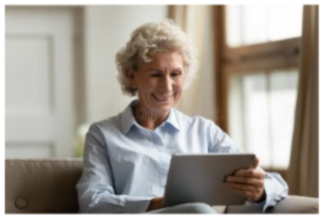
*Table 2 – Personas.*

| **Name / Picture** | **Details** | **Goal** |
|---|---|---|
| Joanna  | Joana wants to use the app to hear her granddaughter's stories. | "I want to feel loved". |
| Beatriz | Beatriz wants to record stories for her grandma to listen to them. | "I want to show how much I love grandma even when I am far away". |

## 3.3 User Stories

The user stories are as described in Table 3:

*Table 3 – User stories.*

| **Beatriz** |  **Joanna** |
|---|---|
| As a recording user, I want to record stories to communicate with my grandmother. | As a listening user, I want to be able to hear the story my granddaughter sent me so I do not feel alone. |
| As a recording user, I want to have a login to enter the app. | As a listening user, I want to have a login to enter the app. |
| As a recording user, I want to save my story so I do not lose what I recorded. | As a listening user, I want to enter the app without difficulty hearing the story. |
| As a recording user, I want to share my story with my grandmother so she can hear it. | As a listening user, I want a clean and user-friendly interface so I can browse without help from someone else. |

## 3.4 Functional Requirements

The functional requirements are described in Table 4. Please consult Table 13 in appendix 9.1.2 for the full list of functional requirements.

*Table 4 – Functional requirements.*

| **Requirement ID** | **Requirement Statement** | **Category (Must-have / Nice-to-have)** |
|---|---|---|
| FR01 | The app shall have a set-up page. | Must |
| FR02 | The app shall have a page for the user to choose between listening to a story or recording a story. | Must |
| FR03 | The app shall have a Library page for the user to select a story to listen to. | Must |
| FR04 | The app shall have a story player page. | Must |
| FR05 | The app shall have a recording page. | Must |
| FR06 | The app shall have an "about your story page" so the recording user can add information about the story recorded. | Must |
| FR07 | The app shall have a "Send a story to" page so the user can select the recipient for the recorded story. | Must |
| FR08 | The app shall have a navigation menu. | Must |
| FR09 | The app shall have a chat page listing all chat conversations. | Nice |
| FR10 | The app shall have a user profile page. | Must |
| FR11 | The app shall have authentication persistence, so users do not have to log in every time they close the app and reopen it. | Nice |
| FR12 | Error case: the user email input is not registered.<br>The field for email input should become red, and an error message should be displayed at the base of the field with the content: "There is no account with this email". | Nice |
| FR13 | Error case: the user email input is not valid.<br>The field for email input should become red, and an error message should be displayed at the base of the field with the content: "Insert a valid email address". | Nice |
| FR14 | Error case: the password is incorrect.<br>The password field turns red, and an error message appears at the base of the field with the content: "Incorrect password". | Nice |

# 4 DESIGN

This chapter of the report concentrates on the design methodology of the application, its software architecture, and user interface design.

## 4.1 Design methodology

For the current project, the design methodology chosen was agile, from the iterative category. As just one person is doing the job, none of the team tools available was used, for example, ceremonies of daily scrum or sprint review. There are two reasons why this choice was made: most technologies used were previously unfamiliar, and the requirement for delivering software that works even with limited functionalities.

For that, some SCRUM principles were used. One of them was the product backlog that helped to keep the focus on the main priorities and to build them in order. With this one, the backlog refinement was also used to organize and administrate the product backlog. Another one was timeboxing which helped to prioritize actions and to avoid perfectionism. Finally, the refactoring technique was also used to keep the code more maintainable and straightforward, which helped to modify its preserving functionality.

## 4.2 Software architecture

The software architecture of the application includes an overview and the main parts of the software with their relationships.

### 4.2.1 Overview

The software architecture chosen for the application development was the layered architecture (Figure 1). The main reason for this choice was that it has a high level of two crucial patterns: testability and ease of development. Both are important for bringing not only easier testing but also simplicity. The application is a classic 2-tier architecture.

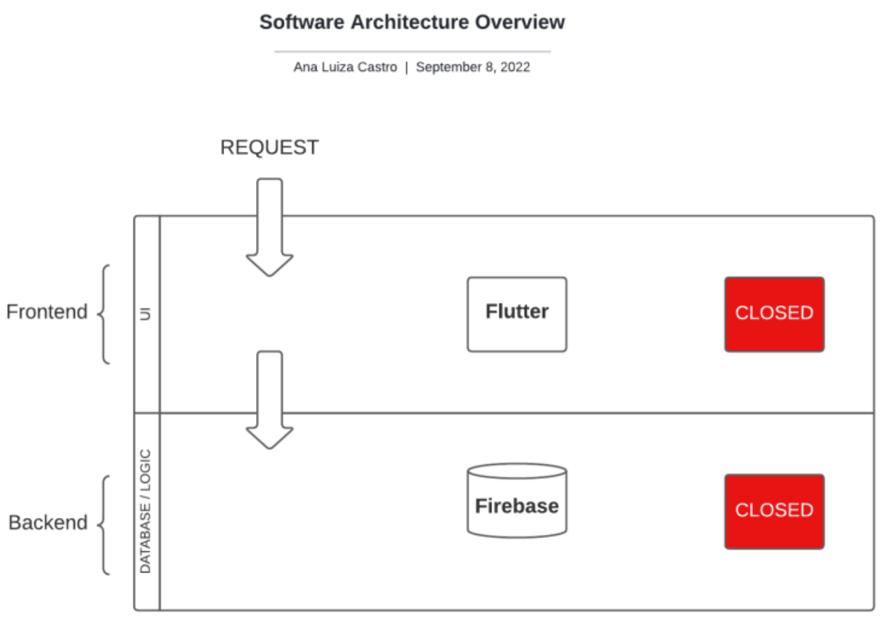


*Figure 1 – Software Architecture Overview.*

### 4.2.2 Flutter Software Architecture

Flutter also has a layered architecture (Figure 2). According to the Flutter developers' team: "It exists as a series of independent libraries that each depend on the underlying layer. No layer has privileged access to the layer below, and every part of the framework level is designed to be optional and replaceable" (Flutter architectural overview).

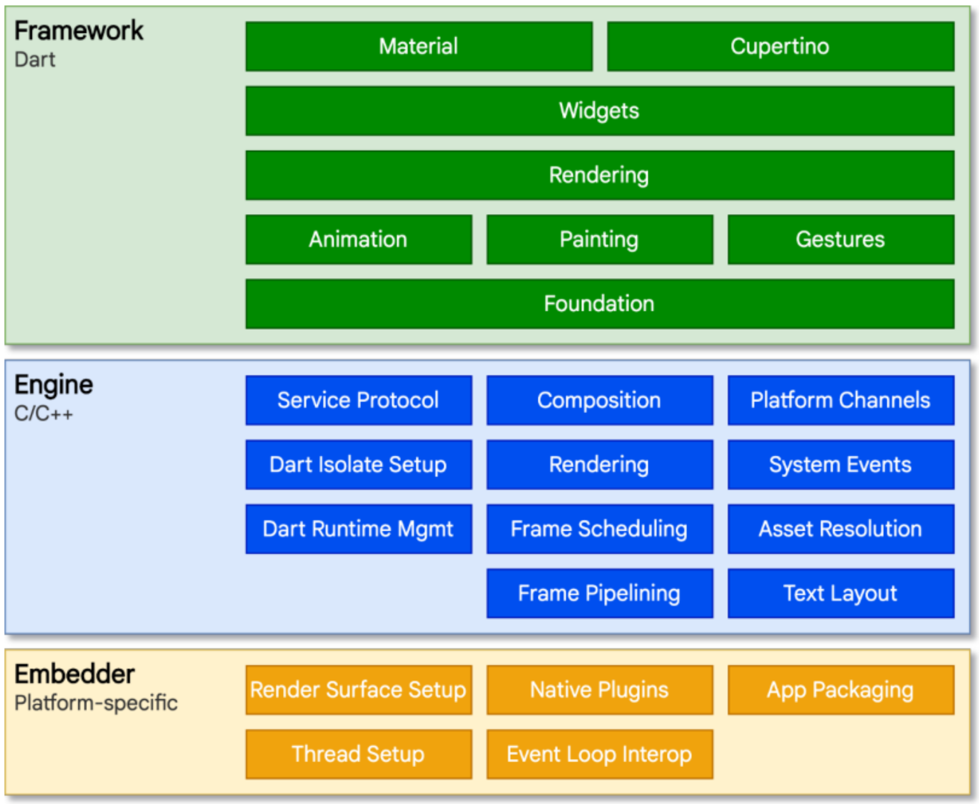


*Figure 2 – Flutter Software Architecture.*

*Source: Flutter, 2022. Available in: https://docs.flutter.dev/resources/architectural-overview.*

### 4.2.3 Database design

Figure 3 shows a diagram of the database structure implemented.

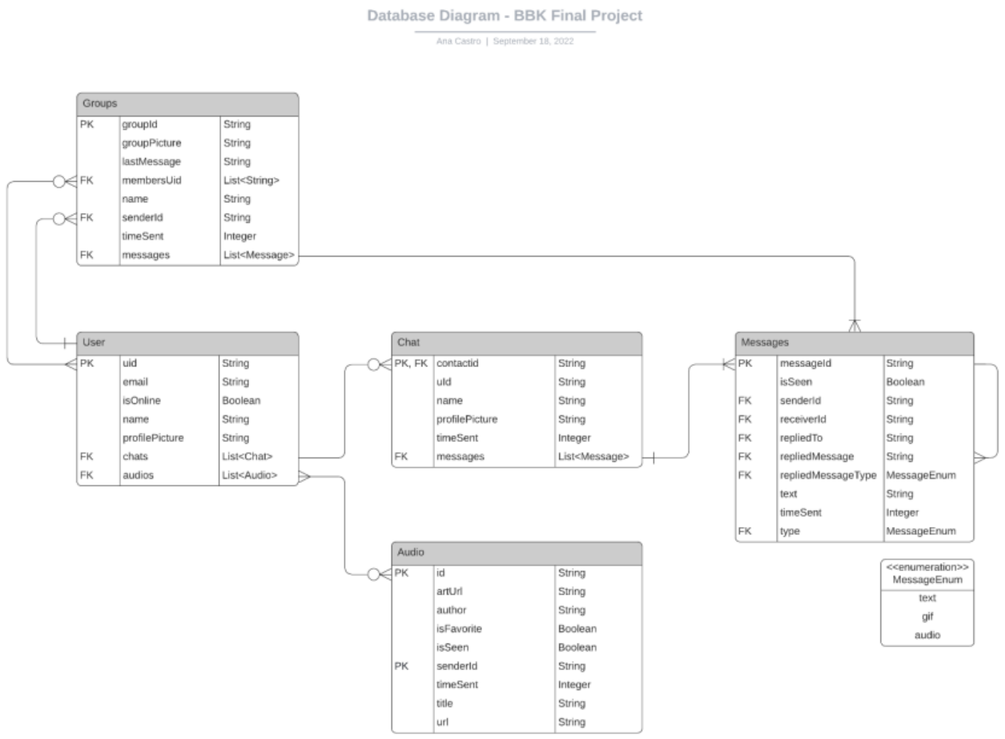


*Figure 3 – Database diagram.*

The relationships shown in Figure 3 are detailed below:

- A user can be the creator of multiple groups, but a group can only have one creator.
- A user can belong to zero, one or many groups, and a group cannot contain less than two users.
- A group can have from zero to many messages, but a message can only belong to a single group.
- A user can have from zero to many chats, but each chat is only associated with a single user and their contact (opposite user or group).
- A chat can have from one to many messages, but a message can only belong to a single chat.
- A user can have from zero to many audios (received or sent), and audio can belong to many users.

The description of each database table field can be seen in tables from 5 to 9.

*Table 5 – Description of the fields of the User data table.*

| **User** | | |
|---|---|---|
| **Field** | **Type** | **Description** |
| audios | String | The email address of the user. |
| isOnline | bool | It records whether the user is online in the chat. |
| name | String | The name of the user. |
| profilePicture | String | The URL of the user's profile picture. |
| uid | String | The id of the user. |
| chats | List<chats> | The list of chats associated with the user. |
| audios | list<audios> | The list of audios associated with the user. |

*Table 6 – Description of the fields of the Chat data table.*

| **Chat** | | |
|---|---|---|
| **Field** | **Type** | **Description** |
| contactId | String | The id of the group or opposite individual. |
| lastMessage | String | The text content associated with the last message. |
| name | String | The name of the group or opposite individual. |
| profilePicture | String | The profile picture's URL of the group or opposite individual. |
| timeSent | int | The date and time of the last message in milliseconds since the Unix epoch (1970-01-01-00:00:00). |
| messages | List<Messages> | Messages collection associated with this chat. |

*Table 7 – Description of the fields of the Audio data table.*

| **Audio** | | |
|---|---|---|
| **Field** | **Type** | **Description** |
| id | String | The id of the audio. |
| artUrl | String | The URL of the image of the story cover. |
| author | String | The name of the author of the story. |
| isFavorite | boolean | Records whether this audio was marked as favorite. |
| isSeen | boolean | Whether this audio was listened to by the receiver. |
| senderId | String | The user's id of the sender. |
| timeSent | integer | Date and time in which the audio was sent in milliseconds since Unix epoch (1970-01-01-00:00:00). |
| title | String | The title of the story. |
| url | String | Url of the respective audio file. |

*Table 8 – Description of the fields of the Message data table.*

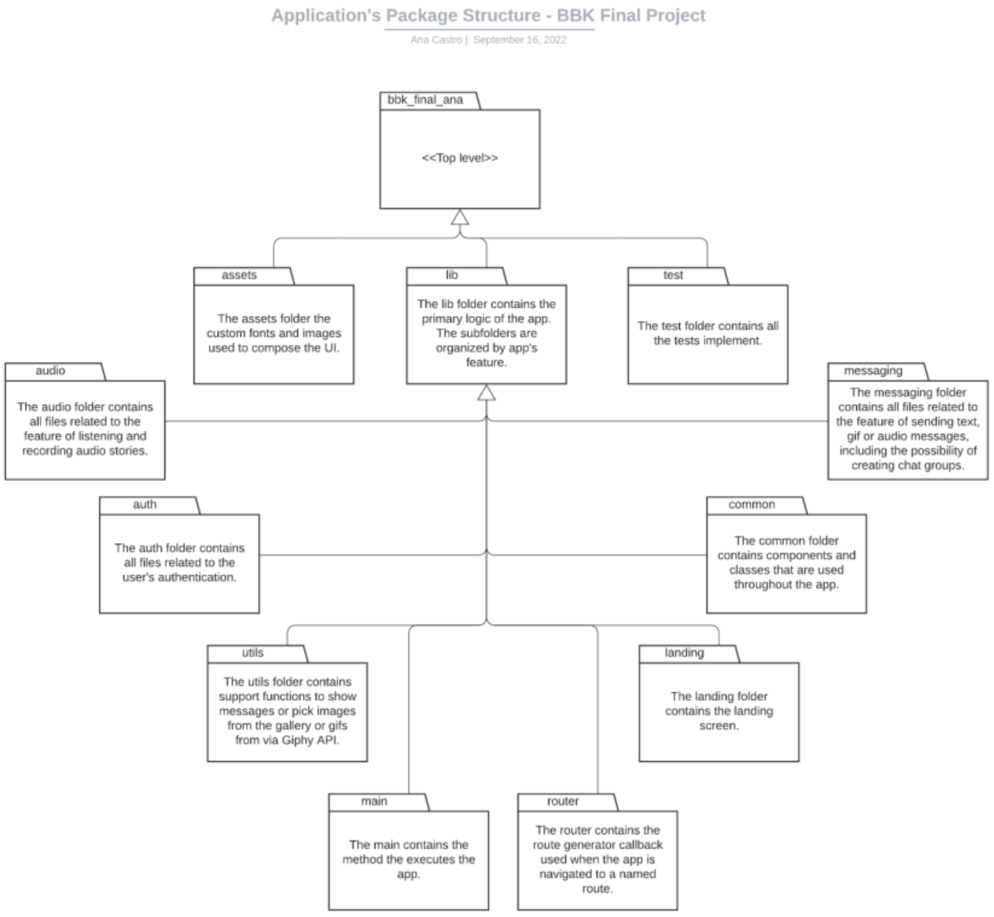
| **Message** | | |
|---|---|---|
| **Field** | **Type** | **Description** |
| isSeen | bool | Registers whether the message was seen by the receiver. |
| messageId | String | The id of the message. |
| receiverId | String | The user's id of the receiver. |
| repliedMessage | String | The id of the message that was replied to (if any). |
| repliedMessageType | MessageEnum | The type of the message replied to (if any). |
| repliedTo | String | The user's id of the sender of the message replied to (if any). |
| senderId | String | The user's id of the sender |
| text | String | The string associated with the message. |
| timeSent | int | The date and time when the message was sent. |
| type | MessageEnum | The type of the message, i.e. 'text', 'gif', or 'audio'. |

*Table 9 – Description of the fields of the Group data table.*

| **Group** | | |
|---|---|---|
| **Field** | **Type** | **Description** |
| groupId | String | The id of the group. |
| groupPicture | String | The URL of the group profile picture. |
| lastMessage | String | The text content of the last message. |
| membersUid | List<String> | The list of ids of the group's members. |
| name | String | The name of the group. |
| senderId | String | The id of the user that created the group. |
| timeSent | int | The date and time of the last message in milliseconds since Unix epoch (1970-01-01-00:00:00). |
| messages | List<Messages> | Messages collection associated with this group. |

### 4.2.4 Application's package structure

The application's package structure follows the Flutter layout convention (Package layout conventions). Figure 4 shows the structure implemented, with the description of the primary objective of each file and directory in the package.



Application's Package Structure - BBK Final Project
Ana Castro | September 16, 2022
bbk_final_ana
<<Top level>>
assets
The assets folder the custom fonts and images used to compose the UI.
lib
The lib folder contains the primary logic of the app. The subfolders are organized by app's feature.
test
The test folder contains all the tests implement.
audio
The audio folder contains all files related to the feature of listening and recording audio stories.
messaging
The messaging folder contains all files related to the feature of sending text, gif or audio messages, including the possibility of creating chat groups.
auth
The auth folder contains all files related to the user's authentication.
common
The common folder contains components and classes that are used throughout the app.
utils
The utils folder contains support functions to show messages or pick images from the gallery or gifs from via Giphy API.
landing
The landing folder contains the landing screen.
main
The main contains the method the executes the app.
router
The router contains the route generator callback used when the app is navigated to a named route.

*Figure 4 – Application's package structure.*

Some directories and files generated automatically by the IDE (Android Studio) were omitted. In addition to the structure above, every package has a file named *pubspec.yaml* in the root directory. This file contains the list of the package's dependencies.

Additionally, the iOS and Android directories contain the files generated automatically by the IDE that "convert" a flutter project into an iOS and Android project. Considering the scope proposed for this project, the only manual changes made were in the android folder, specifically in the *build.gradle* and *AndroidManifest.xml* files, to adjust permissions and versions necessary to run the external libraries imported.

### 4.2.5 Design patterns applied

As described in Figure 4, the lib folder contains the primary logic of the app. App's features are organized in respective subfolders. The internal structure of the subfolders follows the model-view-controller (MVC) design pattern (whenever applicable).

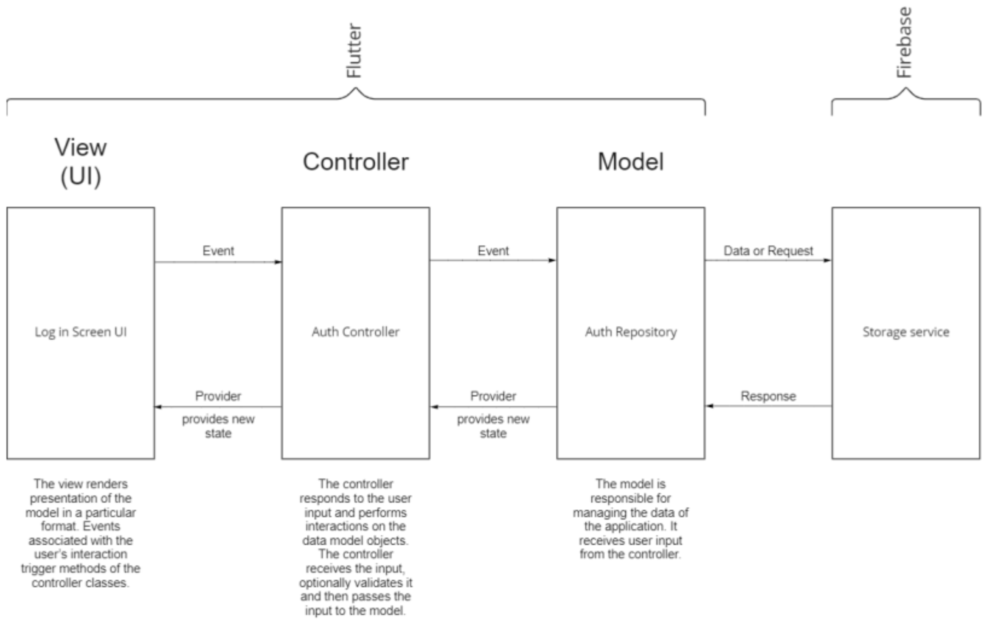
Table Table 10 presents, as an example, the internal structure and the function of its components of the folder related to the audio recording and playing feature.

*Table 10 – Sub-structure of a feature folder. Example of the audio feature.*

| Feature | Sub-folder | Description |
|---|---|---|
| audio | controller | This folder contains the classes used to intermediate the interaction between the model classes in the repository folder and the UI classes in the screens folder. |
|  | enums | This folder contains the Enumerations used to specify the state of the program. |
|  | notifiers | This folder contains the classes used to update and store the application's state and notify the UI when the state changes. The controller classes call those classes to update the state of the program. |
|  | repository | This folder contains the model class that contains the methods called by the controller and interacts with the database services fetching and updating data. |
|  | screens | This folder contains the classes that implement the application's user interface (UI). The UI lets the user know the application's state and interact with it. Events associated with the user's interaction trigger methods of the controller classes. |
|  | widgets | This folder contains the graphical components of the screens that compose the UI. |

The provider or observer pattern “enables a subscriber to register with and receive notifications from a provider. It is suitable for any scenario that requires push-based notification” (Observer Design Pattern). The provider pattern is used with the MVC pattern to notify the controller classes and the UI about changes in the app’s state. App’s state is the state that is not ephemeral, that needs to be shared across many parts of the app, and often is kept between user sessions.

Figure 5 presents an app’s feature architecture using the MVC design pattern associated with the provider pattern.



The view renders presentation of the model in a particular format. Events associated with the user's interaction trigger methods of the controller classes.

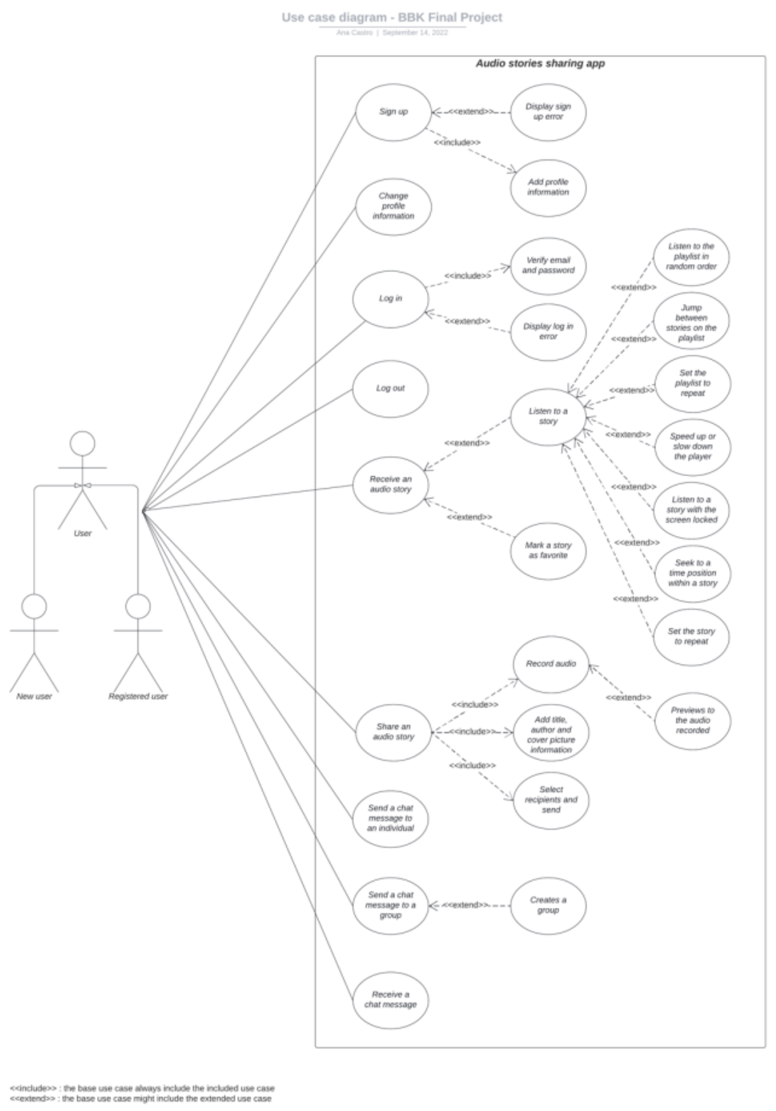
The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

The model is responsible for managing the data of the application. It receives user input from the controller.

*Figure 5 – Example of an app’s feature architecture using the MVC design pattern associated with the provider/observer pattern. The descriptions of functions of the view, controller, and model found in (Buschmann et al, 1996) were used.*

### 4.2.6 Use case diagram

A diagram of the use cases can be found in Figure 6.



Use case diagram - BBK Final Project
Ana Castro | September 14, 2022
Audio stories sharing app
Sign up
<<extend>>
Display sign up error
<<include>>
Add profile information
Change profile information
Log in
<<include>>
Verify email and password
<<extend>>
Display log in error
Log out
Receive an audio story
<<extend>>
Listen to a story
<<extend>>
Mark a story as favorite
<<extend>>
Listen to the playlist in random order
<<extend>>
Jump between stories on the playlist
<<extend>>
Set the playlist to repeat
<<extend>>
Speed up or slow down the player
<<extend>>
Listen to a story with the screen locked
<<extend>>
Seek to a time position within a story
<<extend>>
Set the story to repeat
User
New user
Registered user
Share an audio story
<<include>>
Record audio
<<extend>>
Previews to the audio recorded
<<include>>
Add title, author and cover picture information
<<include>>
Select recipients and send
Send a chat message to an individual
Send a chat message to a group
<<extend>>
Creates a group
Receive a chat message

<<include>> : the base use case always include the included use case
<<extend>> : the base use case might include the extended use case

*Figure 6 – Use case diagram.*

## 4.3 User Interface design

The user interface design was done applying the notion of universal design created for the architectural realm by Ronald L. Mace but which can be transferred for the design of applications. “Universal design is the design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design” (Mace, 1985 *apud* Kalbag, 2017, p. 9).

One of the universal design principles was choosing readable typography and large text font. Making all the text in the app larger makes the app more inviting and easier to read. It is essential because one of our personas is an older adult, and this user may have eyesight degeneration due to age. That is also why Sabiá is an audio exchange app because it favors listening instead of the optical component. In addition, these characteristics not only favor older adults that may have reduced eyesight but also other humans of different ages with the same type of disability. Figure 7 is an example of the typography and font size chosen for the application:

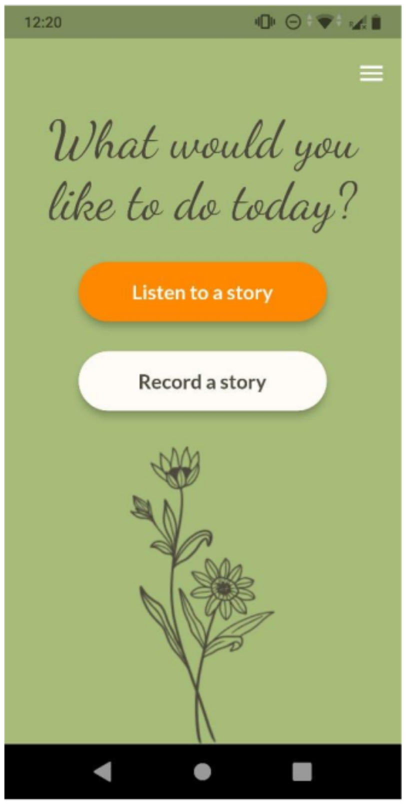
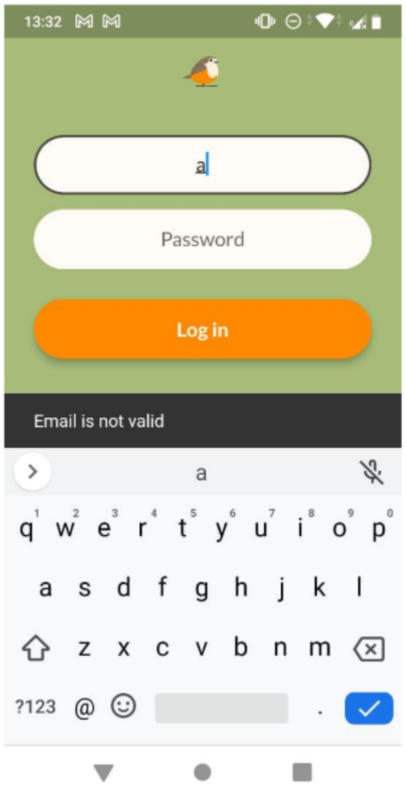


*Figure 7 – Example of typography and font size.*

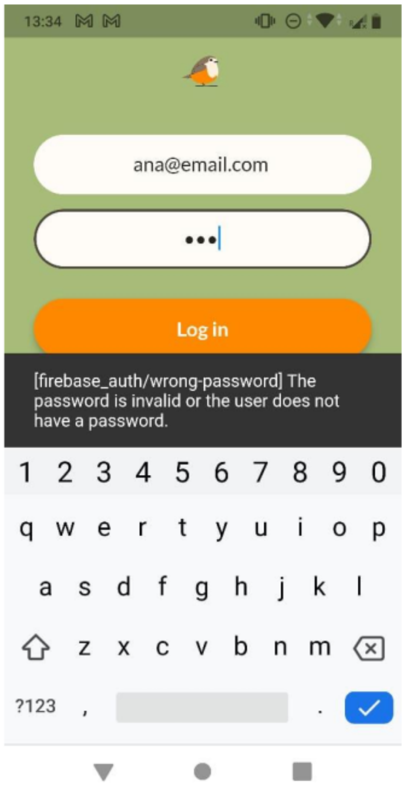
Another universal design principle used was how the error messages appear. In this case, they are clearly written and suggest what the user should do to rectify the error, as shown in Figure 8. There might be doubt of why the error message is not given in red, as usual. That is because people with visual impairments sometimes cannot recognize the red color. Because of that, we chose a black and white notification.



Email is not valid

*Figure 8– Error message example.*

Although there are still errors that need to be fixed, in Figure 9, it is possible to see that there is still a technical message appearing related to Firebase. This type of occasion can make the user confused and frustrated:



[firebase_auth/wrong-password] The password is invalid or the user does not have a password.

*Figure 9 – Technical error message example.*

In addition to these, two other characteristics were considered. The first was the user journey, which was planned to be straightforward. It can be checked in the user manual in appendix 9.2. The second one is that a mobile application was chosen over a web application because smartphones are cheaper than desktops or notebooks, so it is possible to reach more people. Besides, Android is also more affordable than iOS, thus our preferable operating system.

Another critical aspect of the user interface is that the personas were always being considered while developing the app. That is essential for another tool that was used in this project, which was Emotion Design. The idea is based on Walter's (2011) work about designing with emotion. Essentially, the context is that "positive emotional stimuli can be disarming. It builds engagement with your users, which can make the design experience feel like a chat with a friend or a trusted confidant" (Walter, 2011, p. 21).

And that was key because, during the context research, an essential factor was discovered: that one of the main difficulties of the older adults was being afraid of making mistakes while dealing with technology, which led to frustration. Thus, the user interface was made to be inviting and warm to bring the user closer to the app.

An interesting idea of Walter (2011) is to have a Maslow's Hierarchy of Needs, which can be seen in Figure 10, but for the users' needs, shown in Figure 11:

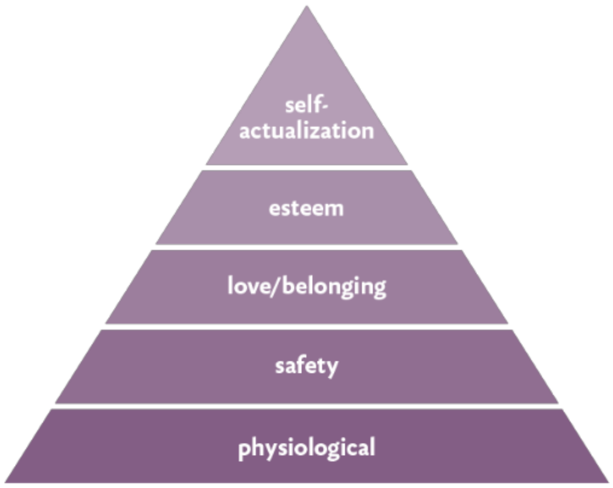


*Figure 10 – Maslow's hierarchy of needs.*
*Available in: (Walter, 2011, p. 13).*

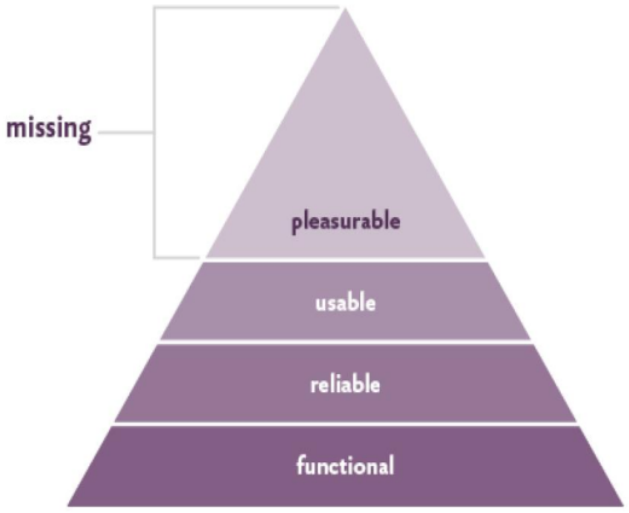


*Figure 11 – Hierarchy of users' needs.*
*Available in: (Walter, 2011, p. 13).*

The main idea is that the applications already have three foundational parts: functionality, reliability, and usability. But there is a missing but essential part related to the user's pleasure while dealing with the application. And that was what the user interface of the Sabiá app aimed to achieve (Figure 12).

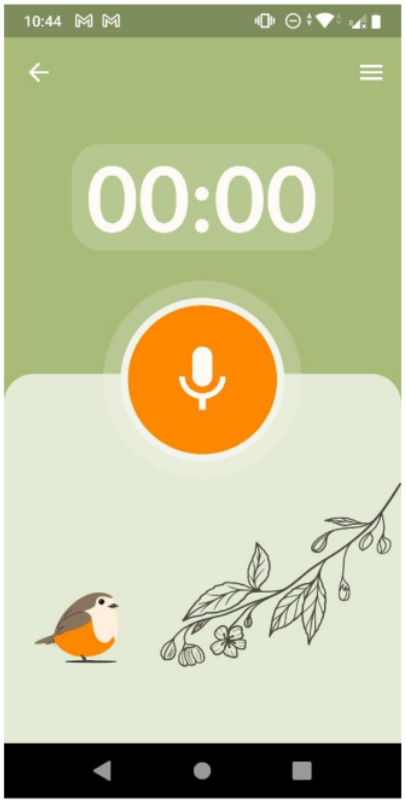


*Figure 12 – Example of Sabiá's user interface.*

# 5 IMPLEMENTATION

This chapter describes the technologies and languages used. It also discusses each stage of the project implementation. Each stage is described in terms of the objective to accomplish, the implementation process, and the challenges faced.

## 5.1 About the technologies, tools and languages used

This section concentrates on the tools and languages used: Flutter, Dart and Firebase.

### 5.1.1 Flutter

Flutter is a set of portable User Interface tools created by Google. It is developed in C, C++, Dart and Skia Graphics Engine, a compact graphics library. The default programming language used by Flutter is Dart. It is common to see these two technologies being used together. Dart is a scripting language created in 2011 by Google to replace JavaScript.

Also, Flutter is open source, and all its code is available on GitHub. Some of the creations with Flutter are Google Ads and Google Green Tea, as well as apps from Alibaba, Abbey Road Studios and Tencent.

Flutter has three main advantages:

- The first is to create applications quickly. Starting with Stateful Hot Reload, an automatic update of the app that enables the project file to be saved almost instantly and without losing the status of the application. With Flutter, it is possible to use several customizable widgets already developed reactively. Widgets are an essential point to speed up development. Since the core idea of Flutter is to use widgets to build the user interface in addition, it can be integrated with several IDEs and editors, for example, Android Studio, XCode and VSCode.
- The second is to create aesthetic and flexible User Interfaces. It allows complete control of every pixel on the screen as it brings widgets, rendering, animation, and gestures into the framework, making the design more flexible and personalized.
- And the third is to maintain the application's native performance. Apps built in Flutter are built directly into native ARM, use the GPU, and can access platform APIs and services. In addition, it can be integrated with already developed applications.

In summary, it is possible to create hybrid applications and maintain native performance with Flutter. With just one code, create an app that will run on Android and iOS and maintain native performance on both.

Other tools for cross-platform development are React Native, Ionic, and Xamarin. Xamarin is Microsoft's platform for building a hundred per cent native mobile apps. Apps for Android, iOS, Windows, Apple Watch, Smartwatch, Google Glass, and Apple TV can be developed by accessing the native APIs of each of the platforms. Xamarin aims to share the same code base for building apps on multiple platforms. In addition, Xamarin is also open source.

According to the Xamarin documentation, applications generated with the platform are native. They contain standard UI controls, have access to all the functionality the platform exposes through its APIs, and have the same performance as native applications. A nice feature is an interoperability that Xamarin provides, being able to directly invoke Objective-C, Java, C++ and C along with their libraries. Some projects and companies that use Xamarin are Azure App, UPS, BBVA and BBC Good Food.

### 5.1.2 Dart

Dart is an object-oriented, multi-paradigm programming language created by Google. It is very versatile and can be used to develop mobile and desktop applications and create scripts and the backend. The language's most popular framework is Flutter, which can be used in IntelliJ IDEA, VSCode, Sublime, Atom, etc.

### 5.1.2 Firebase

Firebase is linked with mobile application development. It can be classified as a BaaS, Backend as a Service, a cloud computing service that serves as Middleware, providing developers with a way to connect their mobile and web applications and cloud services using APIs and SDKs. BaaS makes it possible to completely abstract the server-side infrastructure, which allows developers to focus on the user experience instead of dealing with the backend infrastructure and coding. Today Firebase is the leading platform for Google's mobile development and is part of the Google Cloud Platform suite of products.

Firebase is also a database but has at least eighteen products within its platform. Through Firebase, it is possible to send and receive messages and notifications. It is still possible to create segmentations, customize the content for the user and send messages using the time zone. In addition, Firebase integrates with the Analytics area, where all information about sent messages, such as engagement and conversion, can be tracked. It can be done without coding using Firebase Cloud Messaging (FCM).

In terms of programming language, frameworks, and other technologies, it is possible to use several in conjunction with Firebase, such as Java, Swift, Objective-C, Python, JavaScript, NodeJS, C++, React, etc.

One of the top products for development solutions is Cloud Firestore, which is the no-SQL database with which it is possible to store, query and practically synchronize data. The synchronization part is essential because there is no need for a server for the application, as it is possible to use a backend code that manipulates the database. Cloud Functions is responsible for executing code on the backend without having to manage servers. With Firebase, it is possible to have a complete package of solutions.

Another critical development solution is Firebase Authentication, which facilitates the user registration and login process using platforms such as Google, Facebook and GitHub.

There are competitors, like Parse, bought by Facebook, which is open source. There is also Back4app which is a complete BaaS platform. But Firebase is currently the dominant and most complete platform on the market.

## 5.2 Description of the final product

### 5.2.1 Software development methodology

As mentioned before, the development methodology adopted was iterative and incremental software prototyping due to most technologies being previously unfamiliar. In each phase, simplified prototypes of components were developed before being improved and incorporated into the overall program. For example, a simple audio recorder and player were developed initially to become familiar with the audio library used before implementing the complete audio functionality of the application.

The implementation process was divided into sprints to deliver a specific functionality set. Features were built up incrementally, adding more functionality during each sprint (Figure 13). Within each sprint, the functionalities developed were debugged and improved in iterations.

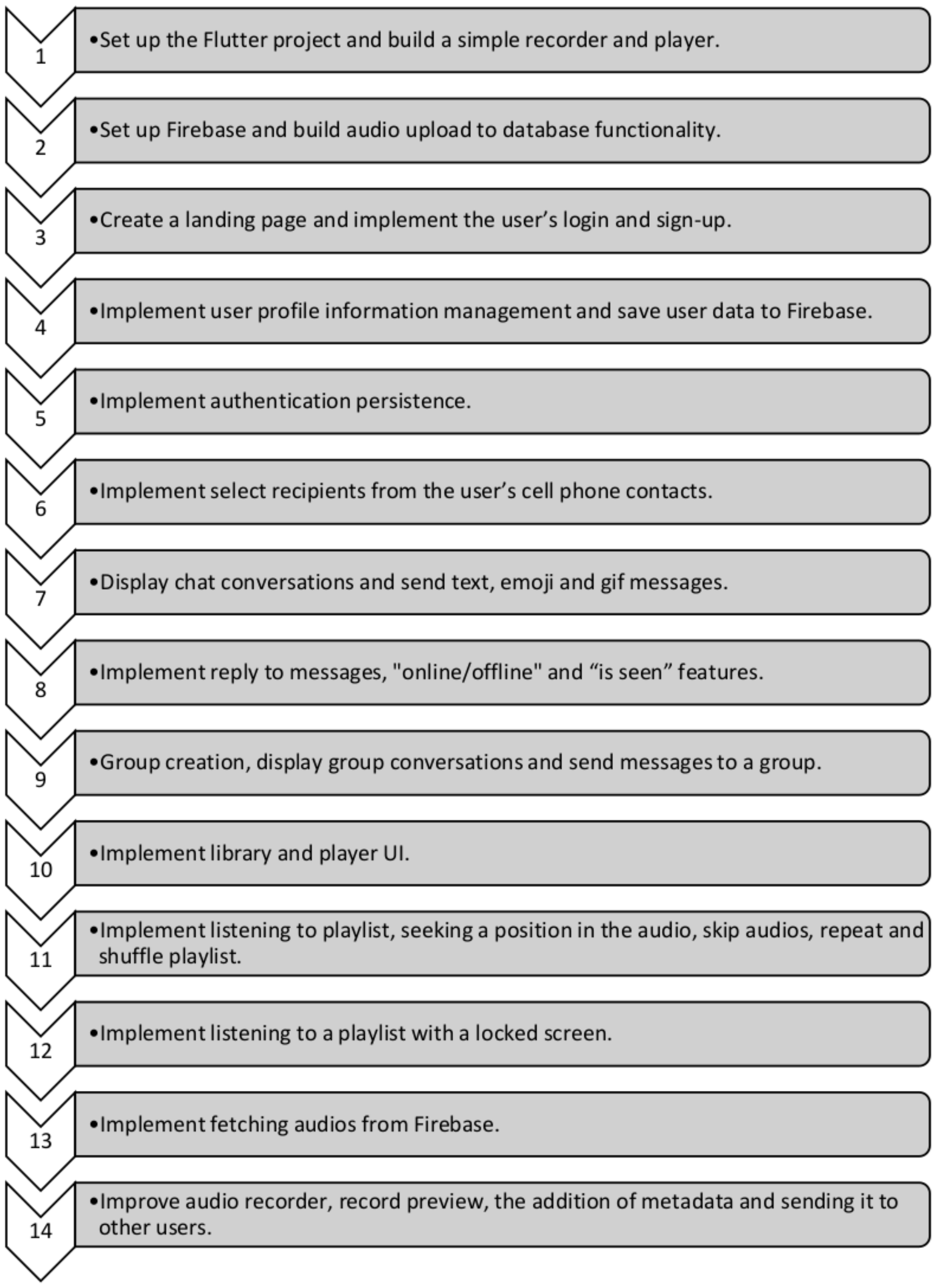

1
•Set up the Flutter project and build a simple recorder and player.
2
•Set up Firebase and build audio upload to database functionality.
3
•Create a landing page and implement the user's login and sign-up.
4
•Implement user profile information management and save user data to Firebase.
5
•Implement authentication persistence.
6
•Implement select recipients from the user's cell phone contacts.
7
•Display chat conversations and send text, emoji and gif messages.
8
•Implement reply to messages, "online/offline" and "is seen" features.
9
•Group creation, display group conversations and send messages to a group.
10
•Implement library and player UI.
11
•Implement listening to playlist, seeking a position in the audio, skip audios, repeat and shuffle playlist.
12
•Implement listening to a playlist with a locked screen.
13
•Implement fetching audios from Firebase.
14
•Improve audio recorder, record preview, the addition of metadata and sending it to other users.

*Figure 13 – Implementation Sprints*

The following subsections will detail each sprint's objectives, deliverables and development highlights.

### 5.2.2 Sprint 1: Set up the Flutter project and build a simple recorder and player

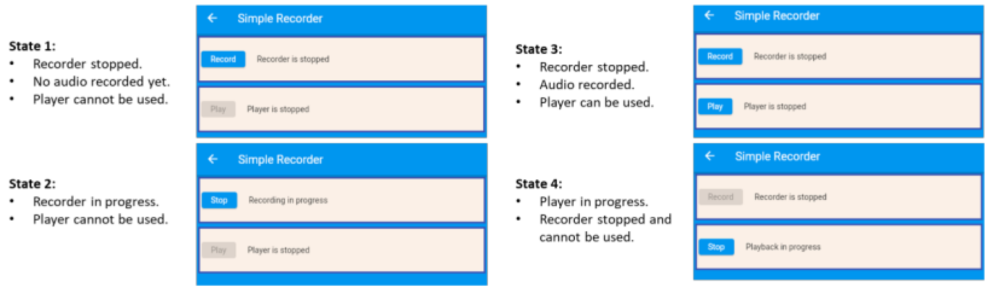
#### 5.2.2.1 Objectives

- Set up the base Flutter project.
- Build a simple recorder to record audio and save the file locally in a temporary cell phone directory.
- Build a simple player that would allow listening to the audio recorded.

#### 5.2.2.2 Implementation highlights

After installing and setting up the Dart SDK, the Flutter SDK, the Android Studio IDE with the respective Flutter plugin, and the Android emulator, I started a standard new Flutter project that comes with some base code (Write your first Flutter app).

I used the external package *flutter_sound* to implement the audio recorder and player. The examples provided by the package author were used as a reference for implementing an audio recorder class, a player class and a simple UI for testing the implementation. Figure 14 shows the different states of the application in the process of recording or playing recorded audio. The audio files were saved locally in a temporary directory.


**State 1:**
- Recorder stopped.
- No audio recorded yet.
- Player cannot be used.

**State 2:**
- Recorder in progress.
- Player cannot be used.

**State 3:**
- Recorder stopped.
- Audio recorded.
- Player can be used.

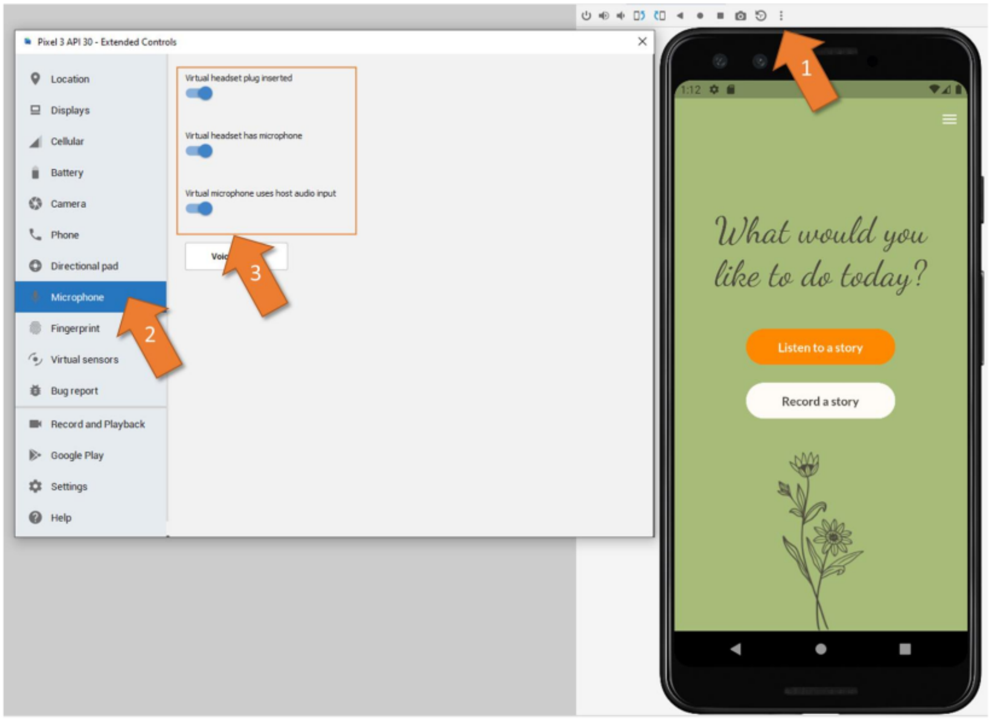
**State 4:**
- Player in progress.
- Recorder stopped and cannot be used.

*Figure 14 - Different application states recording or playing recorded audio in sprint 1.*

#### 5.2.2.3 Challenges

Installing and setting up the Dart SDK, the Flutter SDK, and the Android Studio IDE with the respective Flutter plugin is complex and time-consuming. The references (Windows install) and (How to Install and Setup Flutter for App Development on Windows) were instrumental in overcoming the challenges.

Recording and listening to the first audio took me three days. After implementing and extensively debugging the code of an elementary recorder and player, I still could not hear the audio recording. I started investigating issues outside the code implemented. I discovered that I could not use a Bluetooth headset but had to use a wired one. Also, I had to set up the Android emulator to accept the inputs from the computer headset microphone (Figure 15).



Virtual headset plug inserted

Virtual headset has microphone

Virtual microphone uses host audio input

*Figure 15 - Setting up the Android emulator to receive the headset microphone input.*

### *5.2.3* Sprint 2: Set up Firebase and build audio upload to database functionality

#### 5.2.3.1 Objectives

- Set up a Firebase project.
- Link the Firebase project with the app.
- Implement the functionality of uploading the recorded file to Firebase.

#### 5.2.3.2 Implementation highlights

To set up the Firebase project for Android and link it with the Flutter app, I followed the tutorial from Yu (2021).

Cloud Storage for Firebase is an object storage service associated with Firebase (Cloud Firebase Storage). To use it in a Flutter project is necessary to add the external packages *firebase_core* and *firebase_storage* as project dependencies. The *FirebaseStorage* is the class associated with the Cloud Storage for Firebase. The class *CommonFirebaseStoreRepository* (Figure 16) was created to store files in the *FirebaseStorage*.

```
import 'dart:io';
import 'package:firebase_storage/firebase_storage.dart';


class CommonFirebaseStoreRepository {
  final FirebaseStorage firebaseStorage;

  CommonFirebaseStoreRepository({required this.firebaseStorage});

  /// Stores a file in the [FirebaseStorage].
  /// [path] is the path where the file will be stored.
  /// [file] is the file to be stored.
  /// Returns the url of the file stored.
  Future<String> storeFileToFirebase(String path, File file) async
{
    UploadTask uploadTask =
firebaseStorage.ref().child(path).putFile(file);
    TaskSnapshot snapshot = await uploadTask;
    String downloadUrl = await snapshot.ref.getDownloadURL();
    return downloadUrl;
  }
}
```

*Figure 16 – Code snippet of the class responsible for storing files in the FirebaseStorage.*

#### 5.2.3.3 Challenges

It is not straightforward to link the Firebase with the Flutter project. Even following the step-by-step guidance provided by the Firebase console and the tutorial from Yu (2021), the project was not running. Debugging the errors via the application's console, I figured out that I had to change to minimum Android SDK versions of the project to make it compatible with Firebase (Figure 17).

```
// Code snippet from build.gradle file in the
// Android app directory

...

android {
    compileSdkVersion 33 // I had to change this
    ndkVersion flutter.ndkVersion

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    defaultConfig {
        applicationId "bbk.student.anacastro.bbk_final_project"
        minSdkVersion 24 // I had to change this
        targetSdkVersion 33 // I had to change this
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }

    buildTypes {
        release {
            signingConfig signingConfigs.debug
        }
    }
}
```

*Figure 17 – Code snippet from "build.gradle" file in the Android app directory, adjusted for compatibility with the Firebase project.*

### 5.2.4 Sprint 3: Create a landing page and implement the user's login and sign-up

#### 5.2.4.1 Objectives

- Create a landing page.
- Implement the user login with email and password.
- Implement use sign-up with email and password.

#### 5.2.4.2 Implementation highlights

Due to the problem being tackled by this project, having a pleasant and accessible user interface is one of the priorities. Therefore, I opted to implement the landing page with some animations and a friendly design.

The built-in curved animation (Curves class) and the *animated_text_kit* external package were used to implement the landing page. After passing a ticker (like a clock ticker) to an animation controller, Flutter's curved animation will provide a value that will vary from 0 to 1 within a specified time frame. This value is used to control the visibility of elements on the screen, generating a visual animated effect (Figure 18).

```
  @override
  Widget build(BuildContext context) {
    return ScreenBasicStructure(
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 24.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Expanded(...),
                animation.value > 0.6
                    ? const Expanded(...) // Shown after some time
                    : const SizedBox(),// Invisble element
              ],
            ),
            const SizedBox(height: 12.0),
            animation.value < 0.6
                ? const AnimatedTextRow()// Shown initially
                : const ButtonsColumn(), // Shown after some time
          ],
        ),
      ),
    );
  }
}
```

*Figure 18 – Code snippet from the landing page. It shows an example of the use of the animation value that varies with time to control the visibility of elements, resulting in an animated visual effect.*

The *firebase_auth* package was used to implement the app's authentication process. This package provides backend services to authenticate users and supports authentication using passwords, phone numbers, and identity providers like Google, Facebook and Twitter (Firebase Authentication).

The *AuthRepository* class was created to manage the app's authentication process. The authentication process consists of signing up new users, logging in existing users, logging out users, saving, editing, or getting the user's profile and status information. Figure 19 shows the code snippets of the methods of the *AuthRepository* class to log in and sign up users using the *FirebaseAuth*.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:bbk_final_ana/utils/utils.dart';
/// Other imports related to other methods
(...)


class AuthRepository {
  final FirebaseAuth auth;
  final FirebaseFirestore firestore;

  AuthRepository({required this.auth, required this.firestore});

  /// Logs user in using email and password
  Future<bool> signInWithEmail(
      BuildContext context, String email, String password) async {
    bool didSucceed = true;
    try {
      final user = await auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );
    } on Exception catch (e) {
      didSucceed = false;
      showSnackBar(context: context, content: e.toString());
    }
    return didSucceed;
  }

  /// Sign user up using email and password
  Future<bool> createUserWithEmailAndPassword(
      BuildContext context, String email, String password) async {
    bool didSucceed = true;
    try {
      final newUser = await auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );
    } on Exception catch (e) {
      didSucceed = false;
      showSnackBar(context: context, content: e.toString());
    }
    return didSucceed;
  }

/// Other methods below
(...)

}
```

*Figure 19 – Code snippet of the methods of the AuthRepository class to log in and sign up users using the FirebaseAuth.*

#### 5.2.4.3 Challenges

As explained in section 4.2.5, the MVC and provider design patterns were used to comply with the SOLID design principles (The Principles of OOD). Although very important, the use of such design patterns can make the code more complex. Due to my lack of experience, their implementation was challenging.

In addition, I chose the use *flutter_riverpod* package to implement the provider design pattern to share the application state across different parts. *Riverpod* is a state-management library that catches programming errors at compile time rather than runtime, removes nesting for listening/combining objects and ensures that the code is testable (Riverpod 1.0.3+1). Despite the benefits, using *Riverpod* comes with a learning curve, which was also challenging.

### 5.2.5 Sprint 4: Implement user profile information management and save user data to Firebase

#### 5.2.5.1 Objectives

- Implement the initial user's profile set-up (name and picture) after signing up.
- Implement the possibility of changing the profile information.
- Save the user data to Firebase.

#### 5.2.5.2 Implementation highlights

The *image_picker* package was used to pick images from the cellphone gallery. Figure 20 shows the *pickImageFromGallery* method developed using the *image_picker* package. This method will return *null* if the user does not select any file from the gallery and returns to the app.

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


/// Function to pick an image from the cellphone gallery
/// using the OS.
Future<File?> pickImageFromGallery(BuildContext context) async {
  File? image;
  try {
    final pickedImage =
        await ImagePicker().pickImage(source: ImageSource.gallery);
    if (pickedImage != null) {
      image = File(pickedImage.path);
    }
  } catch (e) {
    showSnackBar(context: context, content: e.toString());
  }
  return image;
}
```

*Figure 20 – Code snippet of the pickImageFromGallery method developed using the image_picker package. This method will return null if the user does not pick any file from the gallery and returns to the app.*

The *Cloud Firestore* is a cloud-hosted, NoSQL document database to store, sync, and query the app's data (Cloud Firestore). Figure 21 shows the method developed to save users' data to Firebase, which sits under the AuthRepository class.

```
import 'dart:io';

import 'package:bbk_final_ana/common/constants/constants.dart';
import 'package:bbk_final_ana/models/user_model.dart';
import 'package:bbk_final_ana/utils/utils.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

import
'../../common/repositories/common_firestore_repository.dart';

class AuthRepository {
  final FirebaseAuth auth;
  final FirebaseFirestore firestore;

  AuthRepository({required this.auth, required this.firestore});

(...)
/// More methods above

  /// Saves user's information to firebase
  void saveUserDataToFirebase({
    required String name,
    required File? profilePicture,
    required BuildContext context,
    required CommonFirebaseStoreRepository firebaseStoreRepository,
  }) async {
    try {
      String uid = auth.currentUser!.uid;
      String photoUrl = kStandardProfilePicUrl;
      if (profilePicture != null) {
        photoUrl = await
firebaseStoreRepository.storeFileToFirebase(
            'profilePicture/$uid', profilePicture);
      }
      var user = UserModel(
        name: name,
        uid: uid,
        profilePicture: photoUrl,
        isOnline: true,
        email: auth.currentUser!.email.toString(),
        groupId: [],
      );
      await
firestore.collection('users').doc(uid).set(user.toMap());
    } on Exception catch (e) {
      showSnackBar(context: context, content: e.toString());
    }
  }

/// More methods below
(...)

}
```

*Figure 21 – Code snippet of the method developed to save users' data to Firebase, which sits under the AuthRepository class.*

#### 5.2.5.3 Challenges

The biggest challenge in this sprint was to address the possibility of the user opening the cellphone's gallery to pick a profile picture but not selecting one. If the user has a previous profile picture, that picture should not be changed if this situation happens. A standard profile (Figure 22) should be attributed if the user does not have a profile picture yet. The implementation of this logic was somewhat challenging.

*Figure 22 – Standard profile picture designed with Figma (Figma). A standard profile picture should be attributed if the user does not have a profile picture yet and has not selected any from the cellphone's gallery.*

### 5.2.6 Sprint 5: Implement authentication persistence

#### 5.2.6.1 Objective

- Implement authentication persistence so the users do not have to log in when they close and reopen the app.

#### 5.2.6.2 Implementation highlights

Some steps were necessary to implement the authentication persistence. First, a method to get the current user's data was implemented in the *AuthRepository* class (Figure 23).

```
import 'package:bbk_final_ana/models/user_model.dart';
import 'package:firebase_auth/firebase_auth.dart';
(...)
/// More imports


class AuthRepository {
  final FirebaseAuth auth;
  final FirebaseFirestore firestore;

  AuthRepository({required this.auth, required this.firestore});

(...)
/// More methods above

  /// Gets current user data from firebase
  Future<UserModel?> getCurrentUserData() async {
    var userData =
        await
  firestore.collection('users').doc(auth.currentUser?.uid).get();
    UserModel? user;
    if (userData.data() != null) {
      user = UserModel.fromMap(userData.data()!);
    }
    return user;
  }


/// More methods below
(...)

}
```

*Figure 23 – Code snippet of the method in the AuthRepository class to get the current user's data.*

Following the MVC pattern, a method called the *getCurrentUserData* method above was developed in the *AuthController* class. Then a *userDataAuthProvider* (Figure 24) was created to provide the current user data to the app's root.

```
final userDataAuthProvider = FutureProvider((ref) {
  final authController = ref.watch(authControllerProvider);
  return authController.getCurrentUserData();
});
```

*Figure 24 – Code snippet of the provider of the current user's data.*

The app's root widget watches the *userDataAuthProvider*, and if the user is logged in, it skips the landing page (*WelcomeScreen*) and goes directly to an internal screen of the app

(*InitialDecisionScreen*). Otherwise, it returns the *WelcomeScreen*, where the user will be able to log in or sign up. See Figure 25.

```dart
/// main() executes the app
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  await initAudioService();
  runApp(const ProviderScope(child: MyApp()));
}

/// [MyApp] is the origin/root of the app's widget tree.
class MyApp extends ConsumerWidget {
  const MyApp({super.key});

  // This widgets is the root of the application.
  @override
  Widget build(BuildContext context, WidgetRef ref) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Sabiá',
      theme: ThemeData(...),
      onGenerateRoute: (settings) => generateRoute(settings),

      // The code below implements the user authentication
      // persistence.
      home: ref.watch(userDataAuthProvider).when(
          data: (user) {
            if (user == null) {
              return const WelcomeScreen();
            }
            return const InitialDecisionScreen();
          },
          error: (e, trace) {
            return ErrorScreen(error: e.toString());
          },
          loading: () => const LoaderScreen()),
    );
  }
}
```

*Figure 25 – Code snippet of the implementation of the authentication persistence in the app's root.*

#### 5.2.6.3 Challenges

Again, the biggest challenge in this sprint was the learning curve related to using the *Riverpod* package to implement the provider associated with the MVC design patterns.

### 5.2.7 Sprint 6: Implement select recipients from the user's cell phone contacts

#### 5.2.7.1 Objective

- Implement the function of selecting users that are registered both in the app and the cell phone contacts with the same email.

#### 5.2.7.2 Implementation highlights

Unlike other audiobook apps, this project focuses on the exchange of audio stories among people who actually know each other. Due to that, the app should only allow users to select recipients for sending messages and audio from the intersection between the users registered in the app and the contacts on their respective cell phones.

As email is the primary identifier used for user authentication, the contact in the cell phone must contain the same email to show up in the available recipients' list. Figure 26 shows the code snippet of the method that returns the users whose email is the in the cell phone's contact and registered in the app. The package *flutter_contacts* was used to access the cellphone's contacts.

```
import 'package:bbk_final_ana/models/user_model.dart';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_contacts/flutter_contacts.dart';

class SelectContactRepository {
  final FirebaseFirestore firestore;

  SelectContactRepository({required this.firestore});

  /// Get the list of the users that are both in the cellphone
  /// contacts and in the app's database with the same email.
  Future<List<UserModel>> getUserContacts() async {
    List<UserModel> userContacts = [];
    try {
      // Requests permission to access the phone's contacts
      if (await FlutterContacts.requestPermission()) {
        var allDeviceContacts =
            await FlutterContacts.
                      getContacts(withProperties: true);
        List<String> myContactsEmailAddresses = [];
        for (var contact in allDeviceContacts) {
          for (var email in contact.emails) {
            myContactsEmailAddresses.add(email.address);
          }
        }
        var userCollection = await
firestore.collection('users').get();
        for (var document in userCollection.docs) {
          var userData = UserModel.fromMap(document.data());
          String registeredEmail = userData.email;
          if (myContactsEmailAddresses.contains(registeredEmail)) {
            userContacts.add(userData);
          }
        }
      }
    } catch (e) {
      debugPrint(e.toString());
    }
    return userContacts;
  }
}
```

*Figure 26 – Code snippet of the method that returns the users whose email is the in the cell phone's contact and registered in the app.*

### 5.2.7.3 Challenges

The biggest challenge in this stage was that many emails could be registered under the same contact on the cell phone. The method developed had to consider that at least one of them matched with the email of a user registered in the app so that this user would show up in the possible recipients' list.

### 5.2.8 Sprint 7: Display chat conversations and send text, emoji and gif messages

#### 5.2.8.1 Objectives

- Implement selecting a contact and starting a new chat conversation.
- Implement typing and sending a new text message to a contact.
- Implement fetching messages from a conversation from Firebase and displaying them in the correct disposition on the chat screen.
- Implement fetching 1-2-1 conversations from Firebase and displaying the list on the UI.
- Implement selecting emojis and sending them to contact.
- Implement selecting GIFs and sending them to contact.

#### 5.2.8.2 Implementation highlights

The functionality of selecting registered users in the app and the device contacts with the same email was developed in the previous sprint (item 5.2.7). After selecting the contact, a user is redirected to the chat screen. The chat conversation is created in the database once the first message is sent.

A text bottom chat field component was developed to allow the user to type a new text message, similar to other messaging apps.

The methods to send a text message, create a chat conversation, save a message in a conversation and fetch the messages sit under the *ChatRepository* class. The external packages *cloud_firestore* and *firebase_auth* were used to get and send chat data to Firebase. Figure 27 shows the method to fetch the stream of messages of a chat conversation as an example.

```
/// Gets the messages of an specific 1-2-1 chat
Stream<List<Message>> getChatStream(String receiverId) {
  return firestore
      .collection('users')
      .doc(auth.currentUser!.uid)
      .collection('chats')
      .doc(receiverId)
      .collection('messages')
      .orderBy('timeSent')
      .snapshots()
      .map((event) {
    List<Message> messages = [];
    for (var document in event.docs) {
      var message = Message.fromMap(document.data());
      messages.add(message);
    }
    return messages;
  });
}
```

*Figure 27 – Code snippet of the method to fetch the stream of messages of a chat conversation. This method sits under the ChatRepository class.*

The package *emoji_picker_flutter* was used to pick emojis. Emojis are Strings and can be sent via the same method used for sending text messages.

The package *enough_giphy_flutter* is used to interact with the Giphy API (Giphy) to allow the user to select GIFs. GIFs are similar to image files hosted online. Their URL is saved to the database and shown on the screen via a cached network image component. This component downloads the web-hosted asset and saves it to the device's cache memory, which prevents having to download the file every time it is shown on the screen. The package *cached_network_image* is used to display cached images from URLs.

### 5.2.8.3 Challenges

In addition to challenges related to the use of the design pattern already described, the main challenge of this sprint was the volume of UI components that had to be produced and debugged. The interaction with different external packages demanded studying a large amount of documentation.

It is important to emphasize that the logic implemented to send messages developed during this sprint is very similar to the one implemented for sending the audio stories. Therefore, the work done in this sprint set the foundations of the primary functionalities of the app.

### 5.2.9 Sprint 8: Implement reply to messages, “online/offline”, and “is seen” features

#### 5.2.9.1 Objectives

- Implement swiping laterally a message received to reply to it.
- Implement a way for users to check if a contact is online or offline to chat.
- Implement a way for the sender to check whether the receiver has seen the messages sent.

#### 5.2.9.2 Implementation highlights

The external package *swipe_to* provides a UI component that allows one to associate a method to the lateral swipe gesture. This component was used to “wrap” the message bubble on the chat UI. Figure 28 shows the code snippet of the *MessageReply* class and provider used to control whether the user is replying to a chat message.

```dart
import 'package:bbk_final_ana/common/enums/message_enum.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

/// [messageReplyProvider] provides a [MessageReply] to the
/// ChatController and the UI. The default state is null. When the
/// user swipes a message bubble in the chat, a new [MessageReply]
/// is created and the state of the [StateProvider] is updated.
/// After saving the message in the database, the state of the
/// [messageReplyProvider] is updated again to null.
final messageReplyProvider = StateProvider<MessageReply?>((ref) =>
null);

/// [MessageReply] controls whether the user is replying to a chat
/// message.
/// [message] the text of the message being replied to.
/// [isMe] is true when the message being replied to is yours,
/// false otherwise.
/// [messageEnum] is the type of the message being replied to.
class MessageReply {
  final String message;
  final bool isMe;
  final MessageEnum messageEnum;

  MessageReply(this.message, this.isMe, this.messageEnum);
}
```

*Figure 28 – Code snippet of the MessageReply class and messageReplyProvider that were used to control whether the user is replying to a chat message. The comments explain the details of the implementation logic.*

The key to implementing the “online/offline” feature is to add a *binding observer* to the application when the user opens the screen with the chat conversations. Binding observers are notified when various application events occur. Whenever the application’s lifecycle state is “resumed”, which means that the application is visible and responding to user input, the user’s state should be changed to online, otherwise offline (Figure 29).

```
/// The [ConversationsScreen] is the screen with the list of group
/// or individual conversations
class ConversationsScreen extends ConsumerStatefulWidget {
  const ConversationsScreen({Key? key}) : super(key: key);
  static const String id = '/conversations';

  @override
  ConsumerState<ConversationsScreen> createState() =>
      _ConversationsScreenState();
}

class _ConversationsScreenState extends
ConsumerState<ConversationsScreen>
    with WidgetsBindingObserver {


  @override
  void initState() {
    super.initState();
    // Starts observing the application's state
    WidgetsBinding.instance.addObserver(this);
  }

  @override
  void dispose() {
    super.dispose();
    // Finishes observing the application's state
    WidgetsBinding.instance.removeObserver(this);
  }

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    super.didChangeAppLifecycleState(state);
    // The following code update the user status (online/offline)
    // when the app is open, paused, closed.
    switch (state) {
      case AppLifecycleState.resumed:
        ref.read(authControllerProvider).setUserState(true);
        break;
      case AppLifecycleState.paused:
      case AppLifecycleState.detached:
      case AppLifecycleState.inactive:
        ref.read(authControllerProvider).setUserState(false);
        break;
    }
  }


// More methods below

(...)

}
```

*Figure 29 – Code snippet of the "online/offline" functionallity.*

The method *setUserState* was added to the *AuthRepository* class to update the user state in the database (Figure 30).

```
/// Sets the user's state as online or offline. The user's state
/// switches to online when the user access the chat. The opposite
/// happens when the user exits the chat
void setUserState(bool isOnline) async {
  await
firestore.collection('users').doc(auth.currentUser!.uid).update({
    'isOnline': isOnline,
  });
}
```

*Figure 30 – Code snippet of the setUserState method that sits under the AuthRepository class.*

The method *setMessageSeen* was created in the *ChatRepository* class to update the *isSeen* field of the message in the database when the recipient opens the chat screen with new messages. Figure 31 shows the code snippet of the verification in the *ChatScreen* to check whether it should be set as seen. A double-blue check icon is displayed on the UI close beside the messages that were seen by the recipient.

```
if (!messageData.isSeen &&
    (messageData.receiverId ==
        firebaseAuth.currentUser!.uid)) {
  ref.read(chatControllerProvider).setChatMessageSeen(
        context,
        widget.receiverId,
        messageData.messageId,
      );
}
```

*Figure 31 – Code snippet of the verification to set a message as seen when a user visualizes it. This snippet can be found on the ChatList UI component.*

#### 5.2.9.3 Challenges

The biggest challenge of this sprint was to figure out how to observe the application's lifecycle state and update the user status accordingly. Fortunately, the Flutter documentation (*WidgetsBindingObserver* class) is comprehensive and provides the necessary information.

### 5.2.10 Sprint 9: Group creation, display group conversations and send messages to a group

#### 5.2.10.1 Objectives

- Implement the creation of groups of users.
- Display the group conversations in a list.
- Implement sending and receiving messages from a group.

#### 5.2.10.2 Implementation highlights

The *GroupRepository* class is responsible for interacting with the database to store new group information. This class contains a single method called *createGroup* that creates a new group based on a selected user list and stores its information in the database. Figure 32 shows a code snippet of this class. The details of the fields related to the group data table can be found in this report's Table 9.

```
import...

/// The [GroupRepository] is responsible for interacting with the
/// database for storing the information of new groups.
class GroupRepository {
  final FirebaseFirestore firestore;
  final FirebaseAuth auth;

  GroupRepository({
    required this.firestore,
    required this.auth,
  });

  /// Creates a new group and stores its information in the
  /// database.
  void createGroup(
    BuildContext context,
    String name,
    File profilePicture,
    List<UserModel> selectedContacts,
    CommonFirebaseStoreRepository firebaseStoreRepository,
  ) async {
    try {
      List<String> userIds = [...selectedContacts.map((e) =>
e.uid)];
      var groupId = const Uuid().v1();
      String profileUrl = await
firebaseStoreRepository.storeFileToFirebase(
        'group/$groupId',
        profilePicture,
      );
      Group group = Group(
        senderId: auth.currentUser!.uid,
        name: name,
        groupId: groupId,
        lastMessage: '',
        groupPicture: profileUrl,
        membersUid: [auth.currentUser!.uid, ...userIds],
        timeSent: DateTime.now(),
      );
      await
firestore.collection('groups').doc(groupId).set(group.toMap());
    } on Exception catch (e) {
      // Error message is displayed
      showSnackBar(context: context, content: e.toString());
    }
  }
}
```

*Figure 32 – Code snippet of the GroupRepository class, responsible for interacting with the database to store new group information.*

The method *getChatGroups* was added to the *ChatRepository* class to fetch the chat group conversation stream from the database. This stream of conversations is displayed on the UI via the ConversationsScreen.

Finally, the methods *sendTextMessage* and *sendGifMessage* of the ChatRepository class were updated to verify if the message's receiver is a group and update the database accordingly.

#### 5.2.10.3 Challenges

The primary challenge of this sprint was to plan the adaptation of the database structure to include groups. Additionally, various points of the UI related to the *ChatScreen* and ConversationsScreen had to be adapted to include different layouts for 1-2-1 chats or group chats.

### 5.2.11 Sprint 10: Implement library and player UI

#### 5.2.11.1 Objectives

- Implement the user interface of the library screen where users can see all audio stories shared.
- Implement player user interface.

#### 5.2.11.2 Implementation highlights

A considerable amount of time was dedicated to designing and implementing the UI of the library screen and the player screen, as those would be the main screens for typical app users.

To give a feel of an actual story library, images to represent covers of the stories shared were included in the layout (Figure 33). The pictures used in this project were downloaded from Canva (Canva), and the idea is to add a library of custom artwork for story covers to the app.

*Figure 33 – Example of the story covers available in the app.*

The audio stories' information was hard-coded, and the covers were added as constant project assets to implement the first version of the library screen UI. In future sprints, the library screen would be linked to the database, and the covers would be hosted online being displayed via cached network image components.

The dynamic elements of the player screen UI were also hard-coded in the first iteration. For example, the names of the story and the author were added as constant Strings.

#### 5.2.11.3 Challenges

The primary challenge of this stage was to incorporate responsiveness to UI to different device screen sizes. The Flutter framework offers many components to build lists dynamically and add responsiveness to components. However, the interaction between "flexible" components and lists is not very easy to implement for more complex UI designs.

## 5.2.12 Sprint 11: Implement listening to playlist, seeking a position in the audio, skipping audios, repeating and shuffling the playlist

### 5.2.12.1 Objectives

- Implement the possibility of listening to a list of audios hosted online in sequence.
- Improve the player to allow seeking a specific position in a track being listened to.
- Implement jumping between tracks within the playlist.
- Implement setting one specific audio to repeat.
- Implement setting the entire playlist to repeat.
- Implement listening to the playlist in random order.
- Implement the possibility to speed up or slow down the player.

### 5.2.12.2 Implementation highlights

The external package *just_audio* was used to play audio from a URL, and the class *PlayerController* was developed to manage the player. Notifiers were created to notify the UI about the player and playlist's states.

Web-hosted audio example files (SoundHelix web audio examples) were used at this implementation stage to test the player and playlist functionalities developed.

### 5.2.12.3 Challenges

The tutorial from Suragch (2021) was instrumental as a reference in implementing the functionalities listed in this sprint. The biggest challenge was to consider the tutorial guidance while maintaining consistency with the MVC and design patterns implemented throughout other parts of the project.

## 5.2.13 Sprint 12: Implement listening to a playlist with a locked screen

### 5.2.13.1 Objective

- Implement listening to the playlist with the device screen locked or while using another app simultaneously with the audio in the background.

#### 5.2.13.2 Implementation highlights

The external package *audio_service* was used to play audio in the background and control the player using the device OS interface.

The class *StandardAudioHandler* was implemented to manage the interaction between the app's player and the device's OS. The external package *audio_session* was also used to configure the interaction of the app's audio features and the audio from other apps installed on the device that might be used simultaneously. This package allows an audio session to specify which audio output should be prioritized depending on the situation.

#### 5.2.13.3 Challenges

A second tutorial from Suragch (2021) was also instrumental as a reference in implementing the functionalities listed in this sprint. Similar to what happened during the previous sprint, the biggest challenge was to consider the tutorial guidance while maintaining consistency with the MVC and design patterns implemented throughout other parts of the project.

Due to the interaction of functionalities implemented in this sprint with the device OS, a bug was preventing audios from playing. The solution was to uninstall the app from the Android device emulator and reinstall it without any changes to the code.

### 5.2.14 Sprint 13: Implement fetching audios from Firebase and marking audio as favorite

#### 5.2.14.1 Objectives

- Implement fetching audios associated with a user from the database.
- Implement a button to mark audio stories as favorites.

#### 5.2.14.2 Implementation highlights

The class *AudioRepository* was implemented to manage the interaction with the database. Figure 34 shows the method *getAudioMessagesStream* used to get the stream of audios from Firebase and the *fetchPlaylist* used by the *PlayerController* class to load the available audios in the user's library into the player's playlist.

```
import...


/// The [AudioRepository] is a concrete implementation of the
/// [PlaylistRepository] that control the interaction with
/// firebase.
class AudioRepository extends PlaylistRepository {
  AudioRepository({
    required this.auth,
    required this.firestore,
  });
  final FirebaseAuth auth;
  final FirebaseFirestore firestore;

  /// Get the stream of [AudioMetadata] from firebase
  @override
  Stream<List<AudioMetadata>> getAudioMessagesStream() {
    return firestore
        .collection('users')
        .doc(auth.currentUser!.uid)
        .collection('audios')
        .orderBy('timeSent', descending: true)
        .snapshots()
        .map((event) {
      List<AudioMetadata> audios = [];
      for (var document in event.docs) {
        var audio = AudioMetadata.fromMap(document.data());
        audios.add(audio);
      }
      return audios;
    });
  }

  /// Fetch the playlist that will be passed to the
  /// PlayerController
  @override
  Future<List<AudioMetadata>> fetchPlaylist() async {
    List<AudioMetadata> playlist = await
getAudioMessagesStream().first;
    return playlist.toList();
  }

  /// Other methods below
  (...)
}
```

*Figure 34 – Code snippet of the AudioRepository class that manages the interaction with the audio database.*

The *AudioRepository* class also contains the method toggleAudioMessageFavorite, which is used to toggle the field *isFavorite* of audio stories. Additionally, the same class contains the method *setAudioMessageSeen*, which is analogous to the method used to implement the “is seen” functionality for chat messages.

#### 5.2.14.3 Challenges

The implementation of the functionalities during this sprint was very similar to the app's messaging feature, so it went smoothly.

### 5.2.15 Sprint 14: Improve audio recorder, record preview, the addition of metadata and sending it to other users

#### 5.2.15.1 Objectives

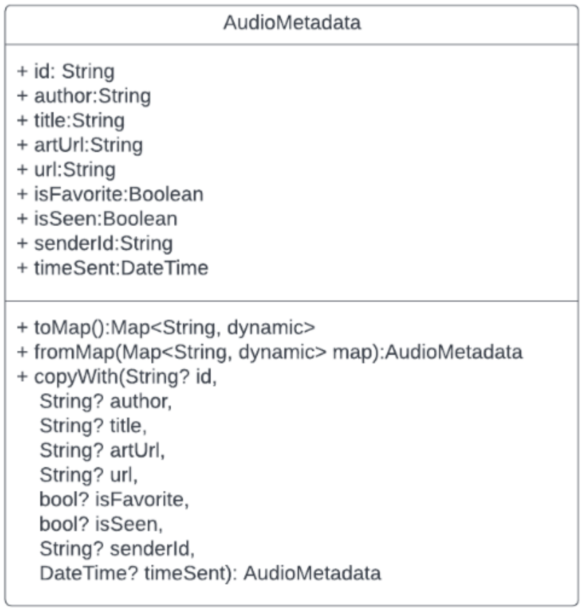
- Improve audio recorder UI.
- Improve audio recording and recording preview.
- Implement the possibility of adding metadata information to the audio recorded, including the author's name, the title and an artwork cover.
- Implement the possibility of selecting users to receive the audio story.

#### 5.2.15.2 Implementation highlights

The *RecorderController* class was created to intermediate the interaction between the recorder external package (*flutter_sound*) and the UI. This class also contains a simple player to preview the audio recorded before sharing it.

The external package *audio_session* was also used to configure the interaction of the recorder and simple player with the device's OS. The audio session configured in the *RecorderController* need to be different from the one configured in the *PlayerController* class for better audio quality.

The *AudioMetadata* class is used to model audio stories in the program (Figure 35). The description of the fields of this class can be seen in Table 7. The class *AuthorTitleCoverScreen* provides the UI for adding metadata data information to the audio recording before sharing it.



```
AudioMetadata
+ id: String
+ author:String
+ title:String
+ artUrl:String
+ url:String
+ isFavorite:Boolean
+ isSeen:Boolean
+ senderId:String
+ timeSent:DateTime

+ toMap():Map<String, dynamic>
+ fromMap(Map<String, dynamic> map):AudioMetadata
+ copyWith(String? id,
   String? author,
   String? title,
   String? artUrl,
   String? url,
   bool? isFavorite,
   bool? isSeen,
   String? senderId,
   DateTime? timeSent): AudioMetadata
```

*Figure 35 – AudioMetadata class.*

Selecting users and sending the audio was implemented following the same logic used to select users and send chat messages. The method *sendAudioMessage* was included in the *ChatRepository* class for this purpose.

#### 5.2.15.3 Challenges

The biggest challenge during this sprint was understanding the technical documentation of the *flutter_sound* package to configure the audio recording parameters correctly and debug the application.

In some cases, while testing the app in an Android device emulator, the recorder did not record the last second of the voice. I believe that happened because of the poor performance of my computer while running the emulator. I did not notice the same bug while testing the app on a physical device. Still, to mitigate the possibility of missing the last second of the audio recorded, I implemented an artificial delay between the moment the button to stop the recorder is tapped and the moment the function to stop the recorder is called (Figure 36).

```
/// Stops the recorder.
void stopRecorder() async {
  // This delay is to ensure the last second is captured by the
recorder.
  await Future.delayed(const Duration(seconds: 1));
  await _recorder!.stopRecorder();
  recordButtonNotifier.value = RecorderStateEnum.recorded;
}
```

*Figure 36 – Code snippet of the method to stop the recorder from the RecorderController class. An artificial delay was added to ensure the last second of the voice was captured before stopping the recorder.*

### 5.2.16 Summary of the external libraries used

The app's dependencies are listed below:

- ***flutter_sound*** is used for recording and previewing recorded audio.
- ***permission_handler*** is used for requesting android permissions and checking their status.
- ***path_provider*** is used for getting the application's current directory and saving local files.
- ***intl*** is used to format the string shown on the recorder's timer.
- ***just_audio*** is used to play audio.
- ***audio_session*** is used to configure the interaction of the audio features of the cellphone's OS.
- ***firebase_core*** is necessary for using the other Firebase services.
- ***firebase_auth*** is used for authenticating users.
- ***firebase_storage*** is used for storing files in the app's database.
- ***cloud_firestore*** is the NoSQL document database to store, sync, and query the app's data.
- ***animated_text_kit*** is used for animating the text on the landing page.
- ***flutter_progress_hud*** is used to show a spinner while the authentication process is ongoing during sign-up and login.
- ***image_picker*** is used to pick images from the cellphone gallery.
- ***flutter_riverpod*** is used to implement the provider design pattern to share the application state across different parts.
- ***email_validator*** is used to validate the email typed by the user before signing up.
- ***flutter_contacts*** is used to access the cellphone's contacts.

- ***uuid*** is used to generate unique ids for different elements (messages, audios and groups).
- ***cached_network_image*** is used to display cached images from URLs.
- ***enough_giphy_flutter*** is used to pick GIFs from Giphy.
- ***swipe_to*** is used to implement the possibility of swiping a message bubble in the chat to reply to it.
- ***emoji_picker_flutter*** is used to pick emojis.
- ***audio_video_progress_bar*** is used to display the total duration and progress of the audio player.
- ***audio_service*** is used to play audio in the background and control the player using the Android interface.
- ***equatable*** is used to compare instances of the same class during unit tests.

# 6 TESTING AND EVALUATION

This report chapter discusses aspects related to testing and evaluating the application developed.

## 6.1 Testing

The following subsections present:

- The types of tests available in the Flutter framework.
- The external libraries that were used for testing the app.
- The approach adopted to test the application developed.
- The testing results.

### 6.1.1 Types of tests available in the Flutter framework

Automated testing in Flutter is divided into three categories. The following descriptions were composed based on the article Testing Flutter apps (Testing Flutter apps):

- Unit tests: They test a single function, method, or class. The goal of a unit test is to verify the correctness of a unit of logic under a variety of conditions. External dependencies of the unit under test are generally mocked out. Unit tests generally do not read from or write to disk, render to screen, or receive user actions from outside the process running the test.
- Widget or component tests: They test a single widget (UI component). A widget test aims to verify that the widget's UI looks and interacts as expected. Testing a widget involves multiple classes and requires a test environment that provides the appropriate widget lifecycle context.
- Integration tests: They test a complete app or a large part of an app. An integration test aims to verify that all the widgets and services being tested work together as expected. Furthermore, integration tests can be used to verify the app's performance.

Table 11 shows the trade-offs between different kinds of tests:

*Table 11 – Tests trade-off. Source: Testing Flutter apps (Testing Flutter apps)*

| Attribute | Unit | Widget (Component) | Integration |
|---|---|---|---|
| Confidence | Low | Higher | Highest |
| Maintenance cost | Low | Higher | Highest |
| Dependencies | Few | More | Most |
| Execution speed | Quick | Quick | Slow |

### 6.1.2 Summary of external libraries used for testing

The external packages used for automated testing are listed below:

- ***test*** is the flutter package for unit testing.
- ***mockito*** is a package for mocking class dependencies.
- ***build_runner*** is used the generate the mocked classes files market with mockito annotations.
- ***rxdart*** is used to create a StreamController to mock the *FirebaseAuth* response.
- ***fake_cloud_firestore*** is used to mock a *FirebaseFirestore* response in tests.
- ***firebase_auth_mocks*** is used to mock a *FirebaseAuth* response in tests.
- ***firebase_storage_mocks*** is used to mock a *FirebaseStorage* response in tests.

### 6.1.3 The approach adopted to test the application developed

The following testing approach was adopted in this project:

- All model classes were subjected to unit testing.
- Classes and respective methods related to user authentication were subjected to both unit and component testing.
- The welcome, error and loading pages were subjected to component testing.
- Manual user acceptance testing was carried out based on a pre-determined set of scenarios and expected results.

Time limitations did not allow for developing automated testing for all components or implementing automated integration testing. Still, the entire user interface and functional requirements were covered by manual user acceptance testing.

### 6.1.4 Test results

Appendix 9.3 contains the test plan adopted with the recorded tests performed and to be developed in future iterations.

Appendix 9.4 presents the entire script of manual user acceptance test scenarios and their results.

Appendix 9.5 contains screens shots showing evidence of the results of the automated tests implemented.

### 6.1.5 Automated tests implementation highlights

This section aims to provide examples of the unit and components tests implemented. The complete set of tests can be found in the test folder in the project package available on the GitHub repository.

#### 6.1.5.1 Unit tests

Figure 37 shows a code snippet of a unit test implemented for the method toMap of the AudioMetadata class. A total of three unit tests were implemented for this class.

```
import 'package:bbk_final_ana/models/audio_metadata.dart';
import 'package:test/test.dart';

void main() {
  group('Testing AudioMetadata class', () {
    test('Testing toMap', () {
      final map = <String, dynamic>{
        'id': '1',
        'author': 'name',
        'title': 'nice',
        'artUrl': 'http://1',
        'url': 'http://2',
        'senderId': '1',
        'timeSent': DateTime(1994).millisecondsSinceEpoch,
        'isFavorite': false,
        'isSeen': false,
      };
      final audioMetadata = AudioMetadata(
        id: map['id'],
        author: map['author'],
        title: map['title'],
        artUrl: map['artUrl'],
        url: map['url'],
        senderId: map['senderId'],
        timeSent:
DateTime.fromMillisecondsSinceEpoch(map['timeSent']),
        isFavorite: map['isFavorite'],
        isSeen: map['isSeen'],
      );
      expect(audioMetadata.toMap(), map);
    });
  /// Other tests below
  });
}
```

*Figure 37 – Code snippet of one unit test implemented for the AudioMetadata class.*

#### 6.1.5.2 Component tests

Figure 38 shows the one component test example for the *AuthRespository* class when a user's login should fail, and an error message is displayed. Note that the dependencies of *AuthRepository* class on the *FirebaseAuth* and the *FirebaseFirestore* classes had to be mocked.

Mocked dependencies mimic the behavior of the existing class's dependencies. Instances of mocked classes can be injected into the classes being tested. The need for mocking dependencies is described in the Flutter documentation (Mock dependencies using Mockito).

```
import...

@GenerateMocks([FirebaseAuth, FirebaseFirestore, User,
UserCredential])
main() {
  group('AuthRepository unit tests', () {

    /// More tests above

    testWidgets(
        'Testing signInWithEmail. Log in fails with incorrect
password.',
        (tester) async {
      final auth = MockFirebaseAuth();
      final firestore = FakeFirebaseFirestore();
      final authRepository = AuthRepository(
       auth: auth,
       firestore: firestore,
      );


      // The function below mocks the FirebaseAuth
      //signInWithEmailAndPassword method when it
      //throws an exception
      when(auth.signInWithEmailAndPassword(
              email: 'ana@email.com', password: '123'))
          .thenThrow(Exception('**wrong-password**'));

      // The primary objective of the code below is to
      // generate the BuildContext to be passed as
      // parameter to the authRepository's method.
      await tester.pumpWidget(const MaterialApp(
        home: Scaffold(
          body: SizedBox(),
        ),
      ));
      final BuildContext context =
                   tester.element(find.byType(SizedBox));

      bool loggedIn =
          await authRepository.signInWithEmail(context,
'ana@email.com', '123');
      await tester.pumpAndSettle(const Duration(seconds: 1));
      expect(loggedIn, false);
      // Check if snack bar is shown
      expect(find.text(Exception('**wrong-password**').toString()),
          findsOneWidget);
    });

    /// More tests below

  });
}
```

*Figure 38 – Code snippet of one component test implemented for the AuthRepository class.*

## 6.2 Evaluation

This section is dedicated to the project evaluation, and it is divided into three parts: Critical evaluation, lessons learnt and future developments.

### 6.2.1 Critical evaluation

The process of writing this dissertation was quite challenging. There was little time to develop, document, and write an academic report covering all the application steps. There were months dedicated solely to this project.

However, the baggage and experience acquired on this journey are immeasurable. There are many steps to building a software application. First, deciding which problem would be tackled and going through academic research on the topic. Second, planning the product's development and choosing the language and tools. Third, designing the software architecture and considering the user experience. Finally, writing the code and being able to see the idea working. All this knowledge and learning will undoubtedly contribute to my future journey professionally and personally.

That is because developing software goes far beyond just knowing how to write code and do research on Stack Overflow. It is necessary to understand what is possible to accomplish. It is imperative to know how to manage time. It is vital to have a mindset that sees challenges as opportunities for growth, not obstacles. It is crucial to recognize that one person does not know everything. It is required to know that bugs will always happen. It is also imperative to know that they will be resolved. It is necessary to know that other bugs will then arise. Finally, it is essential to understand how to deal with the challenges that the journey of creating a program entails.

#### 6.2.1.1 Achievements

In the first place, considering the interface, it is simple and intuitive. Besides that, it is also inviting and ludic. In addition, the design is well done because a good user experience was the primary goal. An example is that the user can see a difference between the stories already listened to and the new ones because these appear with the tag "new". Another example is the user manual, which is simple, direct, and pleasant.

In the second place, taking the code into account, everything that was implemented is working. SOLID principles were respected, and dependency injection was used, both responsible for a more maintainable code. And specific packages were used for each function, avoiding thus redoing work and enabling to save time.

In the third place, concerning tests, extensive manual integration tests were performed. Also, unit tests were implemented for the main classes, and all the code has been extensively debugged.

For clarification, one may ask about the purpose of a chat in an application designed mainly for people without digital literacy. Still, considering there may be more resourceful users, the chat provides another form of exchange for these. The advantage of this over using an external application, such as WhatsApp, is because it is an environment directed just for that, while in WhatsApp, the possibility of distraction with other subjects is greater.

#### 6.2.1.2 Drawbacks

The challenges related to each specific sprint were described previously in this report. Additionally, because of time-consuming activities, I ran out of time for doing tests with real users, which is a significant loss. This is the main drawback of the project.

#### 6.2.1.3 Lessons learnt

More experience was gained with the provider pattern throughout the development. It was initially decided to use the Riverpod package. However, this package adds complexity to the code, and a different strategy for the application's state management would probably be adopted. The approach detailed in the article "Flutter state management for minimalists" (Flutter state management for minimalists) would be a more suitable solution.

Another lesson learnt was that uncertainty forced the adoption of a strategy of putting all the efforts into development first, stopping to document the code later. This decision resulted in a less efficient process for managing the project. One example of an unpredictable situation that led to this decision was that, in one moment, the code logic was correct, but the application was not working as expected. After much effort, it was discovered that it was necessary to uninstall and reinstall the application on the cell phone using Android. Once that was done, the software worked.

Something similar happened related to the tests. An agile implementation methodology was adopted, and discoveries were made throughout the project about how different parts should be implemented. Because of that, it was decided to start implementing the main code and then develop the tests. However, suppose I had more experience and clarity regarding implementing the parts to meet the functional requirements. In that case, a test-driven development could have been an option resulting in higher testing coverage.

### 6.2.2 Possible further developments and improvements

For the tests, it would be good to extend the coverage of unit tests; implement automatic integration tests, and extend the coverage of unit testing mainly to classes where there are dependencies. The implementation of automated tests can be an improvement. However, depending on user feedback, more radical changes may be necessary for developing the software as a product. This could lead to the need to change the integration tests, which would slow down the development process. In addition, it would be essential to experiment with real users and map how they react to and interact with the app, and through that feedback, improve the app.

Besides, it would be nice to manage the necessary permissions so that the code also runs on iOS. This was not done because it was estimated that it would add 10% more time to the process, equating to one to two more weeks of work. As the deadline for delivery was short, the option of not running the risk of delay was taken. In this regard, the preference given to Android is due to its better adaptation to the application personas and broader representation in the global market.

For the notifications, it would be nice to receive notifications on Android and chat and know the number of unread messages. The goal would be to improve user interaction and engagement. In addition, on the library screen, it would be nice to see the notification of new messages. For the chat, it would be nice to be able to click on the user's photo and enlarge it and also get to see all members of a group.

It is imperative to have a "forgot your password" functionality. Another improvement would be to better master the recording package to improve audio quality, with less noise and more intensity.

For both users, the one who records and the one who listens, it would be good to have a search option for stories and messages. In addition, allowing login using a phone number instead of email and creating invite functionality for new users would be good.

For the listener, it would be nice to be able to download the audio for offline listening. In addition, it would be nice to be able to organize the playlist in a selected order (for now, it is only possible to listen to the playlist in chronological or random order).

For the recorder, it would be nice if the user could: seek a specific position of the audio while previewing it; edit the audio recorded; create chapter tags for the audio and save it as a draft. Besides, it would be nice to share the audio more than once. It would also be good to have the story's text on the screen while speaking and have a more extensive artwork cover database. In addition, putting the story's title on the cover would be good.

# 7 CONCLUSIONS

In summary, the project's achievements were delivering a good user experience, with a straightforward user journey, software that works and an uncomplicated and pleasant interface. Besides that, code-wise, the implementation works and best practices were used, for example, following SOLID principles and using dependency injection. In addition, concerning tests, the code was broadly debugged and manually tested. The two main lessons learnt were that it is more efficient to document the code while developing and that a test-driven development would probably result in more substantial testing coverage. Respecting what could be improved, many advances could be made. Still, the main ones are testing the application with real users to get feedback for improvements and to log in with a phone number instead of email because that makes the sign-up process less demanding for users that do not have digital literacy.

# 8 REFERENCES

**Audio Stories Sharing App Sabiá – Manual**. https://www.youtube.com/watch?v=_As_m1PQnBs. 10/09/22, 11:41.

BUSCHMANN, Frank *et al*. **Pattern-Oriented Software Architecture**, Volume 1, A System of Patterns. New York: John Wiley & Sons, 1996.

**Canva**. https://www.canva.com/about/. 02/09/22, 14:00.

CASTRO, Ana Luiza Silva de. Audio exchange application to tackle loneliness of older adults. – London: Birkbeck University/ Department of Computer Science and Information Systems, 2022.

**Cloud Firebase Storage**. https://firebase.google.com/docs/storage. 10/09/22, 15:55.

**Cloud Firestore**. https://firebase.google.com/docs/firestore. 10/09/22, 15:30.

**Curves class**. https://api.flutter.dev/flutter/animation/Curves-class.html. 10/09/22, 10:35.

**Figma**. https://www.figma.com/about/. 06/07/22, 11:00.

**Firebase Authentication**. https://firebase.google.com/docs/auth. 11/09/22, 12:20.

**Flutter architectural overview.** https://docs.flutter.dev/resources/architectural-overview. 08/09/22, 15:00.

**Flutter state management for minimalists**. https://suragch.medium.com/flutter-state-management-for-minimalists-4c71a2f2f0c1. 12/09/22, 13:47.

**Giphy**. https://developers.giphy.com/. 10/09/22, 14:30.

**How to Install and Setup Flutter for App Development on Windows**. https://www.youtube.com/watch?v=Z2ugnpCQuyw. 08/09/22, 08:52.

KALBAG, Laura. **Accessibility for everyone**. New York: A Book Apart, 2017.

**Mock dependencies using Mockito**. 13/09/22, 18:00.

NAEIM, Mahdi; REZAEISHARIF, Ali; KAMRAN, Aziz. COVID-19 has made the elderly lonelier. *Dementia and Geriatric Cognitive Disorders Extra*, v. 11, n. 1, 2021, (p. 26-28). Available in: <https://www.karger.com/Article/FullText/514181>. Access in: 03/03/22.

**Observer Design Pattern**. https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern. 09/09/22, 17:52.

**Package layout conventions**. https://dart.dev/tools/pub/package-layout. 09/09/22, 16:55.

**Riverpod 1.0.3+1**. https://pub.dev/packages/riverpod. 11/09/22, 13:50.

**SoundHelix web audio examples**. https://www.soundhelix.com/audio-examples. 18/09/2022, 22:14.

SURAGCH. Background audio in Flutter with Audio Service and Just Audio. Article available on: https://suragch.medium.com/background-audio-in-flutter-with-audio-service-and-just-audio-3cce17b4a7d. Last access: 18/09/2022.

SURAGCH. *Managing playlists in Flutter with Just Audio*. Article available on: https://suragch.medium.com/managing-playlists-in-flutter-with-just-audio-c4b8f2af12eb. Last access: 18/09/2022.

**Testing Flutter Apps**. https://docs.flutter.dev/testing. 13/09/22, 15:11.

**The Principles of OOD**. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod. 11/09/22, 13:43.

WALTER, Aarron. **Designing for emotion**. New York: A book apart, 2011.

**Widgets Binding Observer class**. https://api.flutter.dev/flutter/widgets/WidgetsBindingObserver-class.html. 08/09/22, 15:55.

**Windows install**. https://docs.flutter.dev/get-started/install/windows. 10/09/22, 16:05.

**Write your first Flutter app**. https://docs.flutter.dev/get-started/testdrive?tab=androidstudio#create-app. 09/09/22, 15:00.
YU, Angela. **The Complete Flutter Development Bootcamp with Dart**. 2021.

# 9 APPENDIX

## 9.1 Requirements lists

### 9.1.1 Non-functional requirements full list

*Table 12 – Non-functional requirements full list.*

| Requirement ID | Requirement Statement | Category (Must-have / Nice-to-have) | Evaluation |
|---|---|---|---|
| NFR01 Time | The app, the documentation and the report shall be ready by 19/09/22. | Must | The time requirement was completed with success. |
| NFR02 Usability | User interface: shall be user-friendly. Users shall be able to navigate the app without external help. | Must | The usability requirement was completed with partial success. Someone with relative technological know-how may be needed to assist the elderly during the set-up process. |
| NFR03 Security | - Login;<br>- Password. | Must | The security requirement was completed with success. |
| NFR04 Documentation | - System documentation;<br>- Training material (manual). | Must | The documentation requirement was completed with success. |

### 9.1.2 Functional requirements full list

*Table 13 – Functional requirements full list.*

| Requirement ID | Requirement Statement | Category (Must-have / Nice-to-have) | Evaluation |
|---|---|---|---|
| FR01 | The app shall have a set-up page. | Must | Requirement completed with success. |
| FR01 – a | The set-up page shall contain a field for the user email. | Must | Requirement completed with success. |
| FR01 – b | The set-up page shall contain a field for the user password. | Must | Requirement completed with success. |
| FR01 – c | The set-up page shall contain a "Sign up" button. | Must | Requirement completed with success. |
| FR01 – d | The set-up page shall have user input for the image and the user name. | Must | Requirement completed with success. |
| FR02 | The app shall have a page for the user to choose between listening to a story or recording a story. | Must | Requirement completed with success. |
| FR03 | The app shall have a page – Library - for the user listening to a story. | Must | Requirement completed with success. |
| FR03 - a | The most recent stories shall be on the top of the page. | Nice | Requirement completed with success. |
| FR03 - b | The user shall be able to categorize a story as favorite. | Nice | Requirement completed with success. |
| FR03 - c | The user shall be able to click on a story and listen to it. | Must | Requirement completed with success. |
| FR04 | The app shall have a story player page. | Must | Requirement completed with success. |
| FR04 – a | The story player page shall have a play/pause button. | Must | Requirement completed with success. |
| FR04 – b | The story player page shall have the next and previous buttons. | Must | Requirement completed with success. |
| FR04 – c | The story player page shall have a repeat button. | Nice | Requirement completed with success. |
| FR04 – d | The story player page shall enable the user to repeat the story. | Nice | Requirement completed with success. |
| FR04 – e | The story player page shall enable the user to repeat the playlist. | Nice | Requirement completed with success. |
| FR04 – f | The story player page shall have a shuffle button. | Nice | Requirement completed with success. |

| | | | |
|---|---|---|---|
| FR04 – g | The story player page shall have a seek bar. | Must | Requirement completed with success. |
| FR04 – h | The story player page shall have a velocity button. | Must | Requirement completed with success. |
| FR04 – i | The story player page shall enable the user to block the screen and keep listening to a story. | Nice | Requirement completed with success. |
| FR04 – j | The story player page shall enable the user to navigate the stories on the playlist. | Nice | Requirement completed with success. |
| FR04 – k | The story player page shall enable the user to use the android menu to control the player. | Nice | Requirement completed with success. |
| FR05 | The app shall have a recording page. | Must | Requirement completed with success. |
| FR05 – a | The recording page shall have a recorder button. | Must | Requirement completed with success. |
| FR05 – b | The recording page shall enable the user to listen to the record. | Must | Requirement completed with success. |
| FR05 – c | The recording page shall have a refresh button. | Must | Requirement completed with success. |
| FR05 – d | The recording page shall have a done button. | Must | Requirement completed with success. |
| FR06 | The app shall have an "About your story page". | Must | Requirement completed with success. |
| FR06 – a | The about your story page shall have a field for the story's title name and one for the author's name. | Must | Requirement completed with success. |
| FR06 – b | The about your story page shall enable the user to select a cover for the story. | Must | Requirement completed with success. |
| FR07 | The app shall have a "Send a story to" page. | Must | Requirement completed with success. |
| FR07 – a | The "Send a story to" page shall enable the sender user to select one or more receiver users. | Must | Requirement completed with success. |
| FR07 – b | The "Send a story to" page shall have a send button. | Must | Requirement completed with success. |
| FR08 | The app shall have a navigation menu. | Must | Requirement completed with success. |
| FR08 – a | The navigation menu shall enable the user to access the library. | Must | Requirement completed with success. |
| FR08 – b | The navigation menu shall enable the user to record a story. | Must | Requirement completed with success. |
| FR08 – c | The navigation menu shall enable the user to access the chat. | Must | Requirement completed with success. |

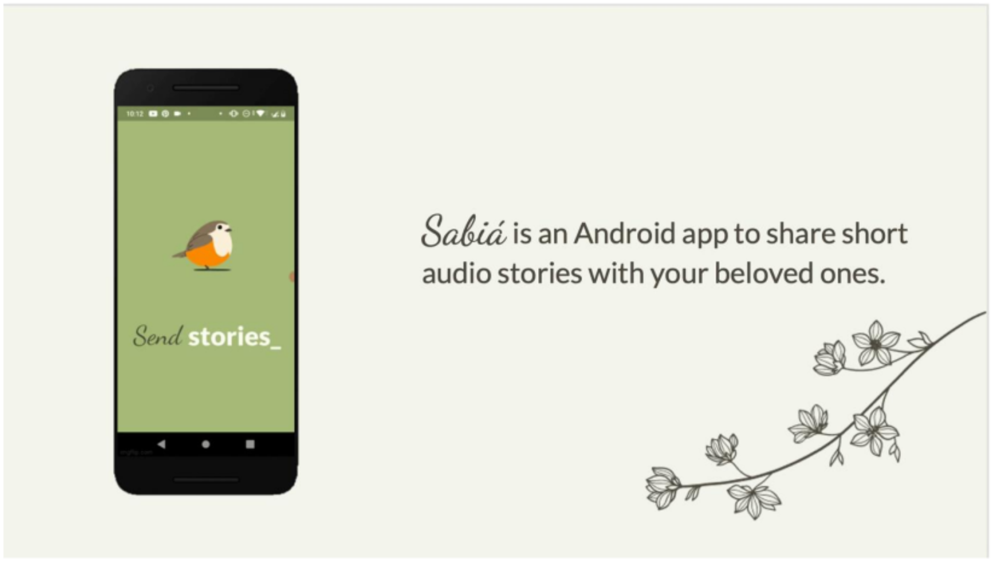
| | | | |
|---|---|---|---|
| FR08 – d | The navigation menu shall enable the user to create a group. | Must | Requirement completed with success. |
| FR08 – e | The navigation menu shall enable the user to access the profile. | Must | Requirement completed with success. |
| FR08 – f | The navigation menu shall enable the user to log out. | Must | Requirement completed with success. |
| FR09 | The app shall have a chat page. | Nice | Requirement completed with success. |
| FR09 – a | The chat page shall enable the users to send messages. | Nice | Requirement completed with success. |
| FR09 – b | The chat page shall enable the users to select a new message button or an existing conversation. | Nice | Requirement completed with success. |
| FR09 – c | The chat page shall enable the users to send a text. | Must | Requirement completed with success. |
| FR09 – d | The chat page shall enable the users to send emojis and GIFs. | Nice | Requirement completed with success. |
| FR09 – e | The chat page shall have a group chat page. | Nice | Requirement completed with success. |
| FR09 – f | The group chat page shall enable the user to create groups of contacts. | Nice | Requirement completed with success. |
| FR09 – f) a | The group chat page shall enable the user to select a photo for the group. | Nice | Requirement completed with success. |
| FR09 – f) b | The group chat page shall enable the user to select a name for the group. | Nice | Requirement completed with success. |
| FR09 – f) c | The group chat page shall display the group created. | Nice | Requirement completed with success. |
| FR09 – f) d | The group chat page shall enable the user to send text, emojis and GIFs. | Nice | Requirement completed with success. |
| FR10 | The app shall have a user profile page. | Must | Requirement completed with success. |
| FR10 – a | The user profile page shall enable the user to change the photo and the name. | Must | Requirement completed with success. |
| FR11 | The app shall have authentication persistence. | Nice | Requirement completed with success. |
| FR12 | Error case: the user email input is not registered.<br>The field for email input should become red, and an error message should be displayed at the base of the field with the content: "There is no account with this email". | Nice | Requirement completed with success. |

|  |  |  |  |
|---|---|---|---|
| FR13 | Error case: the user email input is not valid. The field for email input should become red, and an error message should be displayed at the base of the field with the content: "Insert a valid email address". | Nice | Requirement completed with success. |
| FR14 | Error case: the password is incorrect. The password field turns red, and an error message appears at the base of the field with the content: "Incorrect password". | Nice | Requirement completed with success. |

## 9.2 User Manual

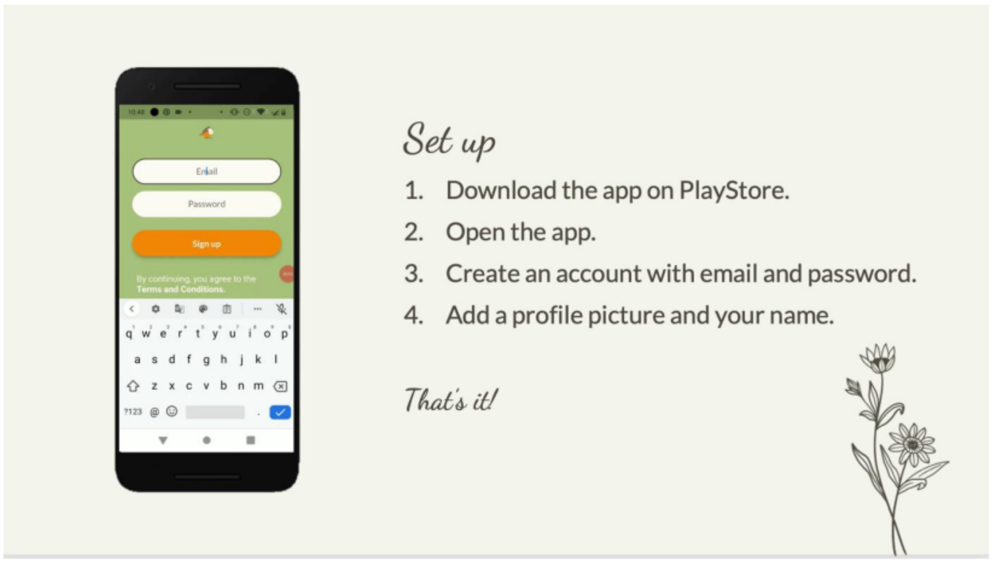
This section describes the user manual, which can also be seen as a video at Audio Stories Sharing App Sabiá – Manual.

*Figure 39 – First page.*



Sabiá is an Android app to share short audio stories with your beloved ones.

*Figure 40 – Contextualization.*

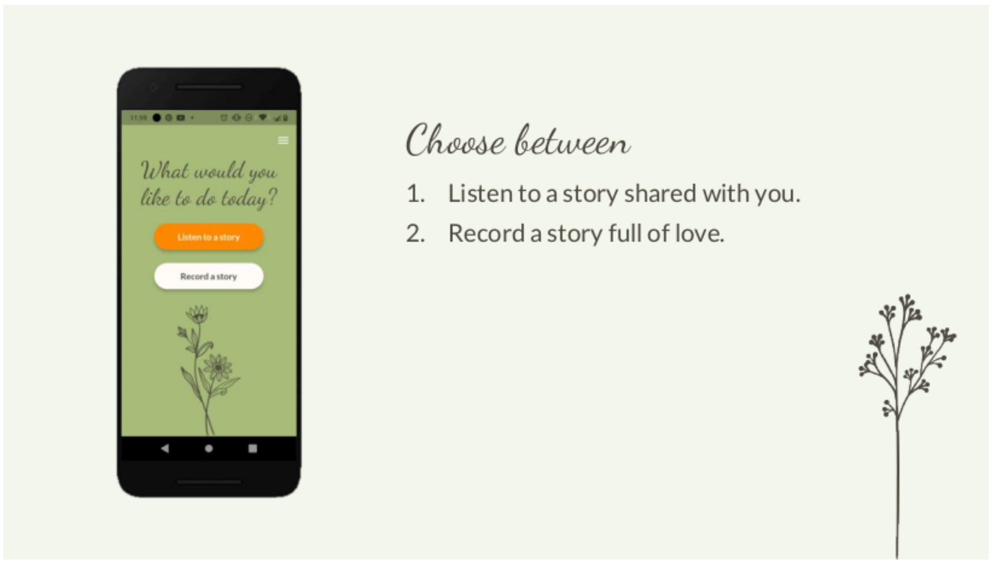


Set up

1. Download the app on PlayStore.
2. Open the app.
3. Create an account with email and password.
4. Add a profile picture and your name.

That's it!

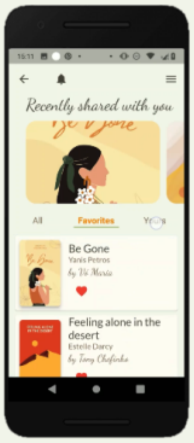
*Figure 41 – Set up.*



Choose between

1. Listen to a story shared with you.
2. Record a story full of love.

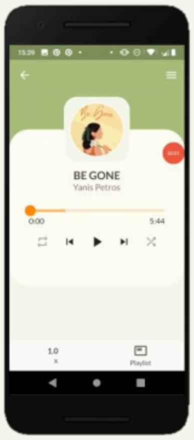
*Figure 42 – Choosing an option.*



## Listening to a story

1. Stories will be organized in the library.
2. The five most recent stories will be on the top.
3. Below, each tab shows:
   a. All stories shared with you.
   b. Your favorites.
   c. The stories you recorded.
4. Tap on a story to listen to it.

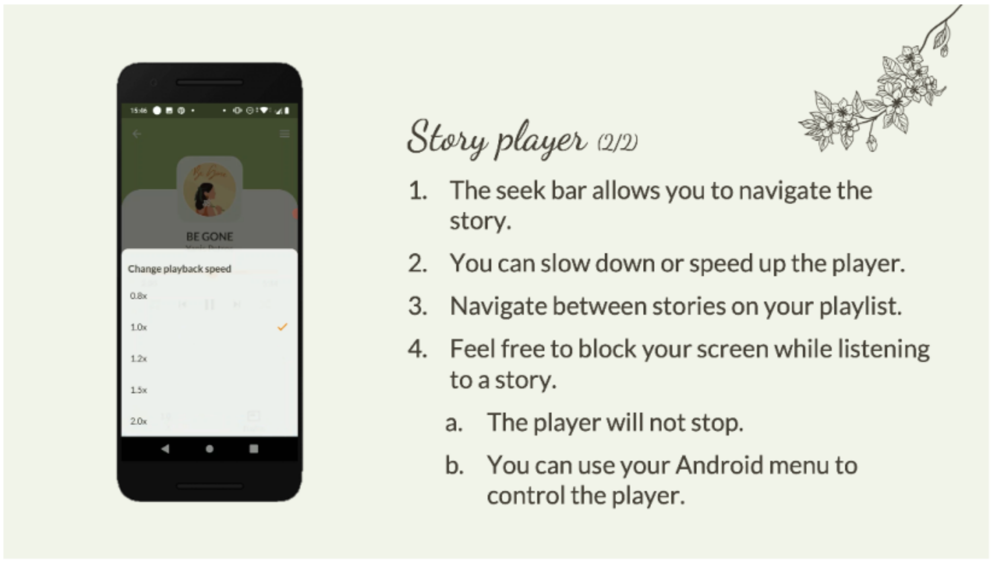
*Figure 43 – Listening to a story.*



## Story player (1/2)

1. Tap the play button to play or stop the story.
2. The next and previous buttons allow you to jump between stories on your playlist.
3. Tap the repeat button once to set the current story to repeat.
4. Tap it twice to set the entire playlist to repeat.
5. Tap it again to disable the repeat mode.
6. Tap the shuffle button to listen to the playlist in a random order.

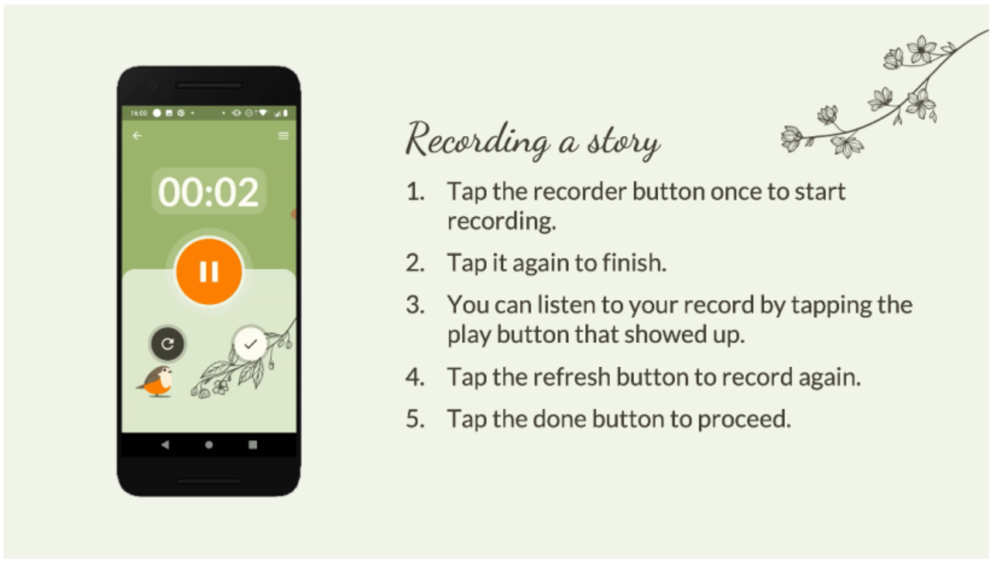
*Figure 44 – Story player (1/2).*



## Story player (2/2)

1. The seek bar allows you to navigate the story.
2. You can slow down or speed up the player.
3. Navigate between stories on your playlist.
4. Feel free to block your screen while listening to a story.
   a. The player will not stop.
   b. You can use your Android menu to control the player.

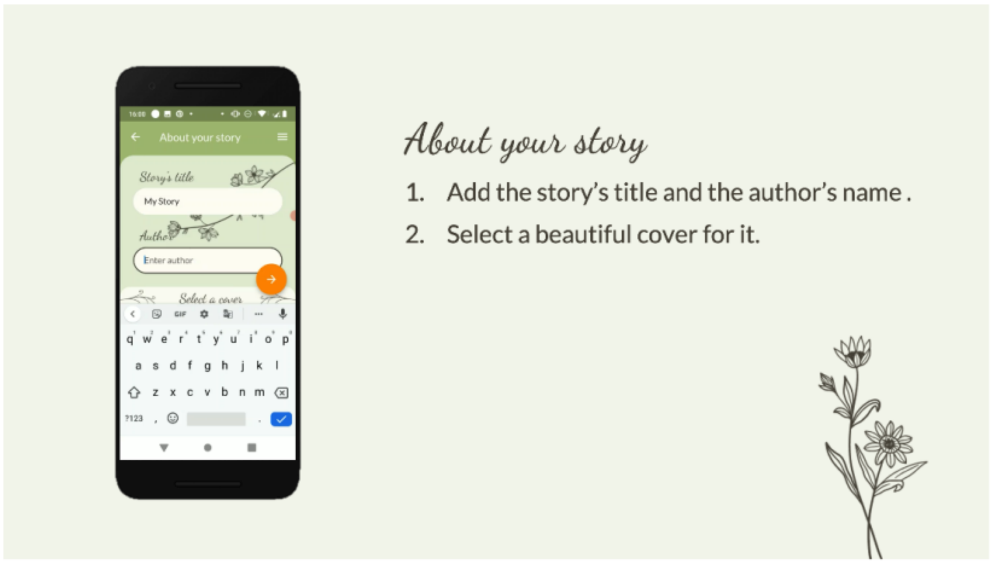
*Figure 45 – Story player (2/2).*



## Recording a story

1. Tap the recorder button once to start recording.
2. Tap it again to finish.
3. You can listen to your record by tapping the play button that showed up.
4. Tap the refresh button to record again.
5. Tap the done button to proceed.

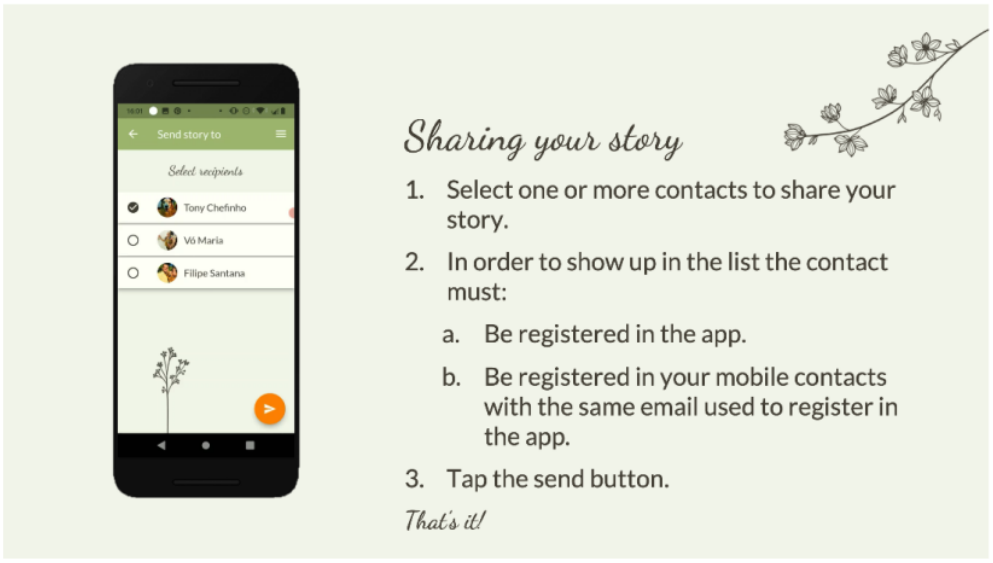
*Figure 46 – Recording a story.*



## About your story

1. Add the story's title and the author's name .
2. Select a beautiful cover for it.

*Figure 47 – About the story.*

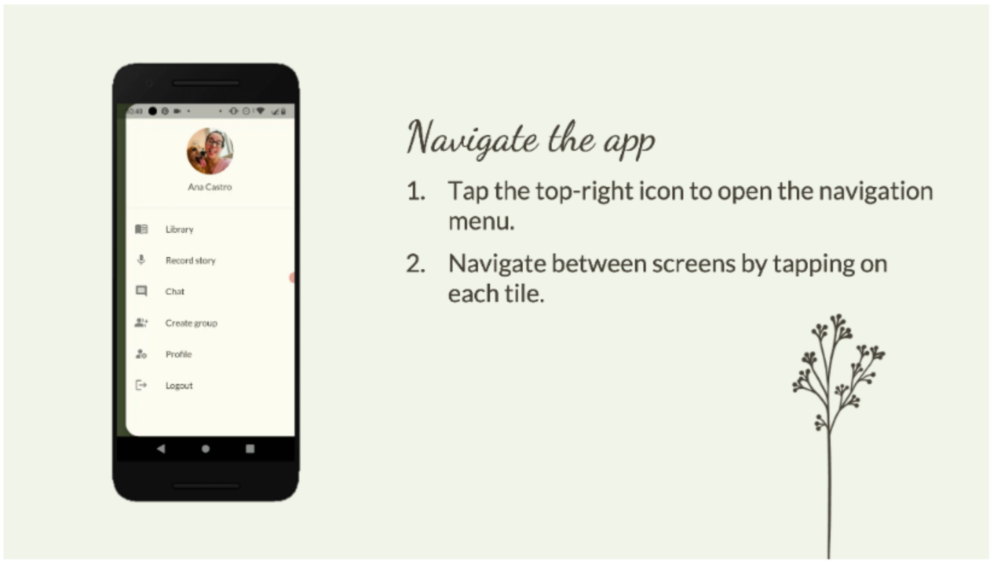


## Sharing your story

1. Select one or more contacts to share your story.
2. In order to show up in the list the contact must:
   a. Be registered in the app.
   b. Be registered in your mobile contacts with the same email used to register in the app.
3. Tap the send button.

*That's it!*

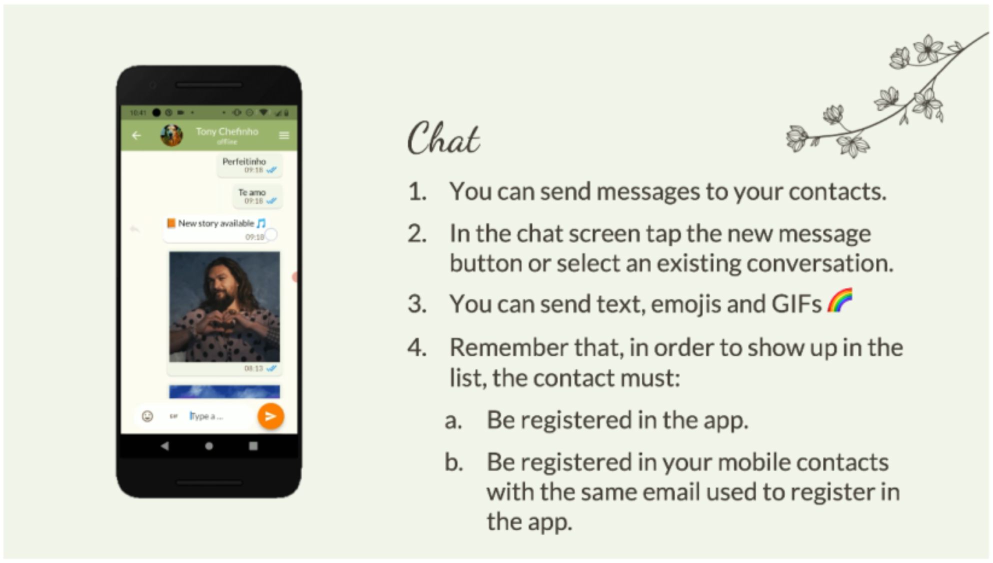
*Figure 48 – Sharing the story.*



## Navigate the app

1. Tap the top-right icon to open the navigation menu.
2. Navigate between screens by tapping on each tile.

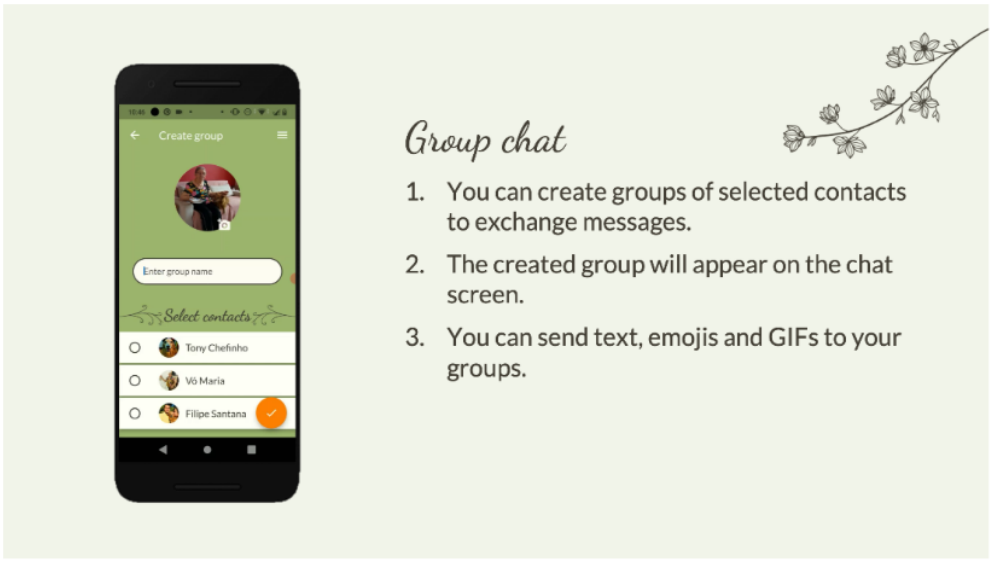
*Figure 49 – Navigate the app*.



## Chat

1. You can send messages to your contacts.
2. In the chat screen tap the new message button or select an existing conversation.
3. You can send text, emojis and GIFs 🌈
4. Remember that, in order to show up in the list, the contact must:
   a. Be registered in the app.
   b. Be registered in your mobile contacts with the same email used to register in the app.

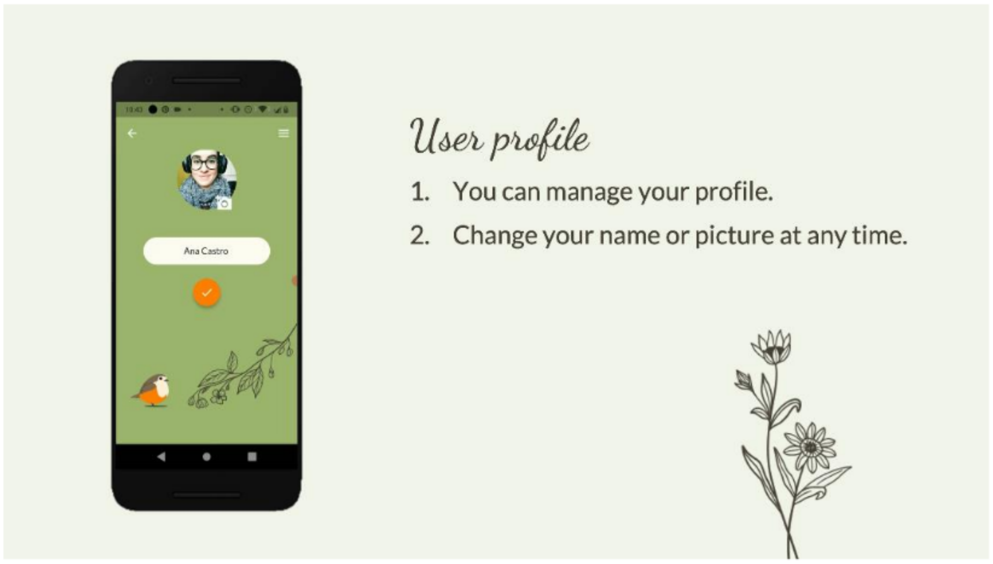
*Figure 50 – Chat*.



## Group chat

1. You can create groups of selected contacts to exchange messages.
2. The created group will appear on the chat screen.
3. You can send text, emojis and GIFs to your groups.

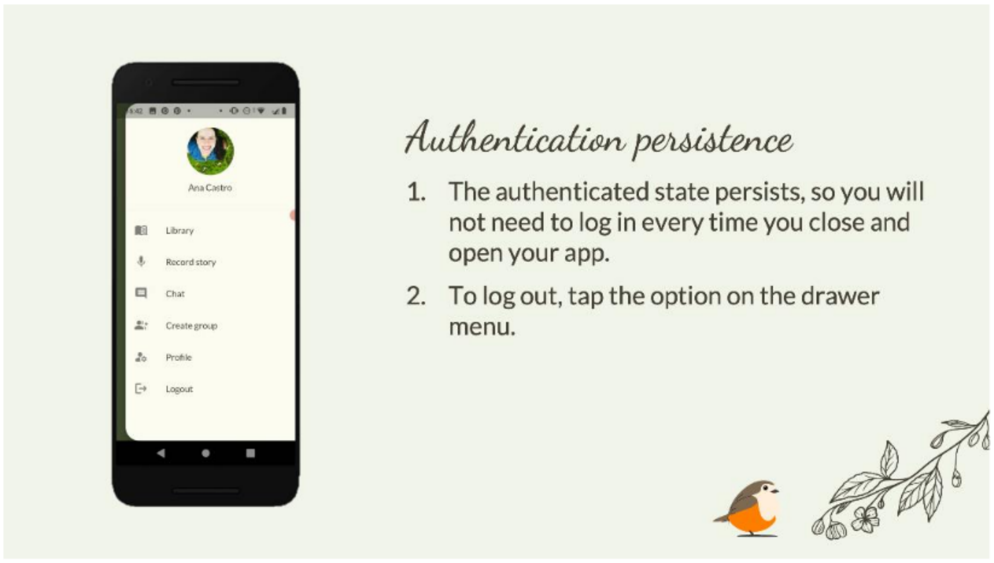
*Figure 51 – Group chat.*



## User profile

1. You can manage your profile.
2. Change your name or picture at any time.

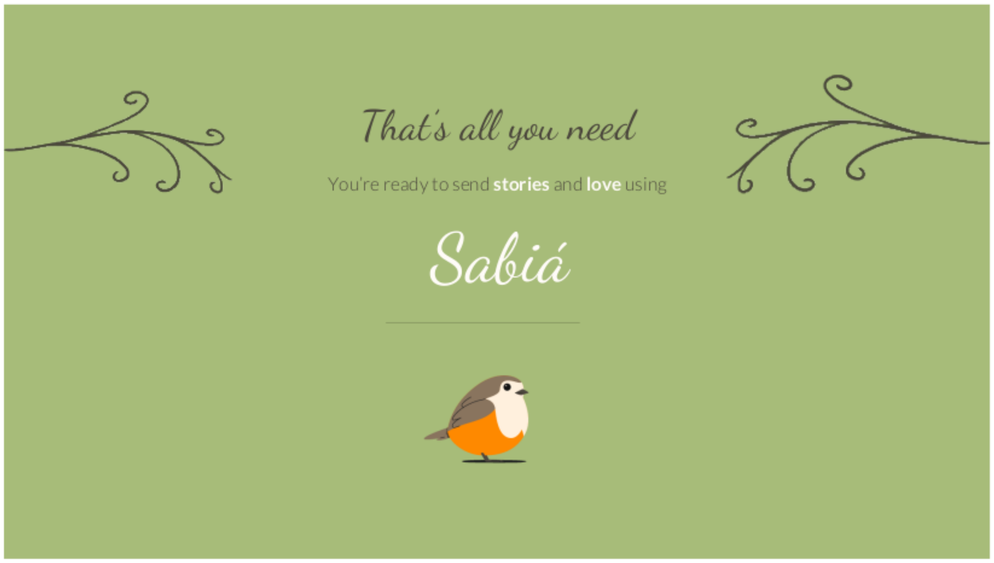
*Figure 52 – User profile.*



## Authentication persistence

1. The authenticated state persists, so you will not need to log in every time you close and open your app.
2. To log out, tap the option on the drawer menu.

*Figure 53 – Authentication persistence.*

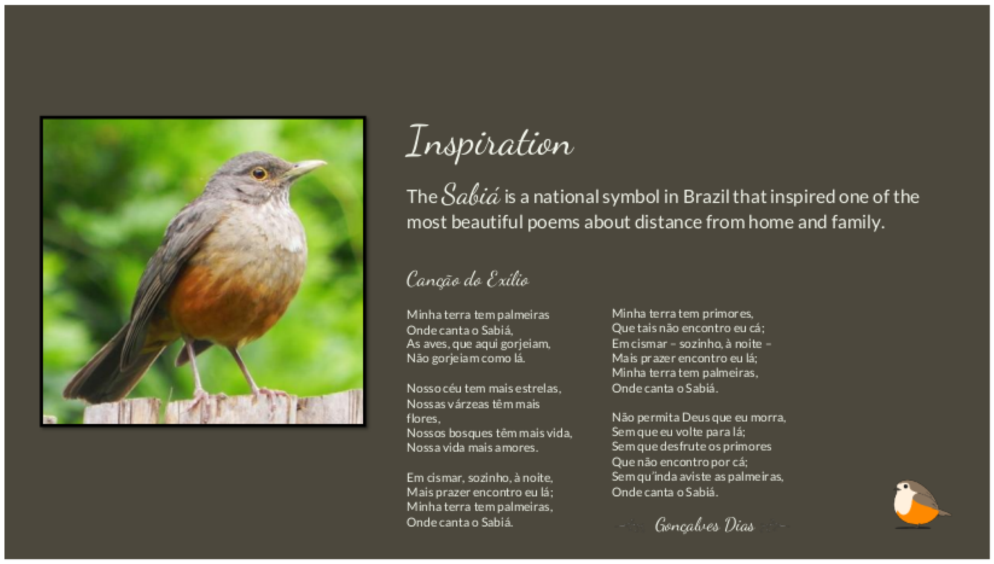


## That's all you need

You're ready to send **stories** and **love** using

Sabiá

*Figure 54 – End page.*



## Inspiration

The *Sabiá* is a national symbol in Brazil that inspired one of the most beautiful poems about distance from home and family.

*Canção do Exílio*

Minha terra tem palmeiras
Onde canta o Sabiá,
As aves, que aqui gorjeiam,
Não gorjeiam como lá.

Nosso céu tem mais estrelas,
Nossas várzeas têm mais
flores,
Nossos bosques têm mais vida,
Nossa vida mais amores.

Em cismar, sozinho, à noite,
Mais prazer encontro eu lá;
Minha terra tem palmeiras,
Onde canta o Sabiá.

Minha terra tem primores,
Que tais não encontro eu cá;
Em cismar – sozinho, à noite –
Mais prazer encontro eu lá;
Minha terra tem palmeiras,
Onde canta o Sabiá.

Não permita Deus que eu morra,
Sem que eu volte para lá;
Sem que desfrute os primores
Que não encontro por cá;
Sem qu'inda aviste as palmeiras,
Onde canta o Sabiá.

*Gonçalves Dias*

*Figure 55 – Inspiration.*

## 9.3 Test plan

*Table 14 – Test plan. Part 1 out of 5.*

| Feature | Path | File | Unit test | Component test | Manual integration test |
|---|---|---|---|---|---|
| audio | audio/controller | audio_handler.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | | player_controller.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | | recorded_audio_handler.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | | recorder_controller.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | audio/enums | library_filters_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | player_state_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | recorder_state_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | library_filters_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | repeat_button_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | audio/notifiers | audio_metadata_notifier.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | play_button_notifer.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | player_progress_notifer.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | player_progress_state.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | recoder_progress_notifier.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | record_button_notifier.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | recorder_progress_state.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | | repeat_button_notifier.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | audio/repository | playlist_repository.dart | ☐ To do | ☐ To do | ☑ Done |

*Table 15 – Test plan. Part 2 out of 5.*

| Feature | Path | File | Unit test | Component test | Manual integration test |
|---|---|---|---|---|---|
| audio | audio/screens | author_title_cover_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | | initial_decision_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | | library_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | | player_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | | recorder_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | | send_to_screen.dart | ☐ To do | ☐ To do | ☑ Done |
| | audio/widgets | audio_control_buttons.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | audio_progress_bar.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | current_audio_artwork.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | current_audio_author.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | current_audio_title.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | done_recording_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | list_header.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | next_audio_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | play_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | player_botton_bar.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | player_speed_list.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | playlist.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | pevious_audio_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | recorder_button.dart | ⃠ N/A | ☐ To do | ☑ Done |

*Table 16 – Test plan. Part 3 out of 5.*

| Feature | Path | File | Unit test | Component test | Manual integration test |
|---|---|---|---|---|---|
| audio | audio/widgets | recorder_secondary_buttons_bar.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | recoder_timer.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | recorder_ui.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | repeat_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | restart_recording_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | shuffle_button.dart | ⃠ N/A | ☐ To do | ☑ Done |
| auth | auth/controller | auth_controller.dart | ☑ Done | ☑ Done | ☑ Done |
| | auth/repository | auth_repository.dart | ☑ Done | ☑ Done | ☑ Done |
| | auth/screens | edit_user_info_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | login_screen.dart | ⃠ N/A | ☑ Done | ☑ Done |
| | | registration_screen.dart | ⃠ N/A | ☑ Done | ☑ Done |
| | | user_info_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| common | common/constants | constants.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | common/enums | message_enum.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | common/providers | message_reply_provider.dart | ⃠ N/A | ⃠ N/A | ☑ Done |
| | common/repositories | common_firestore_repository.dart | ☑ Done | ☑ Done | ☑ Done |
| | common/screens | error_screen.dart | ⃠ N/A | ☑ Done | ☑ Done |
| | | loader_screen.dart | ⃠ N/A | ☑ Done | ☑ Done |

*Table 17 – Test plan. Part 4 out of 5.*

| Feature | Path | File | Unit test | Component test | Manual integration test |
|---|---|---|---|---|---|
| common | common/widgets | circular_cached_nework_image.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | drawer_menu.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | field_title.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | rounded_button_primary.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | rounded_button_secondary.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | screen_basic_structure.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | square_cached_network_image.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | standard_circular_profile_avatar.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | standard_circular_progress_indicator.dart | ⃠ N/A | ☐ To do | ☑ Done |
| landing | landing/screens | welcome_screen.dart | ⃠ N/A | ☑ Done | ☑ Done |
| messaging | messaging/chat/controller | chat_controller.dart | ☐ To do | ☐ To do | ☑ Done |
| | messaging/chat/repository | chat_repository.dart | ☐ To do | ☐ To do | ☑ Done |
| | messaging/chat/screens | chat_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | messaging/chat/widgets | bottom_chat_field.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | chat_list.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | display_text_image_gif.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | message_reply_preview.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | my_message_card.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | | sender_message_card.dart | ⃠ N/A | ☐ To do | ☑ Done |

*Table 18 – Test plan. Part 5 out of 5.*

| Feature | Path | File | Unit test | Component test | Manual integration test |
|---|---|---|---|---|---|
| messaging | messaging/conversations/screens | conversations_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | messaging/conversations/widgets | conversations_list.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | messaging/group/controller | group_controller.dart | ☐ To do | ☐ To do | ☑ Done |
| | messaging/group/repository | group_repository.dart | ☐ To do | ☐ To do | ☑ Done |
| | messaging/group/screen | create_group_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | messaging/group/widgets | select_user_contact_group.dart | ⃠ N/A | ☐ To do | ☑ Done |
| | messaging/select_contacts/controller | select_contact_controller.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | messaging/select_contacts/repository | select_contact_repository.dart | ☐ To do | ⃠ N/A | ☑ Done |
| | messaging/select_contacts/screens | select_user_contact_screen.dart | ⃠ N/A | ☐ To do | ☑ Done |
| models | models | audio_metadata.dart | ☑ Done | ⃠ N/A | ☑ Done |
| | | chat_conversation.dart | ☑ Done | ⃠ N/A | ☑ Done |
| | | group.dart | ☑ Done | ⃠ N/A | ☑ Done |
| | | message.dart | ☑ Done | ⃠ N/A | ☑ Done |
| | | user_model.dart | ☑ Done | ⃠ N/A | ☑ Done |
| utils | utils | utils.dart | ⃠ N/A | ☑ Done | ☑ Done |

## 9.4 User acceptance tests

*Table 19 – User acceptance tests*

| Test ID | Feature tested | Test Scenario | Expected result | Pass/Fail |
|---|---|---|---|---|
| 1 | Sign up | User signs up with valid email and password. | User succeeds. The profile screen is shown. | Pass |
| 2 | Sign up | User attempts to sign up with invalid email and password. | User does not succeed. An error message is displayed. | Pass |
| 3 | Sign up | User attempts to sign up with an email already registered in the app. | User does not succeed. An error message is displayed. | Pass |
| 4 | Login | User logs in with email and correct password. | User succeeds. The initial decision screen is shown. | Pass |
| 5 | Login | User attempts to log in with email and incorrect password. | User does not succeed. An error message is displayed. | Pass |
| 6 | Listen to a story | User tap the "Listen to a story" on the initial decision screen. | The library screen is displayed. | Pass |
| 7 | Listen to a story | User has three audios in the database. | The library screen displays three audios. | Pass |
| 8 | Listen to a story | User marks an audio as favorite. | The fill of the heart icon turns red and the audio can also be seen in the favorites tab. | Pass |
| 9 | Listen to a story | User taps the favorite tab on the library screen. | Only audios marked as favorite are displayed in the screen. | Pass |
| 10 | Listen to a story | User taps the all tab on the library screen. | All audios received by the user are displayed ordered by the most recent date. | Pass |
| 11 | Listen to a story | User receives new audio. | The audio shows up in the library screen with an icon "new" besides it. | Pass |
| 12 | Listen to a story | User receives six audios. | Only five audio covers are displayed in the "recently shared with you" section of the library screen. | Pass |
| 13 | Listen to a story | User taps on a story in the library screen. | The player screen shows up with the information of the selected audio. | Pass |

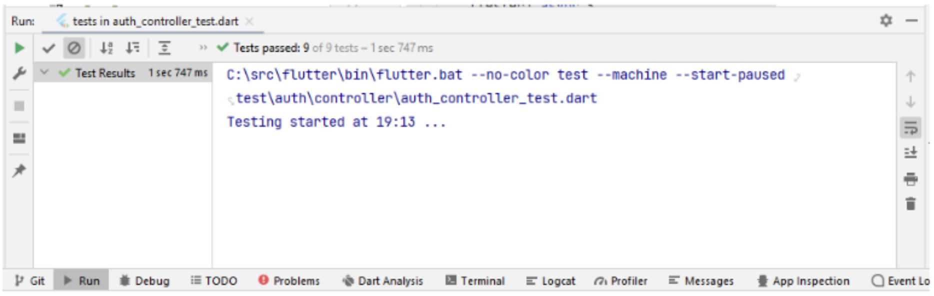
| | | | | |
|---|---|---|---|---|
| 14 | Listen to a story | User taps on the play button of the player screen. | The audio starts to play. The play button becomes a stop button. The audio plays until the end and then the next audio in the playlist starts to play. | Pass |
| 15 | Listen to a story | User taps on the stop button of the player screen. | The audio stops. The stop button become a play button. | Pass |
| 16 | Listen to a story | User taps on the next button of the player screen. | The metadata of the next audio in the playlist is displayed on the screen. | Pass |
| 17 | Listen to a story | User taps on the previous button of the player screen. | The metadata of the previous audio in the playlist is displayed on the screen. | Pass |
| 18 | Listen to a story | User is in the first audio of the playlist and taps the previous button of the player screen. | Nothing happens. The button is gray and not active. | Pass |
| 19 | Listen to a story | User is in the last audio of the playlist and taps the next button of the player screen. | Nothing happens. The button is gray and not active. | Pass |
| | Listen to a story | User taps on the "1x" in the botton left corner of the player screen. | A menu with different player speeds is displayed. | Pass |
| 20 | Listen to a story | Audio is playing and the user selects a higher speed on the player screen. | Audio is played faster. | Pass |
| 21 | Listen to a story | Audio is playing and the user selects a lower speed on the player screen. | Audio is played slower. | Pass |
| 22 | Listen to a story | User taps on the playlist button on the player screen. | A list with all the audios in the playlist is displayed. | Pass |
| 23 | Listen to a story | User taps on the name of a different audio within the playlist. | The player skips to the selected audio. | Pass |
| 24 | Listen to a story | User taps the shuffle button on the player screen. | The order of the audios seing via playlist button randomly changes. The audios are played following the new sequence. | Pass |
| 25 | Listen to a story | User slides the seek bar to a different position on the player screen. | The audio position skips to the minute and second shown. | Pass |

| | | | | |
|---|---|---|---|---|
| 26 | Listen to a story | User taps once the repeat button on the player screen. | The current audio plays until the end and then plays again. The audio repeats until stopped on the the repeat mode is deactivated. | Pass |
| 27 | Listen to a story | User taps twice the repeat button on the player screen. | The playlist plays until the end and then repeat starting by the first song. | Pass |
| 28 | Listen to a story | User is listenning to an audio and locks the device screen. | The audio keeps playing. The user can use the device's OS to pause, play, skip to next or previous audios in the playlist. | Pass |
| 29 | Listen to a story | User is listenning to an audio and changes to another app. | The audio keeps playing. The user can use the device's OS to pause, play, skip to next or previous audios in the playlist. | Pass |
| 30 | Record a story | User tap the "Record a story" on the initial decision screen. | The recorder screen is displayed. | Pass |
| 31 | Record a story | The user taps the record button on the recorder screen. | The recorder starts recording. The timer shows the current recording duration. The record button becomes a stop button. | Pass |
| 32 | Record a story | The user taps the stop button on the recorder screen after being recording for a period of time. | The recorder stops recording. The timer shows the final recording duration. The record button becomes a play button. | Pass |
| 33 | Record a story | The user taps the play button on the recorder screen after having recording an audio. | The audio recorded is played. The play button becomes a stop button. | Pass |
| 34 | Record a story | The user taps the stop button on the recorder screen after playing the recorded audio. | The recording playback stops. The stop button becomes a play button. | Pass |
| 35 | Record a story | The user tap the refresh button on the left of the recorder screen after having recorded an audio. | The recording is erased. The timer is set to 00:00. The play button becomes a record button. | Pass |

| | | | | |
|---|---|---|---|---|
| 36 | Record a story | The user tap the done button on the right of the recorder screen after having recorded an audio. | The screen to add the title, author and select a cover is displayed. | Pass |
| 37 | Record a story | The user inputs an author, title and selects a cover for the audio recorded and taps done. | The screen to select the audio recipients is displayed. | Pass |
| 38 | Record a story | The user does not input one the author or the title for the audio recorded and taps done. | Nothing happens. | Pass |
| 39 | Record a story | The user inputs an author, title but does not select a cover for the audio recorded and taps done. | Nothing happens. | Pass |
| 40 | Record a story | The user selects one or more contact as recipients for the audio recorded and taps send. | The library screen is displayed. The new audio can be seen in the yours tab. The other recipients receive the audio shared. | Pass |
| 41 | Set up profile | After signing up the user taps on the camera icon to select a profile picture. | The device's gallery is open and picture can be chosen. | Pass |
| 42 | Set up profile | After signing up the user does not select a profile picture from the galery nor adds a name and taps done. | Nothing happens. | Pass |
| 43 | Set up profile | After signing up the user does not select a profile picture from the galery but adds a name and taps done. | The initial decision screen is displayed. The standard profile picture is attributed to the user, which can be seen in the drawer menu. | Pass |
| 44 | Manage profile | User taps on the user profile option in the drawer menu. | The user profile screen is displayed with the user current profile picture and name. | Pass |
| 45 | Manage profile | User taps on the edit buton on the profile screen. | User can change name and select a new profile picture from the galery. | Pass |
| 46 | Chat | User taps on the chat option in the drawer menu. | The conversations screen is displayed with the list of all user's 1-2-1 and group chats. | Pass |

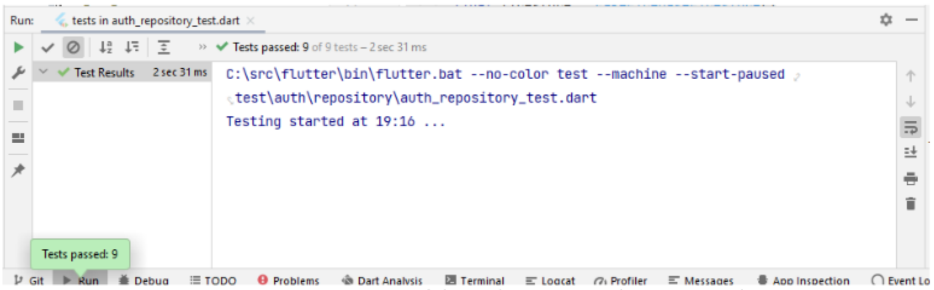
| | | | | |
|---|---|---|---|---|
| 47 | Chat | User taps on a conversation on the conversation screen. | The chat screen is displayed with all the messages of the user on the right side and from other users on the left side. | Pass |
| 48 | Chat | User taps on the new message button on the conversations screen. | A screen to select a contact is displayed. The contact selected are registered in the app and also in the user device's contacts with the same email. | Pass |
| 49 | Chat | User taps on a contact on the select contact screen. | The chat screen is displayed and the selected contact profile picture and name can be seen on the top. | Pass |
| 50 | Chat | User writes message in the botton text field on the chat screen as presses send. | The message is sent. The message show up on the right hand side of the screen. | Pass |
| 51 | Chat | User taps on the emoji button the botton text field on the chat screen. | A container for selecting emojis is shows up. | Pass |
| 52 | Chat | User taps on the GIF button the botton text field on the chat screen. | A container for selecting GIFs shows up. The user can search for GIF using the search field. | Pass |
| 53 | Chat | User taps on a GIF on the container displayed. | The GIF is send. It is displayed on the right hand side of the chat screen. | Pass |
| 54 | Create group chat | User taps on the "create group" option of the drawer menu. | The screen to add a group profile picture, a name and select contacts is displayed. | Pass |
| 55 | Create group chat | User selects a picture from the galery for the group, adds a name, selects users and clicks on done. | The group is created. It can be seen in the conversations screen. | Pass |
| 56 | Authentication persistence | The logged in user closes the app and reopens it. | The initial decision screen is displayed. The user is still logged in. | Pass |
| 57 | Log out | The logged in user tap on the "log out" option of the drawer menu. | The user is logged out. The welcome screen is displayed with the animation "send stories", "send love". | Pass |

## 9.5 Evidence of automated tests' results


Run: tests in auth_controller_test.dart
Tests passed: 9 of 9 tests – 1 sec 747 ms
Test Results 1 sec 747 ms
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

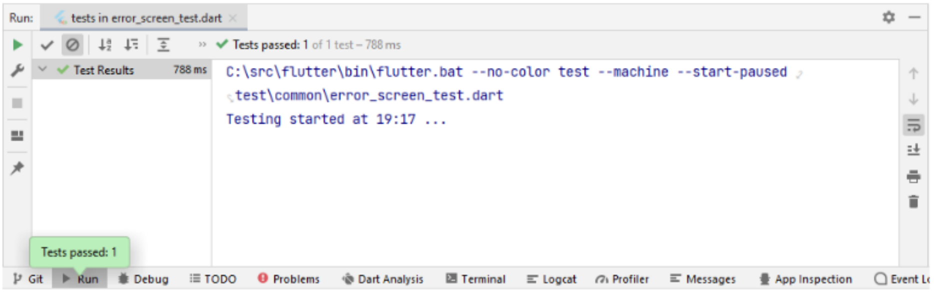
```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
test\auth\controller\auth_controller_test.dart
Testing started at 19:13 ...
```

*Figure 56 – Snippet of the AuthController class tests results*


Run: tests in auth_repository_test.dart
Tests passed: 9 of 9 tests – 2 sec 31 ms
Test Results 2 sec 31 ms
Tests passed: 9
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

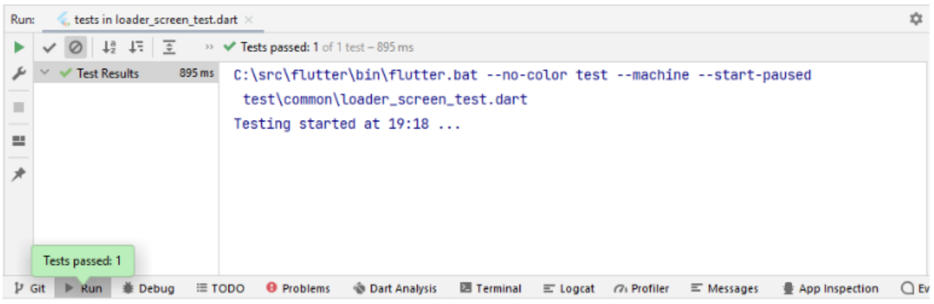
```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
test\auth\repository\auth_repository_test.dart
Testing started at 19:16 ...
```

*Figure 57 – Snippet of the AuthRepository class tests results*


Run: tests in error_screen_test.dart
Tests passed: 1 of 1 test – 788 ms
Test Results 788 ms
Tests passed: 1
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

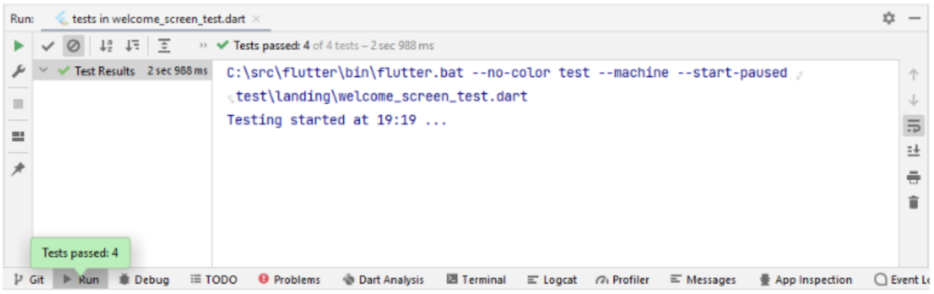
```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
test\common\error_screen_test.dart
Testing started at 19:17 ...
```

*Figure 58 – Snippet of the ErrorScreen class tests results*



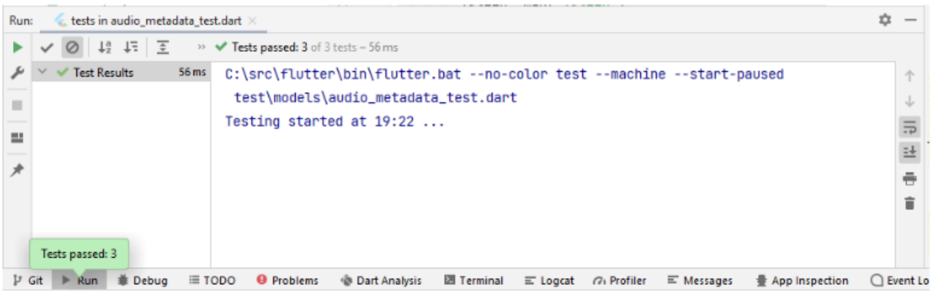
```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\common\loader_screen_test.dart
Testing started at 19:18 ...
```

Figure 59 – Snippet of the LoaderScreen class tests results



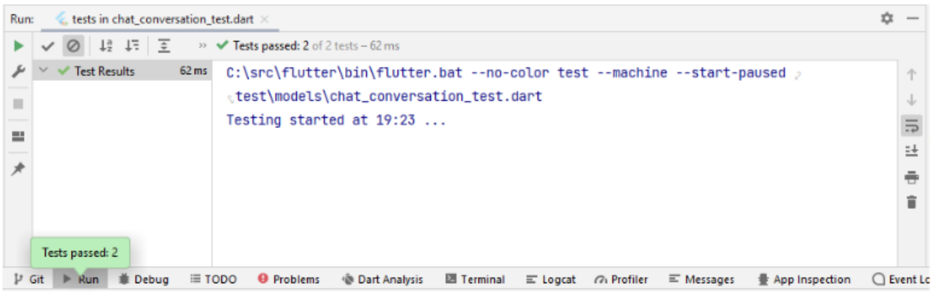
```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\landing\welcome_screen_test.dart
Testing started at 19:19 ...
```

Figure 60 – Snippet of the WelcomeScreen class tests results



```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\models\audio_metadata_test.dart
Testing started at 19:22 ...
```

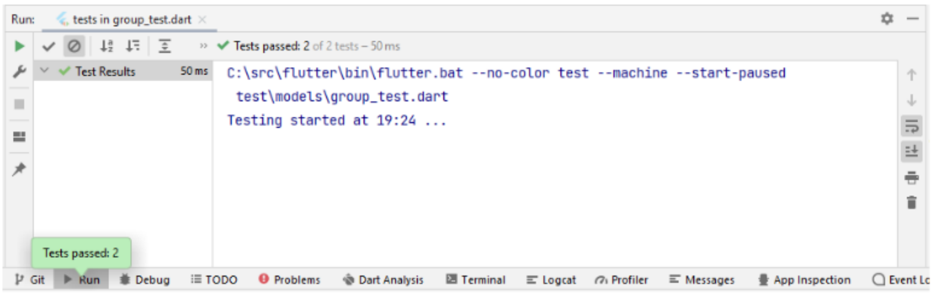
Figure 61 – Snippet of the AudioMetadata class tests results



Run: tests in chat_conversation_test.dart
Tests passed: 2 of 2 tests – 62 ms
Test Results 62 ms
Tests passed: 2
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\models\chat_conversation_test.dart
Testing started at 19:23 ...
```

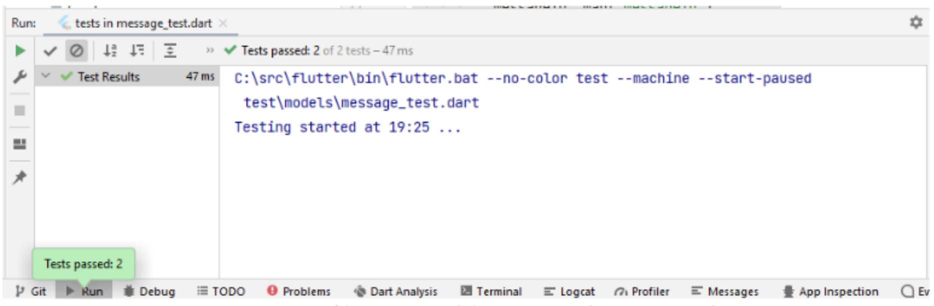
Figure 62 – Snippet of the ChatConversation class tests results



Run: tests in group_test.dart
Tests passed: 2 of 2 tests – 50 ms
Test Results 50 ms
Tests passed: 2
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\models\group_test.dart
Testing started at 19:24 ...
```

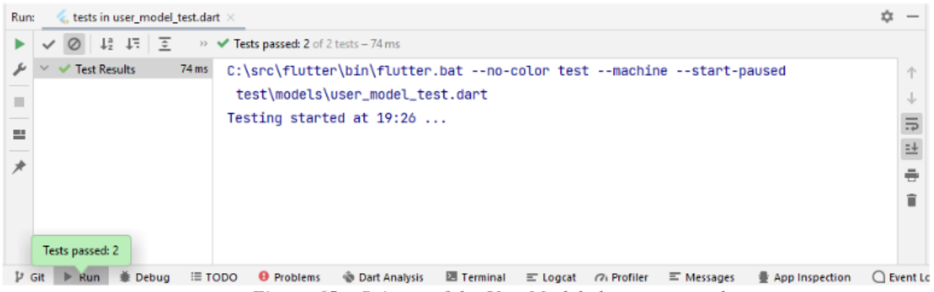
Figure 63 – Snippet of the Group class tests results



Run: tests in message_test.dart
Tests passed: 2 of 2 tests – 47 ms
Test Results 47 ms
Tests passed: 2
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Ev

```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\models\message_test.dart
Testing started at 19:25 ...
```

Figure 64 – Snippet of the Message class tests results



Run: tests in user_model_test.dart
Tests passed: 2 of 2 tests – 74 ms
Test Results 74 ms
Tests passed: 2
Git Run Debug TODO Problems Dart Analysis Terminal Logcat Profiler Messages App Inspection Event Lo

```
C:\src\flutter\bin\flutter.bat --no-color test --machine --start-paused
 test\models\user_model_test.dart
Testing started at 19:26 ...
```

Figure 65 – Snippet of the UserModel class tests results

# PROJECT_REPORT_ANA_LUIZA_CASTRO

GRADEMARK REPORT

FINAL GRADE

72/100

GENERAL COMMENTS

**Instructor**

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64

PAGE 65

PAGE 66

PAGE 67

PAGE 68

PAGE 69

PAGE 70

PAGE 71

PAGE 72

PAGE 73

PAGE 74

PAGE 75

PAGE 76

PAGE 77

PAGE 78

PAGE 79

PAGE 80

PAGE 81

PAGE 82

PAGE 83

PAGE 84

PAGE 85

PAGE 86

PAGE 87

PAGE 88

PAGE 89

PAGE 90

PAGE 91

PAGE 92

PAGE 93

PAGE 94

PAGE 95

PAGE 96

PAGE 97

PAGE 98

PAGE 99

PAGE 100

PAGE 101

PAGE 102

PAGE 103

PAGE 104

PAGE 105

PAGE 106

PAGE 107

GRADING FORM: MSC PROJECT (21/22) ZW

| | |
|---|---|
| ANA LUIZA SILVA DE CASTRO | **72** |

**SPECIFICATION (20)**

*Distinction: The specification and design of the system/software clearly demonstrates how to meet the requirements, and all components fit together in a coherent way. Merit: A good attempt has been made in the specification and design of the system/software. Pass: Before starting the implementation, a specification and design of the system/software is laid out. Borderline Fail: Some attempt has been made in the specification and design of the system/software but not in sufficient detail or too much irrelevant information has been included. Fail: Very* *little or no attempt has been made in the specification and design of the system/software.*

The specification and design are comprehensive and fit together in a coherent way. I appreciate the thought that went into the design especially the accessibility considerations described on Page 30.

**16**

**IMPLEMENTATION (30)**

*Distinction: The key stages of the implementation or research are clearly explained. The implementation or research for a challenging problem is carried out to a high standard. Merit: The implementation or research for a challening problem has been partially successful or has been carried out to a high standard for a less ambitious project. Merit: The implementation or research for a challenging problem has been partially successful or has been carriedout to a high standard for a less ambitious project. Pass: The key stages of the implementation or research are explained. The implementation or research is sound. Borderline fail: Some attempt to explain the key stages of the implementation or research. The implementation or research has been only partially successful. Fail: Little or no attempt to explain the key stages of the implementation or research. The implementation or research only addressed simple aspects of the problem.*

This is a relatively straightforward design that has been implemented well.

**18**

**TESTING (30)**

*Distinction: The solution demonstrates deep insight into the problem/research question. Reflections on the contribution and its limitations are fully justified. Key results are accurately and critically analysed. Relevant conclusions are drawn. Merit: The report has a clear* *and logical argument to support the contribution and its limitations. It discusses how the software meets the requirements and is well reinforced by evidence. Pass: The report attempts to provide a clear and justified reflection upon the contribution and its limitations. It discusses how the software meets the specified requirements. Borderline fail: Some attempt at reflection upon the contribution and its limitations but the argument may be difficult to identify or follow. Some attempt to discuss how the software meets the specified requirements. Fail: Little or no attempt at reflection upon the contribution and its limitations. Little or no attempt to discuss how the software meets the specified requirements.*

The report describes a thorough testing procedure and comments on its limitations.

**25**

**PRESENTATION (20)**

*Distinction: The report is self-contained, providing background and context. It is well-organised and demonstrates the concepts presented. Complex issues are explained clearly. The solution is well-justified. The report cites relevant resources using consistent style and is of professional quality. Merit: Relevant background research and context are presented. Material is generally well-organised and clearly presented. The solution is justified. The report cites relevant resources using an appropriate consistent referencing style. Pass: The report is coherent in style and structure and includes citations. It communicates the main project outcomes. Borderline fail: Some aspects of the material may be poorly presented and badly-organised. Outcomes are not clearly communicated. Referencing is limited or incomplete. Fail: Material is poorly presented and badly-organised. The student's contribution is unclear. Referencing is limited or incomplete.*

This is an unusual presentation for a report, particularly organising material around sprints,

**13**

which I found to be rather engaging. The references are not entirely consistent. I am not clear what "closed" is meant to represent in Figure 1. The description of the implementation is far too long at 35 pages.