

Design and Implementation of Mobile Applications



POLITECNICO
MILANO 1863

FlashQuiz

Design document

Authors:

Erik Galler

Vemund Thallaug Lund

2022 - 2023

Contents

Introduction	3
Purpose	3
The design document	3
Functionalities	3
Architectural design	5
Overview	5
Database structure	5
Controllers	6
Authentication controller	6
Firestore controller	6
External services	6
Firebase Firestore	6
Firebase Authentication	6
Open Trivia Database	7
User interface design	7
User interface	9
Welcome page	9
Sign up and sign in page	10
Home screen / create quiz	11
Delete quiz interface	12
Quiz information page	13
Add question to regular quiz	14
Add questions to multiple choice quiz	15
Store screen	17
Upload screen within a published quiz and the possibility of downloading	18
Upload a quiz the user has created	19
Profile screen with the possibility of changing the name	20
Development plan, implementation and testing	21
Overview	21
User testing	21
Unit testing	22
References	23

Introduction

Purpose

As fourth year students we know how efficient you have to be during exam periods and throughout the year to master the curriculum in time. Many of our colleagues struggle with this. Therefore, our solution to this problem is the app “FlashQuiz”. This will serve as an app where you can create your own quizzes on different subjects, test yourself, and even download quizzes your peers have created and uploaded to the app.

The design document

The purpose of this design document is to present the details of the app “FlashQuiz”. It will include the overall architecture of the application and the user interface. Furthermore the document was important during development as it ensured that both of us were on the same page concerning the architecture, designs and testing.

Functionalities

The main objective of this app is to help other students, and also normal people, aiding them to learn in school or just do a quiz for fun. Therefore there is a number of functionalities each user must have available. Each user is able to perform the following actions:

- Welcome screen
 - Screen that welcomes the user
 - Gives the user the option to either ‘Sign up’ or ‘Sign in’
- Login screen
 - Log in with his/hers existing account
- Register screen
 - Register a new account with a valid email and password
- Home screen
 - Take either your own or downloaded quizzes
 - Choose how many questions of the given quiz you want to take
 - Create two different types of quizzes, with three possible question types. Before creating the questions, the user has to choose the name, description, theme and difficulty of the quiz.
 - Multiple choice quiz
 - An arbitrary number of incorrect answers
 - Normal quiz
 - Regular question with an answer
 - Sentence with a keyword which is hidden for quiz taker
 - Delete a quiz either created by user or downloaded
 - Search for quizzes created by own user
- Store screen
 - The store screen is where you can download or upload quizzes.
 - You can also rate the quizzes you have downloaded.
 - Upload quizzes created by the user
 - Download quizzes from the internet (fetched from sites via API)
 - Rate other quizzes
 - View ratings of other published quizzes
 - Search for uploaded quizzes
- Profile screen
 - Change users display name
 - Logout

Architectural design

Overview

We wanted to make an architecture that wasn't too complex for our project, but rather an architecture that met the needs of the application. We also wanted the application to be compatible on both Android and iOS.

Technology stack

- Frontend
 - React Native
 - TypeScript
 - ESLint + Prettier
 - Redux
 - React Native Paper
 - Expo
- Backend
 - Firebase Firestore
 - Firebase Authentication
- Test framework
 - Jest

Database structure

- User
 - ID → Primary key
 - Name → Display name in app
 - Email → Email for the account
 - Password →
- Quiz
 - CreatorID → ID of the account creating the quiz
 - CreatorName → Name of the creator
 - Date → Date of when it was created
 - Title → The title of the quiz

- Description → Description of the quiz
- Difficulty → How difficult the quiz is
- Theme → Describes the theme of the quiz
- Downloads → Number of downloads the quiz has
- Questions → A list of questions in the quiz, each with its own answer
- Ratings → The different kind of ratings the quiz has
- Type → What kind of type the quiz is

Controllers

Authentication controller

Authentication controller is responsible for handling all communication with Firebase Authentication, with regards to the users. We used the *signInWithEmailAndPassword* which takes care of the login request with an email and password.

Firestore controller

Firestore controller is responsible for handling all communication with Firebase Firestore, for handling the link to the database. We used the collection and doc controller. These handle the access and management of documents as well as updating and deleting data.

External services

Here are the external services used in the application.

Firebase Firestore

Since we wanted an architecture, as well as a database, that met the needs of the application, *Firebase* was a good choice. Here we have implemented the Firebase Firestore. It's a cloud-based NoSQL database that can store and retrieve data easily. Moreover, it's known for its ease of use, scalability and flexibility, which worked perfect for us.

Firestore Authentication

One of the main benefits of using Firestore Authentication is that it's cross-platform compatible. We were therefore easily able to add users, for both iOS and Android users. It also supports multiple authentication and is cloud-based. This means that it supports providers such as email/password, phone number and third-party apps like Google, Facebook and Apple. We went for the email/password combination. And since it's cloud-based we could scale the authentication to meet the needs of the application.

Open Trivia Database

Since the number of quizzes in the store is dependent on users uploading quizzes, the initial store would be approximately empty. Therefore we decided to populate the store with quizzes from the internet via an API. We chose the *Open Trivia Database* with over 4000 verified questions, varying difficulties and multiple categories.

User interface design

A good user interface is crucial as it influences the usability, accessibility and overall experience of the app. We therefore wanted to make a UI which was easy to understand, use and learn from. For the design sketches we used *Figma*, which is a cloud-based service that allows us to work in real-time collaboration. It also has a wide range of features and tools in order to create just what we wanted. A screenshot of our Figma-file can be seen below with the rest of the designs.

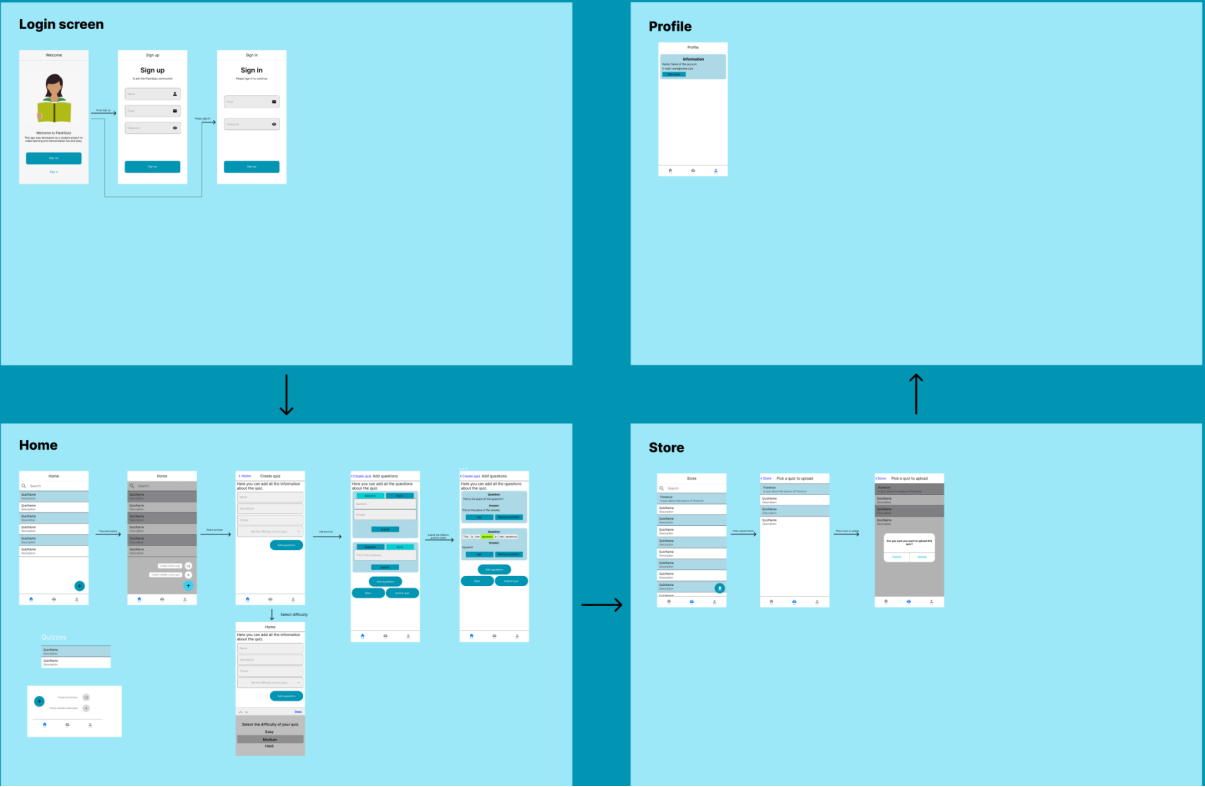


Figure 1: Figma file

User interface

Welcome page

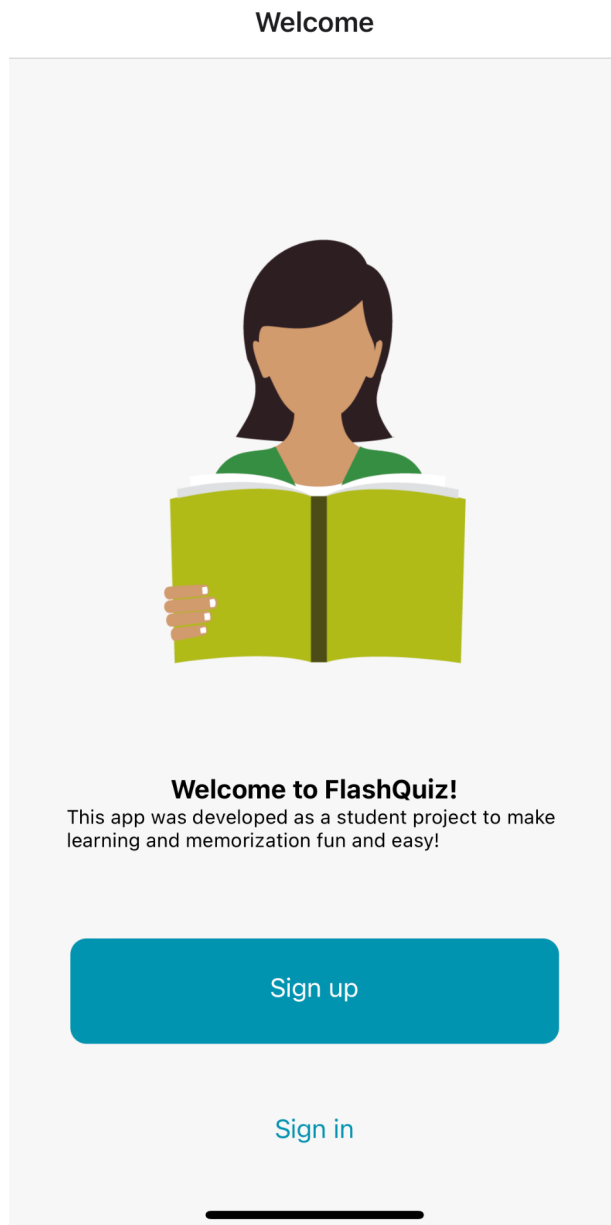


Figure 2: Welcome page

Sign up and sign in page

[< Welcome](#)

Sign in

Sign in

Please sign in to continue

Email

Password

Sign In

[< Welcome](#)

Sign up

Sign up

To join the FlashQuiz community!

Name

Email

Password

Sign up

Figure 3: Sign up and sign in page

Home screen / create quiz

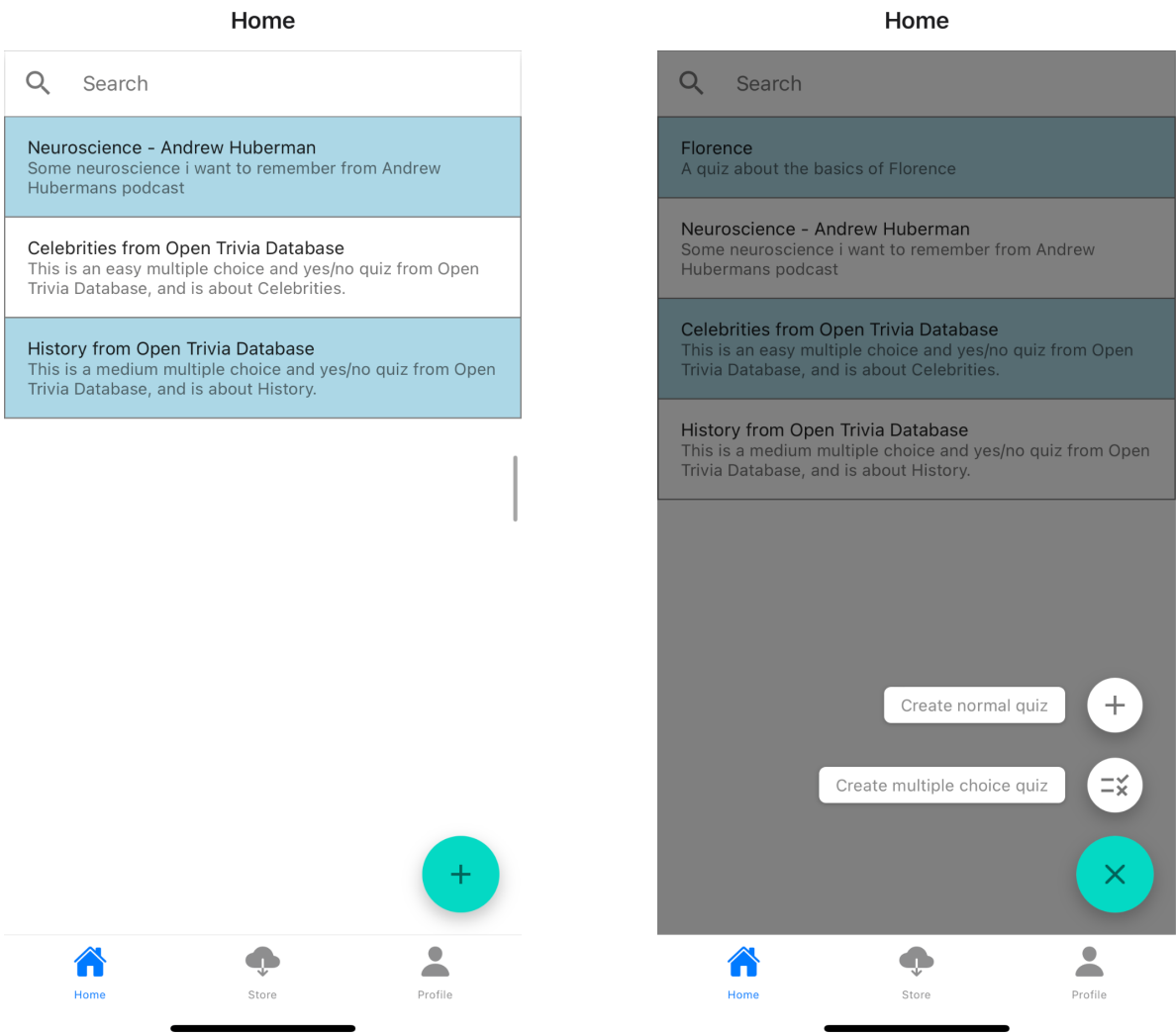


Figure 4: Home screen

Delete quiz interface

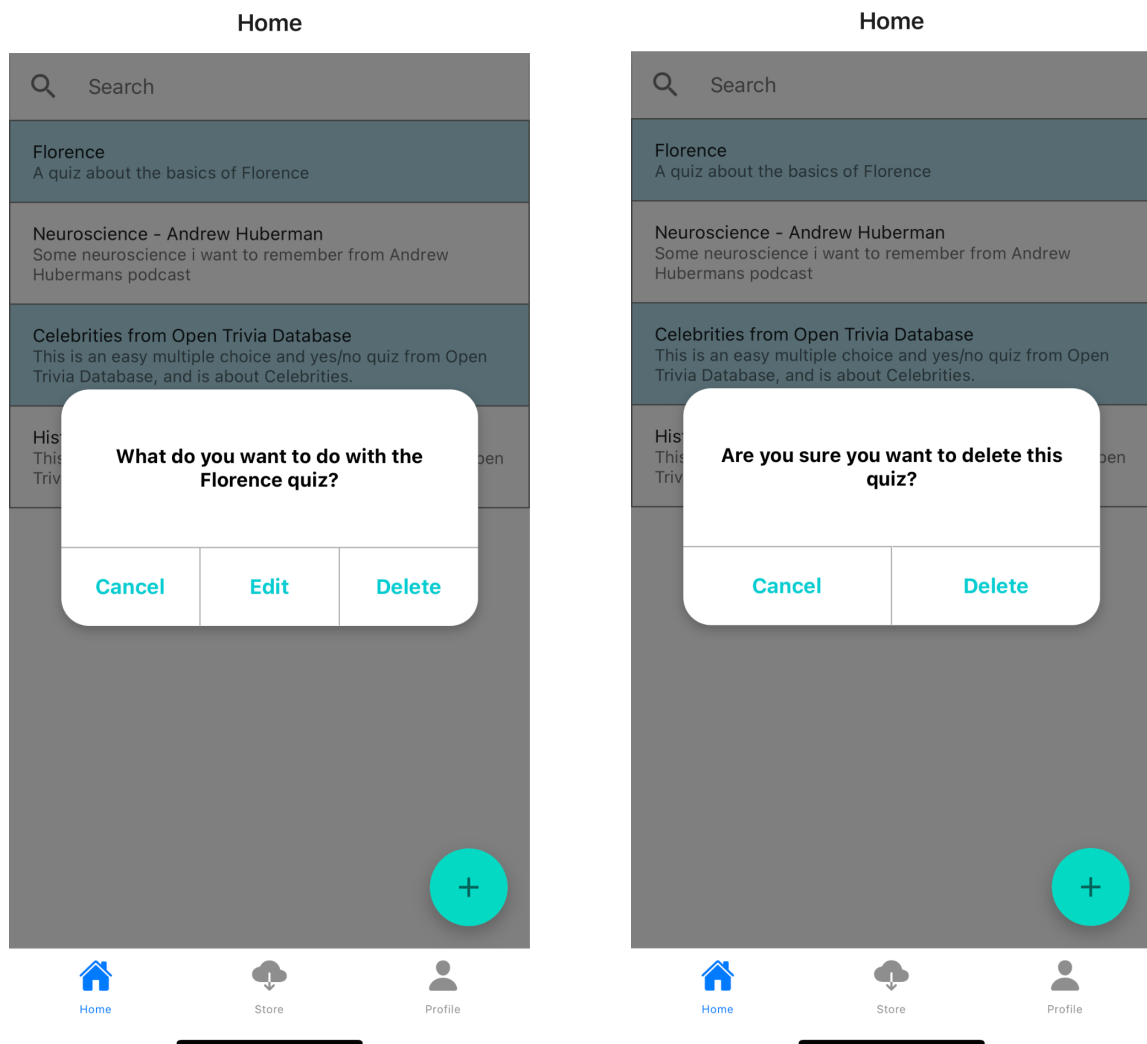



Figure 5: Delete quiz on home screen


Quiz information page


[< Home](#) [Create Quiz](#)

Here you can add all the information about the quiz.



[Add Questions](#)


Home


Store



Profile

Figure 6: Create quiz screen

Add question to regular quiz

[Create Quiz](#) Add Questions

Here you can add all the questions for your quiz.

Question

Form

Question

Answer

Submit

Add question

Back

Submit Quiz

[Create Quiz](#) Add Questions

Here you can add all the questions for your quiz.

Question:

This is a question

Answer:

This is the answer

Edit

Remove question

Question:

This is a sentence and this is the keyword

Answer:

keyword

Edit

Remove question

Add question

[Home](#)

[Store](#)

[Profile](#)

[Home](#)

[Store](#)

[Profile](#)

Figure 7: Add questions to quiz you are creating

Add questions to multiple choice quiz

The figure displays two screenshots of a mobile application interface for adding multiple choice questions to a quiz. Both screenshots show the same layout, but the right one is filled with example data.

Left Screenshot (Initial State):

- Header:** A blue back arrow and the text "Add Multiple Choice Questions".
- Text:** "Here you can add all the questions for your quiz."
- Form:** A light blue rounded rectangle containing:
 - A text input field labeled "Question".
 - A text input field labeled "Possible answer".
 - Two buttons: "Add answer alternative" and "Submit".
- Buttons:** Below the form is a large blue button labeled "Add question". At the bottom are two more blue buttons: "Back" and "Submit Quiz".
- Footer:** Three icons: a house (Home), a cloud with a download arrow (Store), and a person (Profile).

Right Screenshot (Filled State):

- Header:** A blue back arrow and the text "Add Multiple Choice Questions".
- Text:** "Here you can add all the questions for your quiz."
- Form:** A light blue rounded rectangle containing:
 - A text input field labeled "Question" with the text "This is the question?".
 - A text input field labeled "Possible answer" with the text "This is wrong".
 - A text input field labeled "Possible answer" with the text "This is also wrong".
 - A text input field labeled "Possible answer" with the text "This is correct".
 - Two buttons: "Add answer alternative" and "Submit".
- Buttons:** Below the form is a large blue button labeled "Add question". At the bottom are two more blue buttons: "Back" and "Submit Quiz".
- Footer:** Three icons: a house (Home), a cloud with a download arrow (Store), and a person (Profile).

Figure 8: Add questions for a multiple choice quiz

< Add Multiple Choice Questions

Here you can add all the questions for your quiz.

Question:

This is the question?

Answer:

A: This is wrong

B: This is also wrong

C: This is correct

Edit

Add question

Back

Submit Quiz



Home



Store



Profile

Figure 9: Highlighting the correct answer on a multiple choice question

Store screen

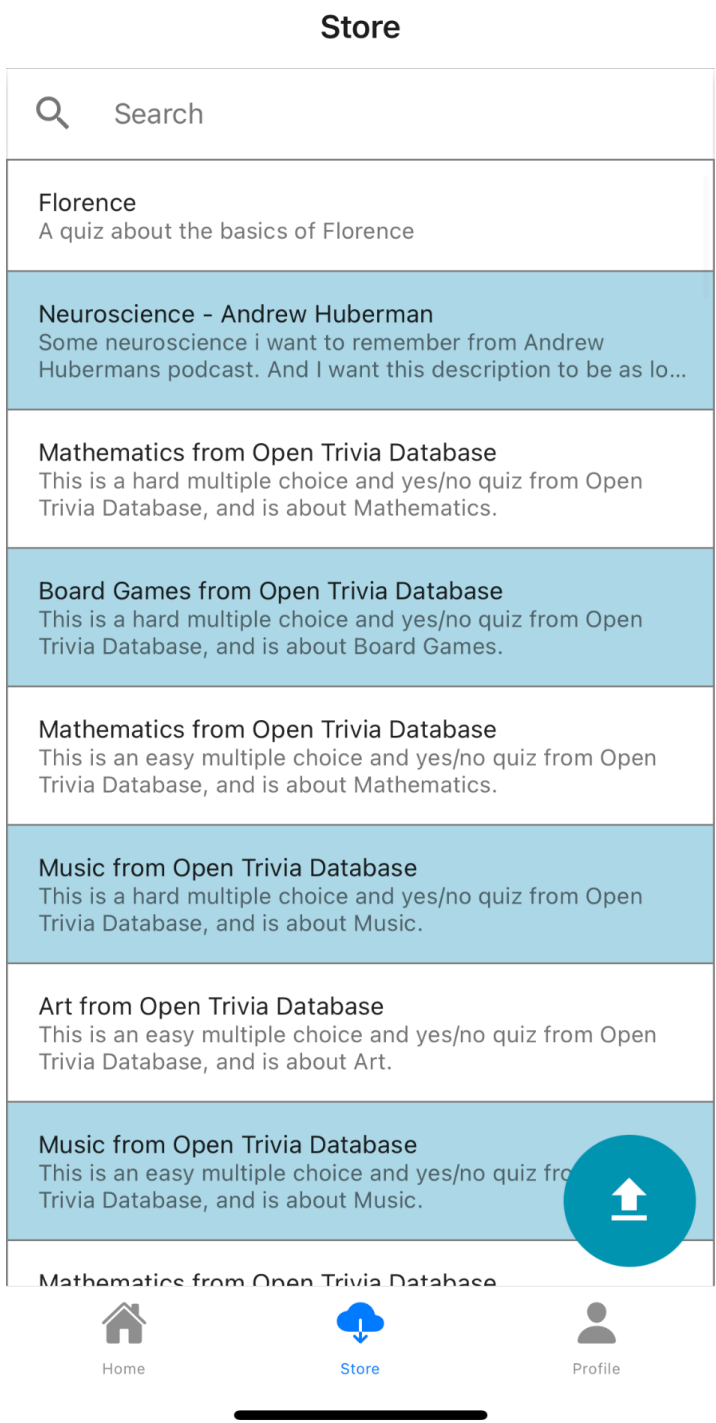


Figure 10: Store screen

Upload screen within a published quiz and the possibility of downloading

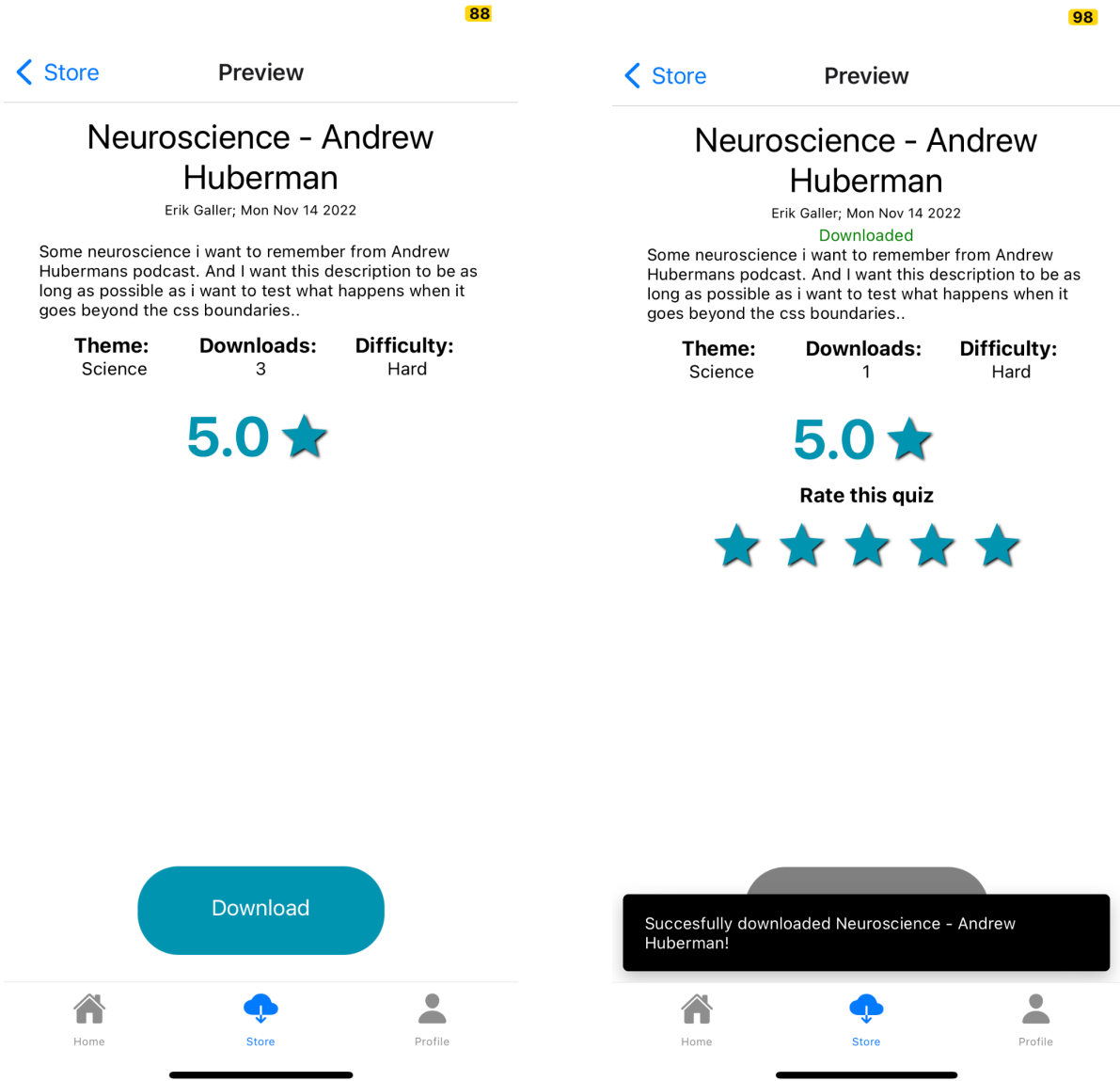


Figure 11: Viewing of a uploaded quiz

Upload a quiz the user has created

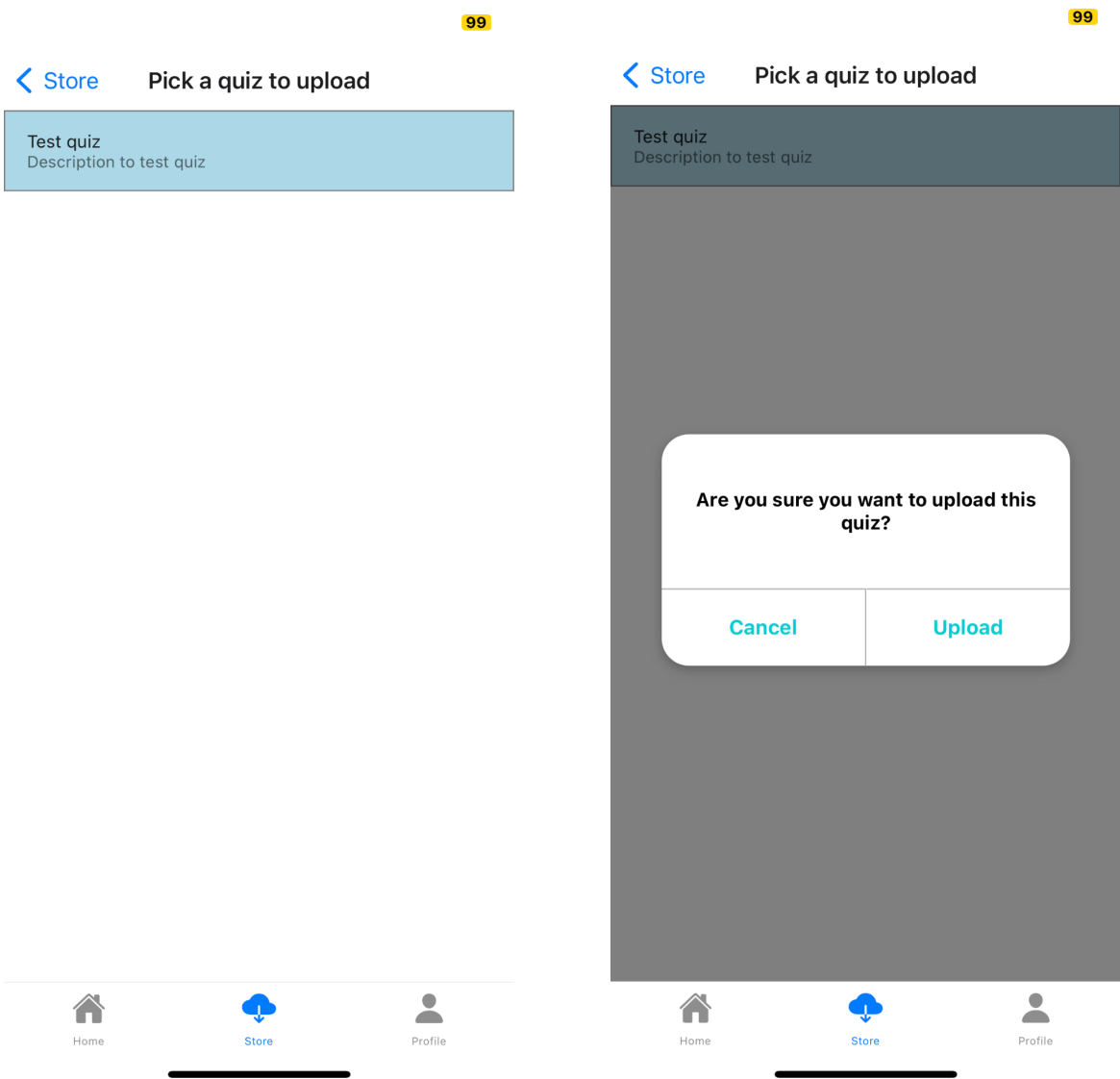


Figure 12: Upload a quiz you created

Profile screen with the possibility of changing the name

87

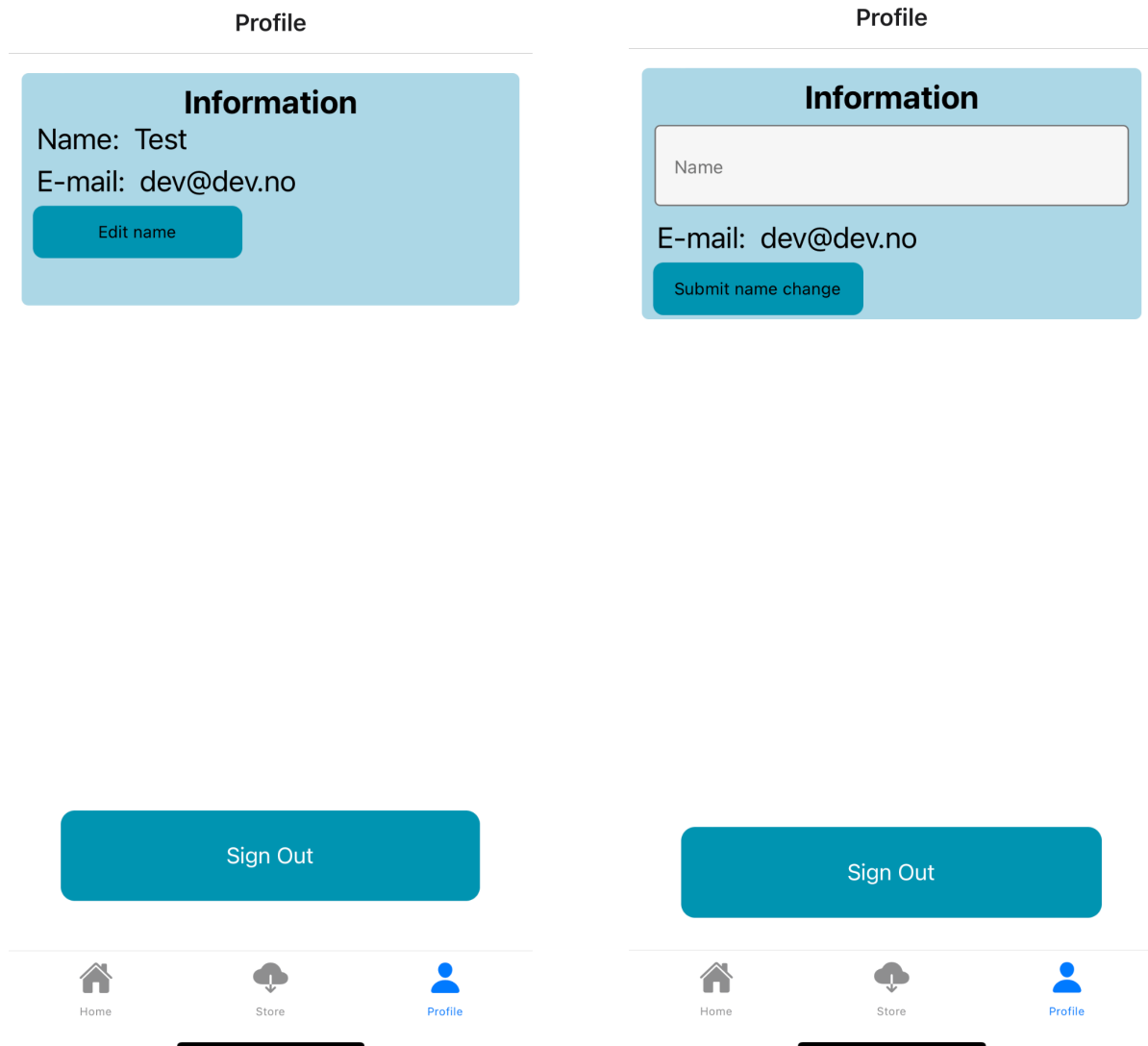


Figure 13: Profile screen

Development plan, implementation and testing

Overview

The development of the app was divided into several phases:

- A sketch of the App and it's basic functionalities
- Frontend setup
- Development of the three main screens and navbar
- Development of database and classes
- Development of the main components to cover the main features of the app
- Development of user authentication
- Final development of minor functionalities and general improvements
- Testing

User testing

After the base functionality was developed we performed a series of user testing. We handed the phone to a roomie, a family member or a friend, and asked them to perform a task on the app, like create a quiz or download one. At first most of these tests were focused on the design and not the functionalities itself, but this changed later in the process. In this way we achieved very good feedback from someone outside of the development crew, which was vital in the process of obtaining a user friendly app.

One of the things we changed from our earlier versions, was how a user continued to the next question in a quiz after answering. Initially we tried to implement this with some slider feature, but most users got confused and tried

to simply tap on the screen instead. Therefore we implemented an invisible touchable opacity that registers any tap on the screen, and directs the user to the next question.

We also asked some users how we could improve the user learning experience, and one of our test users suggested a summary screen at the end of every quiz, to remind you of your mistakes. We thought this was a great idea, and implemented it right away.

Unit testing

For this we used *Jest*, as we have worked with this framework before.

Furthermore, we also performed unit testing on different parts of code. This ensured that the component worked the way we wanted, isolated from the rest of the application. It also improves code quality and is crucial if code is refactored or changed.

```
> flash-quiz-app@1.0.0 test
> jest

PASS tests/generateDropdownData.test.ts
PASS tests/dursteinfeldShuffle.test.ts
PASS tests/emailReg.test.ts
PASS tests/getNumberOfChar.test.ts

Test Suites: 4 passed, 4 total
Tests:       31 passed, 31 total
Snapshots:   0 total
Time:        2.892 s
Ran all test suites.
○ (base) → flash-quiz-app git:(main) x █
```

Figure 14: Screenshot of the tests

References

- [React Native](#)
- [Figma](#)
- [Firebase documentation](#)
- [JEST documentation](#)
- [Open Trivia Database](#)
- [Axios](#)
- [Expo](#)
- [React Native Paper](#)
- [Redux](#)