WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

BACHELOR THESIS

SUPERVISOR:
Lect. Dr. Liviu-Octavian Mafteiu-Scai

GRADUATE:
Adrian-Ioan Tuns

TIMIŞOARA
2021

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: Computer Science in English

Public Speaking Improvement Assistant

SUPERVISOR:
Lect. Dr. Liviu-Octavian Mafteiu-Scai

GRADUATE:
Adrian-Ioan Tuns

TIMIŞOARA
2021

Abstract

As public speaking is an essential part of many people's lives nowadays, several methods were developed and tried through the years to improve one's ability to speak in public. Being in the Information Age, the methods for improving this specific ability have slowly moved towards the side of the digital world.

The act of public speaking can be explained as giving a speech face to face to a live audience, a speech which has to be formulated and expressed in the best possible way, in order to entirely transmit the idea of the oration. Therefore, in order to solve this real-world problem a mobile application was developed, application that aids the orator in preparing to hold a speech for an audience, by rehearsing and correcting the mistakes that may appear in the expression of the ideas or in the usage of the words, while also providing a responsive, intuitive and user-friendly interface.

Contents

Chapter 1

Introduction

1.1 Problem Description

Public speaking refers to giving a speech in any form of speaking, be it formally or informally, between an audience and the speaker. "The study of public speaking began about 2500 years ago in ancient Athens. Men were required to give speeches as part of their civic duties." [15]. It is worth mentioning that with the act of public speaking, a form of public speaking anxiety also appeared, which is a common form of social anxiety, one third of the general population reporting excessive anxiety when speaking to a larger audience, and a third of this group reporting some clinically significant distress or impairment due to their anxiety [3].

It also has to be noted that several benefits exists in studying public speaking, as presented in the article "The Benefits and Necessity of Public Speaking Education" [1]. Among many others, a strengthening of critical thinking skills, a reduction in speaking anxiety and an increase in self-confidence, a development in delivering a message as effectively as possible, an improving in learning how to take an audience oriented perspective, in learning to excite and engage people, a development in listening more intently and effectively, which in turn helps the speaker in providing more useful feedback to others, and lastly and more crucially, in helping empower oneself to be a leader, represent the most notable benefits of studying and engaging in this act.

As it can be noted in the Related Work section, several applications that focus on different problems in the field of this topic exist, with an observable preponderance for the iOS platform and with a lack of free services.

The motivation of this bachelor thesis is to present the way a software application that aim to improve one's capacity to hold a public speech is build. This application is meant to be a free, open-source mobile Android application that can be used in several languages and dialects, leaving room for extending to iOS, macOS, Windows, Linux and web platforms.

1.2 Thesis Structure

The bachelor thesis is divided in six chapters. An informal description of the problem and of the solution is presented in the first part of chapter 1, the last part explaining the main use case. The chapter 2 presents the existing applications

related to the thesis subject. In chapter 3, the functional and non-functional requirements are presented, alongside the chosen architecture. Chapter 4 details the solution, presenting a detailed explanation of the architecture, the technologies used and previously considered or tried, the external and internal perspective of the application and the actual implementation of several features. Chapter 5 presents the user's manual, regarding how the application should be used, with a subsection dedicated to explaining how the project should be run locally, if it is desired. The chapter 6 presents the conclusions of the thesis and the future work.

1.3 Running Example

The main scenario of how the application should be used is as follows: before the user starts rehearsing his already prepared speech, the application should be started and it will listen the speech, converting the spoken words into readable text that will be showed on the screen, highlighting the filler words and making a statistic afterwards, regarding what should be improved, while also providing suggestions on how to achieve that. An option to save the speech result can also be used from the statistics screen.

To ensure a quality analysis, filler words in the desired language should be set before attempting a speech rehearsal.

Chapter 2

Related Work

2.1 Related Work and Applications

Several applications were created in order to aid people in improving their abilities to hold a speech, to speak in public or to win over the public speaking anxiety. These applications can be organized in several categories, as follows: applications that serve as personalised speech coaches, which track the speaker usage of filler words, the talking pace and the pronunciation, such as Ummo, LikeSo or Orai, applications that analyze the user's pitch and the volume of his voice, such as Voice Analyst and applications that fully simulate a situation where the user has to speak in front of a virtual audience of different kinds, making use of Virtual Reality technology, while also providing an exposure as a treatment measure for the anxiety of speaking in public, applications such as Virtual Speech.

2.1.1 Personalized Speech Coach Applications

The first category includes applications that track the usage of filler words and the pronunciation. While this type of applications can't help with the anxiety of public speaking, they can help on improving the actual content of the speech.

Ummo [7] is a paid application developed for iOS that wants to provide an increased self-awareness, therefore changing the way its users communicate, while also becoming everybody's personal speech coach, any time and anywhere. This application was created by several students from Harvard and MIT that wanted to become better communicators and also wanted to make speech coaching accessible to everyone.

In this application, the user registers what filler words and phrases should be tracked, while also having the option to set features like curse-word detection on or off at any point. Before starting a speech, the record button has to be pressed to start and again at the end to stop recording. As the user talks, Ummo analyses the speech, generating a feedback at the end of the speech, measuring several metrics such as the word count, pace, volume of the voice, filler words usage and the clarity of voice.

LikeSo [5] is the category-defining and highly popular application of the Say It Media Inc., company created to improve communications skills by bringing voice and AI-powered tools to higher education and enterprise learning and development. LikeSo is another paid speech coach application made for iOS, powered by a voice

recognition technology, two practice modes and a proprietary scoring system. This application analyses verbal skills in real-time, calculating an articulate score grade, based on a formula that looks at the percentage of non-filler words over total words spoken, and the speaking pace.

It offers two modes of usage, TalkAbout and FreeStyle. In FreeStyle, the user can talk about any subject for up to 30 minutes, having the option to pause or stop at any time. Before starting the mode, words will be selected in the application to train against. An analysis of the talk session and the results will be shown at the end of the talk session. On the other hand, TalkAbout is a conversation game to practice talking on the fly. After choosing a topic from the list of 12 topics, the talk time and the filler words will have to be chosen too. Afterwards a session based on five prompts will begin, the application listening to the answers for each of the prompts and providing a report at the end of the talk session, similar to the first presented mode.

Orai [11] is a free to download application with paid subscription, with a free initial assessment, designed for non-native English speakers and for speakers who struggle with a speech impairments. Available on both Android and iOS, it was created by two international engineering students at Drexel University. Orai records the user talk in a similar manner as the previous applications and gives instant feedback on any speech registered, measuring factors like the talking pace, the filler words usage, the user's energy level, his conciseness, the pauses and the overall confidence of the speaker. Apart from the speech statistics, this application also provides interactive lessons created in collaboration with speech coaches, lessons based on each user's communication goals and needs. The progress can also be tracked over time while using this application and tips on how to improve the communication skills even further will be generated based on the progress so far.

2.1.2 Voice Analysis Applications

Another important category of applications that can aid in improving this essential skill are the applications that can analyse the pitch and the volume of a user's voice. Both of these qualities are important when delivering a speech, as the message can be completely transmitted only if it is presented in the best possible way, with the best voice usage the speaker can achieve. One such application is Voice Analyst [14], a paid application made for both Android and IOS, that uses accurate vocal pitch detection algorithms that can analyse a pitch from 60Hz to 2kHz. Voice Analyst was created by Speech tools Ltd., which worked only on applications regarding speech therapy.

Based on their tests, the accuracy of the pitch detection is within 5 cents (5% of a semitone) across the full spectrum of speech, while the best a human can detect is 8 cents. This application can display the pitch, volume or both as the user speaks. It analyzes the voice to show average, minimum and maximum pitch and volume. The speaking sessions can be saved as recordings which can be sent in an email to be analysed remotely. The application also has integrated cloud services, through third party applications. Minimum and maximum targets for pitch and volume can also be set. This application is most used by speech and language therapists, by people with neurological conditions or brain disease, by singers, clinicient looking to provide remote voice Telehealth, transgenders wanting to modify theirs voices

and by people with voice difficulties such as vocal fold palsy or muscle tension dysphonia.

2.1.3 Public Speaking VR Simulation Applications

The third category is represented by applications based on Virtual Reality technology that simulate a real presentation. One such application is Virtual Speech [15], a free application with digital purchases. Virtual Speech offers "Award-winning off-the-shelf training courses designed to improve skills in the most efficient way", that combines several methods of analysing a speech. In this application the user will enter in simulated environments where he will have to hold a speech, environments such as simulated job interviews with one or several interviewers, meeting rooms with other people, school classes, where the user will talk in front of students, press conferences, television shows where the user is in the role of a guest or even stages, presenting in front of a big audience, while also having TEDx inspired rooms.

After pressing the start analysis option, the presentation will start, during which the user has to talk as much as he considers fit, a report being created afterwards, report that gets more accurate as the time of actual talking is bigger, the report containing information regarding the talking pace, the eye contact with the audience, the filler words usage and a lisenability score, regarding how easy or difficult was to listen to the speech. Audio and visual distractions such a audience coughing, microphone feedback and other effects can be enabled, for a more realistic experience.

The speech can also be recorded and sent to the course administrator, as this application can be used to model entire courses, with teachers and students. During the actual speech rehearsal, a live feedback option can also be turned on, making suggestions about eye contact variety, such as on what side of the audience to focus more, suggestions about the pace of the speech or the volume of the voice. Also, another useful option is to simulate a slides based presentation, using custom presentation slides, that can be uploaded into the account linked to the application and used further. Speech notes, custom questions and branched questions can also be uploaded in the account for being used in the speech simulations. The notes will simply be a text panel that will scroll by itself at a set speed after starting the speech, while the questions can be recorded by the speaker and assigned to avatars from the simulated audience, and the branched questions lets the avatar change his questions based on the user's response.

For the TEDx styled rooms there is a feature that generates specific speech insights. The speaker has to talk at least 10 minute for accurate results and the report that is given afterwards contain informations regarding the measurement of cooperation, modesty, trust, open-mindedness and even how driven, excited, self-assured, or challenged was the speaker.

2.2 Conclusions

As a conclusion, there are several applications that focus on different problems in the field of public speaking, the majority of the applications being developed on the iOS platform, with little free options.

The purpose of the Public Speaking Improvement Assistant is to create an Android application with the possibility to easily extend to other platforms, completely free and open-source, that works in as many languages and dialects as possible, that uses similar statistics measurement as the other speech coach applications, providing the option to save the speeches for further usage and sync them to a cloud account, while also assuring an optimal responsiveness, as response time is of great importance when trying to analyse words in real-time. All of that delivered in a beautiful looking, highly customizable application, that conforms to the Material Design guidelines.

Chapter 3

Application Modelling

3.1 Software Requirements

The requirements know so far for the Public Speaking Improvement Assistant can be organized in both functional and non-functional requirements.

3.1.1 Functional Requirements

The functional requirements are represented of the following: text-to-speech conversion in several languages, real-time analysis of words, generation of statistics based on recognized words, cloud integration, authentication system, option to save speech results locally or to cloud and color customization of the application's components.

3.1.2 Non-functional Requirements

The non-functional requirements: portability through application functionality on Android, with the possibility to easily extend to other platforms too, security of the user data that may exist in the database, integrity of the data, privacy and reliability.

3.2 Redux Architecture

Regarding the architecture to be used for the application, an unidirectional data flow architecture was used, especially so as Flutter is the system development kit used. This unidirectional flow can be achieved through the state management offered by Redux [26] architecture, which makes it easy to develop, maintain and test applications.

Redux is both a pattern and a library used for managing and updating the application state, using events called "actions". It serves as a centralized store for state that needs to be used across the entire application, with rules ensuring that the state can only be updated in a predictable way. Being easy to implement as a 3-tiered architecture for the application needs, as represented in figure 3.2, it offers flexibility, segregating the application into Data, Business and Presentation Layers. For Redux pattern a "Middleware" is needed and "Epics" from Redux-Observable [6] is considered to be used, "Epics" being a side-effects manager for

Redux, allowing the creation of asynchronous actions in reaction to other actions using observables. Figure 3.1 depicts the data flow using Redux architecture.

A notable advantage of this architecture, beside the state management and data flow, is the separation from view logic and view. The view logic can be developed and tested without having visual elements created. The point of contact is represented by the Redux store that uses data models which are common throughout the application. This separation allows for better modularity in development and also offers the possibility to apply the same view logic to different User Interfaces [4].

3.2.1 Redux Data Flow Diagram

Figure depicting the flow of data in a Redux architectural pattern, with Redux-Observable "Epics".

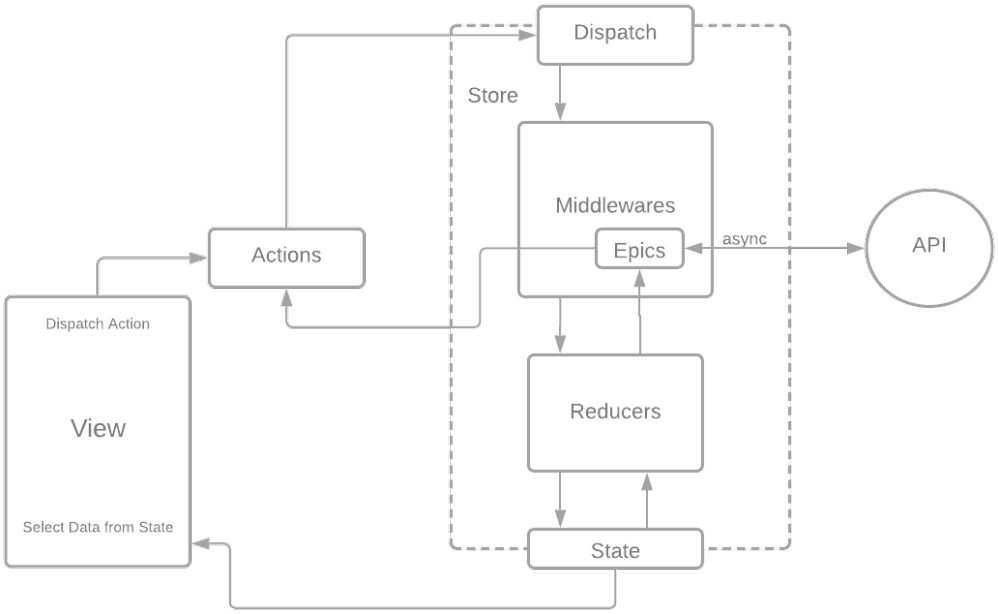

Figure 3.1: Data Flow in Redux Architecture

3.2.2 Redux Three-Tier Architecture Diagram

Application structured in a three-tiered architecture using Redux.

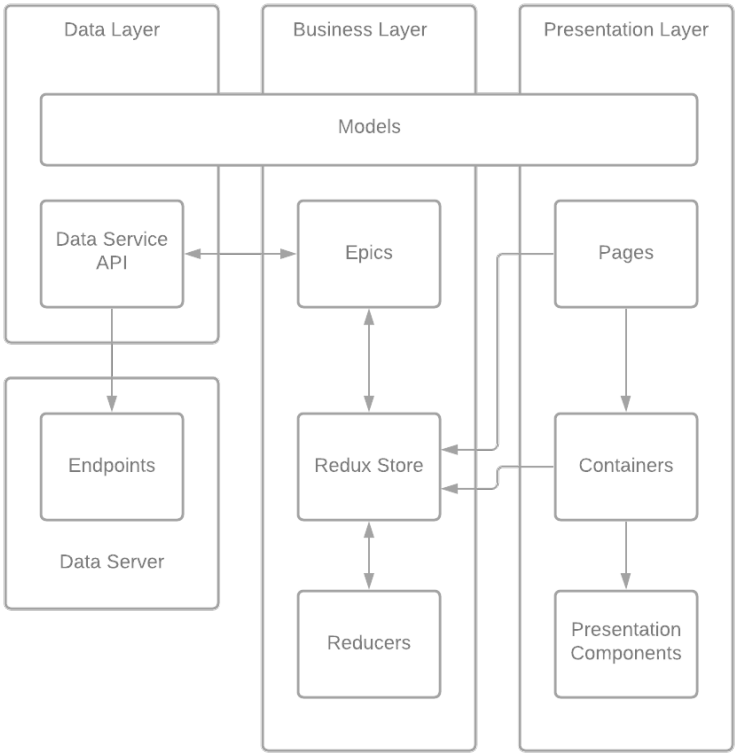

Figure 3.2: Three-Tiered Architecture

Chapter 4

Solution

4.1 Architecture

As mentioned in the Application Modelling section, the Redux [26] architecture was used. Redux ensures that all of the application's state lives inside one container. If the state needs to be changed, a new state needs to be created based on the current state and a requested change.

4.1.1 State

The application state refers to the state that needs to be shared across several parts of the application and that would need to be kept between user sessions, as opposed to the ephemeral state, that is contained in a single UI component. For ensuring that this state cannot be changed directly from the User Interface, it needs to be made immutable.

4.1.2 Action

Actions are the Redux way of modifying the state. They are application events that get dispatched in the Store, to the Middleware component. They also must be immutable and they usually describe a specific change.

4.1.3 Store

The Store is the component that holds a State object which represents the state of the whole application, usually containing smaller states that exist in the application. Each time the state changes, the user interface updates to reflect the new state.

4.1.4 Middleware

Before the reducer is announced that a change must be made, all actions enter the Middleware part. Here some additional code is run, performing a side-effect on the received state, such as communicating with an API or data source. After the processing, the Middleware may dispatch the action further, dispatch a different action or do nothing.

4.1.4.1 Epics

Epics represents a specific type of Middleware, that must always dispatch an action after processing it. It is a very good fit for working with complex asynchronous operations, by receiving a stream of actions and returning a stream of actions It can therefore handle continuous tasks, such as listening for a person speech.

After the data was passed from the Epics to the reducer, the Reducer can notify the Epics of changes, which in turn make the Epics asynchronously communicate with the API, giving back to the Reducer streams of data.

4.1.5 API

API in this application context refers to any User Interface call to a function, be it asynchronous or not, that the application makes.

4.1.6 Reducer

The Reducer is the component where the actual state is modified, updating the Store with a new State.

It can be summarized as a simple equation: state + actions = state.

4.1.7 Container

In the context of this application, a Container is used when reconstructing the UI component, for fetching modified informations.

4.1.8 Advantages and Disadvantages of Redux

The advantages of Redux architecture, from which some were mentioned in previous sections, are the following:

- having a central store, any component can access any state if it requires it
- debugging and testing is easier, as the UI and the data management are separated
- the outcome is predictable, because each actions has a single, clear request

The disadvantages of the Redux architecture are the following:

- boilerplate code must be written, having a restricted design
- no data encapsulation exists, as any component can access data, which could potentially cause security issues
- because state is immutable, the reducer updates the state by returning a new state every time, fact that can cause excessive usage of memory

4.1.9 External Perspective

4.1.9.1 Use Case Diagram and Use Case Details

In figure 4.1, the functional requirements of the application are represented, through the usage of an use case diagram:

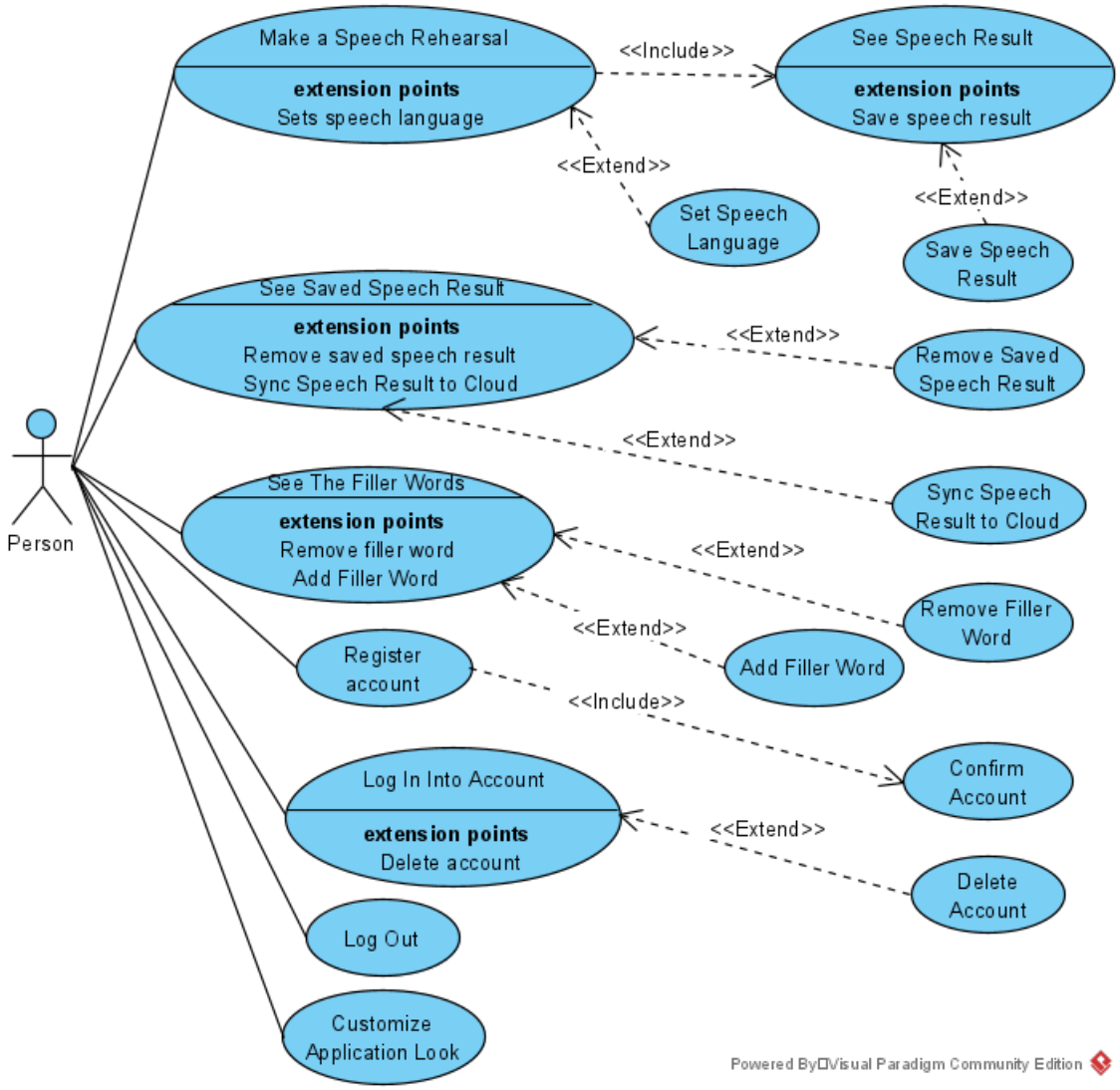

Figure 4.1: Use Case Diagram

In the following part, the "Make a Speech Rehearsal" and the "Customize Application Look" use cases will be explained in detail:

- **UC Name:** Make a Speech Rehearsal

  Goal: To perform a speech rehearsal

  Actors: Any person that uses the application

  Preconditions: The application must have a stable internet connection

  Triggers: The person presses the button that starts the rehearsal

Basic flow of events: The option for speech rehearsal is pressed from the home screen. The person may select the desired language for the speech, and then the button depicting a microphone should be pressed. The person talks and the words spoken are written on the screen, highlighting the words marked as filler words in another Use Case. When the person finishes the speech, the button with the microphone is pressed again and the application will show a screen with statistics regarding the speech, from where the person may save the speech.

Alternative flow of events:

1. If the internet connection is lost at any point, the application will notify the person and no more words will be recognized.
2. If the back button is pressed, the words recognized so far are deleted and the listening process is stopped.

Postconditions: After the speech rehearsal was stopped, the speech result screen is shown.

- **UC Name:** Customize Application Look

  Goal: To customize several application components colors or to switch the theme between light and dark themes

  Actors: Any person that uses the application

  Preconditions: None

  Triggers: The person uses the theme switch or presses the buttons

  Basic flow of events: The person enters the customization menu and either presses the theme switch or taps on a color corresponding to a component from the application. A dialog with color options appears, from where the customization choices can be made, their results being seen in real-time and afterwards, the "Save" button must be pressed.

  Alternate flow of events: If the person presses the "Cancel" button instead of "Save", the previous selected color will be used.

4.1.9.2 Sequence Diagram at System Level

In figure 4.2, the speech rehearsal process can be observed, through the usage of a sequence diagram:

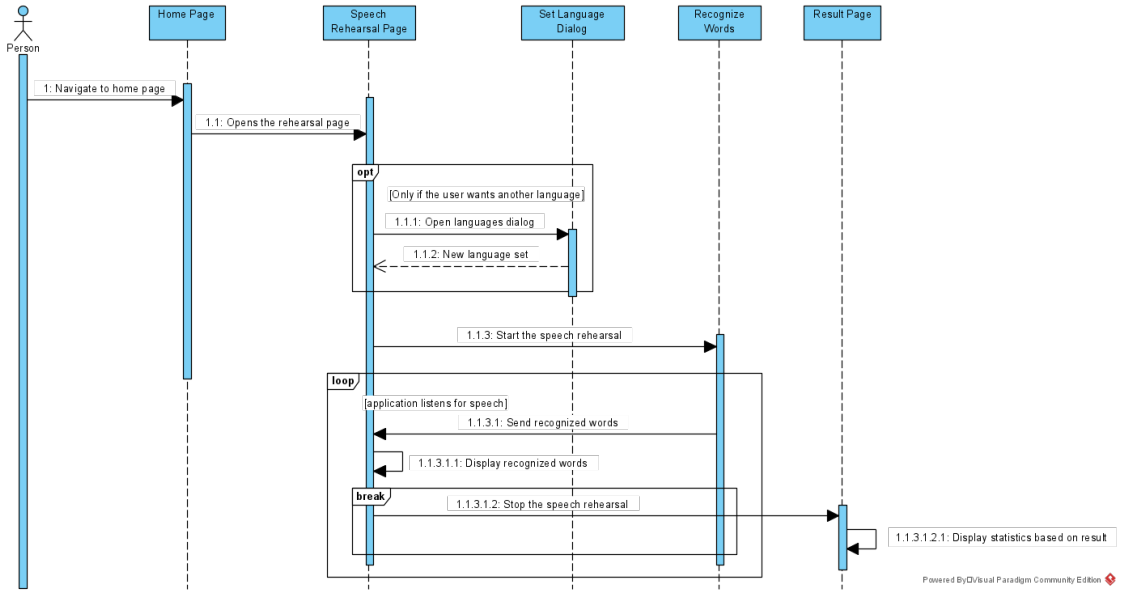

Figure 4.2: Sequence Diagram

4.1.9.3 Description of External Interfaces

The first page 4.3 that appears after the splash screen has two options: "Start Rehearsal" and "Set Filler Words".
The "Set Filler Words" option 4.4 lets the user register what words he wants to train against, in order to take them out of his speech presentation.

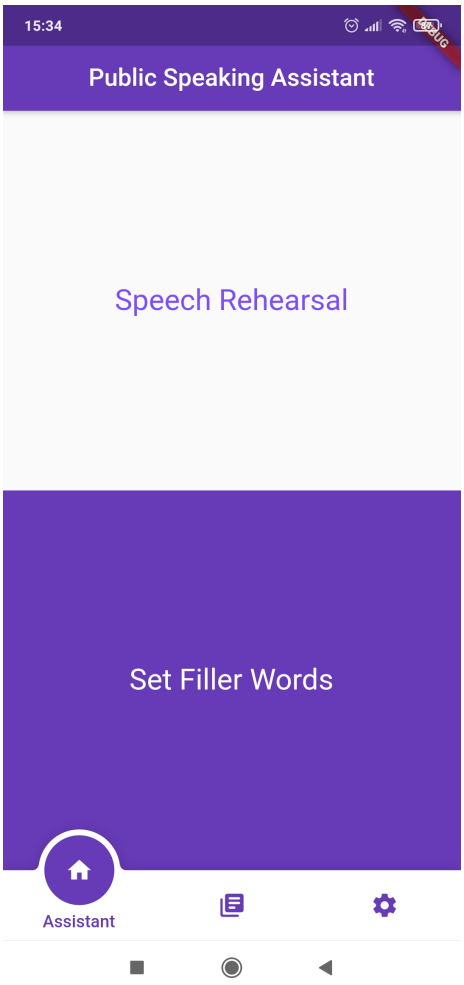

Figure 4.3: Home Page

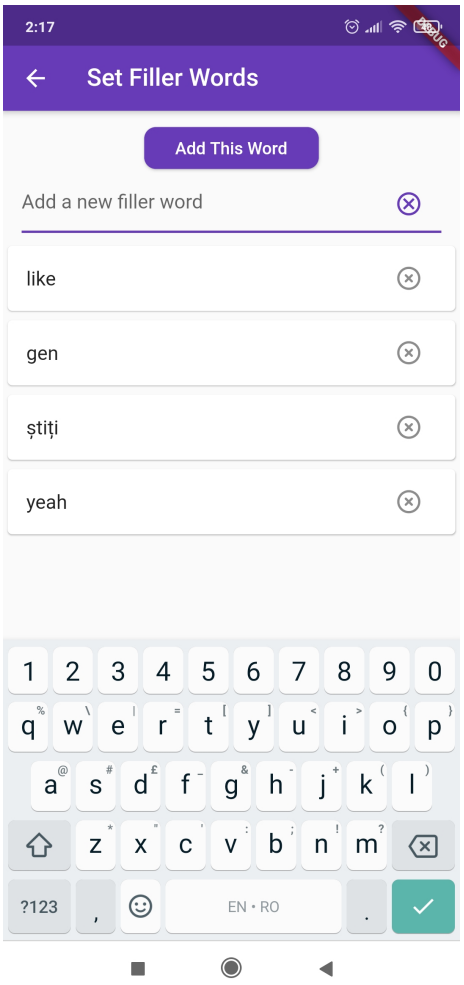

Figure 4.4: Filler Words Page

In order to use the application main feature, a stable internet connection is required. After ensuring that an internet connection exists, the user has to press the "Start Recording" button.
From the application top bar on the rehearsal screen 4.5, the desired language for the speech can be selected from a list of supported languages 4.7. The first row of the rehearsal screen 4.5 shows the currently set speaking language.
After the button was pressed, the recording mode starts and the application listens the user speech, showing the recognized words on the screen as the user speaks, while also highlighting detected filler words 4.6.

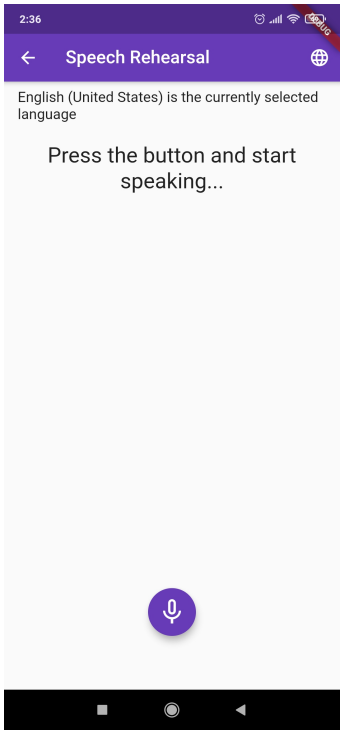

Figure 4.5: Speech Rehearsal Page

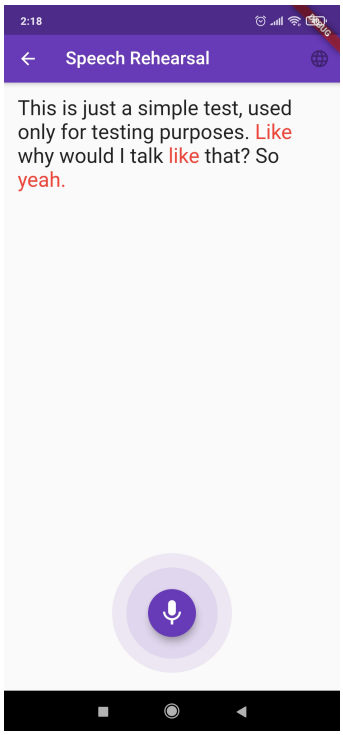

Figure 4.6: Speech Recognition In Action

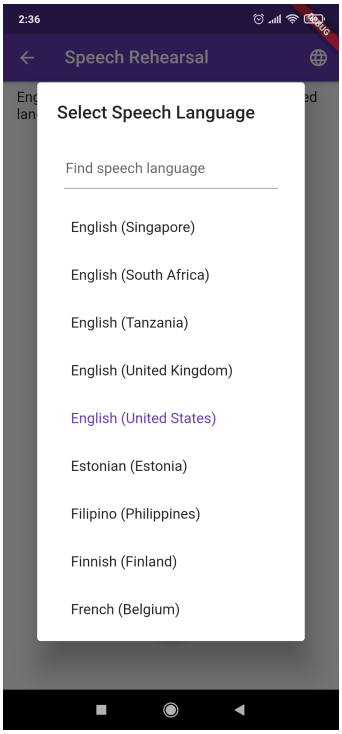

Figure 4.7: Language Selection Dialog

When the user wants to stop the speech rehearsal, the button depicting a microphone must be pressed again and a result screen 4.8 with statistics of the speech is shown. Tips 4.9 about improvement of some statistics can be seen after pressing the items that have an information icon.
The whole speech transcript can also be seen, by selecting the "Full Speech Transcript" option 4.10. The user can then save the result locally, after setting a name or simply discard it and exit back to the home page.

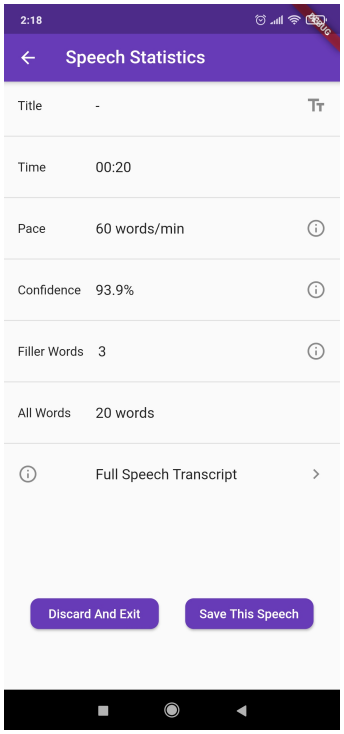

Figure 4.8: Speech Statistics Page

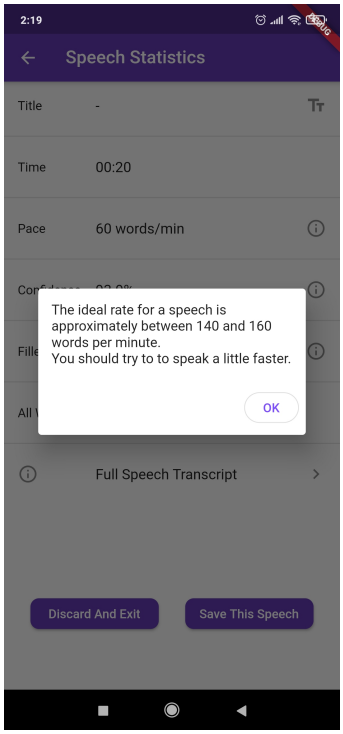

Figure 4.9: Pacing Tip Dialog

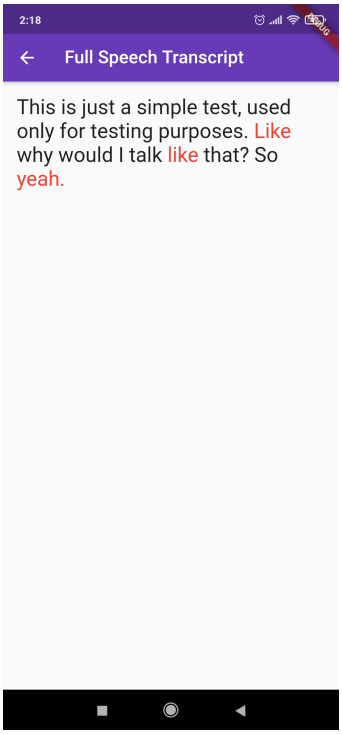

Figure 4.10: Speech Transcript

The saved speech results can be found in the library page 4.11, where the results can be accessed or synced to cloud, if an account was created.
The full transcript 4.12 can be seen after tapping the speech item, with an option to go back to the statistics page present in the application top bar 4.13.

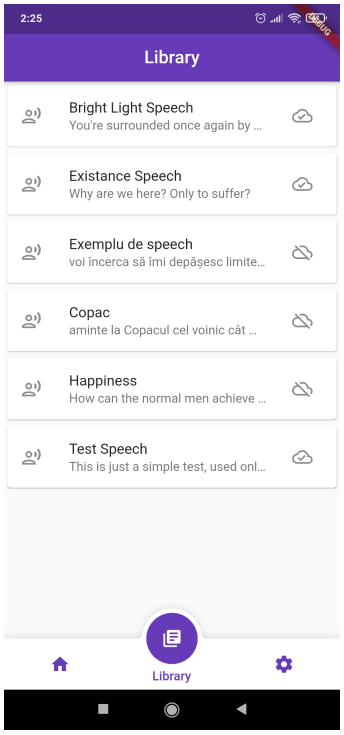

Figure 4.11: Library Main Page

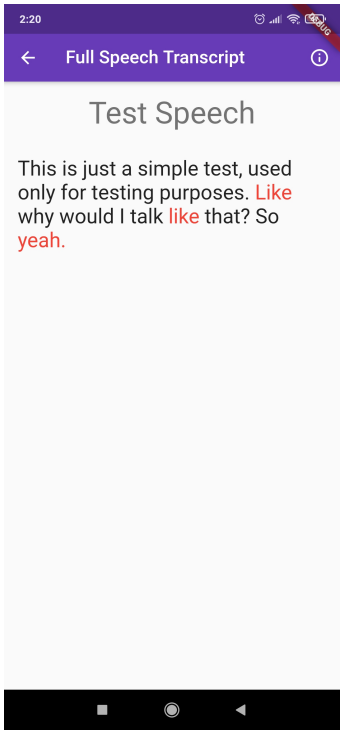

Figure 4.12: Speech Transcript From Library

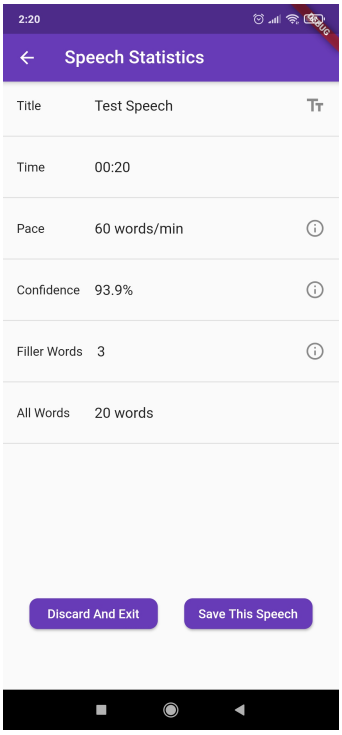

Figure 4.13: Speech Statistics From Library

The settings page 4.14 has several options that can be accessed, such as "Account", "Appearance", "Terms & Conditions", "Privacy Policy" and "Help".
It must also be mentioned that if the user wants to exit the application by pressing the back button, a double back button tap is needed 4.15.

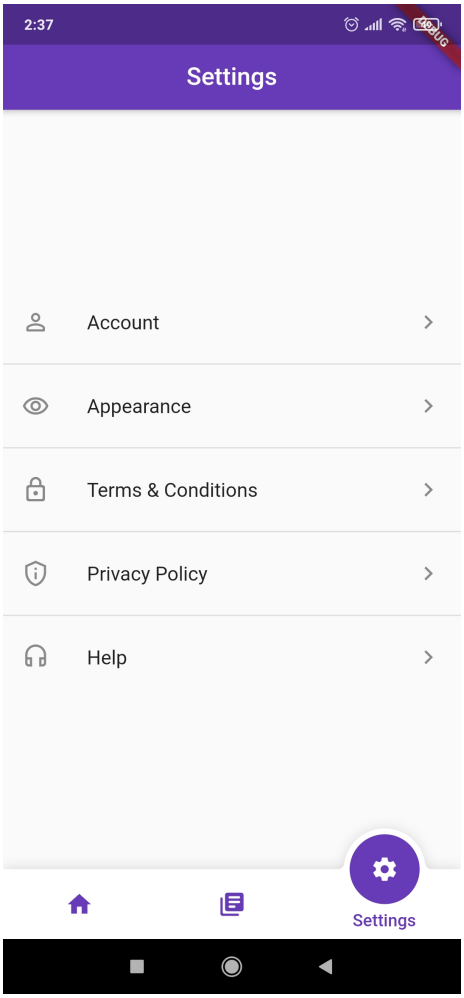

Figure 4.14: Settings Page

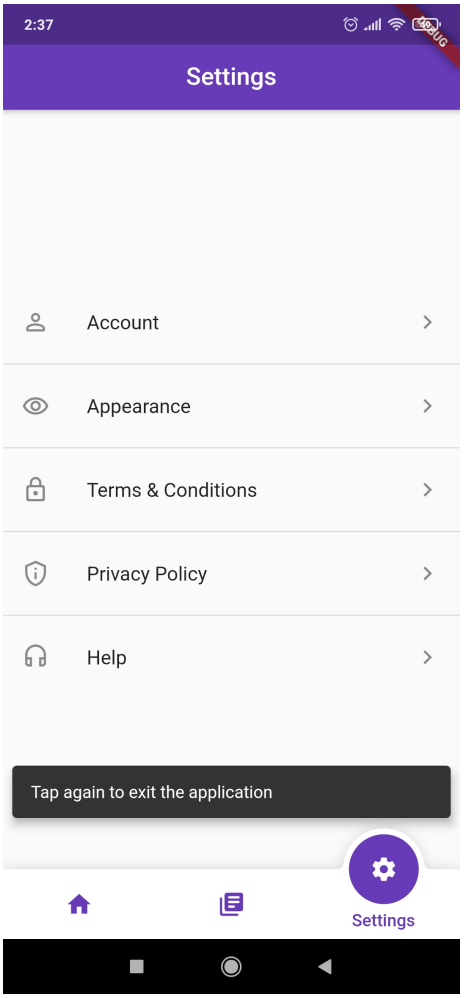

Figure 4.15: Exit on Double Tap

"Account" page 4.16 shows the currently connected account, or the login page 4.17, if no account is connected. A new account can be created either with an email and password, or with a Google account.
After a new account was created, a confirmation email is sent. After confirming the email, the user must authenticate again. Having an account gives the options to sync speech results and to download synced results to a new device.

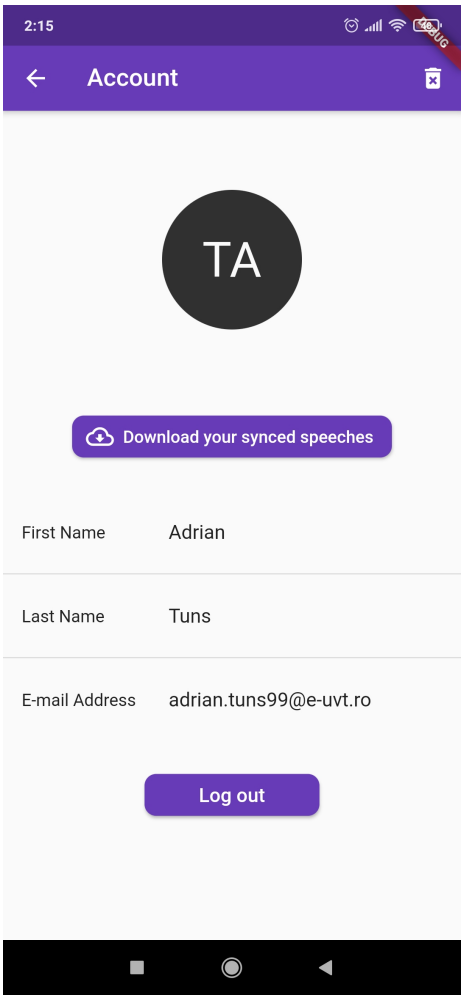

Figure 4.16: Account Page

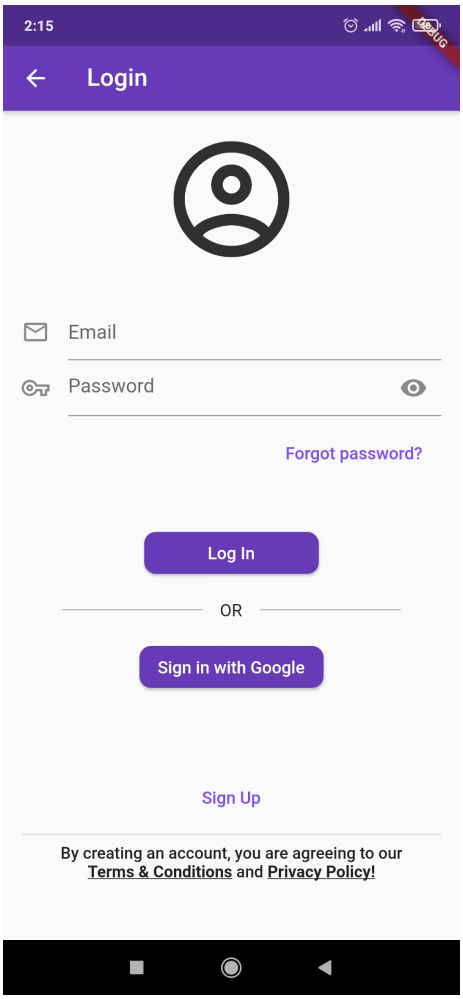

Figure 4.17: Login Page

"Appearance" page 4.18 has a dark mode switch and several customization options for various components from the application, giving the user the possibility to set desired colors for each of them 4.19.

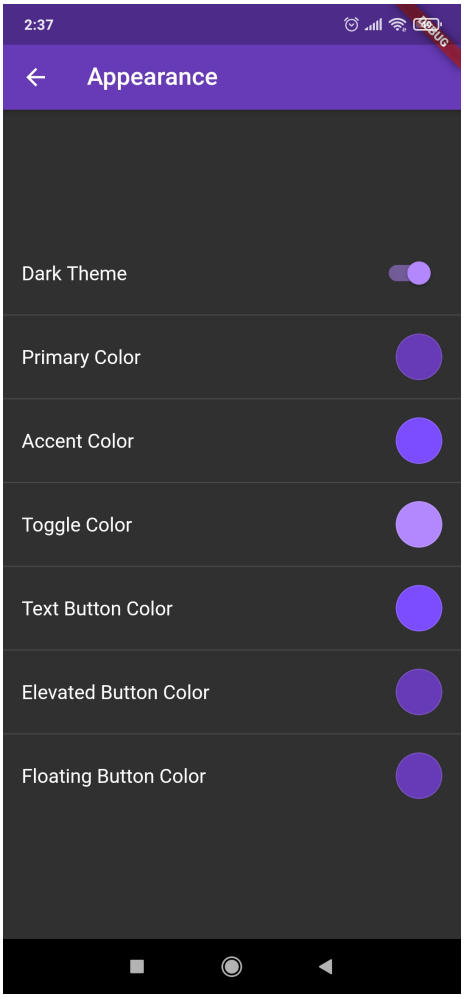

Figure 4.18: Customization Page

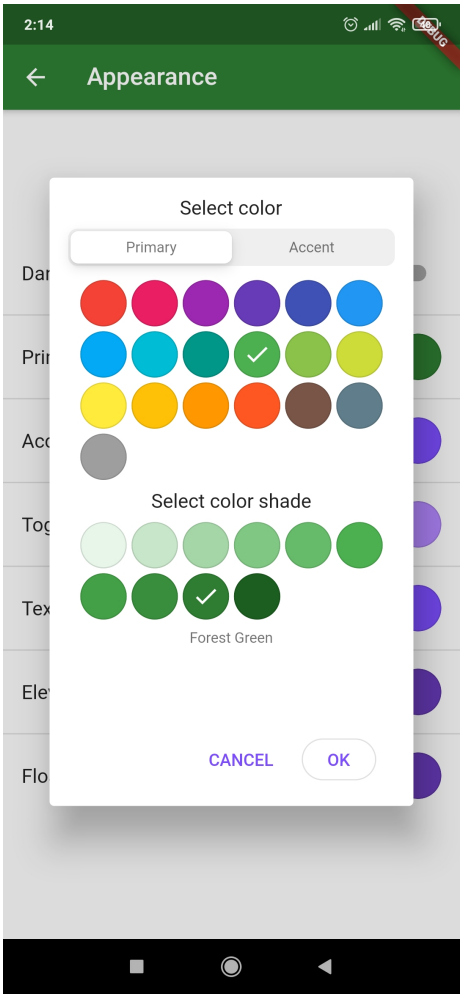

Figure 4.19: Color Selection Dialog

"Terms & Conditions" 4.20, "Privacy Policy" 4.21 and "Help" 4.22 all serve informational purposes.

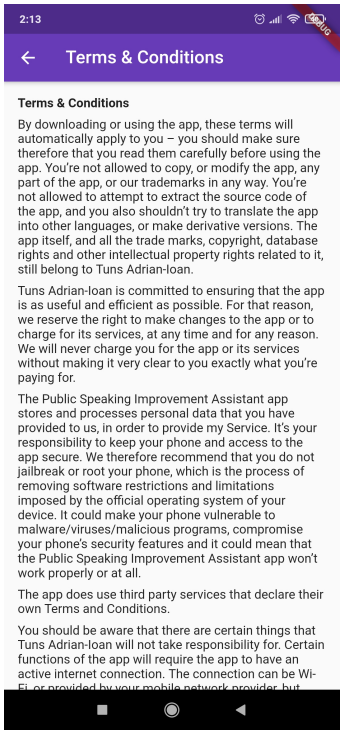

Figure 4.20: Terms & Conditions Page

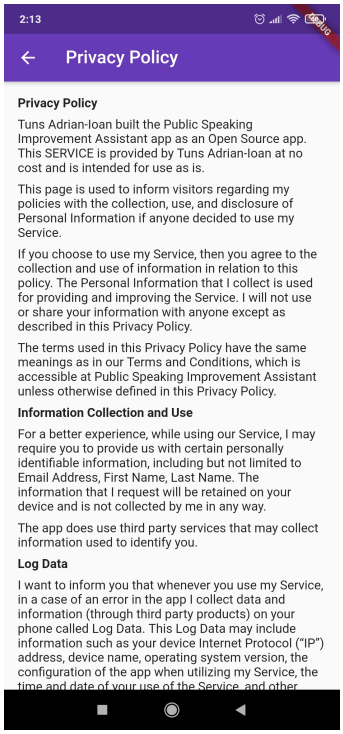

Figure 4.21: Privacy Policy Page

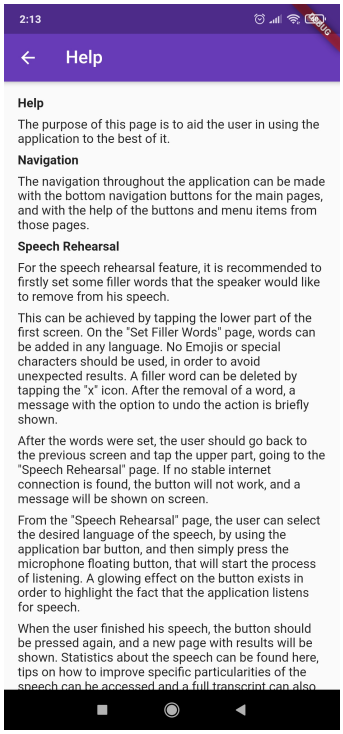

Figure 4.22: User Help Page

4.1.9.4 System Behaviour in Context

In figure 4.23, the process of speech rehearsal it's explained through the usage of an activity diagram:

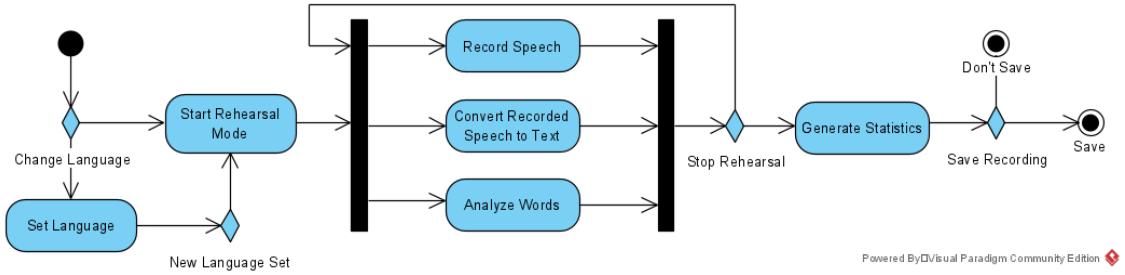

Figure 4.23: Activity Diagram

4.1.10 Internal Perspective

4.1.10.1 Class Diagram

For the internal perspective, the class diagram from figure 4.24 that models the classes used for the state objects from the application is presented. All of the classes have public variables and have a composition relationship, the "AppState" having all the other classes as composites.
The classes have no functions, because dedicated classes that control the interaction between the UI and the business logic are used, as explained in the section Redux Architecture and in section Architecture:

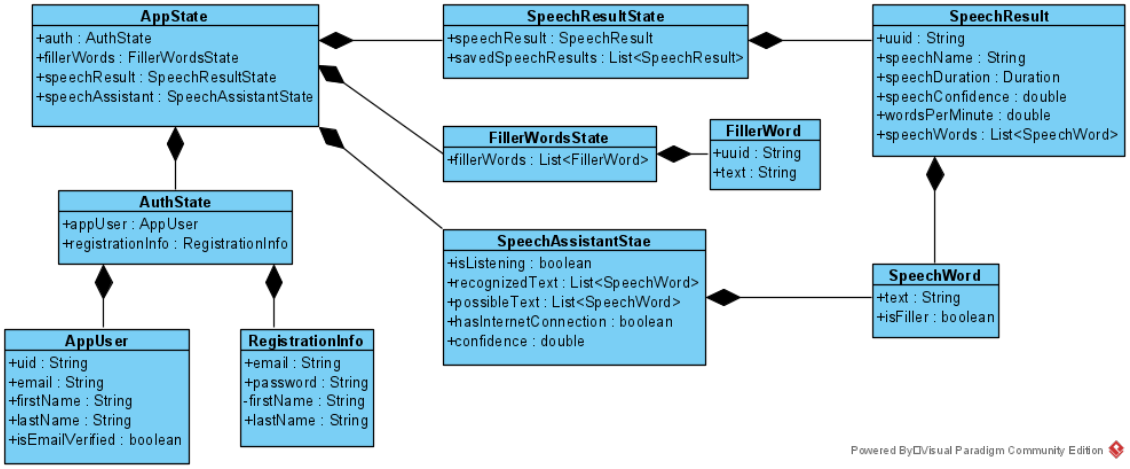

Figure 4.24: Class Diagram

4.1.10.2 Cloud Firestore Database Diagram

The Cloud Firestore database is a NoSQL, document-oriented database. The data is stored in documents and the documents are organized in collections.
The application's database from figure 4.25 has a collection named "users", where all the user documents are stored. A user document has several key-value pairs, with values such as numbers, strings, booleans and arrays of other documents, such as the array of speech results or speech words.

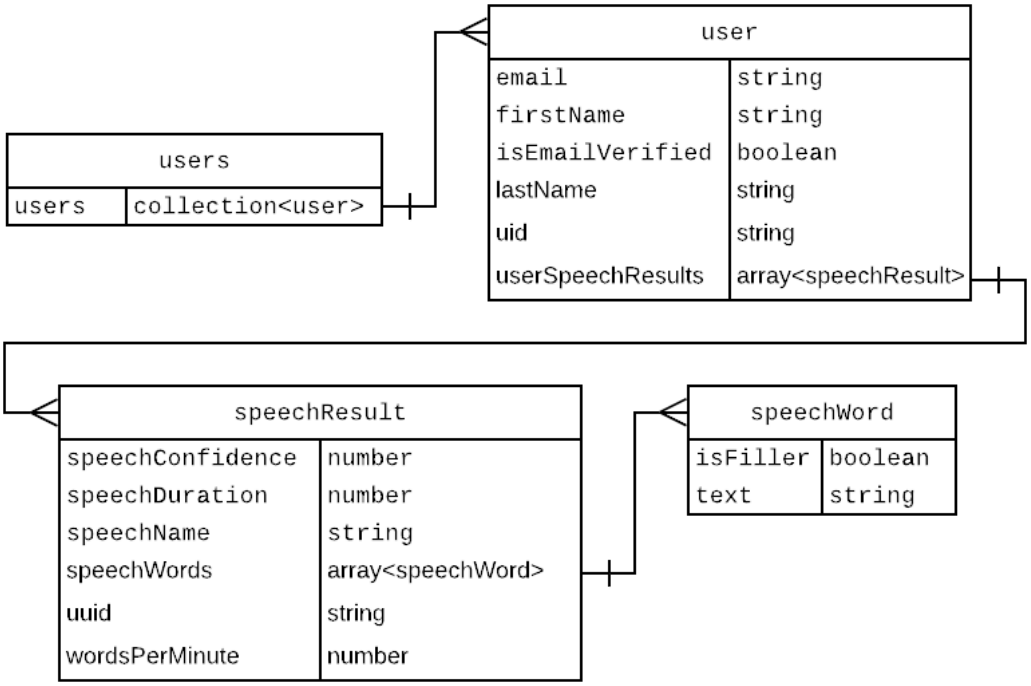

Figure 4.25: Firestore Database Diagram

4.2 Technologies

4.2.1 Technologies Used

4.2.1.1 Software Development Kit

Flutter [10] is used as a Software Development Kit, taking into consideration its abilities to create a beautiful designed application with ease, to easily test it with features such as "hot reload", which displays the code changes without restarting the whole application, affecting only the specific part that was changed and while also creating a fast and efficient application that can be easily expanded to work on other platforms.

Flutter is, as described by the creators, "Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, desktop, and embedded devices from a single codebase". It is an open-source UI software development kit, used in developing applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia

and for web application from a single codebase, that uses the programming language Dart, which is a client-optimized programming language for applications on multiple platforms, also developed by Google.

4.2.1.2 Application Programming Interface

For the speech-to-text feature, an API from Google was used, and more precisely Google Cloud Speech-to-Text API [28]. This API can convert speech into text with ease, in more than 125 languages and variants, with a high precision, making use of Google's deep learning neural network algorithms. This API requires a stable connection to the internet, in order to transform the speech into text. This is a priced service based on the amount of audio successfully processed each month, having a free hour on each month.

4.2.1.3 Local Database

For the local database, Hive [23] was chosen. Hive is a lighweight NoSQL key-value database, written in Dart, having therefore a great compatibility with Flutter in general. It can store both primitive data types and complex data types, through the usage of TypeAdapters, that can be generated from a class. The data is organized in "boxes", which can be compared to an SQL table, but without having a structure.

4.2.1.4 Services

Firebase [9] products were used for the real-time NoSQL database needed when the user wants to sync the local data and for other services such as end-to-end third-party authentication, secure user cloud storage and for analytics and insights of the users application usage.

Google Firebase is an application development platform developed by Google for creating iOS, Android and web applications. It offers options for the application build, for releasing and monitoring and for user engaging.

4.2.2 Technologies Previously Considered

4.2.2.1 Software Development Options

In the beginning, both single platform development options, such as android development with Java and software development kits for application development, such as Flutter and React Native, were considered.

React Native [12] is an open-source mobile application framework created by Facebook, Inc. It is used to develop applications for Android, iOS, macOS, web applications and Windows, by enabling developers to use React's framework, a JavaScript library for building user interfaces, along with native platform capabilities.

4.2.2.2 Speech-to-text Conversion

For the speech-to-text feature, a list of several free to use options was analysed and the best suited one was considered to be Dialogflow API [8]. Owned by Google,

Dialogflow is a natural language understanding platform that makes it easy to design and integrate a conversational user interface into an application.

The second option considered was a flutter plug-in, speech_to_text [29], that exposes the platform specific speech recognition capabilities.

4.3 Implementation

In order to create a flutter application, several plug-ins must be used, a plug-in being a wrapper of native code, that makes platform functionality available to the application [10]. Plug-in packages can be written for all of the supported platforms, such as Android, iOS, web, macOS, Windows or Linux.

One such plug-in that is heavily used for the Epics, is RxDart [27]. RxDart is an implementations of the reactiveX API, used for enabling reactive programming paradigm features in dart. Thorugh reactive programming, asynchronous actions can be created in reaction to other actions using observables.

Before explaining how most of the features work, it must be mentioned that all actions used in the application implement "AppAction", which is an abstract class with an empty constructor, in order to easily handle the results returned from the epics and each ".error" constructor implements "ErrorAction", which is similar to the previously mentioned class, except it has an "error" parameter of type "Object" (base class for all Dart objects), in order to easily and precisely handle any type of error.

4.3.1 Code Generators and Motivation

Dart as a programming language has several drawbacks, such as the lack of serialization, immutability and object comparison.

For solving the above specified problems, the following plug-ins were used: built_value [19], built_collection [18] and freezed [21].

Built_value is a boilerplate code generator plug-in, that requires a class written in a specific form, following this template, only providing the name for the desired class:

```
1 abstract class $CLASS_NAME$ implements Built<$CLASS_NAME$, $CLASS_NAME$Builder>
      {
2   $CLASS_NAME$._();
3   factory $CLASS_NAME$([void Function($CLASS_NAME$Builder) updates]) =
        _$$$CLASS_NAME$;
4 }
```

Listing 4.1: Built_value Generator Template

The generated code assures immutable object models, serialization if a serializer generator was specified (serializer that must use the StandardJsonPlugin, provided by the built_value plug-in) and object comparison, the generated classes being now value types objects, unlike the dart's reference type objects.

Built_collection does the same generation for collections instead, using the builder pattern, to create immutable, comparable and hashable collections.

All the files that will be generated with built_value need to be marked with a "g" as follows: "part 'example.g.dart';".

Freezed is another code generator for immutability, unions and pattern-matching, that was used in order to generate the actions, which do not need the complexity offered by built_value package, yet still need to be immutable. The following template is used for creating a freezed class:

```
@freezed
abstract class $CLASS_NAME$ with _$$$CLASS_NAME$ {
  const factory $CLASS_NAME$() = $CLASS_NAME$$;
}
```

Listing 4.2: Freezed Generator Template

All the files that will be generated with freezed need to be marked with a "freezed" as follows: "part 'example.freezed.dart';".

4.3.2 Application Initialization

When the application is started, all the database boxes are opened, and their instances along with instances of the required plug-ins are sent to the APIs. All the APIs are then sent to create a combined Epic, which is passed to the Store and at the end the Store is returned from the initialization function.

For example, the API that operates on speech result objects is initialized in the following way:

```
  final Box<HiveSpeechResult> speechResultsBox = await Hive.openBox<
      HiveSpeechResult>('speechResults');
  final Uuid uuidInstance = Uuid();

  final SpeechResultApi speechResultApi = SpeechResultApi(
   speechResultsBox: speechResultsBox,
   uuidInstance: uuidInstance,
 );
```

Listing 4.3: API Initialization

The first line opens a database box (table) called "speechResults" that holds object of type "HiveSpeechResult". An instance of Uuid [31] is initialized on the second row. Afterwards, an object that contains the API functions is created to be later passed to the Epics constructor.

The initialization on application start is ensured by having the initialize method called from a mixin attached to the main function. Mixins in dart are a way of reusing a class code without the need to extend or implement them.

```
 Future<void> _initStore() async {
   final List<dynamic> result = await ConcatStream<dynamic>(<Stream<dynamic>>[
     init().asStream(),
     Future<void>.delayed(const Duration(seconds: 2)).asStream(),
   ]).toList();

   _completer.complete(result[0]);
 }
```

Listing 4.4: Store Initialization

The above method is the method from the mixin, that creates two streams, one with the initialize function that returns the Store and one empty stream that waits for 2 seconds, in order to have time to show a splash screen on application start.

The "main" function uses an UI component that rebuilds itself based on the latest snapshot of an asynchronous method. Until the data with the Store from the mixin is received, the splash screen is shown.

4.3.3 Speech Recognition

For achieving the speech-to-text feature, the following plug-ins were needed: google_speech [22], sound_stream [13], connectivity [20], rxDart [27] and tuple [30].

Google_speech plug-in offers the possibility to use the Google Speech API in a flutter application. This API can be used to either recognize speech from a file, or from a continuous stream of data, giving back a stream of recognized words.

Sound_stream is a plug-in used to create a stream of audio data from the microphone without saving it to a file. This stream is then used as an input stream for Google Speech API.

Connectivity was used to continuously check the internet status and to notify when the connection was lost, in order to further notify the user that the rehearsal should be stopped.

Tuple was used to return more values from a function result, going from a pair of values up to 7 values in the same tuple.

Google API can recognize speech for more than 125 languages and variants. In order to specify the desired language, a code that follows a specific format must be used, the languages with their respective codes being specified on the API's documentation page. In order to allow the user to select the language of the speech, a JSON (Javascript Object Notation) with pairs of languages and codes was needed. Therefore, a script in Python was created for obtaining this JSON, using the principle of web scraping. This principle refers to extracting content and data from a website [2].

```
1  import json
2  import html_to_json
3  from urllib.request import urlopen
4  from bs4 import BeautifulSoup
5
6  html = urlopen("https://cloud.google.com/speech-to-text/docs/languages")
7
8  soup = BeautifulSoup(html, features="html.parser")
9
10 gdp_table = soup.find("div", attrs={"id": "lang-table-container"})
11
12 gdp_table_data = str(gdp_table.table).strip('\n')
13
14 json_from_tables = html_to_json.convert_tables(gdp_table_data)
15
16 result_dict = dict((i["Name"], i["BCP-47"]) for i in json_from_tables[0])
17
18 with open('language_codes.json', 'w', encoding='utf-8') as f:
19     json.dump(result_dict, f, ensure_ascii=False, indent=4)
```

Listing 4.5: Web Scraping Script

The script firstly gets the html of the required page, which is then parsed with the "BeautifulSoup" library. The "div" object which contains the required data is found, its new lines are stripped and then it is converted to a JSON. From the

respective JSON, only the required data is further segregated into an object that is then written into a file.

The way is which the speech-to-text feature was realised by following the Redux steps, with emphasis on the Google API speech recognition, will be explained:

1. The class that holds this specific state was written (class that represents a part of the general application state that exists over the entire application)

```
abstract class SpeechAssistantState implements Built<SpeechAssistantState
    , SpeechAssistantStateBuilder> {
factory SpeechAssistantState.initialState() {
  return _$SpeechAssistantState();
}

factory SpeechAssistantState.fromJson(dynamic json) => serializers.
    deserializeWith(serializer, json);

SpeechAssistantState._();

@nullable
bool get isListening;

@nullable
BuiltList<SpeechWord> get recognizedText;

@nullable
BuiltList<SpeechWord> get possibleText;

@nullable
bool get hasInternetConnection;

@nullable
double get confidence;

Map<String, dynamic> get json => serializers.serializeWith(serializer,
    this) as Map<String, dynamic>;

static Serializer<SpeechAssistantState> get serializer =>
    _$speechAssistantStateSerializer;
}
```

Listing 4.6: Speech Assistant State Class

The class was then generated to be immutable with built_value, having getters for fields that define if the application has an internet connection, is listening for speech words, what text was recognized and marked as final and what was marked as a possible transcript and the overall level of confidence of the API in recognizing the words.

2. The function from the API part was then written

```
Stream<Tuple3<List<SpeechWord>, bool, double>> listenForSpeech(
  String languageCode,
  String serviceAccountJson,
  List<String> fillerWords,
) {
  _fillerWords = fillerWords.map((String word) => word.toLowerCase()).
      toList();
```

```
7     _audioStream = BehaviorSubject<List<int>>();
8
9     _audioStreamSubscription = _recorderStream.audioStream.listen((
          Uint8List audioSlice) {
10      _audioStream.add(audioSlice);
11    });
12
13    final ServiceAccount serviceAccount = ServiceAccount.fromString(
          serviceAccountJson);
14    final SpeechToText speechToText = SpeechToText.viaServiceAccount(
          serviceAccount);
15    final RecognitionConfig config = _getConfig(languageCode);
16
17    return speechToText
18        .streamingRecognize(StreamingRecognitionConfig(config: config,
              interimResults: true), _audioStream)
19        .map((response.StreamingRecognizeResponse data) =>
              _getTextFromSpeechWithConfidence(data));
20  }
```

Listing 4.7: Function for Speech Recognition

This function is called after the recorder was initialised and started, in order to have an input from the microphone. It firstly initialises the "_fillerWords" class variable with the list of filler words received as a parameter, converted to lower case, then initialises a "BehaviourSubject" object, which captures the latest item that has been added to the controller, and emits that as the first item to a new listener. Afterwards, an "StreamSubscription" object is initialised to listen for any audio data that was detected by the _recorderStream (which is started with an action beforehand) and to put it further into the "BehaviourSubject" object.

In the following lines, the "ServiceAccount" is initialised with the service account string sent as a parameter, the "SpeechToText" is initialised via the previously created service account and the "RecognitionConfig" object is created, with a specific "languageCode".

A stream containing a tuple with recognized text, a flag to know if the transcript is considered a final version or not and the confidence value of transcript accuracy is then sent to the Middleware, tuple that is computed through the following method, based on the recognized responses from the API.

```
1   Tuple3<List<SpeechWord>, bool, double> _getTextFromSpeechWithConfidence(
        response.StreamingRecognizeResponse data) {
2     final List<String> currentText = data.results.map((response.
          StreamingRecognitionResult e) => //
3         e.alternatives.first.transcript).join('\n').split(' ');
4     currentText.removeWhere((String word) => word.isEmpty);
5
6     if (data.results.first.isFinal) {
7       return Tuple3<List<SpeechWord>, bool, double>(_checkFillerWords(
            currentText), true, _getConfidence(data.results));
8     } else {
9       return Tuple3<List<SpeechWord>, bool, double>(_checkFillerWords(
            currentText), false, null);
10    }
```

```
11  }
```

Listing 4.8: Function for Getting Text and Confidence From Response

The above method shows how the tuple is constructed: every data of "StreamingRecognizeResponse" is transformed, mapping each word of the first possible transcript to the "currentText" list and removing any empty word from the list.

If it is a final version, a tuple with a list of word objects where every word has the information if is is filler or not, a flag for knowing that transcript is final and a value representing the confidence of the transcript is sent to the Middleware. If it's not a final version, a list of words with the same information and the flag that shows that the transcript is partial is sent to the Middleware, in order to continuously have an output on the screen, as the user speaks.

```
1  List<SpeechWord> _checkFillerWords(List<String> words) {
2    return words.map((String word) {
3      // replace anything that is not a word, in order to check for words
           only
4      if (_fillerWords.contains(word.replaceAll(RegExp(r'[^\w]+'), '').
           toLowerCase())) {
5        return SpeechWord((SpeechWordBuilder b) {
6          b
7            ..text = '$word '
8            ..isFiller = true;
9        });
10     } else {
11       return SpeechWord((SpeechWordBuilder b) {
12         b
13           ..text = '$word '
14           ..isFiller = false;
15       });
16     }
17   }).toList();
18 }
```

Listing 4.9: Function for Creating a List With "SpeechWord" Objects

The method presented above shows how the list of "SpeechWord" objects is constructed. Every word from the list of words parameter is checked if it appears in the "_fillerWords" class variable, being then replaced with a newly created "SpeechWord" object, which hold the text and a flag that shows if the word is considered a filler word or not.

Before checking if it appears in the list, the word's text has all the punctuation and spaces removed through the regular expression, being then converted to lower case.

```
1  double _getConfidence(List<response.StreamingRecognitionResult> results)
       {
2    final double currentSum = results.fold<double>(
3        0, (double prev, response.StreamingRecognitionResult result) =>
             prev + result.alternatives.first.confidence);
4
5    if (currentSum == 0.0) {
```

```
6       return currentSum;
7     }
8
9     _confidenceSum += currentSum;
10    _confidenceCount++;
11
12    return _confidenceSum / _confidenceCount;
13  }
```

Listing 4.10: Function for Computing the Confidence Value

In the above method, the sum of each confidence value given to the final transcript words is computed, which is then added to the class variable "_confidenceSum", afterwards the division of the "_confidenceSum" over the class variable "_confidenceCount" being returned.

3. Several actions that describe clear requests are used

   - Initialize recorder - used for initialising the sound_stream plug-in when the recording page is on screen
   - Start recorder - used for starting the data recording from microphone
   - Stop recorder - used for stopping the data recording from microphone
   - Listen for speech - used for initialising the process of listening for words through the google_speech plug-in
   - Stop listening for speech - used for stopping the process of listening for words
   - Listen for internet status - used for initialising the process of listening for internet status, through the connectivity plug-in
   - Stop listening for internet status - used for stopping the process of listening for the internet status

   Taking for example the "ListenForSpeech" action:

```
1 @freezed
2 abstract class ListenForSpeech with _$ListenForSpeech implements AppAction
      {
3   const factory ListenForSpeech({@required String languageCode, @required
        String serviceAccount}) = ListenForSpeech$;
4
5   const factory ListenForSpeech.successful(Tuple2<List<SpeechWord>, double>
        recognizedTextWithConfidence) =
6       ListenForSpeechSuccessful;
7
8   const factory ListenForSpeech.partial(List<SpeechWord> recognizedText) =
        ListenForSpeechPartial;
9
10  @Implements(ErrorAction)
11  const factory ListenForSpeech.error(Object error) = ListenForSpeechError;
12 }
```

Listing 4.11: Complex Action for Speech Listening

Four constructors are created. One through which the action is initiated, that requires the language code to be later passed to the API and the service account, that is needed for using this service. One used when a final transcript is returned, through a tuple with a list of "SpeechWord" objects and a confidence value. One used when a partial transcript is returned, containing only a list of "SpeechWord" objects, and another one containing an error, if something went wrong. The error constructor is needed to notify the application that this needs to be handled.

In contrast, an action used for stopping a listening process would have only one constructor:

```
@freezed
abstract class StopListeningForSpeech with _$StopListeningForSpeech
    implements AppAction {
  const factory StopListeningForSpeech() = StopListeningForSpeech$;
}
```

Listing 4.12: Simple Action for Stopping Speech Listening

4. Next, the action goes to the Middleware part:

```
Stream<AppAction> _listenForSpeech(Stream<dynamic> actions, EpicStore<
    AppState> store) {
  return actions //
      .whereType<ListenForSpeech$>()
      .flatMap((ListenForSpeech$ action) => _api
          .listenForSpeech(
            action.languageCode,
            action.serviceAccount,
            store.state.fillerWords.fillerWords.map((FillerWord e) => e.
                text).toList(),
          )
          .map((Tuple3<List<SpeechWord>, bool, double> recognizedTextTuple
              ) {
            // if the recognized text is marked as final, send it to
                successful constructor; otherwise, to the partial one
            if (recognizedTextTuple.item2) {
              // for the successful constructor, send the average
                  confidence too
              return ListenForSpeech.successful(
                  Tuple2<List<SpeechWord>, double>(recognizedTextTuple.
                      item1, recognizedTextTuple.item3));
            } else {
              return ListenForSpeech.partial(recognizedTextTuple.item1);
            }
          })
          .takeUntil(actions.whereType<StopListeningForSpeech$>()) //
              close the stream on this action
          .onErrorReturnWith((dynamic error) => ListenForSpeech.error(
              error)));
}
```

Listing 4.13: Epics Function for Recognized Speech

Here, a stream of actions and a reference to the store are given as parameters. If the ListenForSpeech$ (where $ marks the fact that it was generated and is

immutable) action was found, it is passed as a request to the API class, alongside a list with the text of the filler words that exist in the state and that can be accessed from the store, mapping the returned results from the API to the "ListenForSpeech.successful" constructor or to the "ListenForSpeech.partial" constructor, based on the flag that says if it is a final transcript or not.

The epics takes actions until the "StopListeningForSpeech$" action was encountered. On an encountered error, the stream sends the error to the "ListenForSpeech.error" constructor.

5. The reducer is now announced if speech words were successfully recognized

```
SpeechAssistantState _listenForSpeechSuccessful(SpeechAssistantState state,
    ListenForSpeechSuccessful action) {
  return state.rebuild((SpeechAssistantStateBuilder b) => b
    ..recognizedText.addAll(action.recognizedTextWithConfidence.item1)
    ..confidence = action.recognizedTextWithConfidence.item2
    ..possibleText = ListBuilder<SpeechWord>());
}
```

Listing 4.14: Reducer for Changing the State on Recognized Text

This function rebuilds the "recognizedText", by adding new recognized words, the confidence value by updating it with the newly sent value and the "possibleText" by replacing it with an empty list.

6. Containers that may exist over an UI component are then changing the data with the one that was changed in the Store

```
class RecognizedTextContainer extends StatelessWidget {
  const RecognizedTextContainer({Key key, @required this.builder}) : super(
      key: key);

  final ViewModelBuilder<List<SpeechWord>> builder;

  @override
  Widget build(BuildContext context) {
    return StoreConnector<AppState, List<SpeechWord>>(
      converter: (Store<AppState> store) => store.state.speechAssistant?.
          recognizedText?.toList() ?? <SpeechWord>[],
      builder: builder,
    );
  }
}
```

Listing 4.15: Container for Retrieving Recognized Text From Store

The container converts the Store into a ViewModel object, which is passed in the builder function, and which announces the UI Component situated in the container that it needs to rebuild itself and to use the new data.

The container used above has access only to the "recognizedText" variable, announcing the component to change only when this variable is modified. Several such containers are used over components that need to know details about a particular variable and that need to use updated data after a modification.

4.3.4 Saving Data Locally and Syncing to Cloud

For assuring data persistence, the NoSQL key-value local database called Hive [23] is used for saving various settings and data and another one provided through the firebase [9] services is used for syncing speech results to a cloud account. Plug-ins that offer the possibility to interact with those services were used.

Without going into detail with the Redux steps, the API functions that handle the database interaction will be emphasized:

1. Saving "speechResult" objects locally

```
Future<List<SpeechResult>> saveSpeechResult({@required SpeechResult
    speechResult}) async {
  final HiveSpeechResult convertedSpeechResult =
      _convertSpeechResultToHive(speechResult: speechResult);

  await _speechResultsBox.put(speechResult.uuid, convertedSpeechResult);
  final List<HiveSpeechResult> hiveSpeechResultList = _speechResultsBox.
      values.toList();

  return _convertHiveListToSpeechResult(hiveSpeechResultList:
      hiveSpeechResultList);
}
```

Listing 4.16: Function to Save a Speech Result Locally

When the action for saving a speech result was dispatched, the above function start the process of saving the "speechResult" object.

Firstly, the "speechResult" object is converted to an "hiveSpeechResult" object that can be saved in the local database, through the following function:

```
HiveSpeechResult _convertSpeechResultToHive({@required SpeechResult
    speechResult}) {
  final HiveSpeechResult hiveSpeechResult = HiveSpeechResult(
    speechResult.speechDuration.inSeconds,
    speechResult.speechConfidence,
    speechResult.speechWords.map((SpeechWord word) => HiveSpeechWord(word
        .text, word.isFiller)).toList(),
    speechResult.speechName,
    speechResult.uuid,
    speechResult.wordsPerMinute,
  );

  return hiveSpeechResult;
}
```

Listing 4.17: Function to Convert a Speech Result Object to a Hive Object

The converter function constructs and returns an object that can be saved in the local database, object that has the "read" and "write" functions, the equal operator and the hashcode overridden through an adapter object, adapter that is registered in the main function in the following way: "Hive.registerAdapter<HiveSpeechResult>(HiveSpeechResultAdapter());"

After the hive object was constructed by mapping each of the "speechResult" parameters to the "hiveSpeechResult" constructor, the box that holds the

speech results is used, by calling the "put" method, adding a new element with the "uuid" as a key and the converted object as a value.

Afterwards, a list with all the saved "hiveSpeechResult" objects is converted to a list with simple "speechResult" objects and returned from the function, in order to further notify the Reducer to update the view.

2. Syncing "speechResult" objects to cloud

When trying to sync a speech result to cloud, the object and a bool flag that tells if the object is synced are needed.

```
1  Future<List<SpeechResult>> syncSpeechResultToCloud(
2      {@required SpeechResult speechResult, @required bool isSynced}) async
           {
3    final User currentUser = _auth.currentUser;
4    if (currentUser == null) {
5      return null;
6    }
7
8    final DocumentSnapshot snapshot = await _firestore.doc('users/${
         currentUser.uid}').get();
9
10   final AppUser user = AppUser.fromJson(snapshot.data());
11   final List<SpeechResult> syncedSpeechResults = user.userSpeechResults.
         toList();
12
13   // if the result is already on cloud, it means it must be removed
14   if (isSynced) {
15     syncedSpeechResults.remove(speechResult);
16   } else {
17     syncedSpeechResults.add(speechResult);
18   }
19
20   final AppUser userUpdated = AppUser((AppUserBuilder b) {
21     b
22       ..uid = user.uid
23       ..email = user.email
24       ..firstName = user.firstName
25       ..lastName = user.lastName
26       ..userSpeechResults = ListBuilder<SpeechResult>(syncedSpeechResults
             )
27       ..isEmailVerified = user.isEmailVerified;
28   });
29
30   await _firestore.doc('users/${user.uid}').set(userUpdated.json);
31
32   return syncedSpeechResults;
33 }
```

Listing 4.18: Function to Sync Speech Result Object to Firestore

First thing in syncing the result to cloud, is checking if a user is currently authenticated. Afterwards, a snapshot with the data of the user's document is retrieved. An "AppUser" object is created from the snapshot and a list with the current synced results instantiated. Based on the flag given as parameter, the "speechResult" object is either saved or removed from the list. A new "AppUser" object with the updated list of speech results is created and the

user document from cloud is updated with the JSON of the object. If no stable internet connection is found, the firebase data persistence will make the syncing operation as soon as a connection is found.

4.3.5 Problems Encountered

Before making the final choice for the speech-to-text feature, the Dialogflow API [8] was considered and the plug-in speech-to-text [29] was tried.

Speech-to-text is a plug-in that exposes the platform capability of speech recognition, being able to work with no internet connection. A problem was encountered during the testing: on Android platform, the maximum waiting time with no recognized words is 5 seconds, after which the speech recognition automatically stops. This limitation made the plug-in unusable for an application that needs the user to prepare a speech, speech which may have several intentional pauses. And for Dialogflow API, no plug-in was found for integrating it in the application.

4.3.6 Code Quality

In order to assure a high code quality, two methods are used.

The first method, is by using Dart's static analysis tool, which requires a file called "analysis_options.yaml" to work. The static analysis finds bugs before executing any piece of code, while also ensuring that the code conforms to the dart style guidelines.

The second method is through the CI principle, of the CI/CD (Continuous Integration/Continuous Delivery) principles, and with the help of Github Actions. Two scripts run on the application code after a commit is pushed to the Github repository, scripts that make sure that that all the code respects the coding standards, that no bugs or security vulnerabilities were found and that the code is properly formatted, following the dart style guidelines, with a maximum code line length of 120 characters.

Chapter 5

User's Manual

5.1 Installing

In order to use the Public Speaking Improvement Assistant, the user must install it through the provided APK (Android Package Kit) installation file.

An Android device with at least Android Lollipop 5.0 is required for this application.

5.1.1 Running the Project Locally

If the project needs to be run locally instead of installing it from the APK, several configurations must be done.

For Windows configuration, the following requirements must be met:

- Operating Systems: Windows 7 SP1 or later (64-bit), x86-64 based
- Disk Space: 1.64 GB (without the disk space for IDE/tools)
- Specific tools:
  - Windows PowerShell 5.0 or newer
  - Git for Windows 2.x, with the Use Git from the Windows Command Prompt option

For macOS configuration, the following requirements must be met:

- Operating Systems: macOS (64-bit)
- Disk Space: 2.8 GB (without the disk space for IDE/tools)
- Specific tools:
  - Git
  - Highly Recommended: Xcode

For other operating systems requirements (Linux, ChromeOS) please see the documentation [10].

The flutter SDK must be installed, by following the steps from the official documentation. Flutter version 1.x is required for this application.

The usage of IntelliJ IDEA IDE [24] is recommended. For IntelliJ IDEA, the following plug-ins should be installed: "Dart", "Flutter" and "Flutter Enhancement Suite".

After configuring the IDE and the project, from the terminal the command "flutter pub get" must be run, in order to get the plug-ins data.

Afterwards, a Firebase account must be created and a new project added [16]. From the project, an Android application should be registered, and a SHA-1 key generated [17] from the IDE and then added to the Firebase Android project settings. The Firebase configuration file should then be downloaded and added to the local project as instructed in the documentation.

For the Google Speech API, a Google Cloud Platform account must be created. After the account creation, the previously created project from Firestore will appear among the existing projects. It must be selected and the Cloud Text-to-Speech API [28] enabled [25]. A service account must then be created and put in the project "/private/account" folder.

After the configurations are done, the project can be started on an Android emulator or on a physical device.

5.2 Navigation

The navigation throughout the application can be made with the bottom navigation buttons for the main pages, and with the help of the buttons and menu items from those pages.

5.3 Speech Rehearsal

For the speech rehearsal feature, it is recommended to firstly set some filler words that the speaker would like to remove from his speech.

This can be achieved by tapping the lower part of the first screen. On the "Set Filler Words" page, words can be added in any language. No Emojis or special characters should be used, in order to avoid unexpected results. A filler word can be deleted by tapping the "x" icon. After the removal of a word, a message with the option to undo the action is briefly shown.

After the words were set, the user should go back to the previous screen and tap the upper part, going to the "Speech Rehearsal" page. If no stable internet connection is found, the button will not work, and an informative message will be shown on the screen.

From the "Speech Rehearsal" page, the user can select the desired language of the speech, by using the application bar button, and then simply press the microphone floating button, that will start the process of listening. A glowing effect on the button exists in order to highlight the fact that the application listens for speech.

When the user finished his speech, the button should be pressed again, and a new page with results will be shown. Statistics about the speech can be found here, tips on how to improve specific particularities of the speech can be accessed and a full transcript can also be read. After setting a title, the speech can be saved locally.

5.4 Library

The "Library" page contains all of the saved speeches. By tapping on one, the user can see the full transcript and can also see the statistics with tips for improvement, by using the button from the application bar, from inside the transcript page.

Any speech can be synced to or unsynced from cloud if an account is currently logged-in, by tapping the icon depicting a cloud.

Speeches can be deleted by swiping from left to right. If a speech was also synced, it will be deleted from cloud too. After the removal of a speech, a message with the option to undo the action is briefly shown.

In the case when no internet connection exists, the syncing will be made as soon as a connection is found.

5.5 Account Creation and Usage

The first option from the "Settings" page stands for the "Account". A new account can be created with an email and a password, or by signing-in with a Google account.

Once an account was created, a confirmation email is sent. After the email was confirmed, the user should log in again, and the account page will be shown.

From the account page, previously synced speeches can be downloaded. The account and all of its data can also be deleted from the application bar button.

In the case the password was forgotten, it can be reset by tapping the "Forgot password?" button, from the login page.

5.6 Appearance Customization

The second option from the "Settings" page shows the Appearance page. From here, the user can switch between light and dark theme, or even select a desired color for several components of the application. By tapping a circle with color corresponding to an option, a dialog with several colors and shades is shown.

5.7 Informative Pages

The pages "Terms & Conditions", "Privacy Policy" and "Help" serve informative purposes.

Chapter 6

Conclusions and Future Work

An application that satisfies the requirements, and more precisely aids a person in preparing a speech that will later be delivered in front of an audience, was developed.

The application follows the strict architecture of Redux and offers several features, from the main feature that helps in the rehearsal of the speech in over 125 languages and dialects, by offering details on the number of filler words used, the clarity of the words and the pace of speaking, to features for saving the results locally and even syncing them to a cloud account, and to customizing the application looks.

For the future, the application needs to be developed further, to be able to detect filler words such as "um" or "eh" too, and to provide a more in-depth analysis for other statistics, such as the volume or the pitch of the voice.

Bibliography

[1] Tony Docan-Morgan and Laura Nelson. *The Benefits and Necessity of Public Speaking Education*. May 2015, pp. 1–16.

[2] Katharine Jarmul and Richard Lawson. *Python Web Scraping - Second Edition: Hands-on Data Scraping and Crawling Using PyQT, Selnium, HTML and Python*. 2nd. Packt Publishing, 2017. ISBN: 1786462583.

[3] Philip Lindner, Alexander Miloff, Simon Fagernäs, Joel Andersen, Martin Sigeman, Gerhard Andersson, Tomas Furmark, and Per Carlbring. *Therapist-led and self-led one-session virtual reality exposure therapy for public speaking anxiety with consumer hardware and software: A randomized controlled trial*. Vol. 61. July 2018. DOI: 10.1016/j.janxdis.2018.07.003.

[4] José Proença. *Web app architecture based on Redux - José Proença - Medium*. Medium, 2018. URL: https://medium.com/@akaztp/web-app-architecture-based-on-redux-1e16294c817a.

[5] *Home - Say It Like So*. [Online; accessed 14. Nov. 2020]. 2019. URL: https://sayitlikeso.com.

[6] *Introduction · redux-observable*. [Online; accessed 5. Jan. 2021]. 2019. URL: https://redux-observable.js.org.

[7] Anshul Bhagi. *Ummo | Speech Coaching App*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://ummoapp.herokuapp.com.

[8] *Dialogflow Documentation | Google Cloud*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://cloud.google.com/dialogflow/docs.

[9] *Firebase*. [Online; accessed 3. Jan. 2021]. 2020. URL: https://firebase.google.com.

[10] *Flutter - Beautiful native apps in record time*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://flutter.dev.

[11] *Orai | AI communication coach and speech coach app*. [Online; accessed 13. Nov. 2020]. 2020. URL: https://www.orai.com.

[12] *React Native*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://reactnative.dev.

[13] *sound_stream | Flutter Package*. [Online; accessed 21. Jun. 2021]. 2020. URL: https://pub.dev/packages/sound_stream.

[14] www. designmilitia. co. uk. *Speech Tools*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://speechtools.co/voice-analyst.

[15] *VirtualSpeech - Soft Skills Training with VR*. [Online; accessed 15. Nov. 2020]. 2020. URL: https://virtualspeech.com.

[16] *Add Firebase to your Flutter app*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://firebase.google.com/docs/flutter/setup?platform=android.

[17] *Authenticating Your Client | Google Play services | Google Developers*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://developers.google.com/android/guides/client-auth.

[18] *built_collection | Dart Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/built_collection.

[19] *built_value | Dart Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/built_value.

[20] *connectivity | Flutter Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/connectivity.

[21] *freezed | Dart Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/freezed#motivation.

[22] *google_speech | Flutter Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/google_speech.

[23] *Hive Docs*. [Online; accessed 20. Jun. 2021]. 2021. URL: https://docs.hivedb.dev/#.

[24] *IntelliJ IDEA*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://www.jetbrains.com/idea.

[25] *Quickstart: Using client libraries*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://cloud.google.com/speech-to-text/docs/quickstart-client-libraries.

[26] *Redux - A predictable state container for JavaScript apps. | Redux*. [Online; accessed 5. Jan. 2021]. 2021. URL: https://redux.js.org.

[27] *rxdart | Dart Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/rxdart.

[28] *Speech-to-Text: Automatic Speech Recognition | Google Cloud*. [Online; accessed 20. Jun. 2021]. 2021. URL: https://cloud.google.com/speech-to-text.

[29] *speech_to_text | Flutter Package*. [Online; accessed 20. Jun. 2021]. 2021. URL: https://pub.dev/packages/speech_to_text.

[30] *tuple | Dart Package*. [Online; accessed 27. Jun. 2021]. 2021. URL: https://pub.dev/packages/tuple.

[31] *uuid | Dart Package*. [Online; accessed 21. Jun. 2021]. 2021. URL: https://pub.dev/packages/uuid.