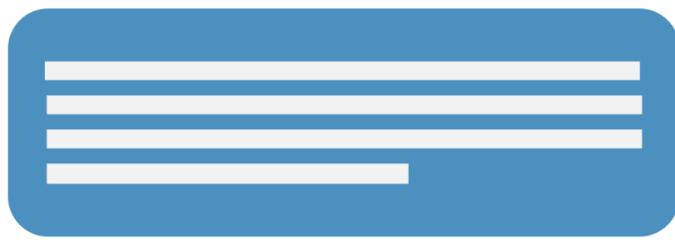


## Bachelor Thesis 2023

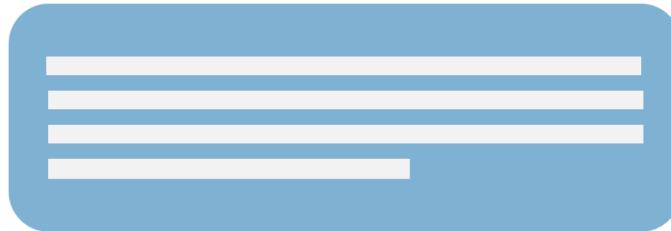
### Automated Chatbot Using Speech-to-Text and Text-to-Speech with Mobile App Integration



Freepik.com



Freepik.com



Student : Natal Bumann

Professor : Dominique Genoud

Submitted on : 28. July 2023

## Abstract

This bachelor thesis deals with developing an automated chatbot using Speech-to-Text (STT) and Text-to-Speech (TTS) technologies integrated into a mobile app. The main objective of the work is to identify and select suitable STT and TTS application programming interfaces (APIs) for the chatbot.

The research methodology included a thorough analysis of different STT APIs considering the criteria of accuracy, speed, cost, and quality of documentation. A similar investigation was also conducted for TTS APIs, evaluating aspects such as quality of voice output, configuration options, language scope, speed, and cost.

After carefully evaluating the STT APIs, OpenAI's Whisper API was selected for its consistent performance across all criteria. Its high accuracy, coupled with fast execution, was particularly compelling. In the case of the TTS API, the choice fell on Microsoft Azure's API, as it, too delivered consistent results in all measured aspects. It also offered the added benefit of generating matching Visemes. Visemes are the visual representations of phonemes in speech, used in lip-syncing and facial animation to match mouth movements with corresponding sounds.

The mobile application was then developed based on these evaluations. The most significant sections of the implementation are explained in detail using code snippets. In the final step, the mobile app underwent user testing and verification.

**Keywords:** chatbot, speech-to-text, text-to-speech, API analysis, mobile app, ChatGPT, OpenAI Whisper, Microsoft Azure.

## Foreword

This bachelor thesis was conducted as part of the Business Informatics degree program at the HES-SO Valais-Wallis. The work was proposed under the supervision of Professor Dominique Genoud.

The work lasted from May 01 to July 28, 2023, and addressed the topic "*Automated Chatbot Using Speech-to-Text and Text-to-Speech with Mobile App Integration.*" I was fascinated by the idea of developing a chatbot that converts speech to text and vice versa and integrating it into a mobile application. This combination of STT and TTS technologies makes it possible to take chatbot interaction to a new level. It opens exciting prospects for usability and application range.

At this point, I would like to thank everyone who supported me during my bachelor's thesis. Thanks go to my supervisor Dominique Genoud. I would also like to thank Sven Zengaffinen for proofreading my work.

## Table of content

List of tables .....	vii
List of Figures.....	viii
List of abbreviations .....	x
1. Introduction.....	1
2. State of the Art .....	3
2.1 Natural Language Processing.....	3
2.2 Chatbots.....	5
2.2.1 What is a chatbot?.....	5
2.2.2 What is ChatGPT? .....	5
2.2.3 Today's Chatbots.....	5
2.3 Speech-to-Text Technology.....	6
2.3.1 Current State of STT Technology .....	7
2.3.2 Advantages of using STT concerning chatbots .....	8
2.4 Text-to-Speech Technology.....	8
2.4.1 Current State of TTS Technology .....	10
2.4.2 Advantages of using TTS concerning chatbots .....	11
3 Methodology .....	12
3.1 Methodology of STT .....	12
3.1.1 Overview of the selected STT-APIs .....	12
3.1.2 Data Acquisition of STT.....	13
3.1.3 Evaluation Criteria of STT .....	14
3.1.3.1 Accuracy.....	15
3.1.3.2 Performance .....	16

---

3.1.3.3 Cost .....	17
3.1.3.4 Usability and documentation quality .....	18
3.2 Methodology of TTS .....	19
3.2.1 Overview of the selected TTS-APIs .....	20
3.2.2 Data Acquisition of TTS.....	21
3.2.3 Evaluation Criteria of TTS .....	21
3.2.3.1 Voice quality .....	21
3.2.3.2 Customization options.....	22
3.2.3.3 Language coverage .....	22
3.2.3.4 Performance .....	22
3.2.3.5 Costs .....	23
4 Results .....	24
4.1 Results of Speech-to-text.....	24
4.1.1 Results Accuracy .....	24
4.1.2 Results performance.....	25
4.1.3 Results costs .....	26
4.1.4 Results usability and documentation quality .....	27
4.1.5 Overall results of STT .....	27
4.2 Results of Text-to-Speech .....	28
4.2.1 Results voice quality .....	28
4.2.2 Results Customization options .....	28
4.2.3 Results language coverage .....	29
4.2.4 Results performance.....	29
4.2.5 Results costs .....	30
4.2.6 Overall result of TTS.....	31

5 Justification.....	32
5.1 Justification of STT-API.....	32
5.2 Justification of TTS-API .....	33
5.3 Justification of NLP API .....	33
5.4 Justification of Avatar Animation API .....	34
6 Technical Basis of the Implementation .....	35
6.1 Technologies .....	35
6.1.1 Flutter with Dart .....	35
6.1.2 Rive Flutter.....	35
6.1.3 Google Firebase .....	35
6.1.4 Python with Flask .....	36
6.1.5 DigitalOcean® .....	36
6.2 Architecture .....	36
6.2.1 System architecture of the test environment.....	37
6.2.2 System architecture of the productive environment .....	37
6.2.3 Software architecture of the Flutter app.....	38
6.3 Functionalities of the App .....	38
6.3.1 Login and Registration .....	39
6.3.2 Main screen .....	40
6.3.3 Open menu .....	43
6.3.4 Settings and profile screen .....	43
6.3.5 Error messages .....	44
7 Implementation.....	44
7.1 Chatbot .....	45
7.1.1 Milestone 1: implementation of login/registration.....	45

7.1.2 Milestone 2: implementation of STT .....	46
7.1.3 Milestone 3: implementation of ChatGPT .....	50
7.1.4 Milestone 4: implementation of TTS .....	51
7.1.5 Milestone 5: animation of the Avatar.....	54
7.1.5.1 Process in Rive .....	56
7.1.5.2 Rive in combination with Flutter .....	60
7.1.6 Milestone 6: implementation of Cloud Firestore .....	60
7.2 Flask Application.....	65
8 Publishing .....	66
8.1 Publishing chatbot application .....	66
8.2 Publishing Flask application.....	68
9 Testing.....	70
9.1 Manual testing.....	70
9.2 Manual testing by external persons .....	71
10 Further improvements .....	73
11 Conclusion .....	75
References .....	77
List of Appendices .....	85

## List of tables

Table 1 STT-APIs and the used version and tariff .....	12
Table 2 Rating scale of usability and documentation quality.....	19
Table 3 TTS-APIs and the used version and tariff .....	20
Table 4 Overview of the average speed values for the respective audio lengths.....	25
Table 5 Overview of the load testing for 100 requests .....	26
Table 6 Costs for each STT-API.....	26
Table 7 Weighted total score for the respective STT-API .....	27
Table 8 Total overview of all STT-API results .....	27
Table 9 Overview of the two categories, Intelligibility and Naturalness.....	28
Table 10 Number of languages (including dialects) supported by various TTS-APIs .....	29
Table 11 Cost of the TTS-APIs .....	30
Table 12 Monthly costs of the TTS-APIs .....	31
Table 13 Total overview of all TTS-API results .....	31
Table 14 The implemented milestones .....	45
Table 15 States of the avatar .....	54

## List of Figures

Figure 1 Abstract representation of the NLP components.....	4
Figure 2 Traditional chatbot vs. modern AI chatbot .....	6
Figure 3 Abstract STT process.....	7
Figure 4 Abstract TTS process.....	9
Figure 5 WER formula.....	15
Figure 6 The formula for the calculation of STT costs .....	18
Figure 7 The formula for the calculation of TTS costs .....	23
Figure 8 Overview of the average WER of the STT-APIs.....	24
Figure 9 Overview of the average WER per language and API.....	25
Figure 10 Bar chart that visualizes the speed test results for the TTS APIs. ....	30
Figure 11 Results of the speed test of the GPT models .....	34
Figure 12 The architecture of the system within the testing environment. ....	37
Figure 13 The architecture of the system within the production environment. ....	37
Figure 14 Mockup of the login and registration page .....	39
Figure 15 Sequence diagram of the login.....	39
Figure 16 Sequence diagram of the registration.....	40
Figure 17 Mockup of the main page and the opened panel.....	41
Figure 18 Sequence diagram of the STT part .....	41
Figure 19 Sequence diagram of the ChatGPT part .....	42
Figure 20 Sequence diagram of the TTS and avatar animation part.....	42
Figure 21 Mockup of the opened menu.....	43
Figure 22 Mockup from the settings and profile page .....	43
Figure 23 Mockup of alert message .....	44
Figure 24 GUI of the login and registration screen .....	46
Figure 25 Login method.....	46
Figure 26 GUI of the main page and the permissions question.....	47
Figure 27 Check the permission method. ....	48
Figure 28 Start recording method. ....	48
Figure 29 Stop recording method. ....	49

---

Figure 30 getTranscriptSTT() method .....	49
Figure 31 Method to call the STT-API.....	50
Figure 32 Method to call the ChatGPT-API.....	51
Figure 33 Method to call the TTS-API.....	52
Figure 34 Method to call the TTS-API.....	53
Figure 35 Methods to get the Visemes. ....	54
Figure 36 GUIs of the different states of the avatar. ....	55
Figure 37 Excerpt from the Microsoft documentation about the Viseme .....	55
Figure 38 Array of the TTS-API .....	56
Figure 39 Illustration of a closed mouth .....	57
Figure 40 Web interface design.....	57
Figure 41 Web interface animation.....	58
Figure 42 Illustration of a made lip animation.....	58
Figure 43 "State Machine" of the chatbot project .....	59
Figure 44 Illustration of the change of state .....	59
Figure 45 Entire Rive implementation in Flutter. ....	60
Figure 46 GUI of a new and a saved chat .....	61
Figure 47 Code for saving the chat.....	62
Figure 48 Code to save the chat automatically.....	62
Figure 49 Showing the code for deleting the chat. ....	63
Figure 50 GUI of the opened menu.....	63
Figure 51 Shows the code for displaying the chats in the menu. ....	64
Figure 52 Firestore Rules .....	65
Figure 53 Representation of the Python code .....	66
Figure 54 Illustration of the YAML-Code .....	67
Figure 55 Illustration of a release on GitHub .....	67
Figure 56 Screenshot of the selected configuration for the Droplet .....	68
Figure 57 Screenshot of the problem with cutting off the answers. ....	71

## List of abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ASR	Automatic Speech Recognition
DNN	Deep Neural Network
DL	Deep Learning
GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
SINF	Service d'Informatique HES-SO
Mbps	Megabits per second
ML	Machine Learning
ms	milliseconds
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
RPM	Requests Per Minute
STT	Speech-to-Text
TTS	Text-to-Speech
UI	User Interface
UX	User Experience
W-A	Weighted Average
WER	Word Error Rate

## 1. Introduction

In a world dominated by technology and artificial intelligence (AI), chatbots have taken on a significant role. They provide an efficient communication platform and optimize the user experience (UX) by enabling people to interact with automated systems in a natural and conversational way. Chatbots have gained substantial popularity in recent years, especially in automated customer support. Nevertheless, they face the challenge of relying on text input, which can be limiting for certain users. To optimize this interaction, this bachelor thesis is dedicated to developing an automated chatbot system using speech recognition and synthesis technologies. The goal is to develop a user-friendly chatbot that is accessible via mobile applications. Special attention is given to the speech-to-text (STT) and text-to-speech (TTS) components to enable smooth communication between the user and the chatbot. The continuous evolution of speech technologies and the increasing use of mobile applications provide new opportunities for chatbot integration. This work explores this synergy and develops a practical application that allows users to operate the chatbot via mobile devices using only their voice. A virtual avatar with synchronized lip movements is integrated into the mobile application to give the chatbot a human-like feel and create an engaging and personal UX.

The structure of this bachelor thesis covers different areas, starting with a theoretical basis and the current state of research in the field of speech technologies and chatbots. Then, detailed comparisons of different STT and TTS application programming interfaces (APIs) are performed to identify the most suitable technologies for the chatbot. The implementation and integration of the chatbot into a mobile application is described, followed by a detailed presentation of the user testing results and possible approaches for further development. The goal of this bachelor thesis is not only to develop a working prototype but also to gain valuable insights into the effectiveness of speech technologies.

The bachelor thesis used various approaches to plan and implement the project effectively. First, an overall plan for the entire work was created. This planning can be viewed in Appendix 23.1. In addition, a Product Backlog was created in which all tasks to be completed were recorded. The Product Backlog served as a central list to get a clear overview of the

requirements and goals of the project. In addition, a daily work journal was kept in which all steps performed per day were documented. The work journal proved extremely useful in ensuring that the project stayed on schedule and that no tasks were overlooked. The exact number of hours per activity can be found in Appendix 23.2. Additionally, a Kanban Board was created for the implementation portion of the project. The Kanban Board helped to visualize the workflow and divide the tasks into different phases. This facilitated the prioritization of tasks and made it possible to focus on the most critical and urgent tasks.

The combination of the product backlog, the work journal, and the Kanban board proved highly effective in moving the project forward in a structured and focused way. The clear planning and ongoing monitoring helped maintain an overview of the project's progress and react promptly to challenges as they arose. The methods proved to be valuable tools for completing the project and achieving the goals set.

In this work, ChatGPT has been used. However, it is important to emphasize that ChatGPT was used wisely. Due to the limited data base and lack of awareness, responses were constantly questioned critically. Using it responsibly made it possible to realize this AI's full potential. ChatGPT was used throughout the bachelor's thesis to convert written text into scientific language. In all other uses of ChatGPT, the corresponding prompt was always presented grayed out. The text conversion prompt was:

*rephrase the following text (here is the text) into scientific language*

## 2. State of the Art

This section explains the basic principles of STT and TTS technologies and highlights their importance for chatbots. To gain a more comprehensive understanding of the underlying mechanisms of STT, ChatGPT, and TTS, this chapter also briefly introduces the basic concepts of natural language processing (NLP). NLP is increasingly important in developing chatbots and AI technologies (Markowitz, 2022). NLP techniques are used to create, analyze, improve, and interpret text (Lutkevich & Burns, 2023). Further explanation on this topic can be found in the following chapter.

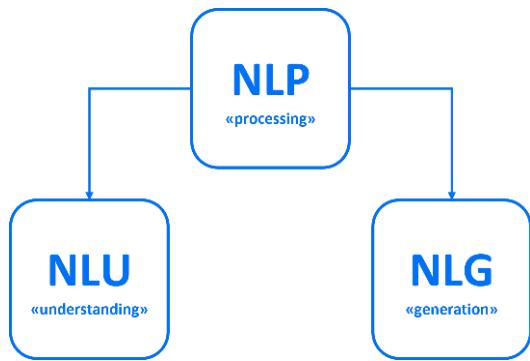
### 2.1 Natural Language Processing

NLP is a subfield of AI that deals with the processing and analysis of natural language (Hyperscience, 2023) and has seen significant growth in recent years due to improved access to large datasets and the emergence of chatbots (Patwardhan, Marrone, & Sansone, 2023). Natural language is the language humans use in everyday speech, such as German, French, or English. These languages evolve from generation to generation and are difficult to capture by strict rules. This presents a challenge for computers because they have difficulty understanding these natural languages (Bird, Klein & Loper, 2009). Therefore, NLP aims to give machines the ability to understand and produce human language in written and spoken forms. NLP uses machine learning (ML) algorithms and deep learning (DL) models to analyze text data and recognize patterns (DeepLearning.AI, 2023; IBM, n.d.-a; Pruneski et al., 2022).

As shown in Figure 1, NLP can be divided into natural language understanding (NLU) and natural language generation (NLG) (Kavlakoglu, 2020).

**Figure 1**

Abstract representation of the NLP components



Source: Own illustration. Adapted from Kavlakoglu (2020).

The process of NLU aims to understand the intention behind the human language. This comprehension process includes tasks such as semantic and syntactic analysis of texts and language to determine the meaning of a sentence. Semantics refers to the meaning of a sentence, while syntax focuses on the grammatical structure of a sentence (Kavlakoglu, 2020). The main goal of NLU is to be able to understand the user's spoken or written text. In contrast, NLG produces text humans can understand (Khurana, Koli, Khatter, & Singh, 2022).

Nowadays, Natural Language Processing (NLP) is used in a wide range of applications. These applications include machine translation, automatic text summarization, and sentiment analysis of texts. Moreover, NLP is also used in the technologies investigated in this bachelor thesis (STT and TTS), about which more detailed information is provided in the following sections (Kavlakoglu, 2020). Furthermore, NLP serves mainly as the basis for modern chatbots (Bhashkar, 2021). Further elaboration on chatbots and NLP is available in the following chapter.

Overall, NLP, NLU, and NLG represent an essential subfield of AI. They improve communication between humans and machines and open new possibilities for efficient data analysis and information processing (DeepLearning.AI, 2023). Through the continuous development of NLP technologies, machines can increasingly better understand human language and enable more natural interactions.

## 2.2 Chatbots

The following two sections were written exclusively by ChatGPT. The output of ChatGPT has undergone additional revision and critical review to ensure that the output generated appears reliable in terms of content. The ChatGPT prompt used can be found before each paragraph.

### 2.2.1 What is a chatbot?

*What is a chatbot? Short explanation.*

A chatbot is a computer-controlled program designed to interact with humans in a way that resembles a natural human conversation. A chatbot can be used on various platforms, such as websites, messaging apps, or social media. The way a chatbot works is usually based on AI and ML. A chatbot can analyze, understand, and respond to user input using algorithms and pattern recognition. It can answer questions, provide information, perform tasks, or help users solve problems. Chatbots are used in many fields, including customer service, marketing, e-commerce, healthcare, and more. They can help increase efficiency, improve the user experience, and help businesses respond to inquiries 24/7.

### 2.2.2 What is ChatGPT?

*What is ChatGPT? Short explanation.*

A prominent example of a chatbot in the modern era is ChatGPT, which is based on the GPT-3.5 or, more recently, the GPT-4 architecture and developed by OpenAI. GPT stands for Generative Pre-trained Transformer. GPT represents a complex algorithm for language models based on ML and deep neural network (DNN) architecture. ChatGPT can answer complex queries, conduct dialogs, generate texts, and support users in various fields of expertise. It can also capture contextual information and respond to previous messages to provide a continuous conversation flow.

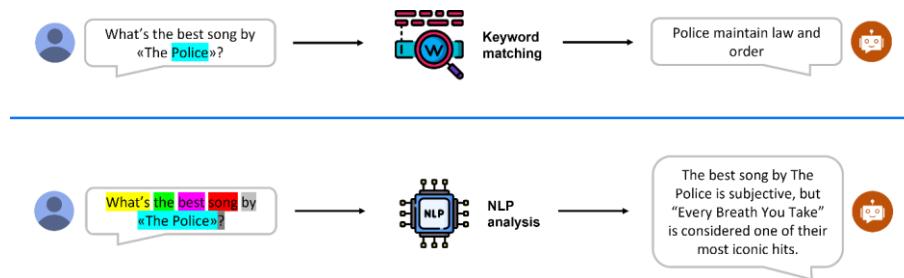
### 2.2.3 Today's Chatbots

In today's world, NLP plays a crucial role in developing chatbots. It is used to provide the most human-like conversation possible. The rapid growth in the field of NLP has led to significant advances in chatbots (Shrivastav, 2023). Unlike traditional chatbots that rely on

simple patterns, keyword matching, and rules to respond to the user, modern AI chatbots based on NLP can understand the direct intention of the user (Amyx, 2023). This difference can be illustrated with a simple example. If you ask a traditional chatbot about the meaning of "*the police*", it only recognizes the keyword "*police*" and explains the police. However, if additionally asked for the best song by "*The Police*", the traditional chatbot might continue to respond with an explanation about the police. This approach has a high potential for error because the traditional chatbot does not really understand the user's actual query (Rajnerowicz, 2023). In contrast, a modern NLP-based chatbot can more accurately interpret the user's query and thus provide more precise and higher-quality answers. Figure 2 illustrates the example again.

**Figure 2**

Traditional chatbot vs. modern AI chatbot



Source: Own illustration. Adapted from Rajnerowicz (2023).

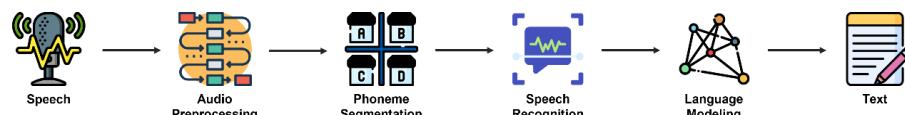
## 2.3 Speech-to-Text Technology

STT technology, or automatic speech recognition (ASR), enables the conversion of spoken human language into written text. This process is realized with the help of various ML and NLP algorithms (Rella, 2023). The distinction between speech recognition and voice recognition is of great importance due to their different technological nature. Voice recognition primarily concerns recognizing and authenticating a particular user based on their unique vocal characteristics. In contrast, speech recognition focuses on converting spoken language into written text (IBM, n.d.-b).

Figure 3 illustrates an abstract version of the STT process. This process begins with recording a person's spoken sounds with a microphone or other audio recording device. This recording is done in analog form, which is then converted to a digital file. The analog-to-digital

conversion plays a key role, as it enables further processing of the audio signal (Simon Says, n.d.).

**Figure 3**  
Abstract STT process



Source: Own illustration.

Second, the audio file undergoes pre-processing, which includes several steps, such as eliminating background noise, normalizing sound frequencies, and improving voice signals. Pre-processing is essential to optimize the quality and comprehensibility of the audio data and, therefore, to increase the precision of speech recognition (Chapuzet, 2022).

Third, after pre-processing, the analyzed audio file is submitted for a detailed analysis. The file is segmented into smaller units called phonemes (Chapuzet, 2022). Phonemes represent a language's most minor meaning-differentiating sound components and serve as building blocks for words (Bathurst, 2020). For example, the phoneme /b/ in the word "bat" can be replaced by the phoneme /p/, which causes the word meaning to change from "bat" to "pat" (Shaywitz, Weiss, Saklofske, & Shaywitz, 2016). It is essential to point out that the number of phonemes varies according to language and dialect (iMerit Technology Services, 2022). For example, British English contains 44 phonemes (McAvan, 2023). Standard French, as spoken near Paris, has about 35 phonemes, according to Lewy (2015).

Fourth and fifth, various modern ML algorithms analyze phonemes and consider a wide range of information, including acoustic features and contextual cues such as prosody and speech patterns. Innovative techniques like NLP have revolutionized the accuracy and performance of converting phoneme sequences into meaningful text (Rella, 2023; Simon Says, n.d.). Ultimately, the output is most likely provided as transcribed text.

### 2.3.1 Current State of STT Technology

Due to the advancing development of AI, STT technologies have made significant progress and use more and more AI techniques. NLP is today already being used in STT recognition,

---

especially to improve the accuracy of text output (Rella, 2023; Simon Says, n.d.). A significant aspect of NLP in STT is the modeling of accents and dialects, as different speech variants can have different pronunciations, emphases, and nuances (DeepLearning.AI, 2023). Thanks to rapid advances in NLP, even greater integration with STT technologies is expected. This will make STT technologies smarter and more applicable to various applications. Such integration can, for example, extensively increase the performance of direct machine translation. Specifically, this means that spoken text can be automatically translated into another language in real-time. A promising example of this technology is the integration of the machine translation function in Skype, which is already being successfully implemented today (Skype, n.d.).

Nevertheless, NLP is being applied in other diverse STT applications, including the implementation in virtual assistants such as Siri or Alexa (Sanghvi, 2023). NLP in these systems facilitates a more precise conversion of spoken language into written text and enables direct comprehension and subsequent generation of responses based on the translated text. This implies that Virtual Assistants possess the potential to enhance their capabilities and accuracy in forthcoming advancements (DeepLearning.AI, 2023; Firth-Butterfield & Ammanath, 2021).

### **2.3.2 Advantages of using STT concerning chatbots**

A chatbot equipped with speech recognition has several notable advantages. First, users can have a conversation with such a chatbot that closely resembles human communication. This creates a more spontaneous and authentic form of interaction than written communication. Second, the integration of speech recognition improves the usability and user experience of the chatbot by allowing users to communicate verbally with the system. In addition, this capability expands accessibility for individuals who have difficulty writing or suffer from motor impairments that limit keyboard use. Thanks to STT technology, these individuals can also reap the benefits of a chatbot (Perdigão, 2023).

## **2.4 Text-to-Speech Technology**

TTS, or speech synthesis, is a technology that allows computers to convert written text into spoken language (Bernard, 2023). It's like teaching a computer to read a text out loud. To better

understand how TTS technology works, looking at a typical flow of a TTS system is helpful. The TTS systems use various algorithms and techniques to convert the written text into a spoken voice output (Rashad, El-Bakry, Isma, & Mastorakis, 2010).

**Figure 4**

Abstract TTS process



Source: Own illustration. It was adapted from Zen (n.d.).

Figure 4 shows a typical flow of a TTS system. It can be seen there that, roughly speaking, a TTS system can be divided into two main parts (NLP and Speech) and three phases: conversion from text to words, conversion from word to phoneme, and conversion of phonemes to sounds (Rashad et al., 2010).

Text normalization, conversion from text to words, is the introductory step in TTS. During text normalization, NLP is used to apply various procedures to convert the input text into an adequate form for speech synthesis. The first step involves segmenting the text into sentences. The next step involves standardizing non-standard words, such as numbers, dates, abbreviations, or acronyms. These non-common words are converted into a sequence of natural words to make them correctly pronounceable. However, non-standard words can be ambiguous. For example, "1997" can be pronounced differently depending on the context. Similarly, abbreviations can either be spelled out or pronounced like regular words. An example is that "St." is read as "saint" ("Src\_08. Speech Synthesis," n.d.). A final step in text normalization is homograph disambiguation. Homographs are words that have the same spelling but different pronunciations (Rashad et al., 2010). ChatGPT has created an example of the disambiguation of homographs.

*simple example for the disambiguation of homographs*

The "bass" (fish) swims in the river. The musician played the "bass" (musical instrument) in the band.

The subsequent step after text normalization is determining each word's corresponding

---

phonemes. This involves matching the previously normalized words with an underlying phoneme library. The final text analysis phase involves prosodic analysis, which examines the features that make up the natural flow of sentences. Phrasing, stress, and intonation produce a fluent and natural-sounding phonetic sequence (Rashad et al., 2010).

Third, the conversion of phonemes into phonetic expressions occurs. The phonemes are used to assign sounds to words, while the system can also generate a phonetic expression for each phoneme (Rashad et al., 2010). There are several methods by which the TTS system can generate audio waveforms. In the current era, neural networks are mainly used for audio data generation (Karagiannakos, 2021; Rashad et al., 2010). A detailed study and explanation of this topic is beyond the scope of this paper. Interested readers may refer to Appendix 1 for additional techniques and further information on this topic.

#### **2.4.1 Current State of TTS Technology**

Obviously, existing models are continuously being refined, which means that the naturalness of pronunciation is continuously increasing as AI improvements progress. However, cloning individual voice profiles is becoming increasingly important (Espinosa, 2023). Videos pop up at regular intervals on platforms such as YouTube and other social media in which, for example, the voice of Humor Simpsons is used to explain American history or the voice of Barack Obama to sing a song by Rihanna. The exceptional similarity of the cloning voice to the original is impressive. These technologies are continuously developing, and various websites and tools already exist that can be used to recreate one's own voice virtually - sometimes with better, sometimes with less good results. Until now, however, it has been necessary to have a considerable amount of audio data available to achieve acceptable results.

On January 5, 2023, Microsoft published a new TTS model called VALL-E. VALL-E is based on a neural approach. According to Wang et al. (2023), it is remarkable for its ability to imitate a speaking voice using a simple 3-second audio recording without ignoring the speaker's emotional nuances and acoustic properties. These features could prove highly significant for rare or endangered languages with limited available data. Furthermore, this approach could also assist people who have lost their natural voice by providing a way to reconstruct their speaking voice.

Two months after the release of VALL-E, Microsoft unveiled an evolution of the VALL-E language model called VALL-E X on March 7, 2023. With the help of the VALL E X model, it is now possible to capture a speaker's voice, e.g., in German, and synthesize high-quality speech output from it in a target language, such as English. In doing so, the characteristics of the speaker's voice, the emotional nuances, and the acoustic environment are preserved, as stated by Microsoft in their publication (Zhang et al., 2023).

## **2.4.2 Advantages of using TTS concerning chatbots**

*give me advantages of text-to-speech in chatbots*

Natural interaction and increased user satisfaction are essential for the success of modern chatbots. Implementing TTS technology plays a significant role in this context, leading to a more natural and fluid conversation. This makes the interaction with the chatbot more pleasant, leading to a positive user experience overall. Another benefit of TTS is the time saved for users, as instead of reading lengthy text messages, they can simply have the responses read to them. This time-saving proves especially useful for busy or mobile users. TTS is an indispensable tool in enabling people with visual impairments or reading barriers to efficiently use the chatbot by allowing them to listen to the answers instead of reading them. Thus, TTS is a crucial tool for optimizing the performance and effectiveness of chatbots in diverse application areas.

### 3 Methodology

The main objective of this chapter is to explain in detail the methodology for carrying out a comparative analysis of different STT-APIs and TTS-APIs.

#### 3.1 Methodology of STT

The following chapter provides a comprehensive overview of the selection process of the STT-APIs to be tested, the methods of data collection, and an explanation of how the various tests were conducted.

##### 3.1.1 Overview of the selected STT-APIs

Several reputable STT-APIs were selected to create a meaningful basis for the comparison. The selected APIs are shown in Table 1.

**Table 1**  
STT-APIs and the used version and tariff

Company	Version	Tariff
 OpenAI <sup>1</sup>	Beta	Standard
 DEEPGRAM <sup>2</sup>	v1.0	Deepgram Nova (Pre-recorded)
 AssemblyAI <sup>3</sup>	Not visible	Standard / Trial
 Google Cloud <sup>4</sup>	v1	Standard1
 Microsoft Azure <sup>5</sup>	Version 3.1	Free (F0) - Standard

Note: All data in this table were collected at 16:15 on June 21.

Source: Own representation.

<sup>1</sup> <https://openai.com/brand#usage-terms>

<sup>2</sup> <https://www.wing.vc/companies/deepgram>

<sup>3</sup> <https://pypi.org/project/assemblyai/>

<sup>4</sup> [https://commons.wikimedia.org/wiki/File:Google\\_Cloud\\_logo.svg](https://commons.wikimedia.org/wiki/File:Google_Cloud_logo.svg)

<sup>5</sup> [https://microsoft.fandom.com/wiki/Microsoft\\_Azure](https://microsoft.fandom.com/wiki/Microsoft_Azure)

---

These APIs already had an established reputation in the industry and had a solid foundation. The familiarity and popularity of the APIs were considered first. However, care was also taken to find a balanced mix of established large companies and emerging young companies. The reasons for selecting Deepgram and AssemblyAI are explained below. No further elaboration is provided for Google, Azure, and OpenAI, as they are already established and widely known companies.

Deepgram is an AI company specializing in speech processing. It provides solutions for converting and interpreting spoken language into text (Deepgram, n.d.). Deepgram claims to be the most powerful STT API in the world, and this claim should be verified. For this reason, Deepgram was selected.

The AI-as-a-service provider, AssemblyAI, has received \$30 million in a 2022 funding round. The company has already raised more than \$63 million in investments (AssemblyAI, n.d.). The company provides TTS conversion and text analytics services. It has experienced significant growth and has doubled its team size by the end of 2022 (Wiggers, 2022). AssemblyAI's goal is to establish itself as a leader in AI-as-a-Service, which is why it was included in the selection of APIs to be tested.

### **3.1.2 Data Acquisition of STT**

As part of the comparative analysis of the STT-APIs, specially created test data was used to evaluate the language support of the APIs and their ability to accurately transcribe languages, dialects, accents, and speaker voices. Ten test persons were selected for each German and English language to ensure a representative data base. Different language backgrounds, genders, and age groups were carefully considered in selecting test persons to ensure a diverse group of participants. Each test person recorded ten predetermined questions and added one spontaneous question. The questions were formulated using ChatGPT to cover various topics and address relevant requirements and scenarios for chatbots. A detailed list of the questions can be found in Appendix 2. The prompt used to generate the questions was as follows:

*Create ten questions I can ask a speech-to-text chatbot to test speech-to-text accuracy. Short and long questions. Type: technology, science, general knowledge, entertainment, and unconventional questions. In the two languages: German and English.*

---

The questions covered various domains such as technology, science, general knowledge, and entertainment. Unconventional questions were also asked to test the versatility of the speech recognition capabilities of the APIs.

Afterward, the test persons used a mobile application developed specifically for this purpose. The visual look of the application is presented in Appendix 3. This app allowed them to make recordings using their smartphone's microphone, which corresponded to the later conditions when using the chatbot. This mobile application created the same level playing field for all test persons and recorded the data in the same format. Before the start of the recordings, test persons were informed of the purpose of the recordings and were given detailed instructions on pronunciation, intonation, and articulation. These standardized instructions and guidelines ensured the audio recordings were meaningful and comparable to the STT-API assessment process. The guidelines given can be found in Appendix 4.

The available data for the cost comparison analysis of the STT-API providers was obtained from the HES-SO's IT service (SINF). The decision to use the SINF as a data source was made based on the potential use of the SINF as a user of an automated chatbot to create a realistic scenario for the cost calculation. However, it should be noted that this data is not directly applicable to any customer support or help desk. The SINF provided the data as a line graph diagram representing the weekly number of requests to the service. The corresponding chart can be viewed in Appendix 5.1. The data collected covered the period from the week of 13.02.2023 to the week of 12.06.2023. Because the semester did not begin until the week of 27.02.2023, only data from that week onward was considered. The chart in Appendix 5.1 shows that the approximate average of weekly requests at SINF is about 80 requests per week. This corresponded to about 240 requests per month at SINF. The average query time was also needed for the cost comparison analysis. This was determined using the rounded average value of all test persons' recordings.

### **3.1.3 Evaluation Criteria of STT**

Various criteria were considered to provide a comprehensive assessment of the STT-APIs. The evaluated criteria included accuracy, performance, cost, usability, and documentation quality. The following sections are organized according to these evaluation criteria.

### 3.1.3.1 Accuracy

Accuracy was evaluated using the Word-Error-Rate (WER) test to provide an objective comparison. The WER test is a common evaluation method in ASR that quantifies the degree of precision of the transcription process (Ali & Renals, 2018). It determines the percentage of misrecognized words compared to the reference transcription. The WER formula can be seen in Figure 5. A low WER value indicates a higher precision of speech recognition by the considered API. This approach allowed an objective comparison of the accuracy of different APIs and facilitated an informed decision for the most appropriate API.

**Figure 5**  
WER formula

$$\text{WER} = \frac{S + D + I}{N}$$

S: number of substitutions  
 D: number of deletions  
 I: number of insertions  
 N: number of words in the reference

Source: Own illustration. It was adapted from Formula of the Word Error Rate (n.d.).

To check the precision, the results of the test persons from the previous chapter, 3.1.2, were compared with the reference transcripts from Appendix 2 to calculate the WER value. For this purpose, a Python script was used, shown in more detail in Appendix 6. The script allowed automatic comparison of the transcripts and reference transcripts to determine the average WER score for each speech recognition API and transcription. The results of each test person were recorded separately in files that provided a comprehensive overview of the test runs and the WER values of the different transcripts and speech recognition APIs. The results of the files were then imported into Excel to perform a comprehensive calculation. This process enabled a quick comparison of the precision of the APIs and facilitated the informed selection of the most appropriate API.

The WER values were visualized using a bar chart, allowing quick identification of differences between the categories and conditions. The clear visual representation helped improve understanding of the results and facilitated the interpretation of the data.

### **3.1.3.2 Performance**

The performance tests were all analyzed under comparable network conditions, focusing on evaluating processing speed and scalability. JMeter version 5.5 was used to measure these specific aspects. JMeter is a widely used performance and load-testing tool that provides extensive user behavior simulation capabilities to evaluate the performance of applications, systems, or services (Apache JMeter™, n.d.).

#### ***Processing speed***

Three test runs were performed with different audio lengths (8 seconds, 25 seconds, 52 seconds) for each STT-API. The measured data was recorded in milliseconds. A simulation of a single user accessing the API was performed. Twenty accesses to each specific STT-API were performed to obtain valid results. The test was conducted over two days to ensure that random influences did not affect the tested APIs.

Microsoft Excel was used to analyze the collected data. First, the data was imported from JMeter. Then, a spreadsheet was created that contained the summary results of the STT APIs. A bar graph was created to visually display the speed of all APIs for the three different audio lengths. This chart allowed a quick comparison of the speeds between the different APIs. This chart allowed for a quick comparison of the speeds between the different APIs. Additionally, a line graph was generated for each test run that represented the values of all twenty calls made. Potential variations and trends over time could be identified by capturing all recorded values. This enabled detailed analysis and identified performance fluctuations or long-term trends in processing speed.

#### ***Scalability***

This analysis evaluated the scalability of the APIs in terms of their capacity to process audio data efficiently under increased load. Another focus was determining the point at which the STT-APIs generate error messages. Tests were performed with 100 and 1000 requests to the STT-APIs. To simulate a realistic test environment, a random wait time was implemented in JMeter that ranged from 2 to 3 seconds for each call.

---

It should be noted that in the data analysis for the 1000 calls, the extreme outliers in the 100 test calls were not included. This methodology was chosen to optimize resources and ensure that only data from representative APIs were used. Nevertheless, if there were unusual discrepancies in the success rate for the 1000 calls, it was decided to investigate these outliers in detail.

A combined bar and line chart was used to plot the first 100 calls to show both the success rate and the response time. By choosing this display format, both aspects of API performance could be visualized in a single chart. The success rate was calculated as a percentage of successful calls, while the response time represented the average time required in milliseconds for a successful call. Only a bar chart was used to visualize the following 1000 calls to the APIs for data collection. In this bar chart, the success rate was represented as a percentage. The deviations at 1000 accesses were analyzed using a line graph to determine the point at which error messages occurred.

It is important to note that the measured times for speed and scalability were averages, and actual speeds could vary depending on specific text content, network conditions, and other factors.

### **3.1.3.3 Cost**

The costs of the various STT-APIs were determined by analyzing the pricing models and rate structures. The used pricing models and rate structures have been listed in Table 1. It should be noted that only the pricing models and rate structures of the tested APIs were compared.

To provide a basis for calculating the costs, the data mentioned in Chapter 3.1.2 was used. The number of monthly requests received by the SINF was 240, and these requests were multiplied by the product of the average recording time of 7 seconds and the cost per request in seconds. Furthermore, this value was multiplied by the average number of 10 requests a user can make to a chatbot. Figure 6 illustrates the formula described earlier.

**Figure 6**

The formula for the calculation of STT costs

$$\text{Costs per month} = (R * ((V * A) * C))$$

R:	Average of Monthly requests
V:	Average length of Voice recordings
A:	API costs per request in seconds
C:	Average number of requests in a chat from a user

Source: Own illustration.

Based on this, a cost calculation was created to determine the estimated monthly expenses for each STT-API provider. This method of cost evaluation allowed for more informed decision-making regarding the cost-effectiveness of the APIs and contributed to the selection of the most cost-effective option that would meet the needs of the chatbot.

The results of the cost evaluation were presented in tabular form to allow for a straightforward cost comparison between the different APIs. Both total cost and cost per minute of transcription were listed.

#### **3.1.3.4 Usability and documentation quality**

The APIs were evaluated in terms of their usability and the quality of the documentation provided. It was essential to determine how easily the APIs could be integrated into existing systems and how well the information provided helped developers use them. Since the quality of usability and documentation was not directly measurable, a rating and measurement system was developed to quantify usability and documentation quality. Several aspects were considered in the evaluation of the project. The rating scale of Table 2 was used to evaluate each aspect. The rating scale was created using the ChatGPT prompt:

*Please provide a rating for the following criteria in relation to API documentation. Consider the availability of developer documentation, sample code, and tutorials, the well-structured and concise nature of the API documentation, the support and accessibility of technical support, and the timeliness and completeness of the information provided. Use a scale of 1 to 10, with 1 being the lowest rating and 10 being the highest. Once you have rated each criterion, please specify the weight you would assign to each criterion in the overall evaluation.*

**Table 2**

Rating scale of usability and documentation quality

Evaluation criteria	Evaluation criterion in detail	Weight
<b>Availability of developer documentation, sample code, and tutorials</b>	1: Very limited or no resources available	30%
	5: Sufficient resources but room for improvement	
	10: Extensive and high-quality resources available that facilitate integration	
<b>Well-structured and concise API documentation</b>	1: Unstructured and challenging to understand	25%
	5: Averagegly structured and concise	
	10: Very well structured and understandable	
<b>Support and accessibility of technical support</b>	1: No or minimal support options	20%
	5: Some support options are available, but room for improvement	
	10: Wide range of support options, quickly accessible and helpful	
<b>Timeliness and completeness of the information provided</b>	1: Outdated or incomplete information	25%
	5: Some updates and gaps in the information provided	
	10: Regular updates and comprehensive information available	

Source: Own representation. Created by ChatGPT.

The ratings were recorded using an Excel spreadsheet. To illustrate the results, a tabular representation was created for each aspect to ensure a precise overview of the ratings. Finally, an aggregated overall rating of all aspects was calculated. The overall rating was determined by applying a weighted average (W-A) method to all aspects. This made it possible to determine which API was to be classified as outstanding in this topic area.

### 3.2 Methodology of TTS

The following chapter presents a comprehensive description of the selection process for the TTS-APIs to be evaluated and the methods used for data collection. Furthermore, it explains how the various tests were carried out.

### 3.2.1 Overview of the selected TTS-APIs

The selected TTS-APIs were carefully chosen to ensure a balanced mix of established companies and emerging start-ups. A listing of the selected TTS-APIs can be found in Table 3.

**Table 3**

TTS-APIs and the used version and tariff

Company	Version	Tariff
<b>ElevenLabs</b> <sup>1</sup>	v1	free
 IBM Watson <sup>2</sup>	v1.0	Version from 31.03.2023
 Google Cloud <sup>3</sup>	v1	Neural2 voices
 Play.ht <sup>4</sup>	V2.0	free
 Microsoft Azure <sup>5</sup>	Version 3.1	Free (F0) - Neural

Note: All data in this table were collected at 15:24 on July 4.

Source: Own representation.

<sup>1</sup> <https://elevenlabs.io/blog/>

<sup>2</sup> <https://zevector.com/ibm-watson-logo/>

<sup>3</sup> [https://commons.wikimedia.org/wiki/File:Google\\_Cloud\\_logo.svg](https://commons.wikimedia.org/wiki/File:Google_Cloud_logo.svg)

<sup>4</sup> <https://brandfetch.com/play.ht>

<sup>5</sup> [https://microsoft.fandom.com/wiki/Microsoft\\_Azure](https://microsoft.fandom.com/wiki/Microsoft_Azure)

In the following, the companies ElevenLabs and play.ht are presented in more detail, and reasons are given why exactly these two companies were selected. There was no further explanation for Google, Azure, and IBM Watson as they were already established companies.

ElevenLabs is an American company that develops TTS software using AI and DL. It was founded in 2022 by Piotr Dabkowski, a former Google machine learning engineer, and Mati Staniszewski, a former operations strategist at Palantir (ElevenLabs, n.d.). Although a comparatively young company, it is experiencing remarkable growth in the speech synthesis space. In 2023, the company plans to raise a funding round of \$18 million at a valuation of \$100 million (Kanetkar, 2023). Based on this positive trend, ElevenLabs API was included in the selection.

Play.ht is a remote startup that provides high-quality TTS synthesis and audio accessibility solutions. Originally launched as a Chrome extension, the company specializes in providing audio content for applications. Today, it helps major companies develop TTS solutions (PlayHT, n.d.). By working with well-known, established companies such as Amazon, PwC, and Volkswagen, play.ht has been included in the selection of APIs.

### **3.2.2 Data Acquisition of TTS**

To create a comprehensive collection of reference texts in multiple languages, English, German, and French were generated using ChatGPT. The prompt used was:

*generate three texts with different lengths of exact max\_tokens=50, max\_tokens=100, and max\_tokens=200. the texts should also be generated in three languages English, German, and French.*

These reference texts were used to evaluate the voice quality of different TTS APIs. Texts of different lengths were created for their future use in a chatbot. The parameters "max\_tokens" were set to 50, 100, and 200, as these are the maximum lengths the chatbot will work with later. Notably, the number of tokens was determined based on the GPT-3.5-Turbo model.

### **3.2.3 Evaluation Criteria of TTS**

To enable a complete evaluation of the TTS-APIs, diverse criteria were considered. The evaluation criteria included acoustic quality of voice, customization options, language coverage, performance, and cost structure. The sections structured according to these evaluation criteria are presented below.

#### **3.2.3.1 Voice quality**

Voice quality is significant in evaluating TTS-API's performance and directly impacts the UX. This study's primary approach to evaluating voice quality was generating and comparing audio samples. For this purpose, samples from the previously created reference texts in Chapter 3.2.2 were converted into speech outputs using a Python script in Appendix 7. The generated audio files were evaluated based on intelligibility and naturalness. Ten subjects were used to evaluate the speech quality to ensure the most subjective assessment possible. These subjects listened to the generated speech outputs of the various TTS-APIs and rated them using a

---

predetermined scale for each criterion. The detailed scales can be seen in Appendix 8.1 and 8.2. The voices used for each API were randomly selected. Voices were selected for English, French, and German, with a male and female voice selected for each language.

The results obtained were then analyzed using Excel. It is important to note that the data were analyzed specifically for the two categories for both genders and the languages used. The overall results for each category, divided by gender, were presented in a table, while bar charts provided a better overview of each category.

#### ***3.2.3.2 Customization options***

To evaluate the customizability of the TTS-APIs, an in-depth analysis of the API documentation was conducted to identify existing customization options for linguistic features such as speech rate, pitch, and stress. Subsequently, these customization options were tabulated per API to ensure a comprehensive overview of the respective possibilities.

#### ***3.2.3.3 Language coverage***

A comprehensive analysis of the languages supported by different TTS-APIs was performed to evaluate the language coverage. For this purpose, the official API documentation was consulted to identify information about the available languages. The number of available languages per API was then tabulated.

#### ***3.2.3.4 Performance***

The JMeter tool was used to quantify the response time of the various TTS-APIs. The reference text defined in Chapter 3.2.2 was used as the basis for the evaluation. Only the text with 200 tokens in English was used since the time for the most extended text length was decisive. The assumption was that if longer texts could be converted quickly, this was also true for shorter texts. Each text was sent individually to each API, and the time for language conversion was measured. This process was repeated 20 times to determine average values and account for possible variations. Times were measured in milliseconds. The collected data were processed in Excel and visualized in a bar chart.

It is important to note that the times measured were averages, and actual speeds could vary

depending on specific text content, network conditions, and other factors.

### **3.2.3.5 Costs**

The cost analysis of the TTS-APIs was performed using the same data base used in Chapter 3.1.2. Using the formula shown in Figure 7, the costs for each API were determined. The selected pricing models and rate structures are listed in Table 3. Monthly costs were determined based on approximately 240 requests per month to the system. These 240 requests per month were multiplied by the product of the assumed average character length of approximately 200 characters per response and the average number of 10 requests per user in a chat. If applicable, this product was reduced by the number of free characters per month and multiplied by the specific API's cost. This provided a realistic estimate of the monthly cost for the named API. The costs were presented in tabular form to allow for quick recording of the corresponding prices. Costs were presented per character and the number of free characters per month.

**Figure 7**

The formula for the calculation of TTS costs

$$\text{Costs per month} = ((R * V * C) - P) * A$$

R:	Average of Monthly requests
V:	Average number of characters per response
P:	Potential free number of characters per month
A:	API costs per character
C:	Average number of requests in a chat from a user

Source: Own illustration.

## 4 Results

This chapter presents the evaluation results of the STT and TTS APIs. The results were illustrated in two primary sections: STT and TTS. These research results provide insights into the merits and limitations of the different APIs and consequently form the basis for selecting the adequate API to implement the chatbot in this bachelor thesis.

### 4.1 Results of Speech-to-text

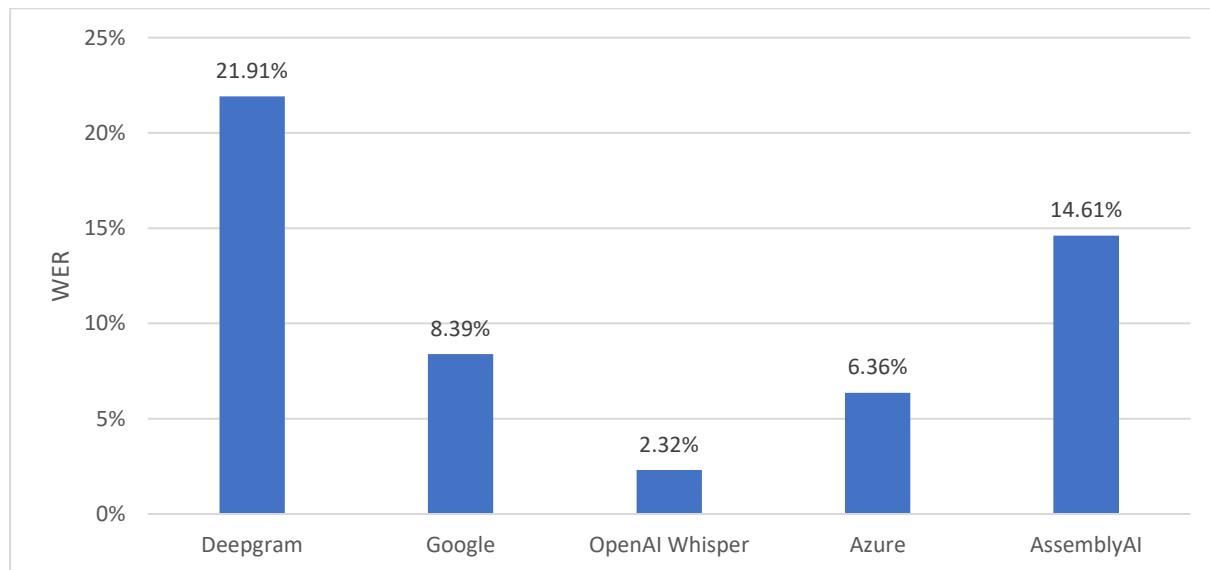
This section presents the results of the STT-API comparison.

#### 4.1.1 Results Accuracy

Figure 8 illustrates remarkable variations in WER among different speech recognition APIs. Among the tested APIs, Deepgram exhibited the highest WER, indicating that Deepgram had comparatively low accuracy in converting spoken language to text in this bachelor thesis. Whisper achieved the best WER, indicating an exceptionally accurate speech recognition capability.

**Figure 8**

Overview of the average WER of the STT-APIs



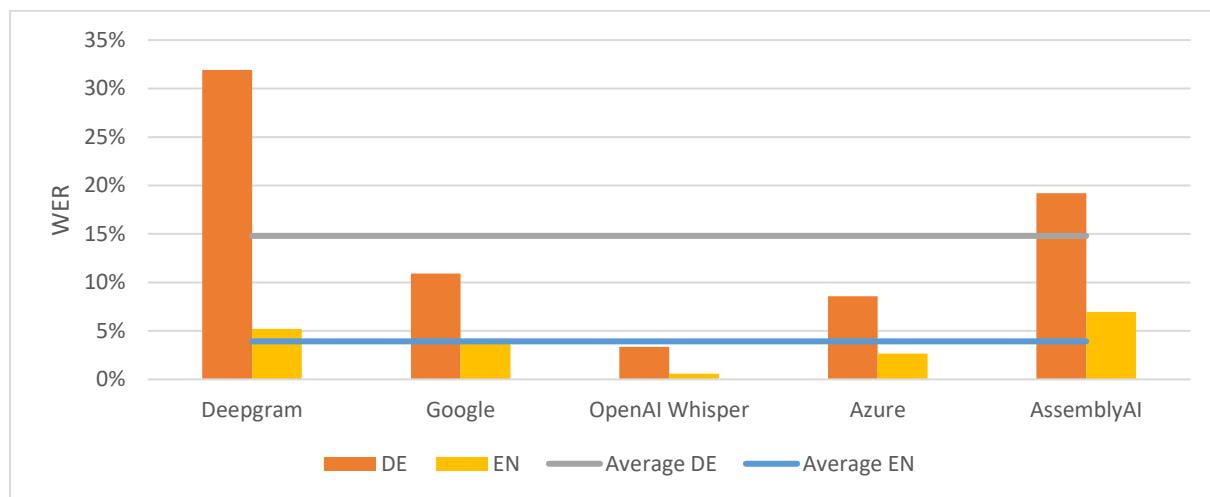
Source: Own illustration. Created with Excel.

A detailed analysis of Figure 9 showed that the accuracy of the STT-APIs researched was

generally lower for German than for English. The WERs showed that all APIs had a higher frequency of errors in German than in English. WER values for German ranged from 3.3% to 31.9%, while WER values for English ranged from 0.58% to 6.9%. The average WER was 14.8% for German and 3.92% for English.

**Figure 9**

Overview of the average WER per language and API



Source: Own illustration. Created with Excel.

#### 4.1.2 Results performance

Table 4 lists the average measured response time of all tested STT-APIs for the corresponding duration of the audio material. A comprehensive examination of the results can be found in Appendix 9.

**Table 4**

Overview of the average speed values for the respective audio lengths

API	8 sec	25 sec	52 sec
<b>Deepgram</b>	1435.30 ms	1568.50 ms	1822.05 ms
<b>Google Speech-to-Text</b>	2016.70 ms	7444.75 ms	16231.00 ms
<b>Microsoft Azure Speech-to-Text</b>	2636.60 ms	10748.10 ms	8317.40 ms
<b>OpenAI Whisper</b>	1508.60 ms	2634.35 ms	3757.15 ms
<b>AssemblyAI</b>	9620.85 ms	11165.00 ms	22256.00 ms

Note: The averaged network parameters across all tests were 276.51 Mbps for download and 235.51 Mbps for upload. The best value for each category is marked in green.

Source: Own representation.

In Table 5, the success rate and response time for 100 requests to the API can be found. The success rate refers to the percentage of successful requests made without errors or failures. A more detailed analysis of the scalability of the different STT APIs can be found in Appendix 10.

**Table 5**

Overview of the load testing for 100 requests

API	8 sec		25 sec	
	Success rate	Ø Response time	Success rate	Ø Response time
<b>Deepgram</b>	100%	1148 ms	82%	1509.15 ms
<b>Google STT</b>	100%	2064.9 ms	88%	6726.49 ms
<b>Microsoft Azure STT</b>	19%	25904.05 ms	2%	13179.62 ms
<b>OpenAI Whisper</b>	93%	1338.58 ms	93%	2533.84 ms
<b>AssemblyAI</b>	100%	171296.92 ms	100%	183939.01 ms

Note: The download speed was 302.45 Mbps, and the upload speed was 271.81 Mbps.

Source: Own representation.

#### 4.1.3 Results costs

All models were offered on a pay-per-use basis. The price information regarding the corresponding API can be seen in Table 6. Table 6 also presents the monthly expenses calculated using the formula detailed in section 3.1.3.3. From the data shown, it was clear that Deepgram and Whisper were the most cost-effective APIs based on the tariffs used.

**Table 6**

Costs for each STT-API

API	Costs per minute (CHF)	Costs per month (CHF)
<b>Deepgram</b>	0.0040	1.12
<b>Google STT</b>	0.022	6.16
<b>Microsoft Azure STT</b>	0.015	4.20
<b>OpenAI Whisper</b>	0.0055	1.54
<b>AssemblyAI</b>	0.014	3.92

Note: All documentation in this table was last accessed no later than May 31 at 15:25.

Source: Own representation.

#### 4.1.4 Results usability and documentation quality

A detailed presentation of the results and corresponding scoring comments is available in Appendix 12. Table 7 contains the final overall score for each API. This table has illustrated that Deepgram and OpenAI APIs have the best results in terms of documentation provided.

**Table 7**

Weighted total score for the respective STT-API

API	Points (W-A)
<b>Deepgram</b>	8.4
<b>Google Speech-to-Text</b>	6.1
<b>Microsoft Azure Speech-to-Text</b>	5.6
<b>OpenAI Whisper</b>	7.25
<b>AssemblyAI</b>	7.2

Source: Own representation.

#### 4.1.5 Overall results of STT

Table 8 presents in summary form the overall results of the STT-APIs. Based on these results, the preferred STT-API was selected. A comprehensive explanation of the justification for this decision can be found in Chapter 5.1.

**Table 8**

Total overview of all STT-API results

Evaluation Criteria		Deep-gram	Google STT	Azure STT	OpenAI Whisper	Assembly
<b>Accuracy (WER-Rate)</b>		21.91%	8.39%	6.36%	2.32%	14.61%
<b>Performance (ms)</b>	6 sec	1435.30	2016.70	2636.60	1508.60	9620.85
	25 sec	1568.50	7444.75	10748.10	2634.35	11165.00
	52 sec	1822.05	16231.00	8317.40	3757.15	22256.00
<b>Costs per month</b>		CHF 1.12	CHF 6.16	CHF 4.20	CHF 1.54	CHF 3.92
<b>Usability</b>		8.4	6.1	5.6	7.25	7.2

Note: The two best results for each category are highlighted in green using the color scheme.

Source: Own representation.

## 4.2 Results of Text-to-Speech

This section presents the results of the TTS-API comparison.

### 4.2.1 Results voice quality

Table 9 shows that Microsoft Azure TTS scored the highest Intelligibility and Naturalness scores for both male and female voices. IBM Watson TTS and ElevenLabs were also close in most categories, with ElevenLabs scoring slightly better regarding naturalness for female voices. It should be noted that ElevenLabs had only provided the English language for the speech quality assessment. This limited language coverage impacted the comparability of the results, as the other TTS-APIs were also tested in German and French. Google TTS and Play.ht scored the lowest overall on both dimensions. Notably, for 4 out of 5 APIs, female voices were perceived as less natural than male voices. This was consistent for each language tested. The corresponding diagrams can be viewed in Appendix 13.

**Table 9**

Overview of the two categories, Intelligibility and Naturalness

API	Intelligibility		Naturalness	
	Man	Woman	Man	Woman
<b>IBM Watson TTS</b>	5.4	5.3	5.3	5.2
<b>Google TTS</b>	5.3	5.1	3.3	2.6
<b>Microsoft Azure TTS</b>	5.6	5.6	5.4	5.3
<b>ElevenLabs</b>	5.2	5.4	3.8	4.5
<b>Play.ht</b>	5.0	4.4	2.9	2.3

Note: The best result per category is in green

Source: Own representation.

### 4.2.2 Results Customization options

Detailed information about the various API configurations of the TTS APIs can be viewed in Appendix 14. These configurations offered extensive customization options regarding the generated speech output. Parameters such as the speech rate, pitch, volume, audio encoding, and sampling rate were available for all APIs. In addition, some APIs offered advanced features, such as the generation of Visemes, which could be of great relevance for the later animation

of avatars. This was particularly possible with the Azure API. Both Azure and Google allowed the use of SSML to make more precise adjustments to the voice output.

#### 4.2.3 Results language coverage

This section shows the language support of different TTS APIs, including different dialects. The results obtained are documented in Table 10. Each API offered a variable number of supported languages and dialects, resulting in differences in terms of their diversity and application breadth.

**Table 10**

Number of languages (including dialects) supported by various TTS-APIs

API	Number of languages
IBM Watson Text-to-Speech	14
Google Text-to-Speech	57
Microsoft Azure Text-to-Speech	147
ElevenLabs	5
Play.ht	10

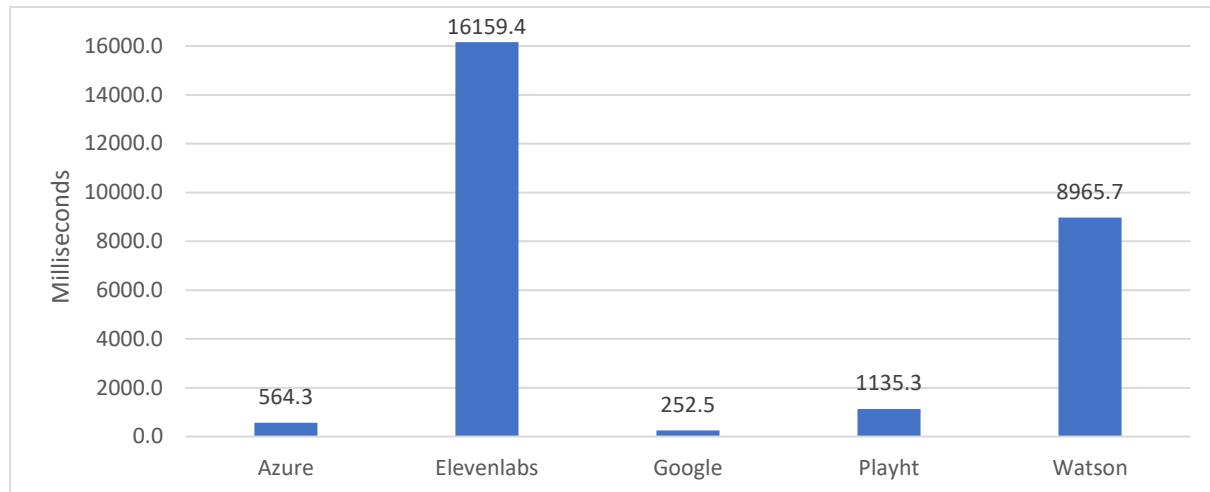
Note: All links in this table were accessed no later than July 4 at 11:34. The best result is in green.  
 Source: Own representation.

#### 4.2.4 Results performance

The bar chart in Figure 10 compares the average speeds of five TTS providers. Microsoft Azure performed well with good speed, while ElevenLabs had the longest processing time. Google was the fastest, and Play.ht and IBM Watson were in the middle. Given the essential importance of speed for a chatbot, Google and Microsoft Azure emerged as winners from this speed test.

**Figure 10**

Bar chart that visualizes the speed test results for the TTS APIs.



Source: Own illustration. Created with Excel.

#### 4.2.5 Results costs

The models from Google, Azure, and IBM followed the principle of usage-based payment, "*pay as you go*," and were billed based on the number of characters used. On the other hand, play.ht, and ElevenLabs offered only pricing models that required a fixed monthly payment. These results can be seen in Table 11.

**Table 11**

Cost of the TTS-APIs

API	Costs (CHF)	Free characters per month
IBM Watson TTS	0.00002 per character	10'000 per month
Google TTS	0.000014 per character	500'000 per month
Microsoft Azure TTS	0.000014358 per character	1'000'000 per month
ElevenLabs	minimum 89 per month	-
Play.ht	minimum 89 per month	-

Note: All links in this table were accessed no later than July 3 at 14:33.

Source: Own representation.

The results presented in Table 12 illustrated the monthly expenses obtained by applying the calculation formula according to section 3.2.3.5. Notably, Google and Azure did not incur any monthly costs based on the data used. In contrast, Play.ht and ElevenLabs recorded remarkable monthly costs.

**Table 12**

Monthly costs of the TTS-APIs

API	Cost per month (CHF)
<b>IBM Watson Text-to-Speech</b>	9.40
<b>Google Text-to-Speech</b>	0
<b>Microsoft Azure Text-to-Speech</b>	0
<b>ElevenLabs</b>	minimum 89 per month
<b>Play.ht</b>	minimum 89 per month

Note: All links in this table were accessed no later than July 3 at 14:33.

Source: Own representation.

#### 4.2.6 Overall result of TTS

Table 13 presents in summary form the overall results of the TTS-APIs. Based on these results, the selection of the preferred TTS-API was made. A comprehensive explanation of the justification for this decision can be found in Chapter 5.2.

**Table 13**

Total overview of all TTS-API results

Evaluation Criteria		IBM Watson	Google TTS	Azure TTS	Eleven-Labs	Play.ht
Voice quality	Intelligibility	5.4	5.2	5.6	5.3	4.7
	Naturalness	5.3	3.0	5.4	4.2	2.6
Customization options		No comprehensive evaluation has been carried out as they are all at a similar level. However, Microsoft Azure has some advantages due to its Viseme functionality.				
Language coverage		14	57	147	5	10
Performance (ms)		8965.7	252.45	564.3	16159.4	1135.3
Costs per month (CHF)		9.40	0	0	min. 89	min. 89

Note: The two best results for each category are highlighted in green using the color scheme.

Source: Own representation.

## 5 Justification

The decision-making section of this bachelor thesis deals with selecting the optimal speech recognition and speech synthesis APIs for the chatbot project. The correct selection of APIs plays a crucial role in ensuring accurate speech recognition and speech synthesis that meet the requirements of the chatbot.

### 5.1 Justification of STT-API

The STT-API was selected based on the results in Table 8. Based on these results, OpenAI's Whisper API was selected as the preferred option. Several compelling arguments were made to support this decision. The following text explains these arguments in detail.

OpenAI's Whisper API was among the leading providers in all evaluation criteria. It ranked at least second in each category analyzed, making it one of the most outstanding options on the market. A key aspect was the high processing speed of the API. Especially for a chatbot, it is essential to convert speech to text quickly and in near real-time to achieve a more natural appearance. In addition, the API stood out for its exceptional precision. Compared to other APIs, Whisper showed superb potential in terms of precision. Whisper's high precision will help ChatGPT provide users with better answers to their converted questions. Additionally, there was a financial benefit as using OpenAI's Whisper API was less expensive than other APIs. This cost savings played an important role, especially in a project with a limited budget.

Furthermore, the implementation of the API was straightforward. OpenAI provided extensive documentation and resources to help developers get started, accelerating the development process, and allowing the project to progress efficiently. One potential drawback of the API was that it was still in beta (as of 21-06-2023) (OpenAI, n.d.), and the number of requests was limited to 50 RPM. This might have been a limitation for some projects. However, the 50 RPM limitation was not critical for the project at hand and did not affect the fulfillment of the requirements.

Due to all these factors, OpenAI's Whisper API proved to be an ideal choice for developing the chatbot in this bachelor thesis.

## 5.2 Justification of TTS-API

Based on the results of Table 13, the decision was made to use the Microsoft Azure API. This decision was justified by several factors, which are explained below.

Microsoft Azure's TTS API stood out for its remarkable performance. When converting text to speech, short audio files were converted in near real-time, which was critical for the efficiency of the project. In addition, Azure offered an attractive cost-benefit ratio. The use of this API was cost-effective compared to other providers and in this case, even free of charge, which positively impacted the project budget. Another critical factor for the selection was the variety of configuration options. The API allowed us to customize various parameters to adapt the voice output to the project's specific needs. The ability to obtain Viseme data using the Microsoft Azure TTS-API was also an essential function. Visemes are mouth movements and shapes corresponding to speech sounds, crucial for animating avatars during the later stages. In addition to the above reasons, the quality of the generated audio files was also convincing. According to the test persons, the generated voice outputs sounded natural and realistic, an important prerequisite for the project's success. In addition, the API offered a wide range of supported languages. This variety made it possible to select the languages needed for the project to reach a wider audience.

Overall, the Azure TTS-API offered an impressive combination of performance, cost, configuration options, Viseme support, audio quality, and language diversity. This decision will help achieve the chatbot's project goals and deliver high-quality voice output.

## 5.3 Justification of NLP API

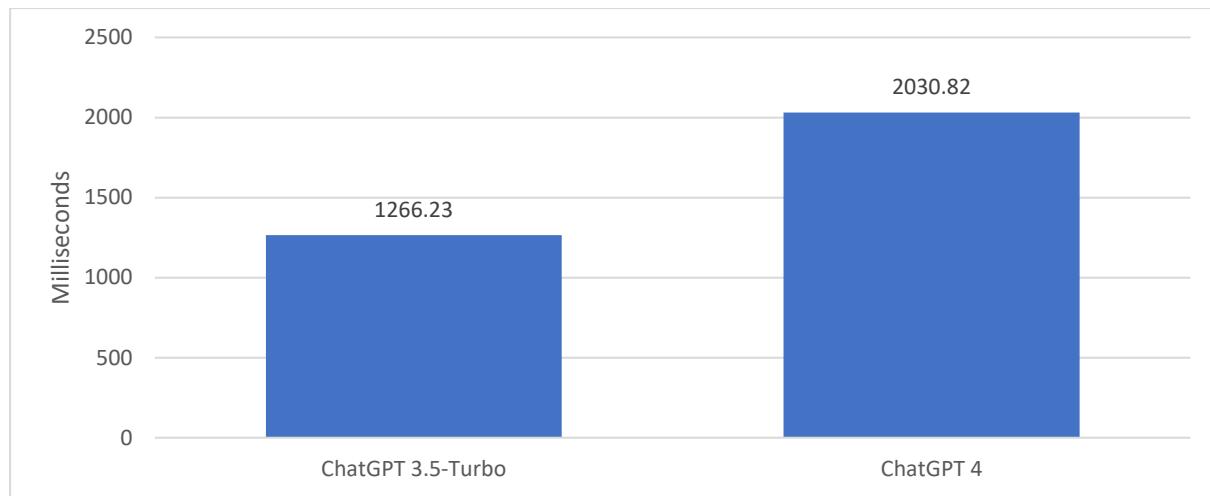
The ChatGPT API was decided to integrate into the chatbot due to its current popularity in the NLP space. On July 6, an email was received from OpenAI (Appendix 15) announcing the availability of Chat GPT 4 through the API. Based on this announcement, it was decided to perform a brief analysis of the GPT-3.5-turbo and GPT-4 models to determine the optimal model for the chatbot. This involved comparing the speed and content of the API responses. Special attention was paid to speed, as the requests to ChatGPT in this chatbot are not very complex. However, high speed is critical for more natural interaction between the user and the

system. According to Figure 11, Model 3.5-Turbo is faster than Model 4.

In Appendix 16, the content of the responses of both models is almost similar. The only notable difference is that Model 4 does not simply limit its generated responses to 100 tokens but attempts to find a matching response within this maximum token range.

**Figure 11**

Results of the speed test of the GPT models



Source: Own illustration.

## 5.4 Justification of Avatar Animation API

Due to limited time resources, performing a comprehensive analysis to select the most appropriate API for character animation was impossible. Nevertheless, research was conducted, and several options were considered. The APIs from the company's "*D-ID*" and "*Synthesia*" were shortlisted but ultimately not considered due to cost.

During the research, Rive emerged as the most promising alternative, backed by Microsoft Azure TTS's ability to generate Viseme. It was shortlisted due to its successful use by well-known companies like Duolingo for animating characters and synchronizing lip movements. A compelling paper from Duolingo demonstrated Rive's effectiveness (Vahidsafa, 2022). Rive's cost structure, offering a free usage option with certain limitations, was crucial for this project with a limited budget. For more information about Rive and its functions, refer to chapter 6.1.2 of the bachelor thesis.

## 6 Technical Basis of the Implementation

This chapter explains the technical foundations for implementing the mobile application. There is a comprehensive presentation of the applied technologies for the development of both the mobile application and the backend server. In addition, the software architecture and system architectures are presented. Furthermore, the various functionalities of the application are described in detail using mockups and UML diagrams.

### 6.1 Technologies

This chapter presents the technologies used for the mobile application and the backend. A short justification for the selection of these technologies is provided.

#### 6.1.1 Flutter with Dart

The decision to use Flutter and Dart was based on the basic knowledge acquired during the Business Informatics course at HES-SO Valais. This knowledge provides a solid foundation for the use of these technologies.

#### 6.1.2 Rive Flutter

Rive, formerly known as Flare (Flare, n.d.), is a development tool that allows developers to create and play interactive animations for applications, games, and websites. It provides the ability to develop animations that can respond to different states and user input. These animations can then be integrated using a lightweight and open-source runtime library, for example, easily into Flutter projects ("Rive | Flutter Package," 2023). Because of the lightweight runtime libraries, the animations within the app consume limited resources and require little power. A detailed description of the use of Rive in the context of animating the avatar in this project can be found in section 7.1.5.

#### 6.1.3 Google Firebase

In this project, the Firebase Authentication and Firestore services were implemented. Firebase Authentication enables simple and secure authentication of users ("Firebase Authentication," 2023). The login function can ensure the user's privacy. Firebase Firestore is

---

used to store and manage the chatbot's data. Firestore is a NoSQL database that provides a scalable and flexible way to store and retrieve data in real-time ("Firebase Firestore," 2023). The chatbot can store user information and chat history in Firestore. This ensures that the chatbot's data is secure, reliable, and can be retrieved anytime. Overall, Firebase Authentication and Firestore provide secure authentication, efficient data management, and a personalized user experience, resulting in a powerful and user-friendly chatbot.

#### **6.1.4 Python with Flask**

In this project, Python was used in combination with the Flask framework. Flask is a lightweight Python-based web framework for developing web applications. It provides a simple yet flexible way to create APIs (Grinberg, 2018). Flask enabled the seamless integration of third-party APIs into the backend, namely the STT API, TTS API, and ChatGPT API. Thanks to Python and Flask, it is possible to ensure that the communication between the mobile app and the third-party APIs is smooth and effective.

#### **6.1.5 DigitalOcean®**

DigitalOcean® is a leading provider of cloud services headquartered in New York City. The company's global data centers offer straightforward and scalable cloud solutions for web developers and businesses (Saleem, 2023). DigitalOcean® was chosen for this project due to its user-friendly approach to enabling developers to create and manage virtual servers in the cloud. Developers can select from various pre-configured operating systems and application stacks to set up development environments and production applications. Another significant factor was DigitalOcean®'s simple and transparent pricing structure, wherein users only pay for the actual resources utilized.

## **6.2 Architecture**

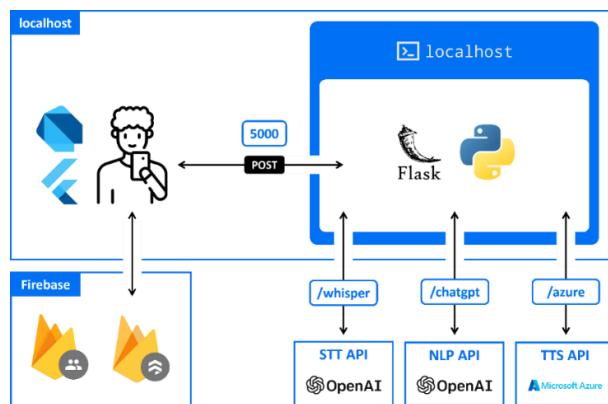
This section is dedicated to analyzing the different used systems and software architectures within the project of this bachelor thesis.

### 6.2.1 System architecture of the test environment

As shown in Figure 12, the tests were carried out in a local test environment for both the front and back end. The smartphone on which the mobile application is installed must be on the same network as the Flask application.

**Figure 12**

The architecture of the system within the testing environment.



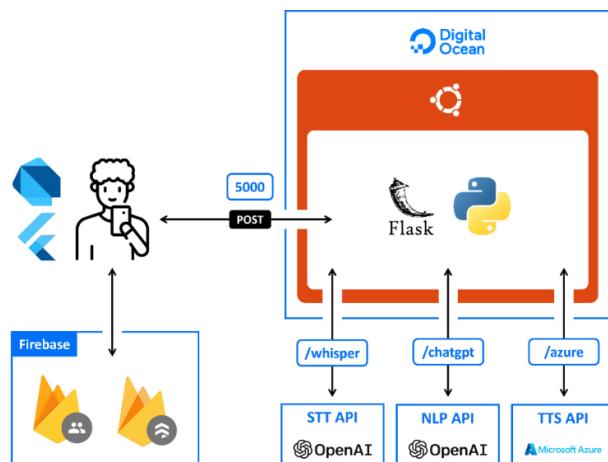
Source: Own illustration.

### 6.2.2 System architecture of the productive environment

The configuration of the productive system in Figure 13 is mainly like the configuration according to Figure 12. The only difference is that the backend is no longer locally but on a droplet on a DigitalOcean® server. The exact configurations can be seen in Figure 56.

**Figure 13**

The architecture of the system within the production environment.



Source: Own illustration.

### 6.2.3 Software architecture of the Flutter app

In this bachelor thesis, Flutter was used in combination with Clean Architecture. The abstraction of business logic and infrastructure details characterizes Clean Architecture. Its primary goal is to minimize dependencies between components to improve code reusability and maintainability. The architecture was conceived by Robert C. Martin in 2012 (Martin, 2012). The following Clean Architecture statement was created with the help of ChatGPT. It should be noted that the generated output of ChatGPT has undergone further revision to ensure it is presented correctly in terms of content.

#### *Brief Explanation of Clean Architecture with Flutter*

A major advantage of Clean Architecture is the clear delineation of responsibilities. Segmenting the application into different layers makes the code more transparent and easier to maintain. The presentation layer is responsible for the user interface, the domain layer contains the business logic, and the data access layer is responsible for accessing data sources. This clear structure makes it easier to understand the code and enables faster identification and resolution of problems. The Clean Architecture also promotes the scalability of the application. The use of interfaces and the modularity of the layers make it easy to add new functionality without making significant changes to the existing code. The individual layers can be developed and replaced independently, which increases the flexibility and extensibility of the application. In this way, the Clean Architecture is also well-equipped for future requirements and changes.

## 6.3 Functionalities of the App

In this chapter, the functionalities of the application are shown. For this purpose, the mockups and the corresponding sequence diagrams of the individual screens of the mobile application are presented. The mockups were created using the Balsamiq Wireframes software, while the UML diagrams were generated using the PlantUML software. It should be noted that the mockups presented may differ from the final application. The use case diagram can be viewed in Appendix 17.1, while the sequence diagram of the entire flow is available in Appendix 17.2.

### 6.3.1 Login and Registration

The chatbot app starts with the login user interface (UI) (Figure 14, left side). To access the app, users must enter their email address and password. If they do not have an account, they can click "Register" to create a new one (Figure 14, right side).

**Figure 14**

Mockup of the login and registration page



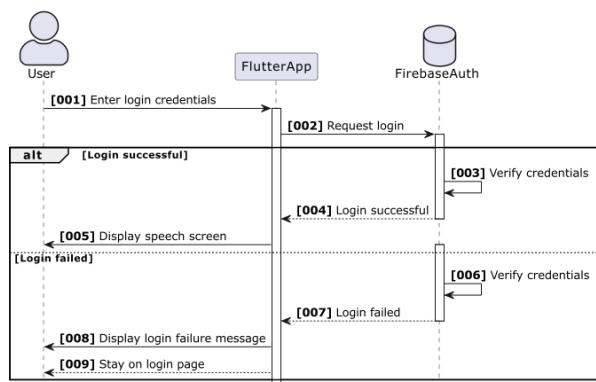
Source: Own illustration. Created with balsamiq.com.

### Sequence diagrams

In the login sequence (Figure 15), users enter their credentials into the Flutter app, which compares the login in the "*FirebaseAuth*" service. Upon successful login, access to the app's features is granted. An error message is displayed in case of a failed login, and the user remains on the login page.

**Figure 15**

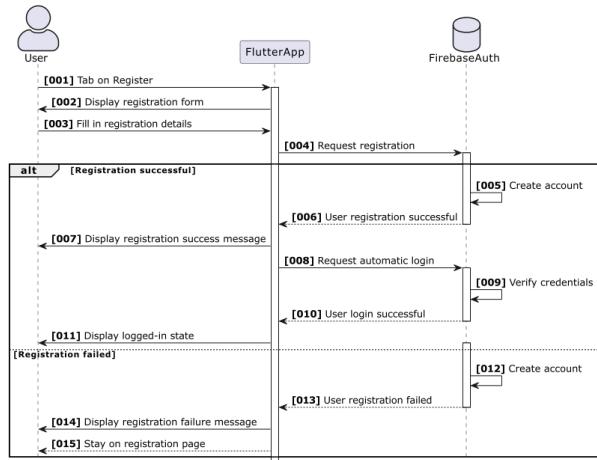
Sequence diagram of the login



Source: Own representation. Created with plantuml.com.

The registration process (Figure 16) involves filling out the form and sending a new user request to "FirebaseAuth". If successful, the user is automatically logged in; if unsuccessful, an error is displayed.

**Figure 16**  
Sequence diagram of the registration



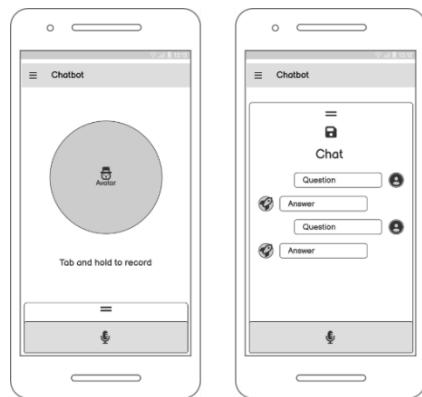
Source: Own representation. Created with plantuml.com.

### 6.3.2 Main screen

Figure 17 on the left shows the basic structure of the main screen's various functionalities. The avatar is in the center area of the screen. At the bottom of the screen is the button that activates the recording function. After a successful audio recording, the three processes STT (Figure 18), ChatGPT (Figure 19), and TTS (Figure 20) are executed one after the other. Above the record button is a slide-up field. Sliding this box up displays the chat. The UI of the chat can be seen on the right side of Figure 17. There you can see that the chat can be deleted. The user's chat entries are displayed on the right side of the chat screen, while ChatGPT's responses are on the left side.

**Figure 17**

Mockup of the main page and the opened panel



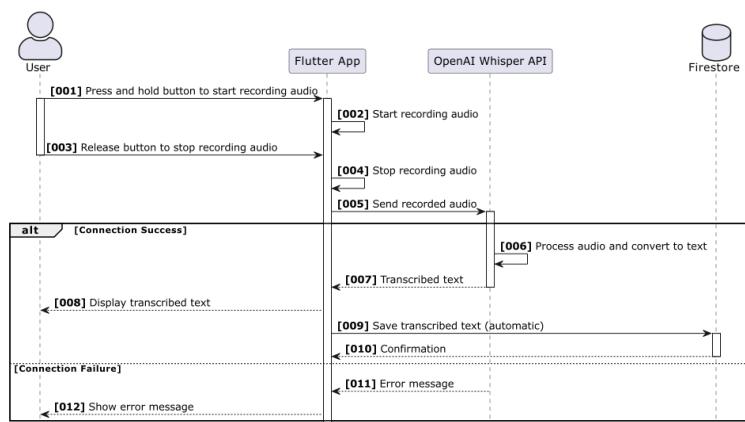
Source: Own illustration. Created with balsamiq.com.

### Sequence diagrams

The Flutter app transmits the recorded audio data to the Whisper API. If the connection is successful, the Whisper API processes the audio data and returns the text to the Flutter app, which displays it to the user. After that, the chat is automatically saved. In case of connection errors, the Whisper API sends an error message to the Flutter app, which displays it to the user.

**Figure 18**

Sequence diagram of the STT part



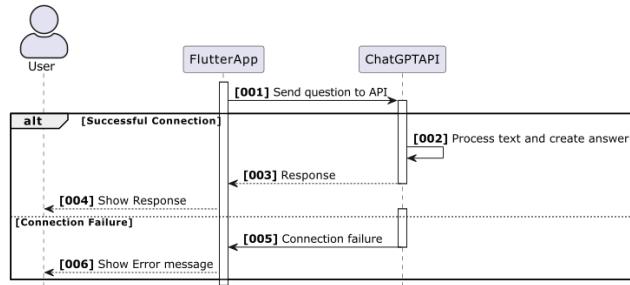
Source: Own representation. Created with plantuml.com.

The sequence diagram 19 shows the communication flow between the chatbot app, the user, and the ChatGPT API. The app sends a question to the API, which processes it and generates a response. If the connection is successful, the app receives the response and

displays it to the user. In case of a connection failure, the app receives an appropriate error message displayed to the user.

**Figure 19**

Sequence diagram of the ChatGPT part

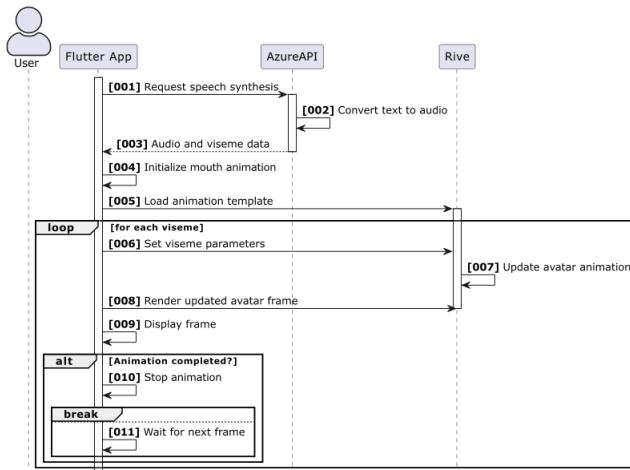


Source: Own representation. Created with plantuml.com.

The sequence diagram 20 illustrates the flow of the chatbot speech synthesis function. The app initiates a request for speech synthesis to the Azure API. The API performs a text-to-speech conversion and passes the generated audio and viseme data back to the app. The app initializes a sequence of mouth animations and loads an animation template from Rive. It then sets the parameters for each visual effect in Rive, which updates the avatar animation and generates an updated avatar image. The updated image is displayed in the app while simultaneously checking to see if the animation is complete. If it is, the animation stops. Otherwise, it waits for the following image.

**Figure 20**

Sequence diagram of the TTS and avatar animation part

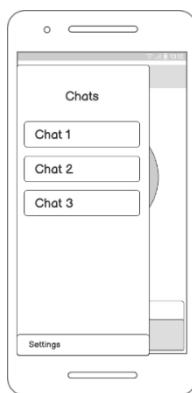


Source: Own representation. Created with plantuml.com.

### 6.3.3 Open menu

The menu is used for quick and easy access. Figure 21 gives an insight into the functions of the menu. The menu allows the user to select a previously saved chat or to create a new one. In addition, the user can navigate to the settings screen via the lower part of the menu. The menu can be closed by clicking on the space on the right side.

**Figure 21**  
Mockup of the opened menu

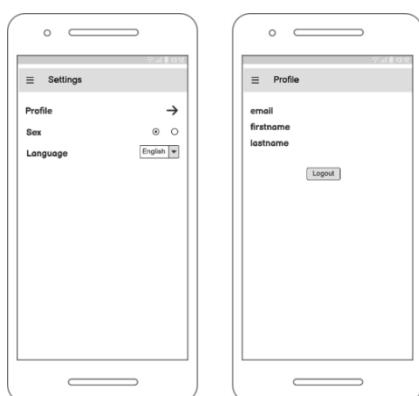


Source: Own illustration.

### 6.3.4 Settings and profile screen

Clicking on the "Settings" menu item displays the UI shown in Figure 22 on the left side. All available settings options are displayed vertically. The first option lets you navigate to the profile page. This page allows logging out (Figure 22, right side).

**Figure 22**  
Mockup from the settings and profile page



Source: Own illustration.

### 6.3.5 Error messages

Error messages are presented consistently throughout the application. A pop-up window like the one shown in Figure 23 is displayed.

**Figure 23**

Mockup of alert message



Source: Own illustration.

## 7 Implementation

This chapter deals with implementing the chatbot and Flask application. Two separate programs were developed in the process. The functionalities were implemented according to the Product Backlog, which can be found in Appendix 18. In addition, a Kanban board was created. An excerpt from the Kanban board used can be found in Appendix 19. The Kanban board allowed a better overview of the outstanding and already completed functionalities. The essential steps of the implementation will be covered in this chapter.

It should be noted that this chapter has been continuously updated according to the latest updates of the application. This means that here is the latest version of the chatbot application with the changes from the tests performed in Chapter 9. Furthermore, some code explanations in this chapter were generated with the help of ChatGPT using the following prompt:

*comment for me this flutter code (Here the code will be placed)*

## 7.1 Chatbot

A private GitHub repository was created to develop the chatbot application to ensure secure code management. Within the repository, two main branches were established: the "*Master*" branch, which served as the stable release, and the "*Dev*" branch, on which active development occurred. Changes were first implemented on the "*Dev*" branch, tested, and iterated as needed. Once a new function or major feature was added and a stable version was reached, the "*Dev*" branch was merged with the "*Master*" branch.

The dart packages used during the development phase of the chatbot are listed in Appendix 20. The chatbot application was implemented in six consecutive milestones. These can be seen in Table 14. The total time spent developing the chatbot system is a total of 94 hours.

**Table 14**  
The implemented milestones

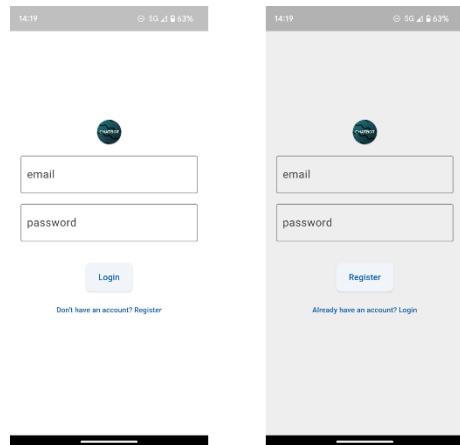
Milestone
<b>1<sup>st</sup>: implementation of login/registration</b>
<b>2<sup>nd</sup>: STT</b>
<b>3<sup>rd</sup>: ChatGPT</b>
<b>4<sup>th</sup>: TTS</b>
<b>5<sup>th</sup>: Animation of the avatar</b>
<b>6<sup>th</sup>: Save and delete chats in Cloud Firestore</b>

Source: Own representation.

### 7.1.1 Milestone 1: implementation of login/registration

The authentication and registration functionalities were realized using the implementation of Firebase. The final GUIs of the two screens for login and registration can be seen in Figure 24.

**Figure 24**  
GUI of the login and registration screen



Source: Own representation. Screenshots of the application

The method shown in Figure 25 is used in Flutter to manage the login functionality. The Auth class authenticates the user based on the specified email address and password. The predefined method "*signInWithEmailAndPassword()*" of Firebase is called in the Auth class. If an error occurs during the sign-in process, specifically a "*FirebaseAuthException*", it is caught in the method, and the "*errorMessage*" state variable is set to suggest the error report. The register with Firebase Authentication has almost the same functionality as the login method. The only difference is that the "*signUpWithEmailAndPassword()*" method is called by Firebase.

**Figure 25**  
Login method

```
login_register_screen.dart

Future<void> signInWithEmailAndPassword() async {
  try {
    // Calls the signInWithEmailAndPassword method from the Auth class
    await Auth().signInWithEmailAndPassword(
      email: _emailController.text.trim(), // Retrieves the email value from the _emailController
      password: _passwordController.text.trim(), // Retrieves the password value from the _passwordController
    );
  } on FirebaseAuthException catch (e) {
    // Catches any FirebaseAuthException that occurs during the sign-in process
    setState(() {
      // Sets the error message to the exception's message
      errorMessage = e.message;
    });
  }
}
```

Source: Own representation. Created with codebeautify.org.

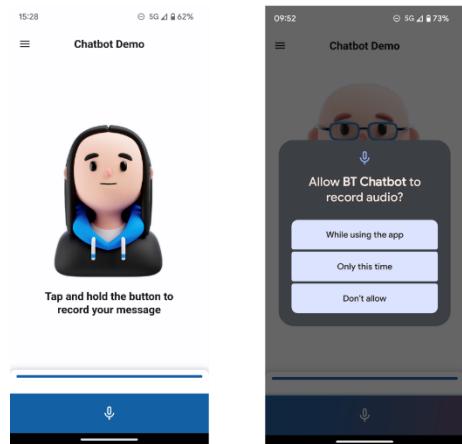
### 7.1.2 Milestone 2: implementation of STT

In this milestone, the UI was first created on which the recording can be performed. The

GUI can be seen in Figure 26 on the left side.

## Figure 26

GUI of the main page and the permissions question

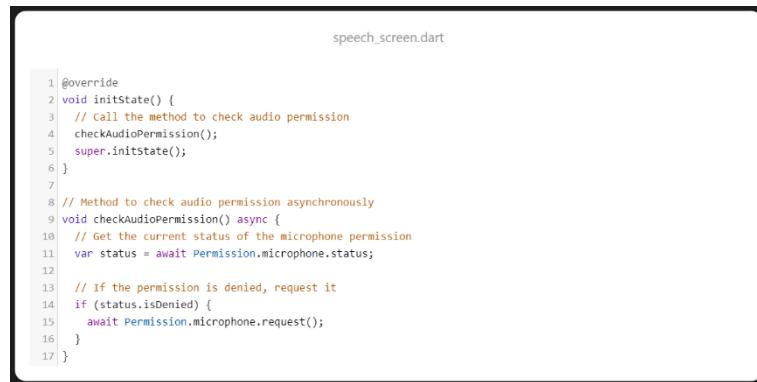


Source: Own representation. Screenshots of the application

When the application is started for the first time, a permission query appears asking whether the use of the microphone should be allowed or not. If permission is not granted, performing a recording later will not be possible. The visual representation of this notification can be seen in Figure 26 on the right. A method was implemented in the code executed when the main page was started. This method uses the dart package "*permission\_handler*" to check whether the permission has already been set. The permission window will not be displayed again if the permission has been set. The implementation of this method is shown in Figure 27. In addition, when the user presses the "*Record*" button, a check is performed to determine whether or not the user is authorized to record data. If the authorization is not present, which means that the user has rejected the corresponding authorization request from Figure 27, an error message will be displayed to the user. The user will then be allowed to open the device's settings directly in the application to permit to use the microphone. This function is shown in Figure 28 on line 5.

## Figure 27

Check the permission method.



```

speech_screen.dart

1 // @override
2 void initState() {
3   // Call the method to check audio permission
4   checkAudioPermission();
5   super.initState();
6 }
7
8 // Method to check audio permission asynchronously
9 void checkAudioPermission() async {
10   // Get the current status of the microphone permission
11   var status = await Permission.microphone.status;
12
13   // If the permission is denied, request it
14   if (status.isDenied) {
15     await Permission.microphone.request();
16   }
17 }
```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

Recording with the microphone was made using the "*record*" dart library. The function "*\_start()*" is executed at the beginning of the recording, as shown in Figure 28. First, it checks whether the authorization to use the microphone is available. When the legitimacy to use the microphone is granted, the recording process is initiated while the avatar is put in the "*listening*" state. The different states of the Avatar can be viewed in chapter 7.1.5.

## Figure 28

Start recording method.



```

speech_screen.dart

1 // Method to start recording audio
2 Future<void> _start() async {
3   try {
4     // Check if the app has permission to record audio
5     if (await _audioRecorder.hasPermission()) {
6       setState(
7         () {
8           _boolListeningInput?.value = recordingProvider.isRecording;
9         },
10      );
11
12     // Start recording audio
13     await _audioRecorder.start();
14   } else {
15     // Show an error dialog if the permission to record audio is denied
16     _showAlertDialog("Permission denied");
17   }
18 } catch (e) {
19   setState(
20     () {
21       _boolListeningInput?.value = false;
22     },
23   );
24   if (kDebugMode) {
25     print(e);
26   }
27 }
28 }
```

Note: Simplified presentation of the code to show the essential things.

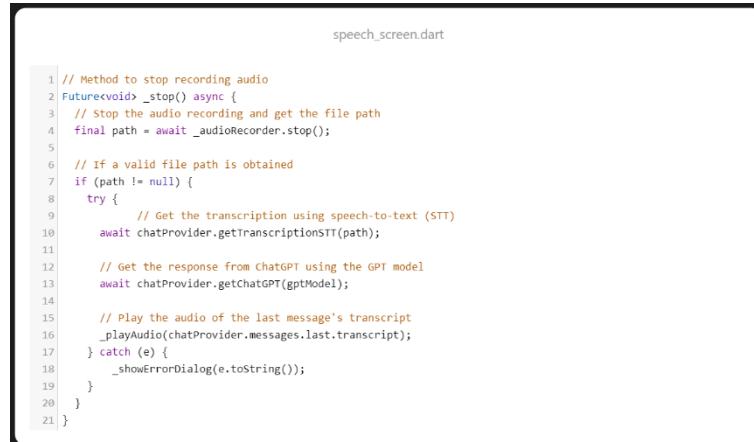
Source: Own representation. Created with codebeautify.org.

The "*\_stop()*" method is executed when the record button is released. This method is shown

in Figure 29. Line 13 in this figure is significant because that is where the method is called to send the recorded audio file to the STT-API. This method is called through the "*ChatProvider*" class. The called method of the "*ChatProvider*" can be seen in Figure 30.

**Figure 29**

Stop recording method.



```

speech_screen.dart

1 // Method to stop recording audio
2 Future<void> _stop() async {
3     // Stop the audio recording and get the file path
4     final path = await _audioRecorder.stop();
5
6     // If a valid file path is obtained
7     if (path != null) {
8         try {
9             // Get the transcription using speech-to-text (STT)
10            await chatProvider.getTranscriptionSTT(path);
11
12            // Get the response from ChatGPT using the GPT model
13            await chatProvider.getChatGPT(gptModel);
14
15            // Play the audio of the last message's transcript
16            _playAudio(chatProvider.messages.last.transcript);
17        } catch (e) {
18            _showErrorDialog(e.toString());
19        }
20    }
21 }

```

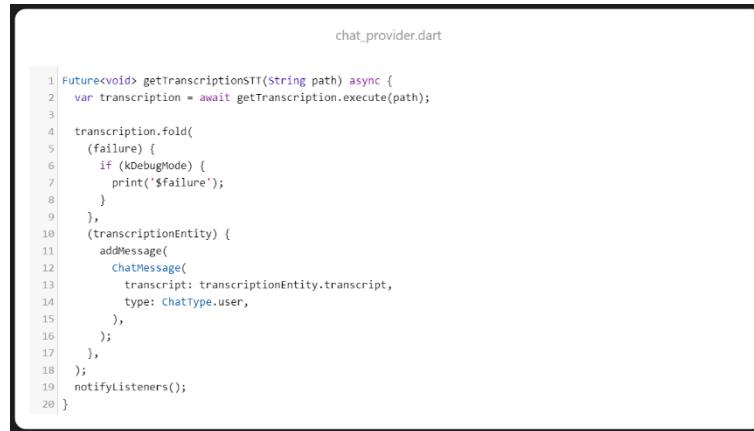
Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

In the method to be seen in Figure 30, the call to the STT API is made. However, the primary focus of this method is to mention "*notifyListeners()*". This represents a function of the "*ChangeNotifier*" that notifies and updates all listeners associated with this "*ChatProvider*". As a result, the chat is automatically updated in all relevant places as appropriate.

**Figure 30**

getTranscriptionSTT() method



```

chat_provider.dart

1 Future<void> getTranscriptionSTT(String path) async {
2     var transcription = await getTranscription.execute(path);
3
4     transcription.fold(
5         (failure) {
6             if (kDebugMode) {
7                 print('$failure');
8             }
9         },
10        (transcriptionEntity) {
11            addMessage(
12                ChatMessage(
13                    transcript: transcriptionEntity.transcript,
14                    type: ChatType.user,
15                ),
16            );
17        },
18    );
19    notifyListeners();
20 }

```

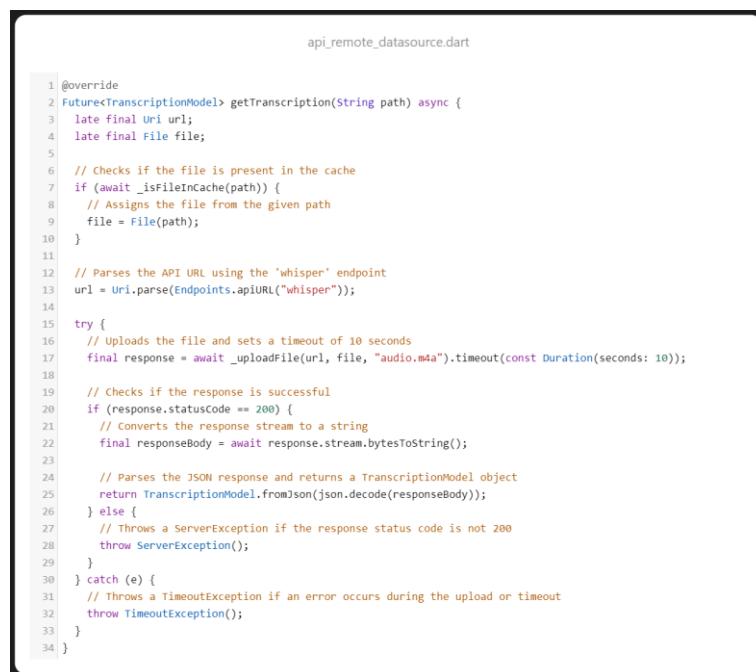
Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

Figure 31 illustrates the procedure for calling the STT-API at the most basic layer, the data layer. At this point, the "`_uploadFile()`" method is used to send an HTTP POST request to the Flask API with the recorded audio file. If there is no response within 10 seconds, a "`TimeoutException`" is thrown, and the corresponding error message is displayed.

### Figure 31

Method to call the STT-API



```

api_remote_datasource.dart

1 @override
2 Future<TranscriptionModel> getTranscription(String path) async {
3   late final Uri url;
4   late final File file;
5
6   // Checks if the file is present in the cache
7   if (await _isFileInCache(path)) {
8     // Assigns the file from the given path
9     file = File(path);
10   }
11
12   // Parses the API URL using the 'whisper' endpoint
13   url = Uri.parseEndpoints.apiUrl("whisper");
14
15   try {
16     // Uploads the file and sets a timeout of 10 seconds
17     final response = await _uploadFile(url, file, "audio.m4a").timeout(const Duration(seconds: 10));
18
19     // Checks if the response is successful
20     if (response.statusCode == 200) {
21       // Converts the response stream to a string
22       final responseBody = await response.stream.bytesToString();
23
24       // Parses the JSON response and returns a TranscriptionModel object
25       return TranscriptionModel.fromJson(json.decode(responseBody));
26     } else {
27       // Throws a ServerException if the response status code is not 200
28       throw ServerException();
29     }
30   } catch (e) {
31     // Throws a TimeoutException if an error occurs during the upload or timeout
32     throw TimeoutException();
33   }
34 }

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

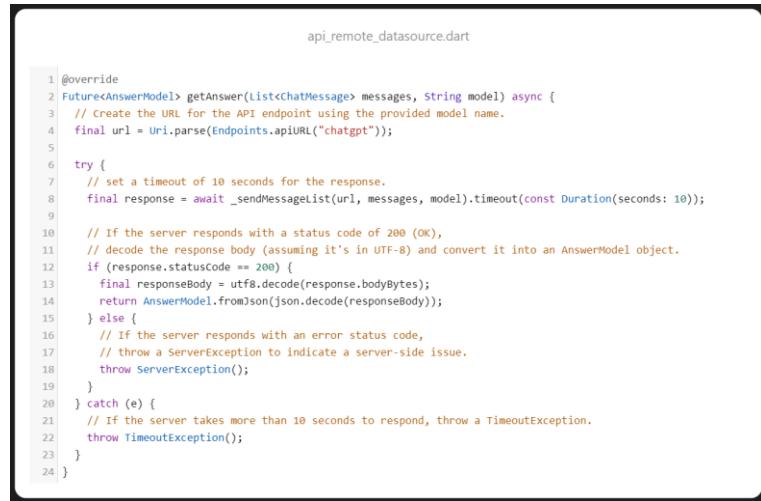
However, receiving a response with a status code of 200 signals a successful call. The received response is then returned to the "`ChatProvider`". After that, the converted text is displayed in the chat, as explained in a previous section, and the process in the "`stop()`" method advances one step. As a result, the method for the following milestone, namely the implementation of ChatGPT, is called.

#### 7.1.3 Milestone 3: implementation of ChatGPT

This API call is like the STT-API call in its basic structure. The previously converted text from the STT-API is submitted as a parameter, to which the ChatGPT API generates a response. Subsequently, the "`ChatProvider`" is updated, which saves the message in the chat. The associated method can be seen in Figure 32.

## Figure 32

Method to call the ChatGPT-API



```

api_remote_datasource.dart

1 @override
2 Future<AnswerModel> getAnswer(List<ChatMessage> messages, String model) async {
3   // Create the URL for the API endpoint using the provided model name.
4   final url = Uri.parseEndpoints.apiUrl("chatgpt"));
5
6   try {
7     // set a timeout of 10 seconds for the response.
8     final response = await _sendMessageList(url, messages, model).timeout(const Duration(seconds: 10));
9
10    // If the server responds with a status code of 200 (OK),
11    // decode the response body (assuming it's in UTF-8) and convert it into an AnswerModel object.
12    if (response.statusCode == 200) {
13      final responseBody = utf8.decode(response.bodyBytes);
14      return AnswerModel.fromJson(json.decode(responseBody));
15    } else {
16      // If the server responds with an error status code,
17      // throw a ServerException to indicate a server-side issue.
18      throw ServerException();
19    }
20  } catch (e) {
21    // If the server takes more than 10 seconds to respond, throw a TimeoutException.
22    throw TimeoutException();
23  }
24}

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

### 7.1.4 Milestone 4: implementation of TTS

The TTS-API is called similarly to the STT and ChatGPT API. This method is also called in the `"_stop()"` method. However, the main difference is that the returned data must first undergo audio conversion to play it later. This process is illustrated in Figure 33 on lines 19 and 31. In addition, the TTS-API also returns an array of different Visemes, which is converted into a list to be used later for animation. This process can be seen in Figure 33 in lines 33 to 37.

### Figure 33

Method to call the TTS-API

```

1 | @override
2 | Future<SpeechModel> getspeech(String answer, bool isMan, String language) async {
3 |   final url = Uri.parse(endpoints.apiUrl("azure"));
4 |
5 |   final response = await httpClient.post(
6 |     url,
7 |     headers: {"Content-Type": "application/json"},
8 |     body: '{"text": "$answer", "isMan": $isMan, "language": "$language"}',
9 |   );
10 |
11 |   if (response.statusCode == 200) {
12 |     final data = json.decode(response.body);
13 |
14 |     // Access the base64 encoded audio and data array
15 |     final audioBase64 = data['audio_file'];
16 |     final dataArray = data['viseme_data'];
17 |
18 |     // Decode base64 to audio bytes
19 |     final audioBytes = base64Decode(audioBase64);
20 |
21 |     late String filePath = "";
22 |
23 |     Directory? appExternalDirectory = await getExternalStorageDirectory();
24 |
25 |     if (appExternalDirectory != null) {
26 |       filePath = '${appExternalDirectory.path}/file.wav';
27 |     }
28 |
29 |     // Specify the desired file path
30 |     File file = File(filePath);
31 |     await file.writeAsBytes(audioBytes);
32 |
33 |     List<Viseme> convertedList = dataArray.map<Viseme>(
34 |       (item) {
35 |         return Viseme(item[0] as double, item[1] as int);
36 |       },
37 |     ).toList();
38 |
39 |     return SpeechModel.fromJson(
40 |       {
41 |         'file': file,
42 |         'visemes': convertedList,
43 |       },
44 |     );
45 |   } else {
46 |     throw ServerException();
47 |   }
48 | }

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

After the audio file and the list are created, the method returns these two components, which are processed further in the "`_playAudio()`" method of Figure 34. The audio player package used can be seen in Appendix 20. Line 32 demonstrates how the playback of the audio is started.

## Figure 34

Method to call the TTS-API



```

speech_screen.dart

1 Future<void> _playAudio(answer) async {
2   try {
3     // Request speech transcription using the GetSpeech service.
4     var transcription = await getSpeech.execute(answer, isMan, language);
5
6     late File file;
7     List<Viseme> visemeEvents = [];
8
9     // Process the response from GetSpeech.
10    transcription.fold(
11      (failure) {
12        if (kDebugMode) {
13          print('$failure');
14        }
15      },
16      (speechEntity) {
17        file = speechEntity.file;
18        visemeEvents = speechEntity.visemes;
19      },
20    );
21
22    // Create an instance of AudioPlayer to play the audio.
23    _audioPlayer = AudioPlayer();
24
25    // Set the file path and load the audio file.
26    await _audioPlayer.setFilePath(file.path);
27    await _audioPlayer.load();
28
29    _createVisemes();
30
31    // Play the audio and handle the completion event to unblock the app.
32    await _audioPlayer.play().then(
33      (value) {
34        blockingProvider.isBlocked = false;
35      },
36    );
37
38    // Dispose of the audio player after playing.
39    _audioPlayer.dispose();
40  } catch (e) {
41    _showErrorDialog(e.toString());
42  }
43}

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

The function "*createVisemes()*" works in collaboration with the method "*getVisemeld()*" but not directly to create a Viseme, as the name of the function might suggest. Instead, it looks for the corresponding Viseme at the audio's time. It does this by capturing the current time position of the audio being played and passing it to the "*getVisemeld()*" method. The "*getVisemeld()*" method, also seen in Figure 35, searches the previously created list of Visemes and returns the corresponding Viseme based on the audio time played. The returned value is then assigned to the variable provided, "*\_numberVisemesInput*". However, the exact functionality of the animation will be explained in more detail in the following chapter.

**Figure 35**

Methods to get the Visemes.

```

speech_screen.dart

1 // This method is used to search visemes based on the audio playback time.
2 void _createVisemes() {
3     // Whenever the audio position changes, this callback will be triggered.
4     _audioplayer.positionstream.listen(
5         (event) {
6             // Retrieve the audio time in milliseconds from the event.
7             double audioTime = event.inMilliseconds.toDouble();
8             // Set the state to update the visual representation of the viseme.
9             setState(
10                 () {
11                     // Call the getVisemeId method to determine the appropriate visemeId
12                     // based on the current audio time, and update the corresponding value.
13                     _numberVisemesInput?.value = getVisemeId(audioTime).toDouble();
14                 },
15             );
16         },
17     );
18 }
19
20 // This method retrieves the visemeId corresponding to the given audioTime.
21 int getVisemeId(double audioTime) {
22     for (int i = visemeEvents.length - 1; i >= 0; i--) {
23         final visemeEvent = visemeEvents[i];
24         // Check if the audio offset of the current visemeEvent is less than or equal to the given audioTime.
25         if (visemeEvent.audioOffset <= audioTime) {
26             // If true, return the corresponding visemeId for the current audioTime.
27             return visemeEvent.visemeId;
28         }
29     }
30     // If no appropriate visemeId is found for the given audioTime, return 0 as a default value.
31     return 0;
32 }

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

### 7.1.5 Milestone 5: animation of the Avatar

In this section, the exposition of the implementation of the avatar animation is done. The avatar's animation is divided into four states that the avatar can assume. These states are described in Table 15 and illustrated in Figure 36. Only the avatar lip sync animation is described in more detail in this chapter for simplification.

**Table 15**

States of the avatar

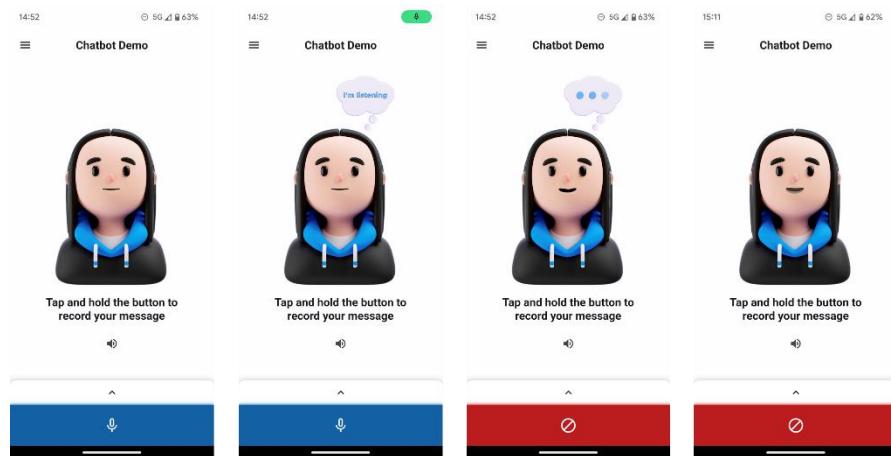
State	Animation
<b>Idle</b>	- Eyebrows move - Avatar blinks
<b>Record audio</b>	- Appearance of a thinking bubble with "I'm listening".
<b>Processing</b>	- Avatar starts smiling - Text content of the thinking bubble changes to three dots
<b>Speech</b>	- Lip sync animation is started

Note: Figure 36 contains the representation of the different states of the avatar.

Source: Own representation.

**Figure 36**

GUIs of the different states of the avatar.



Source: Own representation. Screenshots of the application

The avatar lip sync animation is based on the Visemes provided by the Microsoft Azure TTS API. Accessing the Visemes enables the display of mouth positions corresponding to the different phonemes. Figure 37 shows an example of such a Viseme provided by Microsoft. Here it can be seen that Viseme Id 2 represents the phoneme "a," and the corresponding mouth position is displayed. This mouth position reference pattern serves as a template for reproducing the mouth shape in Rive. Microsoft also provides 21 other mouth positions for various other phonemes, presented on the website <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-speech-synthesis-viseme>.

**Figure 37**

Excerpt from the Microsoft documentation about the Viseme

Visemeld	IPA	Mundposition
2	a	

Source: Screenshot from Yulin-Li (2023)

To synchronize the avatar's lip movements, the first thing to do is to take a closer look at the Azure TTS API, or more precisely, the API's return. Based on the generated audio, the API can automatically create the Visemes. Figure 38 shows the returned Visemes from the API. Each position in this array stores two values. The first value of the array represents the time duration during which the Viseme Id, represented by the second value of the array, retains its validity before being replaced with a new Id. As mentioned in chapter 7.1.4, this array is stored in a list that is later accessed to display the corresponding Viseme correctly.

**Figure 38**

Array of the TTS-API

```
I/flutter ( 4212): [[50.0, 0], [100.0, 20], [175.0, 13], [287.5, 4], [375.0, 6], [462.5, 19], [650.0, 0], [1400.0, 0], [1450.0, 12], [1550.0, 9], [1650.0, 2 0], [1725.0, 1], [1750.0, 19], [1800.0, 11], [1900.0, 1], [1987.5, 15], [2100.0, 6], [2187.5, 15], [2237.5, 19], [2287.5, 6], [2325.0, 7], [2387.5, 7], [241 2.5, 6], [2462.5, 19], [2500.0, 17], [2537.5, 6], [2587.5, 15], [2662.5, 19], [2737.5, 4], [2912.5, 15], [3037.5, 19], [3162.0, 0]]
```

Source: Own representation. Screenshot.

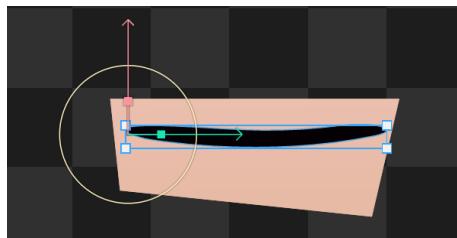
If the method "`_playAudio()`" is called according to Figure 34, this leads to the execution of the previously described methods in Figure 35. In this context, the corresponding Visemes for the played audio are determined. Once the corresponding Viseme has been determined, it is stored in the state variable "`_numberVisemeInput`". This variable is provided by the Rive package and explained in more detail from this step on in the subsection "*7.1.5.2 Rive in combination with Flutter*". To make the concept around Rive more understandable, it is necessary to explain the Rive process from scratch.

### 7.1.5.1 Process in Rive

This section explains how the animations were created using Rive. Rive provides a web-based interface through which the animations can be designed and developed. This web interface is shown in Figure 40 and, in this case, displays the drawing areas for the male and female avatars. Various elements such as images, shapes, text, and other objects can now be added to these artboards for later animation. In particular, the lips were drawn using the Rive tools in this context. This requires a certain amount of practice, and prior knowledge of programs such as Adobe Illustrator or Inkscape is an advantage. Figure 39 shows the drawn lip for Visem id 0, corresponding to a closed mouth condition.

**Figure 39**

Illustration of a closed mouth

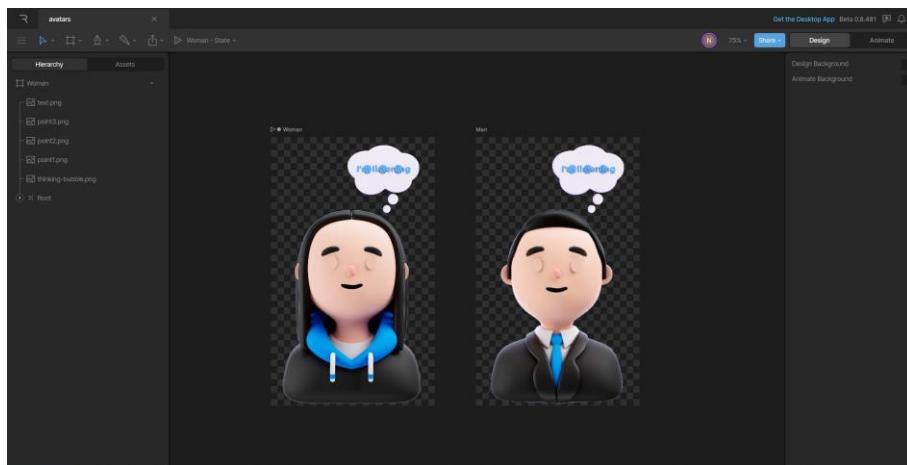


Source: Own representation. Screenshot.

In order to animate the created objects afterward, it is necessary to switch the view in the web interface. This can be achieved by switching from the "*Design*" view, in Figure 40, to the "*Animate*" view in Figure 41.

**Figure 40**

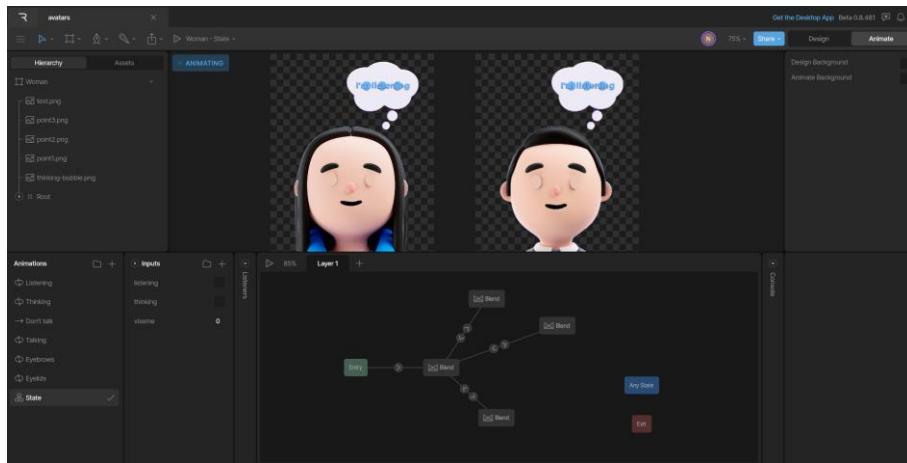
Web interface design



Source: Own representation. Screenshot.

In the "*Animate*" view of Figure 41, the focus is on the lower half of the image. This is where the animation of the objects takes place. The title "*Animations*" can be seen in the lower-left half of the image. In this area, there is the possibility to create either a "*Timeline*" or a "*State Machine*". The "*Timeline*" is used to create the animations for the respective objects. Several objects can be animated at the same time. On the other hand, the "*State Machine*" allows the various "*Timelines*" to be merged to create a comprehensive animation.

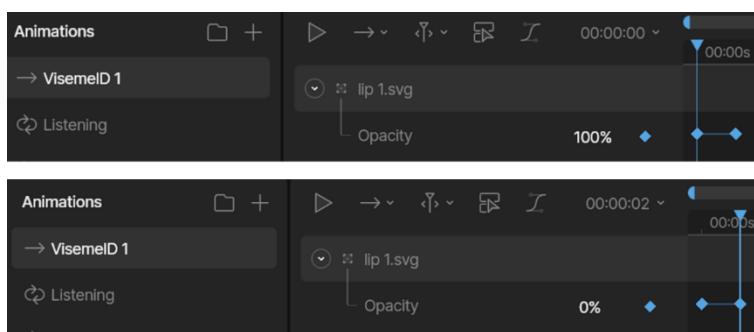
**Figure 41**  
Web interface animation



Source: Own representation. Screenshot.

Figure 42 shows the "*Timeline*" that illustrates the animation of the lip movements for Viseme ID 1. This animation was done through simple settings where the lip shape was set from 100% to 0% after 2 seconds. Such an animation can now be applied to all lip shapes of the respective Viseme.

**Figure 42**  
Illustration of a made lip animation.

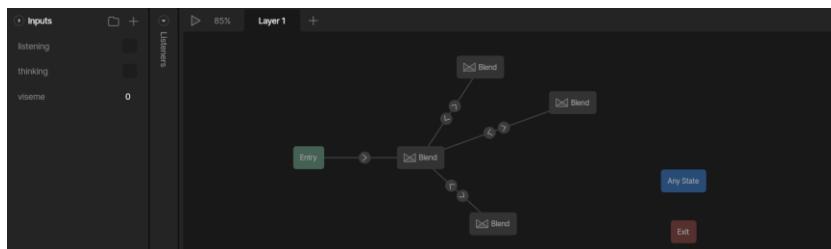


Source: Own representation. Screenshot.

To realize the correct representation of the lip movements for the respectively generated Visemes of the audio, the application of the "*State Machine*" is required. The "*State Machine*" of this project can be seen in Figure 43. On the left side of Figure 43, you can see the "*Inputs*". These "*Inputs*" act as an interface between the animation and the code - in other words, they are used for the interaction between the designer and the programmer. In this project, the "*Inputs*" used consist of two Boolean values representing the avatar's "*listening*" and

"thinking" states. The different states of the avatar can be seen in Table 15. These states are used to change the avatar accordingly while thinking or listening. Thus, the avatar's state can be easily controlled based on these values. Furthermore, a number input field is provided to modify the different values for the Visemes. This input variable was mentioned earlier in this chapter. It is the "`_numberVisemeInput`" variable.

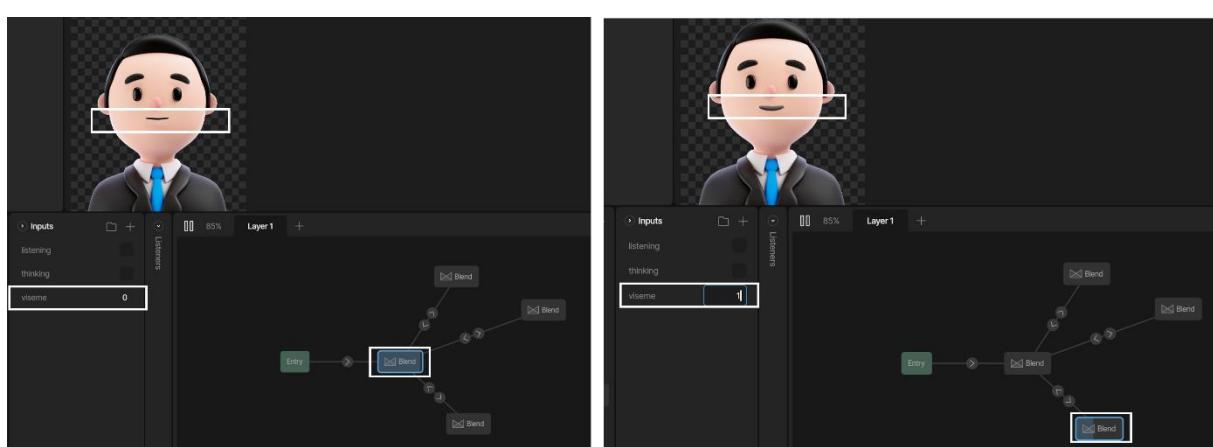
**Figure 43**  
"State Machine" of the chatbot project



Source: Own representation. Screenshot.

In the center of Figure 43, you can see the links between various blend objects. This functionality is best explained with the help of Figure 44. As seen from this illustration, a state change occurs when, for example, the Viseme input is switched from 0 to 1. In parallel, the shape of the avatar's lips changes. With this mechanism, it is possible to synchronize the avatar's lip movements based on the respective Viseme id. After completing all the animations, the file must be exported in .riv format to access the animation in the Flutter environment.

**Figure 44**  
Illustration of the change of state



Source: Own representation. Screenshot.

### 7.1.5.2 Rive in combination with Flutter

To integrate Rive into Flutter, installing the Rive package in Flutter is necessary. The overall required implementation of Rive in Dart code is illustrated in Figure 45. The present code from lines 6 to 13 demonstrates loading the animation into the application. The artboard changes according to the gender provider setting, alternating between representations of a woman and a man. The "`_onRiveInit()`" function, also shown in Figure 45 on line 16, is responsible for initializing and controlling the various input variables. This implies that the identified "found" Viseme, according to the previous description in the previous section, is continuously transmitted as input. This changes the state of the animation and thus leads to a movement of the lips.

**Figure 45**

Entire Rive implementation in Flutter.

```
speech_screen.dart

1 // Declaring three private variables for rive animation inputs
2 rive.SMIInput<double>? _numberVisemesInput;
3 rive.SMIInput<bool>? _boolThinkingInput;
4 rive.SMIInput<bool>? _boolListeningInput;
5
6 SizedBox(
7   width: 400.0,
8   height: 400.0,
9   child: rive.RiveAnimation.asset('assets/animations/avatars.riv',
10     artboard: isMan ? "Man" : "Woman",
11     onInit: _onRiveInit,
12     fit: BoxFit.fitHeight),
13 ),
14
15 /// Callback function called when the Rive animation is initialized and the [artboard] is ready.
16 void _onRiveInit(rive.Artboard artboard) {
17   // Create a StateMachineController using the 'State' state machine in the provided [artboard].
18   final controller = rive.StateMachineController.fromArtboard(artboard, 'State');
19
20   // Add the created controller to the [artboard] so that it can control the animation.
21   artboard.addController(controller!);
22
23   // Find and assign the 'viseme' input of type double from the controller to the [_numberVisemesInput]
24   _numberVisemesInput = controller.findInput<double>('viseme') as rive.SMIInputNumber;
25
26   // Find and assign the 'thinking' input of type bool from the controller to the [_boolThinkingInput]
27   _boolThinkingInput = controller.findInput<bool>('thinking') as rive.SMIInputBool;
28
29   // Find and assign the 'listening' input of type bool from the controller to the [_boolListeningInput]
30   _boolListeningInput = controller.findInput<bool>('listening') as rive.SMIInputBool;
31 }
```

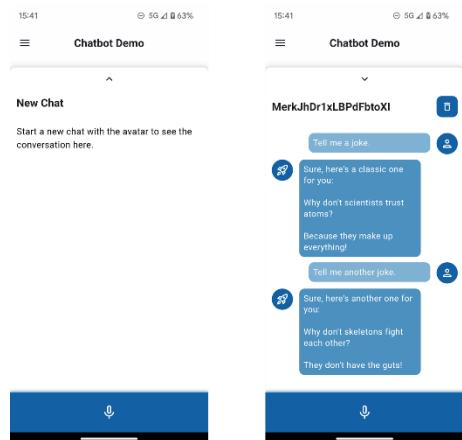
Note: Simplified presentation of the code to show the essential things.  
Source: Own representation. Created with codebeautify.org.

### 7.1.6 Milestone 6: implementation of Cloud Firestore

To store chats permanently, Cloud Firestore has been implemented as a database. The chat history is automatically stored in the database upon the first request to the chatbot. Viewing the chat history requires the user to swipe up the panel. A detailed functional description can be found in the attached user manual in Appendix 21. Figure 46 illustrates the panel open,

with an empty chat on the left. When the user sends a request to the chatbot, the chat history is saved in Firestore. Once the automatic saving is complete, a unique identification number is generated and displayed. This state is visualized on the right side of Figure 46.

**Figure 46**  
GUI of a new and a saved chat



Source: Own representation. Screenshots of the application.

Figure 47 shows the program code used to store chats in Firestore. First, it checks whether the "*ChatId*" passed as a parameter already exists. If it does not exist, a new document is created in Firestore; otherwise, the existing document is updated with this "*ChatId*". Then, the passed chat is converted into a format that Firestore can manage. Finally, the list is saved using Firestore's predefined "*documentReference.set()*" function.

**Figure 47**

Code for saving the chat.

```
firestore.dart

1 Future<String> saveChatMessages(String chatID, List<ChatMessage> messages) async {
2   DocumentReference documentReference;
3
4   if (chatID == "") {
5     // If the chatID is empty, create a new document reference
6     documentReference = FirebaseFirestore.instance
7       .collection('users')
8       .doc(Auth().currentUser!.uid)
9       .collection('chats')
10      .doc();
11   } else {
12     // If the chatID is not empty, use the existing chatID to update the document reference
13     documentReference = FirebaseFirestore.instance
14       .collection('users')
15       .doc(Auth().currentUser!.uid)
16       .collection('chats')
17       .doc(chatID);
18   }
19
20   // Convert the list to a format Firestore understands
21   List<Map<String, dynamic>> messagesMap = messages
22   .map((message) => {
23     'transcript': message.transcript,
24     'type': message.type.toString(), // Assuming type is an enum
25   }).toList();
26
27   // Save the list to Firestore
28   await documentReference.set({'messages': messagesMap});
29
30   return documentReference.id;
31 }
```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

The automatic saving is done with the help of the "*ChatProvider*", which is activated with every new chat entry. The corresponding code that implements the automation of storage is shown in Figure 48. In it, the functionality for saving is integrated into the "*addMessage()*" method of the "*ChatProvider*" class.

**Figure 48**

Code to save the chat automatically.

```
chat_provider.dart

1 void addMessage(ChatMessage message) async {
2   _messages.add(
3     message,
4   );
5   setChatId(
6     await firestoreService().saveChatMessages(chatID, messages),
7   );
8 }
```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

When the chat has been successfully saved, a corresponding functionality should be implemented to delete this chat on request. This functionality is implemented using a delete mechanism activated by pressing a corresponding button. The delete button is shown in Figure 46 on the right side. After the user presses the delete button, the code in Figure 49 initiates

the deletion process and removes the chat in question from the system accordingly.

### Figure 49

Showing the code for deleting the chat.



```
firestore.dart

1 Future<void> deleteChatMessages(String chatID) async {
2   DocumentReference documentReference = FirebaseFirestore.instance
3     .collection('users')
4     .doc(Auth().currentUser!.uid)
5     .collection('chats')
6     .doc(chatID);
7
8   // Delete the document
9   await documentReference.delete();
10 }
```

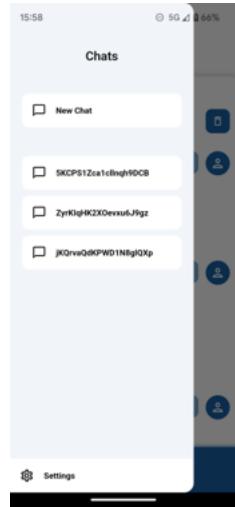
Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

After successfully implementing the functions for saving and deleting the chat, it is necessary to ensure the possibility of displaying the saved chat again. This functionality can be implemented in the chatbot application by opening the menu on the left side. The GUI of the opened menu is shown in Figure 50. Selecting a menu item with the corresponding ID displays the saved chat in the chat panel.

### Figure 50

GUI of the opened menu



Source: Own representation. Screenshots of the application

In the GUI, the chat histories are displayed in the menu using a "*StreamBuilder*". A "*StreamBuilder*" allows the menu to be updated automatically when a new chat is added or deleted. The underlying code for the "*StreamBuilder*" is illustrated in Figure 51. Here, it is

noticeable in line 2 that a method is called that outputs all documents in the "Chat" collection, thus displaying all stored chat IDs. The code on lines 30 to 33 is executed when a user taps a chat menu item. All messages of this selected chat are loaded and passed to the "ChatProvider". This will update all "logged-in clients" with the latest chat messages through this provider.

### Figure 51

Shows the code for displaying the chats in the menu.

```

1 StreamBuilder<List<String>>(
2   stream: FirestoreService().getAllChats(),
3   builder: (context, snapshot) {
4     if (snapshot.connectionState == ConnectionState.waiting) {
5       // While waiting for data to load
6       return const CircularProgressIndicator();
7     } else if (snapshot.hasError) {
8       return Padding(
9         padding: const EdgeInsets.only(top: 20),
10        child: Text(
11          'Error: ${snapshot.error}',
12          style: TextStyle(
13            color: Theme.of(context).colorScheme.error,
14            fontSize: 12),
15        ),
16      );
17    } else {
18      List<String> dataList = snapshot.data ?? [];
19      return ListView.builder(
20        shrinkWrap: true,
21        itemCount: dataList.length,
22        itemBuilder: (context, index) {
23          String chatId = dataList[index];
24          return Container(
25            margin: const EdgeInsets.only(bottom: 10),
26            child: ListTile(
27              leading: const Icon(Icons.chat_bubble_outline),
28              title: Text(chatId),
29              onTap: () async {
30                chatProvider.setChatId(chatId);
31                List<ChatMessage> messages =
32                  await FirestoreService().getAllChatMessages(chatId: chatId);
33                chatProvider.setChatMessages(messages);
34              },
35            ),
36          );
37        },
38      );
39    }
40  },
41 ),

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

In the context of Cloud Firestore, it is of particular importance to consider the security settings for access to the database. Firestore allows the definition of different sets of rules that apply to the database. Figure 52 shows the specific rules defined for this project. Lines 6 through 8 of the rule sets govern access to user data. The present code clarifies that only the user with his individual "UID" can access his data. The same applies to the regulation regarding chats, as defined in lines 11 to 13.

**Figure 52**  
Firestore Rules

```
Cloud Firestore Rules

1 rules_version = '2';
2 service cloud.firestore {
3     match /databases/{database}/documents {
4
5         // Collection 'users' is restricted to only authenticated users
6         match /users/{userId} {
7             allow read, write: if request.auth != null && request.auth.uid == userId;
8         }
9
10        // Collection 'chats' under a specific user is restricted to that user only
11        match /users/{userId}/chats/{chatId} {
12            allow read, write: if request.auth != null && request.auth.uid == userId;
13        }
14    }
15 }
```

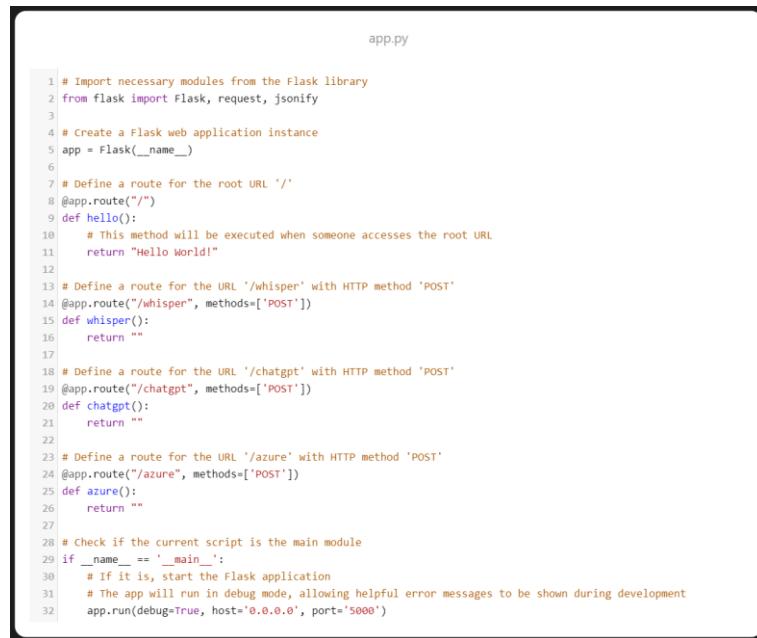
Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

## 7.2 Flask Application

The backend of the mobile application was developed using Python and Flask. For this, the Flask package was successfully installed using the "*pip install Flask*" command via the Python package management system. After the successful installation, the functionalities of Flask could be used. Figure 53 presents a conceptual representation of the Python file and the Flask functions applied. The function "`@app.route("")`" represents an API endpoint in Flask. An endpoint was created for each API accordingly. In these endpoints, each API was called by default according to the documentation of the respective API. To show this code here is beyond the scope of the implementation part. The code for this can be seen in Appendix 24. Finally, the method "`app.run(debug=True, host='0.0.0.0', port='5000')`" was used to specify which IP address and port to run the Flask application. For more details about deploying the backend in production mode, see Chapter 8.

**Figure 53**  
Representation of the Python code



```

app.py

1 # Import necessary modules from the Flask library
2 from flask import Flask, request, jsonify
3
4 # Create a Flask web application instance
5 app = Flask(__name__)
6
7 # Define a route for the root URL '/'
8 @app.route("/")
9 def hello():
10     # This method will be executed when someone accesses the root URL
11     return "Hello World!"
12
13 # Define a route for the URL '/whisper' with HTTP method 'POST'
14 @app.route("/whisper", methods=['POST'])
15 def whisper():
16     return ""
17
18 # Define a route for the URL '/chatgpt' with HTTP method 'POST'
19 @app.route("/chatgpt", methods=['POST'])
20 def chatgpt():
21     return ""
22
23 # Define a route for the URL '/azure' with HTTP method 'POST'
24 @app.route("/azure", methods=['POST'])
25 def azure():
26     return ""
27
28 # Check if the current script is the main module
29 if __name__ == '__main__':
30     # If it is, start the Flask application
31     # The app will run in debug mode, allowing helpful error messages to be shown during development
32     app.run(debug=True, host='0.0.0.0', port='5000')

```

Note: Simplified presentation of the code to show the essential things.

Source: Own representation. Created with codebeautify.org.

## 8 Publishing

In this chapter, a distinction is made between the chatbot application's publication and the backend's publication.

### 8.1 Publishing chatbot application

To streamline and automate the release process, GitHub Actions was integrated. GitHub Actions is a continuous integration and delivery (CI/CD) platform that can automate the build, test, and deployment pipeline (GitHub, n.d.). Using GitHub Actions, automating the build process and APK file creation was possible, resulting in significant time and resource savings by eliminating the need for manual steps. For each major release with significant changes, a new version was automatically created on GitHub by performing a push to the "*master*" branch. The corresponding YAML file code can be seen in Figure 54.

**Figure 54**  
Illustration of the YAML-Code

```

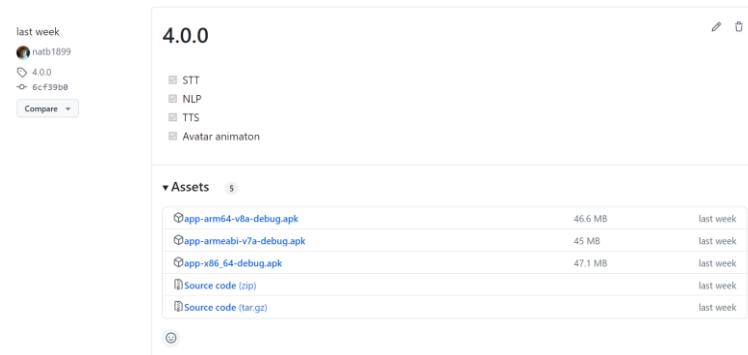
name: Build and Release APK
on:
  push:
    branches:
      - master
jobs:
  build:
    name: Build and Release
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v3
        with:
          distribution: "microsoft"
          java-version: "11"
      - uses: subosito/flutter-action@v2
      - run: flutter pub get
      - run: flutter test
      - run: flutter build apk --debug --split-per-abi
      - uses: ncpipollo/release-action@v1
        with:
          tag: 5.0.0
          artifacts: "build/app/outputs/apk/debug/*.apk"
          token: ${secrets.TOKEN}

```

Source: Own representation. Created with codebeautify.org.

With each new release, an updated APK file was also generated for users to download and test. An exemplary copy of such a release can be seen in graphic 55. The automatically generated APK file was crucial in uploading the chatbot application to Google Play.

**Figure 55**  
Illustration of a release on GitHub



Source: Own representation. Screenshot.

A Google developer account must first be requested to upload an application to Google Play. Identity confirmation usually takes two to three business days before the account can be fully used. Once a developer account is created, uploading the generated APK file and creating an internal test link to download the app easily is possible. The entire process was performed according to the instructions in the following video (HeyFlutter.com, 2021, June 18):

<https://www.youtube.com/watch?v=g0GNuoC0taQ>.

## 8.2 Publishing Flask application

The Flask application was successfully implemented in the production environment on DigitalOcean®. The details regarding DigitalOcean® were explained in chapter 6.1.5. However, during the release process, challenges were encountered with the Docker container, as running the Azure Microsoft API in it was not immediately possible (Eric-Urban, 2023). Attempts were made to submit a request to Microsoft, but no response was received. It was decided to run the Flask application as a background process on an Ubuntu server to overcome this hurdle. This approach allowed the desired application functionality to be maintained without relying on the support of a Docker environment. The Ubuntu server was a reliable foundation for running the application and ensured smooth and stable operation. A brief guide to releasing this application on the server is provided below. The official instructions from DigitalOcean® were used to create the "*Droplet*" (DigitalOcean, n.d.-b). The applied configurations can be seen in Figure 56.

**Figure 56**

Screenshot of the selected configuration for the Droplet

The screenshot shows a list of droplets with one item selected: "ubuntu-s-1vcpu-1gb-fra1-01". Below the list is a detailed configuration table for this specific droplet:

Image	Ubuntu 20.04 (LTS) x64	Region	FRA1
Size	1 vCPU 1GB / 25GB Disk (\$6/mo)	IPv4 IPv6 Private IP	[REDACTED] Enable [REDACTED]
	Resize	VPC	default-fra1

Source: Own representation. Screenshot.

To connect to the created "*Droplet*", the open-source tool PuTTY was used, which was developed as an SSH and Telnet client by Simon Tatham ("PuTTY," n.d.). A public and private key was created to ensure a secure connection to the "*Droplet*", supported by the PuTTYgen application. Interactive mouse movements on the computer were used to generate both a public and private key (DigitalOcean, n.d.-a). The public key was added to the "*Droplet*"

---

configuration, while the associated private key was securely stored on the local device.

After the droplet has been fully configured, a connection has been successfully established using the PuTTY client. A directory named "flask" was created on the server. Within this directory, the Python file created in chapter 7.2 was copied.

The Flask application was started as a background process using the command "*nohup python3 app.py > mylog.log 2>&1 &*". This command has the following meaning:

*explain me the command nohup python3 app.py > mylog.log 2>&1 &*

- "*nohup*": this abbreviation stands for "*no hang-up*" and prevents the started program from being terminated by closing the terminal session. This means the program will continue to run even if the user session is terminated or disconnected from the server.
- "*python3 app.py*" will start the Python3 program named "*app.py*".
- "*>*": This operator redirects the standard output of the Python program.
- "*mylog.log- "*2>&1- "*&***

Now the program runs, and by the command "*ps aux | grep app.py*", all processes on the system can be displayed, which contain the name "*app.py*". In this way, the Flask application process can be identified and terminated, for example.

The web page is called up when the Internet Protocol (IP) address of the "Droplet" is entered into the URL bar of a web browser together with port 5000.

## 9 Testing

In this chapter, the implementation of the test phase of the chatbot application is explained. First, there is an explanation of the self-contained manual tests. Then, the manual tests conducted by different test persons with corresponding feedback are presented.

### 9.1 Manual testing

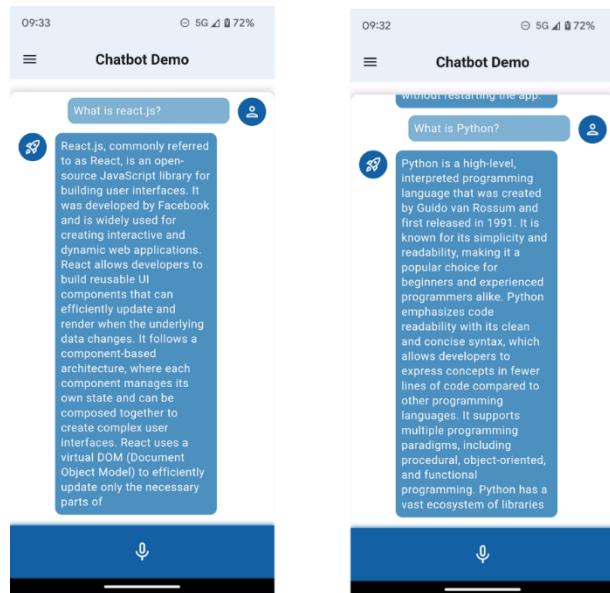
In this section, mainly technical tests are performed and analyzed. First, the total time it takes for the chatbot to process a request and return a response was examined and measured.

To evaluate this, a timer was integrated into the Flutter code to capture the processing time of the chatbot. To do this, 20 queries in English were sent to the chatbot. For mainly short answers, the most common expression when using the chatbot, the average processing time was 3.11 seconds. In contrast, for more extended responses, limited to a maximum of 100 tokens, the chatbot took an average of about 8.68 seconds to process. For German and French, the average processing time was 4.96 seconds and 5.03 seconds, respectively. These results illustrate that processing is almost two seconds faster in English. However, this factor is difficult to influence because many other factors work together. First and foremost, network connections are undoubtedly crucial, as is the speed at which the various APIs can process requests. In addition, the models are probably less well-trained in the German and French languages. Nevertheless, Chapter 10 presents a potential solution for accelerated avatar response time that could potentially be considered in future implementations.

However, another problem could be identified with the longer responses. As can be seen in Figure 57, the responses are truncated after reaching the maximum token count of 100. This is undoubtedly an unsatisfactory limitation.

**Figure 57**

Screenshot of the problem with cutting off the answers.



Source: Own representation. Screenshot of the Application.

Another identified problem arises from observing several requests indicating that the record button should be blocked during processing. This becomes necessary because conflicts between different API calls occur when new requests are made to the chatbot while older requests have not yet been fully processed. One solution to this problem is to block the ability to send new requests until the previous request has been fully processed. Similar approaches have already been implemented in other Chatbots like ChatGPT by OpenAI.

One other problem that has been identified is that the chatbot only accepts requests in the specified language. Otherwise, the chatbot's response is somewhat unintelligible; for example, the English voice is forced to respond in German. This confusion in voice output is due to the limitations of the TTS API, as it provides specific voices for each language. Chapter 10 describes a potential solution approach.

## 9.2 Manual testing by external persons

Six participants were brought in to conduct manual user tests, particularly to investigate usability. The feedback from the test persons can be viewed in Appendix 22. In the following, a summary of the feedback received is presented.

---

The feedback from manual tests was consistently cheerful, highlighting the app's promising potential. However, several areas for improvement have been identified. One recurring concern is the lack of feedback from the chatbot, leaving users uncertain whether it is actively listening or processing their input. Consequently, improving the app's feedback mechanism to display status during interactions is crucial.

Additionally, users expressed difficulties in discovering that the chat window could be dragged upwards. This usability issue should be addressed to improve user navigation within the app.

Another point of criticism is the interruptibility of the avatar. The test subjects emphasized the need for the ability to interrupt the chatbot, especially when it makes incorrect statements or has problems with speech recognition. The lack of this feature forces users to endure the entire chatbot process, which they found somewhat "annoying." Some test subjects even believed that the TTS aspect did not contribute much to the user experience (UX). Therefore, implementing an interrupt or pause function would be very welcome.

The efficiency of the speech recognition and the aesthetically pleasing design were rated positively. However, there were occasional problems with longer processing and waiting times. Fixing these issues will contribute to a more seamless and appealing user experience.

In conclusion, the feedback obtained from test persons offers valuable insights that can be utilized to optimize the app and deliver an improved user experience. Enhancing the visibility of the swipeable chat and refining feedback through the avatar are essential steps. Clearer indications of speech recording and question processing are also warranted. Additionally, introducing options for pausing, restarting, or canceling the chat would address user concerns and contribute to a more user-friendly app.

## 10 Further improvements

In the previous chapter, an analysis revealed that the processing times of the chatbot app sometimes exhibit a certain slowness and thus affect the UX. As a possible solution, the integration of the ChatGPT API and the TTS API as streaming APIs could be considered. This would mean that the response generated by ChatGPT would be continuously sent back to the app instead of waiting until the complete response is generated. Consequently, the response could be sent to the TTS API faster, and the voice response could be generated continuously. This would save processing time and give the chatbot user the impression that the response was generated faster.

Another aspect that shows potential for future improvement concerns overcoming language barriers in connection with the chatbot. A useful further development would be for the chatbot or the STT API to be able to determine the language of the input independently. In this context, there are already approaches that have been implemented by experts in the Whisper API. One possible approach is documented at the following GitHub link <https://github.com/openai/whisper/blob/main/README.md>, updated May 5, 2023, and accessed July 18, 2023. The advantage of such an implementation is that it makes the API much more flexible and can provide responses in arbitrary languages. This is made possible by passing the recognized language to the TTS-API, which can select the appropriate voice for that language. Another aspect regarding the language is that the current application is only available in English. It is advisable to overcome this limitation in the future and make the application multilingual.

Lip synchronization is another essential aspect for modifying the avatar that must be adapted. For the existing 22 Viseme IDs, only four different lip shapes exist at the moment. These lip shapes are also simply displayed consecutively at the moment and not matching the Visemes. Therefore, to ensure complete coverage, another 18 lip shapes should be developed that can also be animated with Rive. The detailed procedure for this is explained in detail in Section 7.1.5.

Perhaps it would be even better and more beneficial to construct a custom GPT model based on a company's specific data in the future. The company could use this model to allow customers to interact with the chatbot avatar instead of a support representative. The avatar would be empowered to answer questions regarding the company's software, website, or general aspects of the business. This approach would represent a potential advancement in customer communication and company interaction.

Due to time constraints, the avatar interruption option was not finalized. Nevertheless, it is advisable to implement this feature to improve the overall experience and give users more control. The option to pause the interaction and resume it later would increase the UX and make the overall experience more positive.

An additional area of use for the chatbot could be integration into an Android Car app or Apple Car Play. Such an implementation would make it possible to use the chatbot while driving.

## 11 Conclusion

In this present bachelor thesis, an automated chatbot was designed and implemented, built on STT and TTS APIs, and integrated into a mobile application. The ChatGPT API was used in the generation of responses for the chatbot. The main objective of the work was to investigate the most suitable STT API and TTS API for implementing this chatbot.

Several criteria were considered when examining the STT APIs, including accuracy, speed, cost, and documentation quality. Each of these criteria played a significant role in evaluating the STT APIs and helped identify their respective advantages and disadvantages. Several criteria were also used to compare the TTS APIs, namely quality of speech output, configuration options, language scope, speed, and cost. The analysis of each of these aspects allowed a differentiated evaluation of the TTS APIs and helped identify the one that best fit the requirements of the chatbot project.

However, in the end, it also turned out that when considering a wide range of aspects, the depth of the analysis for individual criteria was limited. While the broad consideration of the STT and TTS APIs resulted in an overall view, it was not possible to go into the details of individual aspects to the same level. This could potentially lead to some limitations in the detailed evaluation of the APIs. In future work, a more in-depth analysis of individual criteria could be conducted to provide more detailed information and allow for an even more precise evaluation of the APIs. A more focused view could help better understand each API's strengths and weaknesses and make informed decisions that are even better suited to the specific needs of a chatbot project.

Despite this challenge, it was still possible to make an informed decision and identify those STT and TTS APIs that best fit the requirements of the chatbot project. OpenAI's Whisper API proved to be the most optimal solution for speech recognition. This API stood out for its remarkable accuracy and high speed, combined with acceptable cost and well-structured documentation, which enabled the integration of the speech features into the mobile application. After careful consideration, the Microsoft Azure API was selected as the most suitable solution for the TTS conversion. This API achieved consistent results in all measured

---

criteria and offered the remarkable ability to generate Visemes. Together with OpenAI's Whisper API, it formed a robust and viable solution that met the requirements of the chatbot project and enabled the successful integration of the speech features into the mobile application.

An important aspect of this work was also the implementation of the chatbot, integrating the selected STT and TTS APIs into the mobile application. The implementation resulted in a user-friendly chatbot that enabled effective communication with users. In addition, the developed chatbot was tested for usability. Despite the chatbot's successful implementation, there is room for future improvement. Speech recognition and conversion technology are constantly evolving, and new advanced solutions may emerge that offer even more accurate and powerful features. Therefore, it is advisable to continuously follow the developments in this field and regularly update the chatbot to benefit from the latest innovations.

A crucial component when developing and integrating a chatbot into a mobile application is the human factor. It is the human who interacts with the chatbot, and thus the needs and experiences of the users are crucial. The user testing so far has already given some indication that the TTS feature of the chatbot may be perceived as tedious. To further improve the user experience, it is now of great importance to survey a broader range of people and understand their actual needs and preferences. By actively incorporating user voices and requirements into the development process, it is possible to create a chatbot solution that is not only technically capable but also provides an outstanding user experience. Ultimately, this human-centric approach makes the chatbot a valuable tool that enriches the interaction with the mobile application and positively changes the way users interact with technology.

In conclusion, this bachelor thesis has shown that the combination of high-quality STT and TTS APIs provides an effective way to improve user interaction in mobile applications. Selecting the best suitable APIs and successfully integrating them into the chatbot are crucial steps to ensure powerful speech functionality. The lessons learned and experience gained can serve as a guide for future projects looking to develop similar speech applications. As speech technology continues to evolve, new opportunities are opening to design innovative and impactful speech applications that can revolutionize how we interact with technology.

## References

- Alexeyo. (2023, May 12). Speech service quotas and limits - azure cognitive services. Retrieved July 3, 2023, from <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/speech-services-quotas-and-limits>
- Ali, A., & Renals, S. (2018). Word Error Rate Estimation for speech Recognition: e-WER (Vols. 20–24). <https://doi.org/10.18653/v1/p18-2004>
- Amyx, S. (2023, May 22). ChatGPT vs. Traditional Chatbots: Advantages and Differences. Retrieved July 25, 2023, from <https://scottamyx.com/2023/05/23/chatgpt-vs-traditional-chatbots-advantages-and-differences/>
- Apache JMeterTM. (n.d.). Apache JMeterTM. Retrieved July 7, 2023, from <https://jmeter.apache.org/>
- AssemblyAI. (n.d.). AssemblyAI | About. Retrieved July 7, 2023, from <https://www.assemblyai.com/about>
- Bathurst, A. (2020, January 20). Phonemes are the building blocks of words. Retrieved July 20, 2023, from <https://www.babiesandlanguage.com/what-is-a-phoneme/>
- Bernard, J. (2023, June 26). Text-to-Speech and natural language processing. Retrieved July 9, 2023, from <https://typecast.ai/learn/natural-language-processing-tts/#:~:text=Text-to-Speech%20technology%20is,processing%20text%20into%20synthesized%20speech>.
- Bhashkar, K. (2021, December 7). Conversational AI Chatbot using Deep Learning: How Bi-directional LSTM, Machine Reading. . . Medium. Retrieved July 25, 2023, from <https://bhashkarkunal.medium.com>
- Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python (First Edition). O'Reilly Media.

Chapuzet, A. (2022, October 31). Speech Recognition: How it works and what it is made of.

Retrieved June 21, 2023, from <https://vivoka.com/how-to-speech-recognition/>

Deepgram. (n.d.). About us | Deepgram. Retrieved July 7, 2023, from

<https://deepgram.com/about>

DeepLearning.AI. (2023, January 11). Natural language processing (NLP) [a complete guide].

Retrieved May 17, 2023, from <https://www.deeplearning.ai/resources/natural-language-processing/>

DigitalOcean. (n.d.-a). How to Connect to your Droplet with PuTTY on Windows |

DigitalOcean Documentation. Retrieved July 16, 2023, from

<https://docs.digitalocean.com/products/droplets/how-to/connect-with-ssh/putty/>

DigitalOcean. (n.d.-b). How to create a droplet | DigitalOcean Documentation. Retrieved July

16, 2023, from <https://docs.digitalocean.com/products/droplets/how-to/create/>

ElevenLabs. (n.d.). ElevenLabs || Prime Voice AI. Retrieved July 7, 2023, from

<https://beta.elevenlabs.io/about>

Eric-Urban. (2023, April 19). Install and run Speech containers with Docker - Speech service -

Azure Cognitive Services. Retrieved July 16, 2023, from

<https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/speech-container-howto>

Espinosa, M. N. (2023, May 11). State of the art in Voice Cloning: A. . . Retrieved July 10,

2023, from <https://blog.marvik.ai/2023/03/21/state-of-the-art-in-voice-cloning-a-review/>

Firebase Authentication. (2023, July 12). Retrieved July 20, 2023, from

<https://firebase.google.com/docs/auth?hl=en>

Firebase Firestore. (2023, July 18). Retrieved July 20, 2023, from

<https://firebase.google.com/docs/firestore?hl=en>

Firth-Butterfield, K., & Ammanath, B. (2021, November 11). Here's why virtual assistants and chatbots using AI are here to stay. Retrieved July 10, 2023, from <https://www.weforum.org/agenda/2021/11/chatbots-and-other-virtual-assistants-are-here-to-stay-for-good-or-bad/>

Flare. (n.d.). Rive - Bring your apps and games to life with real-time animation. Retrieved July 9, 2023, from <https://flare.rive.app/>

Formula of the word error rate. (n.d.). Retrieved from <https://sonix.ai/packs/media/images/corp/articles/word-error-rate-2017-c5aba7282b39531154f5676a184c7ec4.png>

GitHub. (n.d.). Understanding GitHub Actions - GitHub Docs. Retrieved July 16, 2023, from <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.

Hyperscience. (2023, April 28). Natural language processing (NLP) - what is it and how is it used? Retrieved June 8, 2023, from <https://hyperscience.com/knowledge-base/natural-language-processing/>

IBM. (n.d.-a). What is natural language processing? | IBM. Retrieved May 27, 2023, from <https://www.ibm.com/topics/natural-language-processing>

IBM. (n.d.-b). What is speech recognition? | IBM. Retrieved June 11, 2023, from <https://www.ibm.com/topics/speech-recognition>

iMerit Technology Services. (2022, March 21). The past, present, and future of Speech-to-Text and AI transcription. Retrieved July 10, 2023, from <https://imerit.net/blog/the-past-present-and-future-of-speech-to-text-and-ai-transcription-all-un/a/>

Kanetkar, R. (2023, April 12). Hot AI startup ElevenLabs, founded by ex-Google and palantir staff, is set to raise "8 million at a "00 million valuation. Check out the 14-slide pitch deck it used for its > million pre-seed. Business Insider. Retrieved July 7, 2023, from <https://www.businessinsider.com>

Karagiannakos, S. (2021, May 13). Speech synthesis: A review of the best text to speech architectures with Deep Learning | AI Summer. Retrieved July 10, 2023, from <https://theaisummer.com/text-to-speech/>

Kavlakoglu, E. (2020, November 13). NLP vs. NLU vs. NLG: The differences between three natural language processing concepts. Retrieved July 10, 2023, from <https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/>

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2022). Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3), 3713–3744. <https://doi.org/10.1007/s11042-022-13428-4>

Léwy, N. (2015). Computational psycholinguistics and spoken word recognition in the bilingual and the monolingual. ResearchGate. Retrieved from [https://www.researchgate.net/publication/288990666\\_Computational\\_psycholinguistics\\_and\\_spoken\\_word\\_recognition\\_in\\_the\\_bilingual\\_and\\_the\\_monolingual](https://www.researchgate.net/publication/288990666_Computational_psycholinguistics_and_spoken_word_recognition_in_the_bilingual_and_the_monolingual)

Lutkevich, B., & Burns, E. (2023, January 20). Natural Language Processing (NLP). Retrieved July 25, 2023, from <https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP>

Markowitz, D. (2022, April 15). Google cloud speech AI guide. Retrieved May 12, 2023, from <https://cloud.google.com/blog/products/ai-machine-learning/your-ultimate-guide-to-speech-on-google-cloud>

Martin, R. C. (2012, August 13). Clean Coder Blog. Retrieved July 11, 2023, from <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

McAvan, H. (2023). What are the 44 Phonemes in the English Language? Time for Phonics | DfE Validated Phonics Scheme. Retrieved from <https://timeforphonics.co.uk/what-are-the-44-phonemes-in-the-english-language/>

OpenAI. (n.d.). OpenAI Platform. Retrieved July 6, 2023, from  
<https://platform.openai.com/docs/guides/rate-limits/overview>

Patwardhan, N., Marrone, S., & Sansone, C. (2023). Transformers in the Real World: A survey on NLP applications. *Information*, 14(4), 242. <https://doi.org/10.3390/info14040242>

Perdigão, F. (2023, February 10). Speech Recognition: How You Can Talk to Your Chatbot. Retrieved July 10, 2023, from <https://www.visor.ai/blog/speech-recognition-in-chatbots/>

PlayHT. (n.d.). About Us. Retrieved July 7, 2023, from <https://play.ht/about-us/>

Pruneski, J. A., Pareek, A., Nwachukwu, B. U., Martin, R. K., Kelly, B. T., Karlsson, J., . . . Williams, R. J. (2022). Natural language processing: using artificial intelligence to understand human language in orthopedics. *Knee Surgery, Sports Traumatology, Arthroscopy*, 31(4), 1203–1211. <https://doi.org/10.1007/s00167-022-07272-0>

PuTTY. (n.d.). Retrieved July 16, 2023, from <https://www.putty.org/>

Rajnerowicz, K. (2023, July 6). NLP Chatbot: Complete Guide & How to Build Your Own. Retrieved July 10, 2023, from <https://www.tidio.com/blog/nlp-chatbots/>

Rashad, M., El-Bakry, H. M., Isma, R. I., & Mastorakis, N. E. (2010). An overview of text-to-speech synthesis techniques. ResearchGate. Retrieved from [https://www.researchgate.net/publication/228673642\\_An\\_overview\\_of\\_text-to-speech\\_synthesis\\_techniques](https://www.researchgate.net/publication/228673642_An_overview_of_text-to-speech_synthesis_techniques)

Rella, S. R. (2023, June 12). What is Automatic Speech Recognition? | NVIDIA Technical Blog. Retrieved July 25, 2023, from <https://developer.nvidia.com/blog/essential-guide-to-automatic-speech-recognition-technology/>

rive | Flutter Package. (2023, June 27). Retrieved July 8, 2023, from

<https://pub.dev/packages/rive>

Saleem, S. (2023, June 21). What is DigitalOcean? Retrieved July 12, 2023, from

<https://www.cloudways.com/blog/what-is-digital-ocean/>

Sanghvi, H. (2023, June 13). The AI-Powered application of NLP (Natural Language

Processing) used in industries. Retrieved July 15, 2023, from

<https://syndelltech.com/applications-of-nlp-in-business/>

Shah, A. A., Ansari, A. W., & Das, L. (2023). BiLingual Text to Speech Synthesis System for

Urdu and Sindhi. ResearchGate. Retrieved from

[https://www.researchgate.net/publication/255646057\\_BiLingual\\_Text\\_to\\_Speech\\_Synthesis\\_System\\_for\\_Urdu\\_and\\_Sindhi](https://www.researchgate.net/publication/255646057_BiLingual_Text_to_Speech_Synthesis_System_for_Urdu_and_Sindhi)

Shaywitz, B. A., Weiss, L. G., Saklofske, D. H., & Shaywitz, S. E. (2016). Translating Scientific

Progress in Dyslexia into Twenty-first Century Diagnosis and Interventions. In Elsevier

eBooks (pp. 269–286). <https://doi.org/10.1016/b978-0-12-404697-9.00009-1>

Shrivastav, A. (2023, July 4). Generative AI Chatbots: Gamechanger or Doomslayer to

intelligent conversations. Retrieved July 27, 2023, from

<https://www.kellton.com/kellton-tech-blog/generative-ai-chatbots-gamechanger-or-doomslayer-to-intelligent-conversations>

Simon Says. (n.d.). What is ASR? How Does it Work? Our In-Depth 2023 Guide. Retrieved July

25, 2023, from <https://www.simonsaysai.com/blog/automatic-speech-recognition>

Skype. (n.d.). Translate by Voice or Text in Real-Time with Skype Translator | Skype. Retrieved

July 10, 2023, from <https://www.skype.com/en/features/skype-translator/>

src\_08. Speech Synthesis. (n.d.). Retrieved July 10, 2023, from <https://wikidocs.net/191144>

Vahidsafa, J. (2022, November 10). How Duolingo Animates Its World Characters. Retrieved

July 7, 2023, from <https://blog.duolingo.com/world-character-visemes/>

---

Wang, C., Chen, S., Wu, Y., Zhang, Z., Zhou, L., Liu, S., . . . Wei, F. (2023). Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2301.02111>

Wiggers, K. W. (2022, July 14). Flush with new cash, AssemblyAI looks to grow its AI-as-a-service business. Retrieved July 7, 2023, from <https://techcrunch.com/2022/07/14/flush-with-new-cash-assemblyai-looks-to-grow-its-ai-as-a-service-business/>

Yulin-Li. (2023, July 6). Get facial position with viseme - Azure Cognitive Services. Retrieved from <https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-speech-synthesis-viseme>

Zen, H. (n.d.). Deep Learning in Speech Synthesis [Slide show]. Retrieved from <https://static.googleusercontent.com/media/research.google.com/de//pubs/archive/41539.pdf>

Zhang, Z., Zhou, L., Wang, C., Chen, S., Wu, Y., Liu, S., . . . Wei, F. (2023). Speak Foreign Languages with Your Own Voice: Cross-Lingual Neural Codec Language Modeling. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2303.03926>

## List of Appendices

Appendix 1: Methods to Generate Audio in TTS.....	3
Appendix 2: Reference text for STT comparison .....	4
Appendix 3: GUIs of the app for recording the tests. ....	5
Appendix 4: User Guide to record the test recordings. ....	6
Appendix 5: Data from SINF .....	7
Appendix 5.1: Display the number of requests made to the SINF. ....	7
Appendix 5.2: Display of the number of incidents of SINF. ....	7
Appendix 6: Python code to test the accuracy of STT. ....	8
Appendix 7: Python code to test the voice quality. ....	9
Appendix 8: Voice quality rating scales .....	10
Appendix 8.1: Intelligibility scale .....	10
Appendix 8.2: Naturalness scale.....	10
Appendix 9: Detailed analysis of the processing speed of STT .....	11
Appendix 10: Detailed analysis of the scalability of STT APIs .....	14
Appendix 11: Email from AssemblyAI .....	17
Appendix 12: Results of the analysis of the different documentation.....	18
Appendix 13: Results of the voice quality analysis .....	20
Appendix 13.1: Bar chart of data from Table XY.....	20
Appendix 13.2: Bar chart of data from Table XY.....	20
Appendix 14: Configurations of the different TTS APIs.....	21
Appendix 15: Email from OpenAI.....	22
Appendix 16: Generated text with the GPT models.....	23
Appendix 17: UML diagrams from the Flutter application.....	24

---

Appendix 17.1: Use case diagram.....	24
Appendix 17.2: Sequence diagram.....	25
Appendix 18: Product Backlog .....	26
Appendix 19: Extract from the Kanban board.....	27
Appendix 20: Versions of the used flutter packages .....	28
Appendix 21: User Guide.....	29
Appendix 22: User Feedback.....	34
Appendix 23: Project Management .....	35
Appendix 23.1: Timetable.....	35
Appendix 23.2: Invested hours per activity. ....	35

## Appendix 1: Methods to Generate Audio in TTS

**Table 1**

Synthesis Method	Description	Limitations	References
<b>Articulatory Synthesis</b>	Simulates the behavior of human articulators such as lips, tongue, and larynx to produce speech.	It is challenging to model the behavior of articulators, resulting in lower speech quality compared to other methods.	Shah, Ansari, & Das, 2023
<b>Formant Synthesis</b>	Generates speech using a set of rules developed by linguists to mimic linguistic features. It may sound less natural and exhibit artifacts. Difficulty in specifying synthesis rules precisely.	Speech may sound less natural and exhibit artifacts. Difficulty in specifying synthesis rules precisely.	Shah, Ansari, & Das, 2023
<b>Concatenative Synthesis</b>	Based on concatenating previously recorded speech units stored in a database. Units range from complete sentences to syllables and are usually recorded by speakers.	Requires an extensive recording database to cover all possible combinations of speech units.	Rashad et al., 2010
<b>Neural Networks</b>	The current predominant speech synthesis method uses architectures like WaveNet, DeepVoice3, Tacotron2, and FastSpeech.	Significant increase in speech quality in terms of intelligibility and naturalness.	Karagiannakos, 2021; Rashad et al., 2010

Note: All data in this table were collected at 16:15 on June 21.

Source: Own representation.

## Appendix 2: Reference text for STT comparison

### German

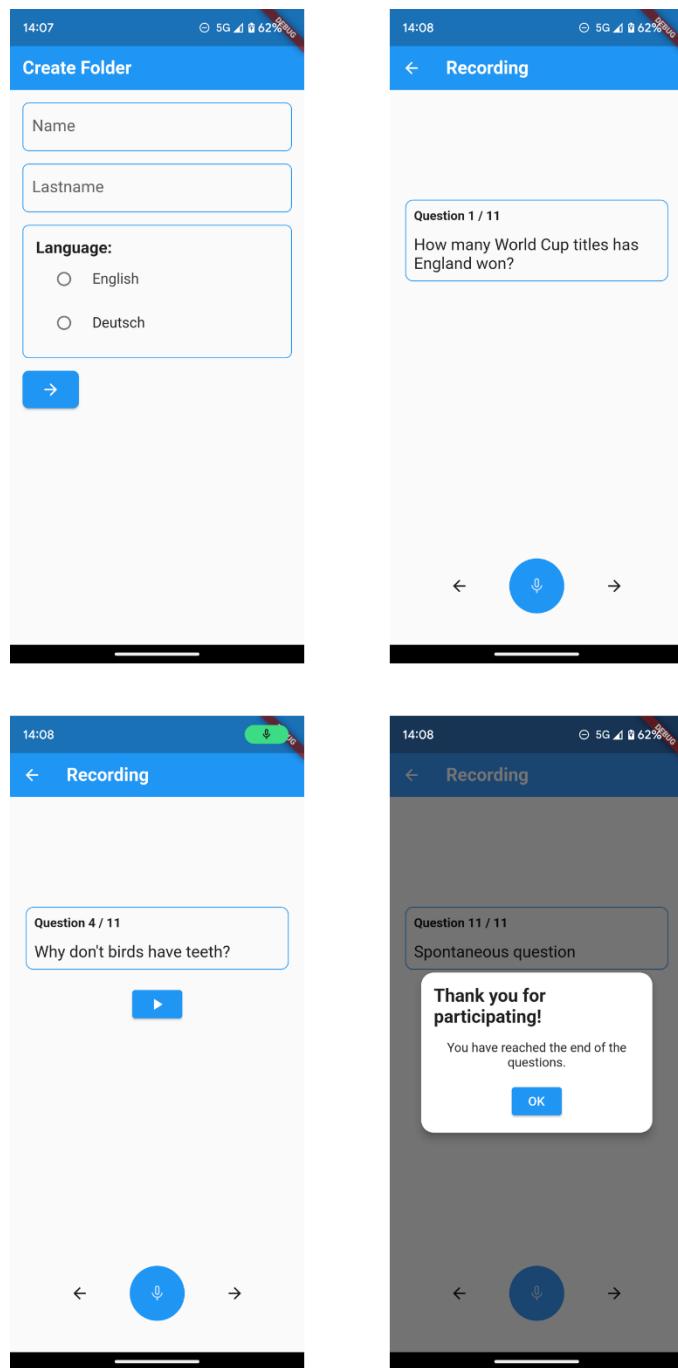
- 1 Wie viele Weltmeistertitel hat Deutschland gewonnen?
- 2 Wer ist der Rekordtorschütze der Bundesliga?
- 3 Wie funktioniert die Blockchain-Technologie und welche Auswirkungen hat sie auf die Finanzbranche?
- 4 Warum haben Vögel keine Zähne?
- 5 Warum hat die Banane eine gebogene Form?
- 6 Welche Auswirkungen hat der Klimawandel auf die Artenvielfalt in den Regenwäldern und welche Maßnahmen können ergriffen werden, um diese zu schützen?
- 7 Was sind die Vor- und Nachteile von Cloud Computing im Vergleich zur lokalen Datenspeicherung?
- 8 Wie viele Planeten gibt es in unserem Sonnensystem?
- 9 Erkläre die Grundprinzipien der künstlichen Intelligenz und wie sie in verschiedenen Bereichen angewendet werden.
- 10 Wie heißt die berühmteste Brücke in San Francisco?
- 11 Spontane Frage

### English

- 1 How many World Cup titles has Germany won?
- 2 Who is the record goal scorer in the Bundesliga?
- 3 How does blockchain technology work, and what impact does it have on the financial industry?
- 4 Why don't birds have teeth?
- 5 Why does the banana have a curved shape?
- 6 What is the impact of climate change on rainforest biodiversity, and what measures can be taken to protect it?
- 7 What are the advantages and disadvantages of cloud computing compared to local data storage?
- 8 How many planets are there in our solar system?
- 9 Explain the basic principles of artificial intelligence and how they are applied in different fields.
- 10 What is the name of the most famous bridge in San Francisco?
- 11 Spontaneous question

## Appendix 3: GUIs of the app for recording the tests.

Figure 1



Source: Own illustration. Screenshots of the application.

## Appendix 4: User Guide to record the test recordings.

Liebe Testperson,

Vielen Dank, dass Sie mich bei meinem Spracherkennungs-Test unterstützen. Ihre Teilnahme ist äusserst wertvoll, um die Leistung der Speech-to-Text APIs zu bewerten und die besten Optionen für einen Chatbot auszuwählen. Speech-to-Text APIs sind Programme, die gesprochene Sprache in Text umwandeln. Sie dienen als Grundlage für die Spracherkennungsfunktionen des Chatbots. Ein Chatbot ist ein computerbasiertes Programm, das menschenähnliche Konversationen führen kann.

Bitte sprechen Sie die Fragen und Aussagen auf natürliche Weise aus, wie Sie es im Alltag tun würden. Es ist nicht erforderlich, besonders deutlich oder übertrieben zu sprechen. Dies hilft mir, die Spracherkennungsfähigkeiten der APIs unter realistischen Bedingungen zu bewerten. Die Sprachdistanz vom Mund zum Smartphone sollte eine natürliche Haltung sein. Ungefähr wie auf dem Bild nebenan.



### Anleitung zur Aufnahme

#### 1. Aufnahme-App herunterladen:

Laden Sie die App AA TEST APP auf Ihr Smartphone herunter und installieren Sie sie.

#### 2. Eingeben der Kontaktdaten

Geben Sie Ihre Kontaktdaten an und wählen Sie die entsprechende Sprache, in der Sie die Fragen einsprechen wollen.

#### 3. Aufnahme durchführen:

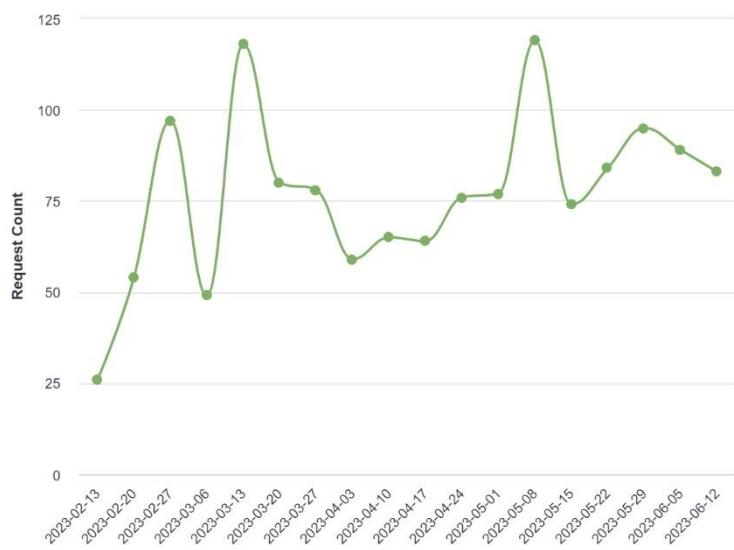
Klicken und halten Sie den Aufnahme-Button in der App gedrückt, um mit der Aufnahme zu beginnen. Halten Sie das Smartphone in angemessenem Abstand und fangen Sie an die angezeigte Frage abzulesen. Sobald sie die Aufnahme beendet haben, klicken Sie auf den Pfeil auf der rechten Seite, um zur nächsten Frage zu gelangen.

Ich danke Ihnen nochmals für Ihre Teilnahme und Ihre Unterstützung. Bei Fragen oder Unklarheiten stehe ich Ihnen gerne zur Verfügung.

## Appendix 5: Data from SINF

### Appendix 5.1: Display the number of requests made to the SINF.

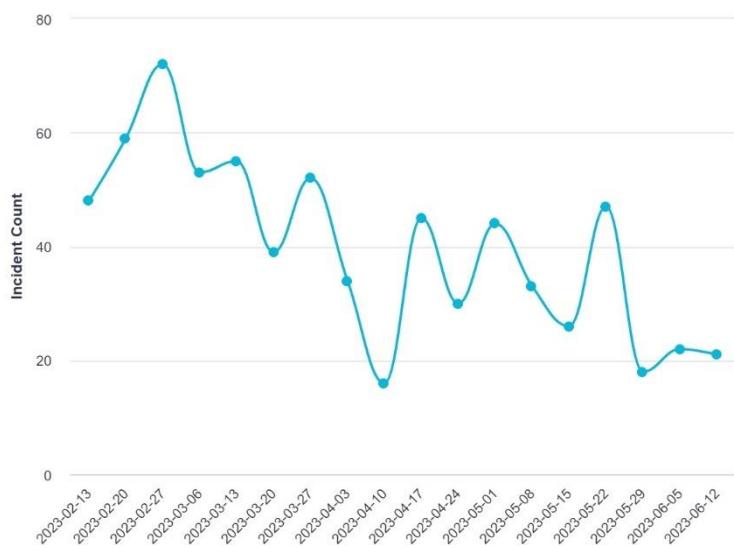
**Figure 2**



Source: The data were provided by the SINF.

### Appendix 5.2: Display of the number of incidents of SINF.

**Figure 3**



Source: The data were provided by the SINF.

## Appendix 6: Python code to test the accuracy of STT.

Figure 4

```

test_accuracy.py

1 import os
2 import requests
3 from unidecode import unidecode
4 from jiwer import wer
5
6 # Define the base URL for your Flask server
7 base_url = "http://192.168.137.1:5001"
8
9 # Define the API endpoints for each speech-to-text API
10 api_endpoints = [
11     "api": "whisper",
12     "api": "googlesT",
13     "api": "deepparam",
14     "api": "azuresST",
15     "api": "assemblyAI",
16 ]
17
18 # Get the current directory
19 current_directory = os.getcwd()
20
21 # Define the folder name where the audio files are stored
22 folder_name = "audio_folder"
23
24 # Define the file name of the ground truth texts
25 ground_truth_file = "ground_truth_en.txt"
26
27 # Construct the folder path
28 folder_path = os.path.join(current_directory, folder_name)
29
30 # Construct the ground truth file path
31 ground_truth_path = os.path.join(current_directory, ground_truth_file)
32
33 # Open the ground truth file and read the texts
34 ground_truth_texts = []
35 with open(ground_truth_path, 'r') as file:
36     for line in file:
37         text = line.strip()
38         ground_truth_texts.append(text)
39
40 # Dictionary to store the total WER and count for each API
41 wer_totals = {api: 0 for api in api_endpoints}
42 wer_counts = {api: 0 for api in api_endpoints}
43
44 # Get the first audio file name
45 first_audio_file = next(iter(sorted(os.listdir(folder_path))), None)
46 if first_audio_file is not None:
47     output_filename = first_audio_file.rsplit("_", 1)[0] + "_output.txt"
48 else:
49     output_filename = "output.txt"
50
51 # Open the output file in write mode
52 output_file = open(output_filename, "w")
53
54 # Iterate through each API and audio file
55 for api, endpoint in api_endpoints.items():
56     api_url = base_url + endpoint
57
58     output_file.write(api_url + "\n")
59
60     for i, filename in enumerate(sorted(os.listdir(folder_path))):
61         audio_file = os.path.join(folder_path, filename)
62
63         if not os.path.isfile(audio_file):
64             continue
65
66         # Prepare the request payload
67         with open(audio_file, 'rb') as file:
68             files = {'file': file}
69             response = requests.post(api_url, files=files)
70
71         # Process the response
72         if response.status_code == 200:
73
74             # Remove leading/trailing whitespaces and double quotes
75             recognized_text = response.text.strip().strip('"')
76
77             # Convert umlauts
78             converted_text = unidecode(recognized_text)
79
80             ground_truth_text = ground_truth_texts[i] if i < len(ground_truth_texts) else ""
81
82             # Calculate the Word Error Rate (WER)
83             error_rate = wer(ground_truth_text, converted_text)
84
85             # Accumulate the WER and count for the API
86             wer_totals[api] += error_rate
87             wer_counts[api] += 1
88
89             print(f"API: {api}, File: {filename}, Text: {response.text}, WER: {error_rate}")
90             output_file.write(f"API: {api}, File: {filename}, Text: {response.text}, WER: {error_rate}\n")
91
92             print(f"API: {api}, File: {filename}, Error: {response.text}")
93             output_file.write(f"API: {api}, File: {filename}, Error: {response.text}\n")
94
95         # Calculate and print the average WER for each API
96         if wer_counts[api] > 0:
97             average_wer = wer_totals[api] / wer_counts[api]
98             output_file.write(f"Average WER: {average_wer}\n")
99
100 # Close the output file
101 output_file.close()

```

Source: Own representation. Created with codebeautify.org.

## Appendix 7: Python code to test the voice quality.

Figure 5

```

test_voice_quality.py

1 import requests
2 import os
3
4 def test_text_to_speech(api_base_url, endpoint_path, text, voice, text_length):
5     url = api_base_url + endpoint_path
6
7     headers = {
8         'Content-Type': 'application/json',
9     }
10
11    data = {
12        'text': text,
13        'voice': voice
14    }
15
16    try:
17        response = requests.post(url, json=data, headers=headers)
18        if response.status_code == 200:
19            content_type = response.headers.get('Content-Type')
20            if 'audio/mp3' in content_type:
21                filename = f"{endpoint_path.replace('/', '')}_{(voice.replace(' ', '_'))}_{(text_length)}.mp3"
22            elif 'audio/wav' in content_type:
23                filename = f"{endpoint_path.replace('/', '')}_{(voice.replace(' ', '_'))}_{(text_length)}.wav"
24            else:
25                print(f"Unsupported file format in API response for '{text}' using '{voice}' voice.")
26                return
27
28            filepath = os.path.join(output_dir, filename)
29            with open(filepath, 'wb') as file:
30                file.write(response.content)
31            print(f"Success! Speech generated for '{text}' using '{voice}' voice. File saved as
32            '{filename}'")
33        else:
34            print(f"Failed to generate speech. Error: {response.status_code} - {response.text}")
35    except requests.exceptions.RequestException as e:
36        print(f"An error occurred: {e}")
37
38 # Define the common base URL for all APIs
39 api_base_url = 'http://192.168.137.1:5001'
40
41 # Define the API endpoint paths for each text-to-speech service
42 api_endpoints = {
43     'API1': '/googletts',
44     'API2': '/azuretts',
45     'API3': '/playht',
46 }
47
48 # Define the voices for each API
49 api_voices = {
50     'API1': [
51         'de-DE-Standard-A',
52         'de-DE-Standard-B',
53     ],
54     'API2': [
55         'de-DE-BerndNeural',
56         'de-DE-KatjaNeural',
57     ],
58     'API3': [
59         'fr-FR-YvesNeural',
60         'fr-FR-EloiseNeural',
61     ],
62 }
63
64 # Define the texts with different lengths
65 texts = {
66     '200_tokens': "La vie est une aventure remplie de surprises et de défis. Chaque jour apporte son lot d'opportunités et d'expériences uniques. Il est important de rester ouvert d'esprit et de saisir ces occasions pour grandir et se développer. Les obstacles peuvent sembler insurmontables parfois, mais c'est en les surmontant que nous découvrons notre véritable force intérieure. Les relations humaines jouent également un rôle essentiel dans notre épanouissement. Elles nous permettent de partager nos joies, nos peines et de construire des souvenirs précieux. Enfin, il est essentiel de prendre soin de soi, tant sur le plan physique que mental. L'équilibre entre le travail, les loisirs et le repos est crucial pour maintenir notre bien-être global. En résumé, la vie est une aventure passionnante qui mérite d'être vécue pleinement, avec optimisme et gratitude."
67 }
68
69 # Specify the directory for storing the response files
70 output_dir = 'response_files'
71 os.makedirs(output_dir, exist_ok=True)
72
73 # Iterate over each API and test different texts, voices, and text lengths
74 for api_name, endpoint_path in api_endpoints.items():
75     print(f"Testing {api_name}...")
76     voices = api_voices.get(api_name, [])
77     for text_length, text in texts.items():
78         for voice in voices:
79             test_text_to_speech(api_base_url, endpoint_path, text, voice, text_length)
80     print(f"Testing {api_name} complete.\n")

```

Source: Own representation. Created with codebeautify.org.

## Appendix 8: Voice quality rating scales

### Appendix 8.1: Intelligibility scale

Table 2

Grading	Description
1	Unintelligible - The speech output is difficult or impossible to understand.
2	Moderately intelligible - Intelligibility is limited because words or parts of sentences are unclear or ambiguous.
3	Average Understandable - Most words and phrases are understandable, but occasional ambiguities exist.
4	Well understandable - The speech output is primarily clear and understandable, only rarely do ambiguities occur.
5	Very intelligible - Intelligibility is very high, with only occasional minor comprehension problems.
6	Excellent Understandable - The voice output is perfectly understandable, with no ambiguities at all.

Source: Own representation. Created by ChatGPT.

### Appendix 8.2: Naturalness scale

Table 3

Grading	Description
1	Artificial - The voice output sounds very mechanical and unnatural.
2	Clearly recognizable - There are signs of artificiality, but the voice is mostly understandable.
3	Average Natural - The voice sounds mostly natural, but there are still some signs of artificiality.
4	Natural - The voice sounds mostly natural, with only minor hints of artificiality.
5	Very Natural - The voice output sounds almost like a real human voice, barely distinguishable from natural voice production.
6	Completely Natural - The voice output is natural and indistinguishable from a human voice.

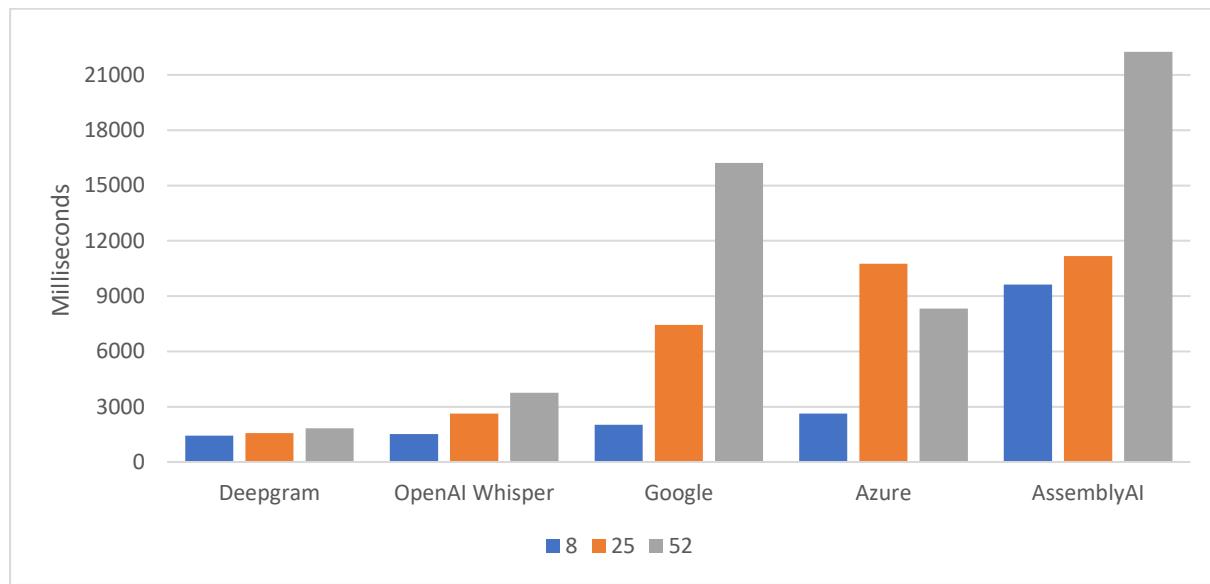
Source: Own representation. Created by ChatGPT.

## Appendix 9: Detailed analysis of the processing speed of STT

The bar chart in Figure 6 shows that AssemblyAI had the most extended processing times compared to the other APIs, regardless of audio length. Deepgram, on the other hand, showed the shortest processing time for all audio files. It was notable that Deepgram produced fast results regardless of audio length. OpenAI Whisper performed well overall but could not keep up with Deepgram for the longer audio files. An interesting observation with Google STT and Microsoft Azure STT was that they showed a significant drop in performance as audio length increased. Notably, Azure had faster speed when processing files with a length of 52 seconds compared to files with a median length of 25 seconds.

**Figure 6**

Bar chart: Average audio length per API

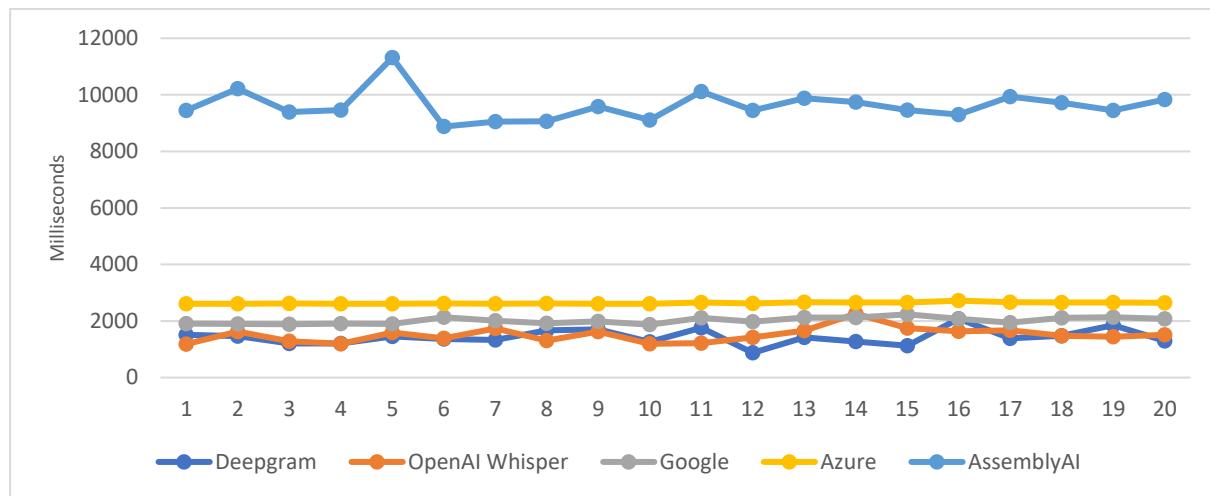


Source: Own illustration.

Line graph 7 gives an additional overview of the processing speed of Speech-to-Text APIs for all twenty calls with an audio length of 8 seconds. A detailed graph analysis revealed no clear spikes or significant fluctuations in processing times. This indicates that each API examined performed consistently when processing 8-second audio files.

**Figure 7**

Line chart: Speed of STT APIs for all 20 calls for an audio length of 8 seconds.



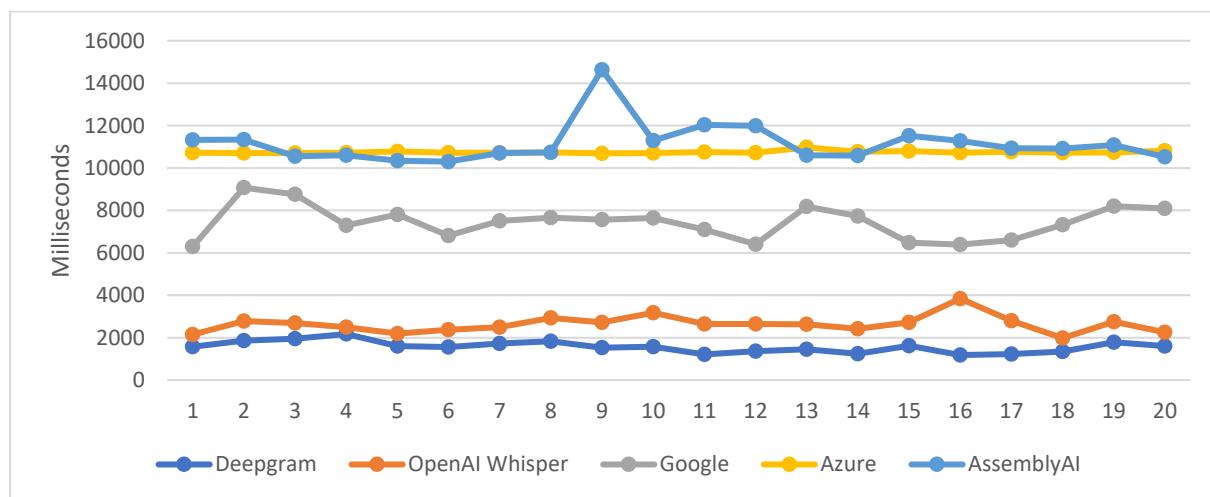
Note: Network specifications: The download speed was 302.45 Mbps, the upload speed was 271.81 Mbps, and the ping value was 5. Measured with <https://www.speedtest.net>

Source: Own illustration.

This line diagram 8 illustrated that OpenAI and Deepgram's APIs achieved continuous and stable performance for an audio file length of 25 seconds. Google's API results were within the average of the five APIs tested. The Google API did not show consistent results across all twenty queries performed. In contrast, the Azure API produced consistent but slower results than the Deepgram, OpenAI, and Google APIs.

**Figure 8**

Line chart: Speed of STT APIs for all 20 calls for an audio length of 25 seconds.



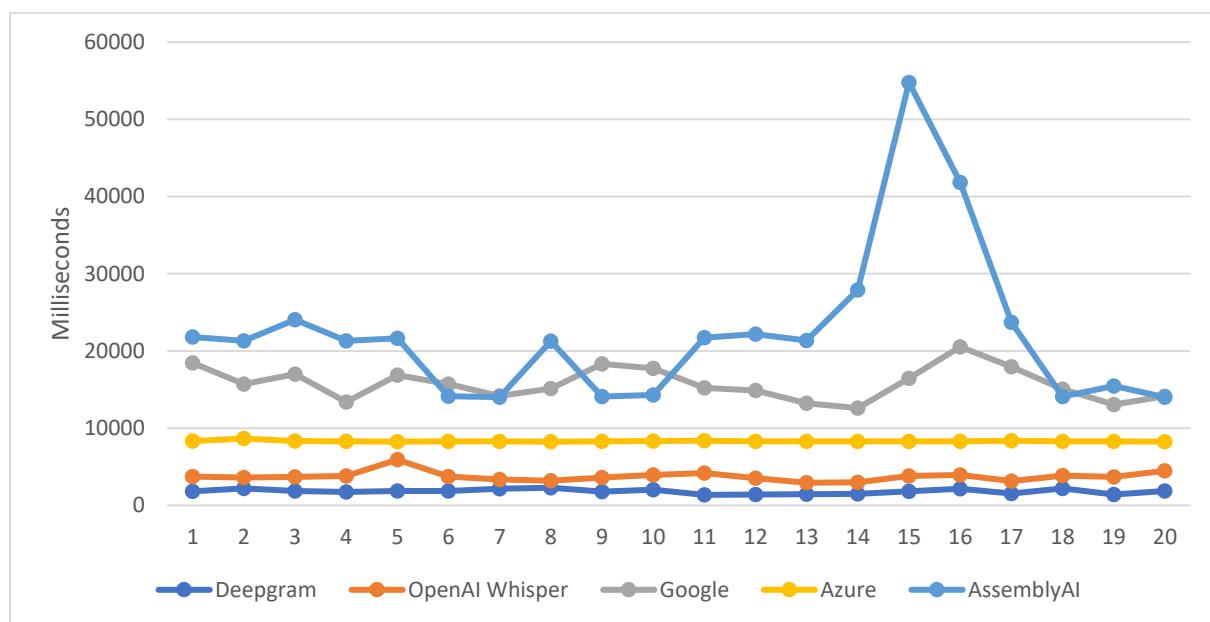
Note: Network specifications: The download speed was 297.71 Mbps, the upload speed was 259.52 Mbps, and the ping value was 5. Measured with <https://www.speedtest.net>

Source: Own illustration.

In the case of the 52-second audio file, a similar observation regarding line diagram 9 was also noted for the 25-second audio file. It was found that the performance of the AssemblyAI and Google APIs was less stable compared to other APIs. In addition, the AssemblyAI API exhibited significant outliers.

**Figure 9**

Line chart: Speed of STT APIs for all 20 calls for an audio length of 52 seconds.



Note: Network specifications: The download speed was 276.51 Mbps, the upload speed was 235.51 Mbps, and the ping value was 6. Measured with <https://www.speedtest.net>

Source: Own illustration.

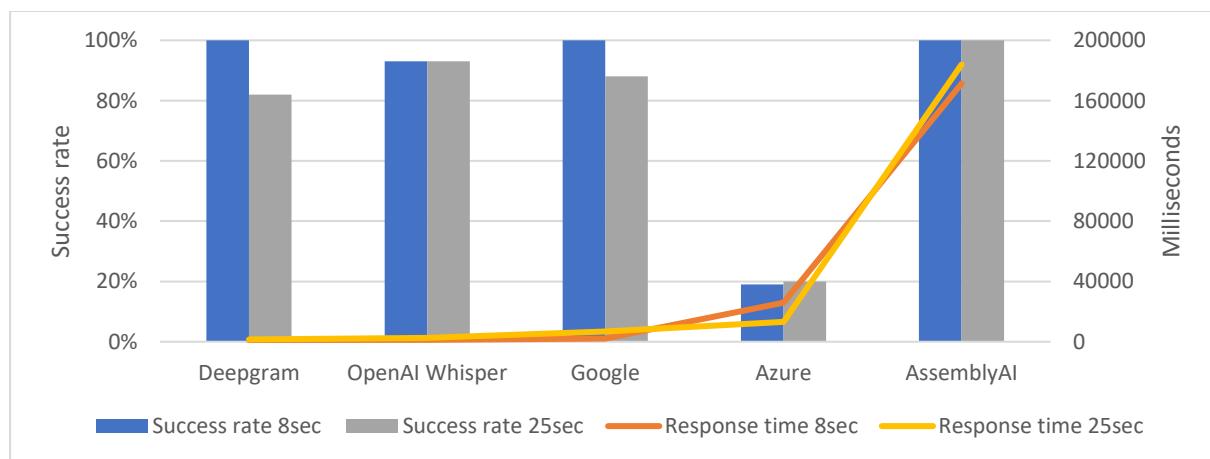
## Appendix 10: Detailed analysis of the scalability of STT APIs

The available results in chart 10 illustrated that the APIs of Deepgram, Google, and AssemblyAI had a success rate of 100% under a high load of 100 API requests and an audio duration of 8 seconds. However, for an audio duration of 25 seconds, only AssemblyAI achieved a 100% success rate. However, AssemblyAI's response times were significantly long, with 171296 milliseconds for 8 seconds and 183939 milliseconds for 25 seconds. This could be attributed to AssemblyAI's API rate cap, which only allowed a limited number of requests per minute (RPM) (see Appendix 11). Instead of an error message, a waiting list was created for all requests, resulting in high response times.

In comparison, OpenAI Whisper showed a consistent success rate of 93% in both time intervals. Azure showed a 19% success rate after 8 seconds and a 20% success rate after 25 seconds. As shown in Graph 10, the response times were high at 25904 milliseconds and 13179 milliseconds, respectively. However, it should be noted that the Azure API tariff tested could also not handle many concurrent RPMs (Alexeyo, 2023). To illustrate the other APIs and for a more detailed overview, the AssemblyAI API was not included in diagram 11. Furthermore, diagram 11 and the results show that the response times of Whisper and Deepgram remained constant even with multiple and faster API requests.

**Figure 10**

Overview of the scaling tests for 100 requests with the audio lengths of 8, 25 sec

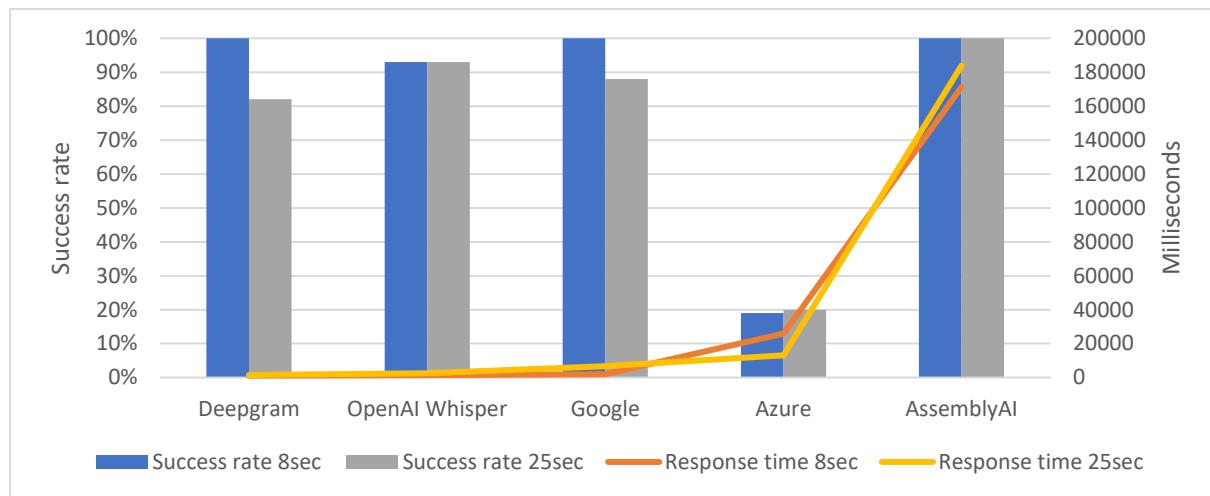


Note: Network specifications: The download speed was 276.51 Mbps, the upload speed was 235.51 Mbps, and the ping value was 6. Measured with <https://www.speedtest.net>

Source: Own illustration.

**Figure 11**

Scaling tests for 100 requests with the audio lengths of 8, 25 sec w/o AssemblyAI



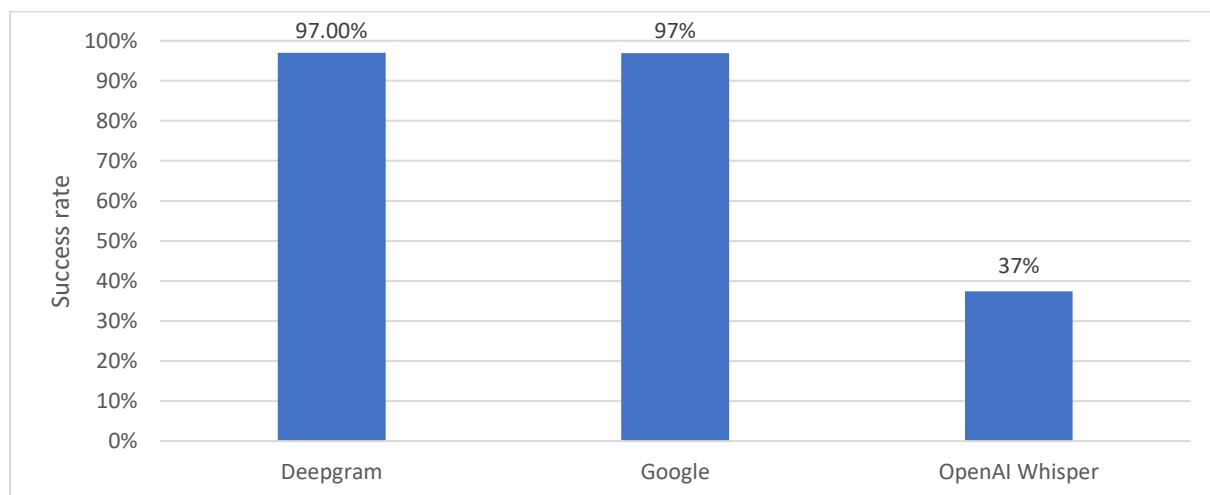
Note: Network specifications: The download speed was 276.51 Mbps, the upload speed was 235.51 Mbps, and the ping value was 6. Measured with <https://www.speedtest.net>

Source: Own illustration.

In the presented Figure 12, the analysis of the results of 1000 requests to the API was illustrated. This analysis revealed that Deepgram had an outstanding success rate of 97%. This implied that 97% of the 1000 requests were successfully recognized and transcribed. In parallel, Google also achieved a 97% success rate. In contrast, OpenAI Whisper revealed a success rate of only 37%, which was significantly lower than the other two APIs.

**Figure 12**

Overview of the success rate with 1000 requests to the API

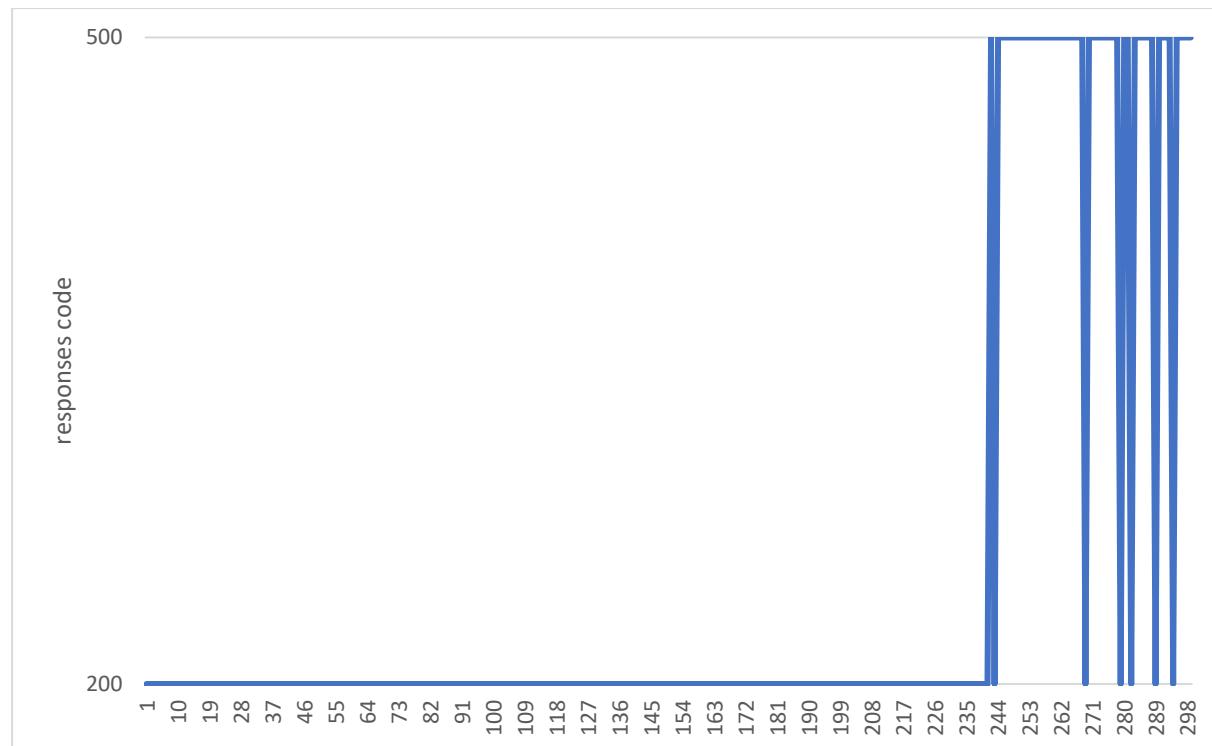


Note: Network specifications: The download speed was 276.51 Mbps, the upload speed was 235.51 Mbps, and the ping value was 5. Measured with <https://www.speedtest.net>

Source: Own illustration.

A detailed examination of OpenAI's Whisper API requests revealed that error messages only appeared after approximately 240 requests. This observation is also illustrated in Figure 13. This could be explained by OpenAI's restriction to a maximum of 50 RPM (OpenAI, n.d.)

**Figure 13**  
Detailed view of the error rate of OpenAI



Note: A measurement was made to determine when the API response code changed from 200 to 500.

Source: Own illustration.

## Appendix 11: Email from AssemblyAI

20.07.23, 11:06

Gmail - You're approaching your monthly Free Tier limit



Natal Bumann &lt;nbumann97@gmail.com&gt;

### You're approaching your monthly Free Tier limit

1 message

AssemblyAI <support@assemblyai.com>  
To: nbumann97@gmail.com

6 June 2023 at 06:47



Hi there,

We wanted to let you know that **you're approaching your Free Tier limit**. On the Free Tier, you can transcribe 3 hours of audio per month for free. Once you hit 3 hours, your account will be temporarily suspended until next month.

#### Upgrade your account for:

- Higher concurrency limit - Transcribe up to 32 audio or video files in parallel
- Access to premium features like Speaker Labels, Topic Detection, PII Redaction, Key Phrase Extraction, Real-Time Streaming Transcription
- 24x7 customer support via chat, Slack, and email
- Detailed reports on your usage and spend

[Upgrade account](#)

If you have any questions, you can email [support@assemblyai.com](mailto:support@assemblyai.com) - or reply directly to this email.

AssemblyAI

PS - If you don't want to get these alerts, you can unsubscribe below!



[Unsubscribe - Unsubscribe Preferences](#)

<https://mail.google.com/mail/u/0/?ik=307a5b83c8&view=pt&search=all&permthid=thread-f:1767927277106306315&simpl=msg-f:1767927277106...> 1/2

## Appendix 12: Results of the analysis of the different documentation

**Table 4**

Results of availability of developer documentation, sample code, and tutorials.

API	Points	Comment
<b>Deepgram</b>	9	Es gibt so Missionen zu erledigen, dass man Deepgram besser kennenlernen kann, sehr ausführlich und übersichtliche Dokumentation, gibt auch einen Blog wo viele Sachen beschrieben und erklärt werden, es gibt einen Playground, wo man die API testen kann, gibt Codebeispiele
<b>Google STT</b>	6	ähnlich wie Azure aber ein bisschen besser, man findet schneller die Dokumentation, man findet Tutorials und auch Beispielcode für verschiedene Programmiersprachen
<b>Microsoft Azure STT</b>	5	Nicht übersichtlich, viel Text und viele Unterpunkte, auf den ersten Blick ein grosses Chaos, man ist eigentlich schneller bei Informationen, wenn man über Google sucht, Sample für STT nicht einfach zu finden, Tutorial ist gut, Man findet nach langem Suchen das Speech Studio dieses ist nicht schlecht aufgebaut und man findet dort Informationen, die man brauchen kann, überall wird halt irgendwo auf andere Microsoft Produkte verwiesen dies macht es sehr unübersichtlich
<b>OpenAI Whisper</b>	8	Übersichtlich, nicht so ausführlich da momentan noch Beta-Version der Doku dies ist aber sehr gut gekennzeichnet, Anhand der Dokumentation ist man fähig die API schnell einzubinden, Beispielcode hat es auch, Tutorial ist kurzgehalten aber wenig ausführlich, Nicht für so viele Programmiersprachen ein Guide, man findet aber ausführliche Erklärungen über das Modell etc. da Open-Source
<b>AssemblyAI</b>	7	Sehr übersichtliche Seite, man findet die Dokumentation sehr schnell, hat einen Schnellstart, um direkt zu starten, hat viele verschiedene Anleitungen für verschiedene Situationen, man hat Beispielcode für verschiedene Programmiersprachen, Tutorials sind sehr gut erklärt, hat eine Suchfunktion (die einigermassen gut funktioniert)

Source: Own representation.

**Table 5**

Shows the results for the category Well-structured and clear API documentation.

API	Points	Comment
<b>Deepgram</b>	10	Es gibt eine spezifisch API-Dokumentation, die sehr ausführlich ist, von Authentifizierung bis Error-Meldung alles vorhanden, ist auch sehr übersichtlich gestaltet, gut gemacht, sogar wo man direkt Abfragen testen kann,
<b>Google STT</b>	6	es eine spezifische API-Dokumentation vorhanden, leider findet man sie nicht so schnell, Dokumentation ist aber ausführlich, man findet die Error Meldung und alles Nötige
<b>Microsoft Azure STT</b>	6	Eine spezifische API-Dokumentation ist zu finden und diese ist auch einigermassen strukturiert und Information lastig, man sieht, wie man mit Error Meldung umgehen soll und welche Error Meldung

<b>OpenAI Whisper</b>	7	Man findet eine sehr gut strukturierte API-Dokumentation, nicht zu viel beschrieben, aber es ist ausreichend, um die API zu verstehen, man sieht die API Error Codes
<b>AssemblyAI</b>	8	Dokumentation ist gut strukturiert und man findet sich gut sehr gut zu recht. Dokumentation ist ausführlich beschrieben. Man findet ein extra Kapitel zu den verschiedenen API-Referenzen wie Errors etc.

Source: Own representation.

**Table 6**

Shows the results for the technical support category support and accessibility.

<b>API</b>	<b>Points</b>	<b>Comment</b>
<b>Deepgram</b>	6	naja, man findet eine Community-Seite wo einem geholfen wird, man kann aber nirgends direkt mit einem Arbeiter über eine Chatbot etc. Kontakt aufnehmen. Man findet eine Q&A, über den Blog findet man auch noch Sachen
<b>Google STT</b>	4	Man findet auch schnell den Support, gibt aber keine offizielle Seite, wo man mit Google-Arbeitern kommunizieren könnte, es wird auf Stack Overflow etc. verwiesen, Q&A vorhanden
<b>Microsoft Azure STT</b>	3	Wie allgemein bei Microsoft sehr schwer irgendwo Kontakt aufzubauen, Es hat ein Q&A ist aber sehr kompliziert aufgebaut.
<b>OpenAI Whisper</b>	8	Support ist extrem gut, 24/7 Chatbot, man sieht direkt welches System von OpenAI laufen und welche nicht, Q&A ist auch vorhanden, gibt direkt häufige Fragen vorgeschlagen
<b>AssemblyAI</b>	8	Support ist 24/7 über einen Chatbot verfügbar, man findet verschiedene Links über die gesamte Seite verteilt, um den Support anzuschreiben, Es hat ein Q&A

Source: Own representation.

**Table 7**

Shows the results for Timeliness and completeness of the information.

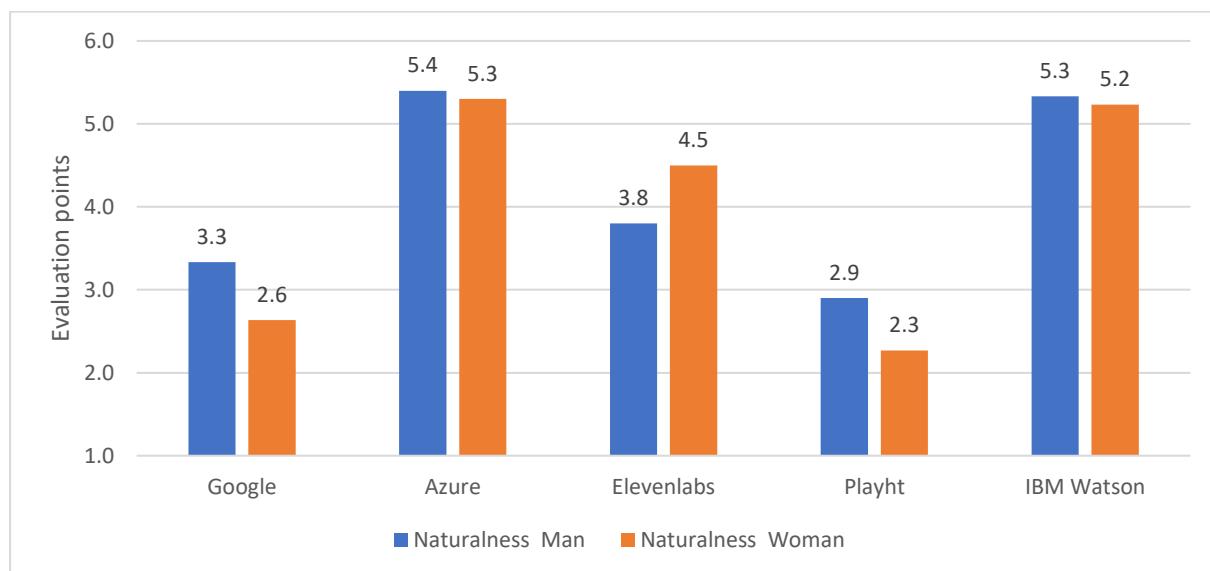
<b>API</b>	<b>Points</b>	<b>Comment</b>
<b>Deepgram</b>	8	sehr gut, man sieht, wann das letzte Mal aktualisiert, und man kann sehen zu welcher Version der API die Dokumentation gehört
<b>Google STT</b>	8	sehr gut, man findet, wenn die Dokumentation zuletzt aktualisiert worden ist und um welche Version es geht
<b>Microsoft Azure STT</b>	8	Sehr gut, wird immer aktuell gehalten, man findet das Datum und sogar wer daran mitgewirkt hat
<b>OpenAI Whisper</b>	6	gut, man hat z.B. direkt gesehen das die Dokumentation über STT noch eine Beta-Version ist, leider findet man kein Datum und keine Versionsnummer der aktuellen Dokumentation
<b>AssemblyAI</b>	6	geht so, Leider kein Datum oder irgendetwas zu finden, wann die Dokumentation das letzte Mal aktualisiert worden ist.

Source: Own representation.

## Appendix 13: Results of the voice quality analysis

### Appendix 13.1: Bar chart of data from Table XY

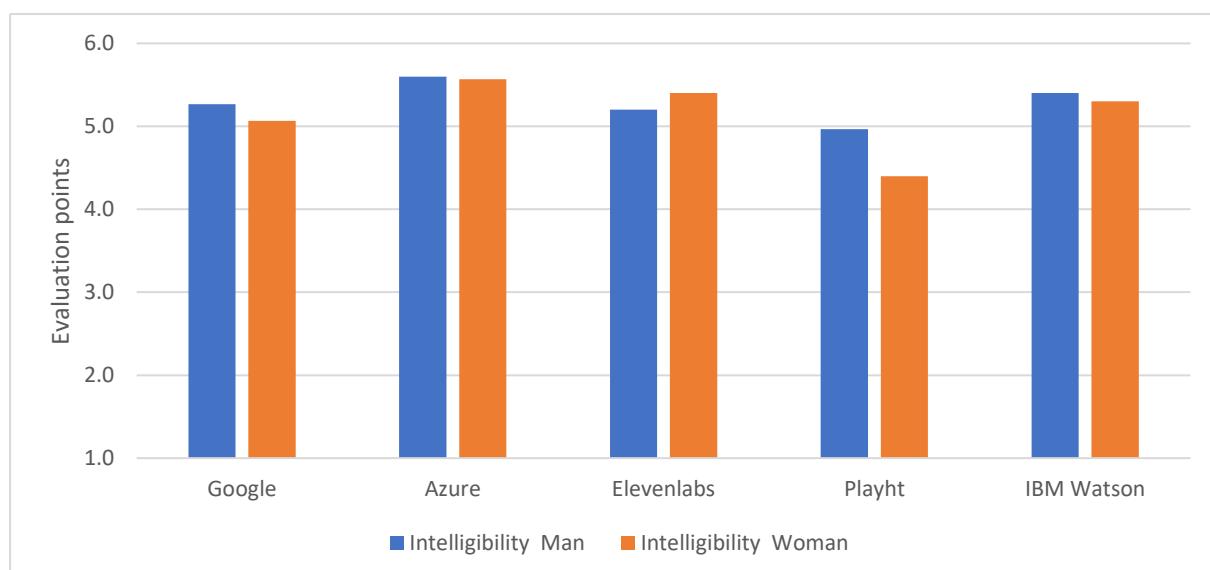
**Figure 14**



Source: Own illustration. Created by Excel.

### Appendix 13.2: Bar chart of data from Table XY

**Figure 15**



Source: Own illustration. Created by Excel.

## Appendix 14: Configurations of the different TTS APIs

**Table 8**

API	Configurations
<b>IBM Watson Text-to-Speech</b>	rate percentage pitch percentage get pronunciation
<b>Google Text-to-Speech</b>	audio Encoding speaking Rate pitch volume Gain Db sample Rate Hertz
<b>Microsoft Azure Text-to-Speech</b>	Visemes Speed Pitch Audio device profile Fine-tuning text-to-speech output with SSML (set emphasis, pronunciation, etc.)
<b>ElevenLabs</b>	stability similarity boost
<b>Play.ht</b>	narration Style global Speed pronunciations trim Silence

Note: All links in this table were accessed no later than July 6 at 13:56.

Source: Own representation.

## Appendix 15: Email from OpenAI



Natal Bumann <nbumann97@gmail.com>

### You now have access to the GPT-4 API

1 message

OpenAI <noreply@tm.openai.com>  
Reply-To: support@openai.com  
To: nbumann97@gmail.com

6 July 2023 at 23:53



### You now have access to the GPT-4 API

As part of the general availability of the GPT-4 API, your OpenAI organization (org-mYnBeeSjGkkVbpu1pHavnvHP) can now access GPT-4 models with 8k context size via the existing OpenAI API.

You can learn more about GPT-4 in our API documentation.

[Try it on the OpenAI Playground](#)

Best,  
The OpenAI team

You received this email because you have an OpenAI API account

## Appendix 16: Generated text with the GPT models

Table 9

Model	Question	Answer
GPT3.5-Turbo	Tell me something about Switzerland	Switzerland is a landlocked country located in Central Europe. It is known for its stunning landscapes, including the Swiss Alps, picturesque lakes, and charming villages. Switzerland is famous for its high quality of life, efficient public transportation system, and political neutrality. It is home to several international organizations, such as the Red Cross and the World Health Organization. Switzerland is renowned for its chocolate, cheese, watches, and banking sector. The country has four official languages: German, French, Italian, and Romans.
GPT3.5-Turbo	What is Flutter?	Flutter is an open-source UI (User Interface) software development kit (SDK) created by Google. It allows developers to build cross-platform applications for mobile, web, and desktop using a single codebase. Flutter uses the Dart programming language and provides a rich set of pre-designed widgets and tools for creating visually appealing and performant applications. It aims to simplify the development process by enabling hot-reload, which allows developers to see the changes instantly without restarting the application. Flutter has gained popularity due to its
GPT-4	Tell me something about Switzerland	Switzerland is a landlocked country located in Central Europe. It is known for its mountains, with the Alps covering about 60% of the total land area. The country has a population of about 8.5 million people and its capital is Bern. Switzerland is famous for its high standard of living, its neutrality and peace, its watches, chocolate, cheese, and its efficient and reliable public transportation system. It is also home to numerous international organizations, including the Red Cross.
GPT-4	What is Flutter?	Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase. It provides a rich set of pre-designed widgets, powerful APIs for 2D rendering, and tools for testing and integration.

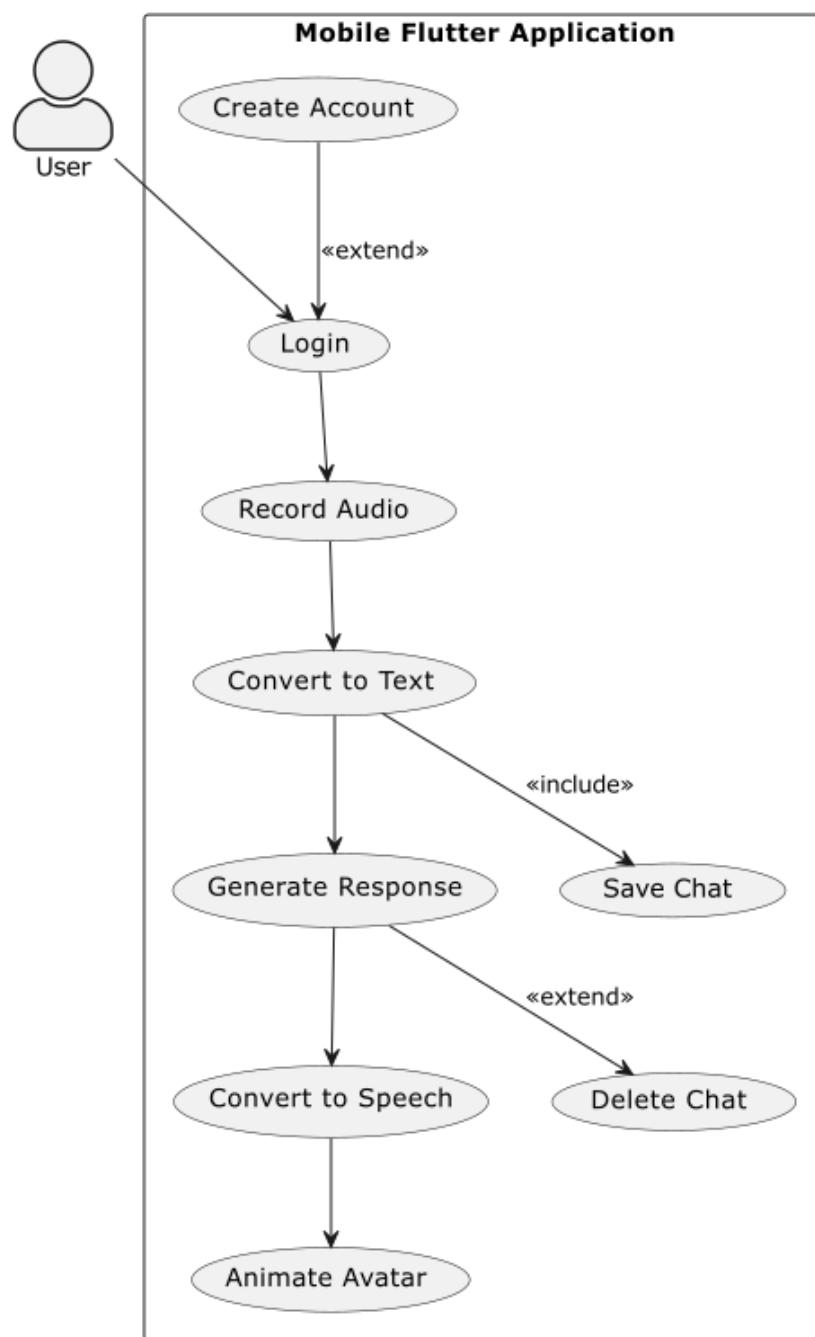
Note: All links in this table were accessed no later than July 3 at 14:33.

Source: Own representation.

## Appendix 17: UML diagrams from the Flutter application

### Appendix 17.1: Use case diagram

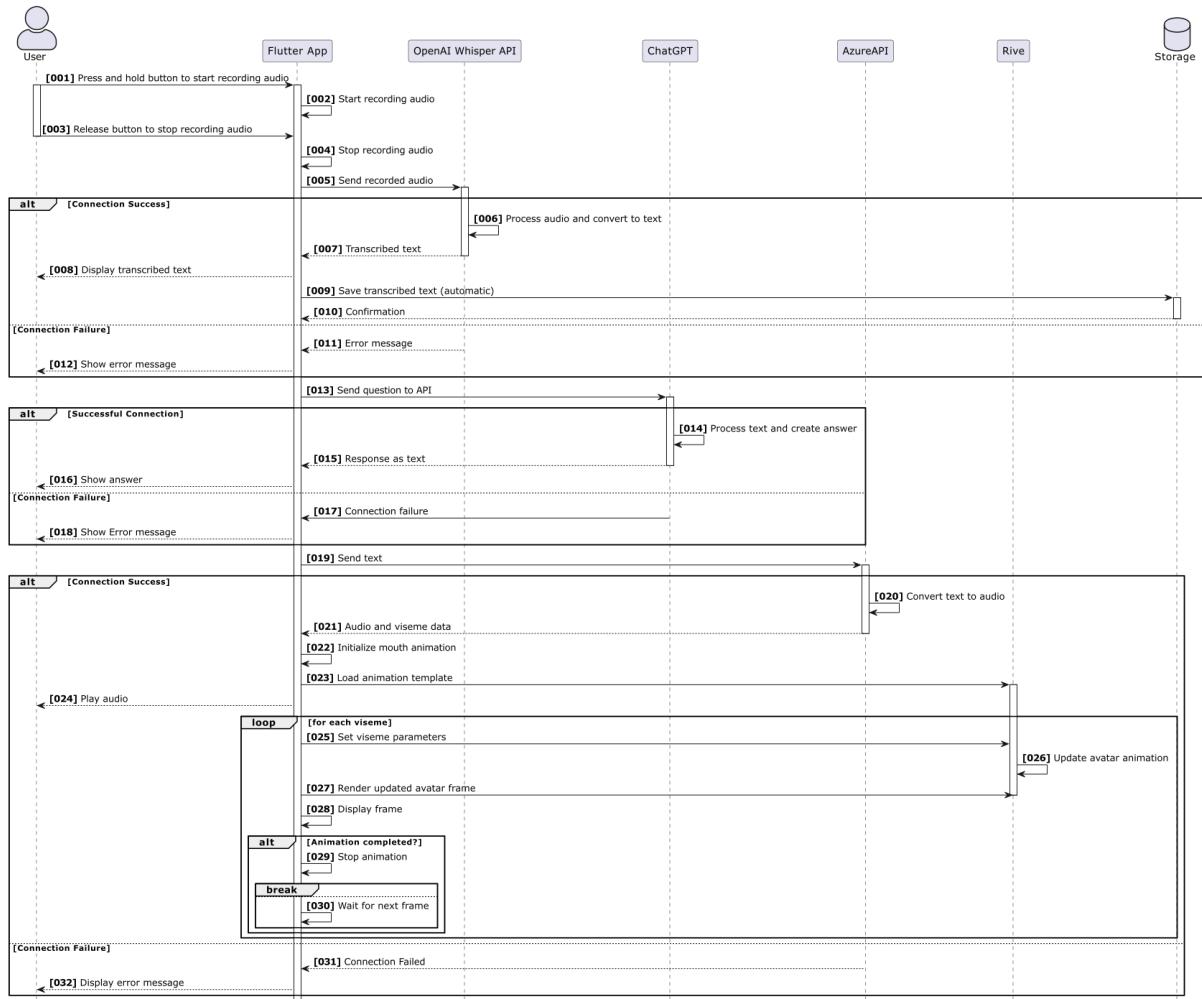
Figure 16



Source: Own illustration.

## Appendix 17.2: Sequence diagram

Figure 17



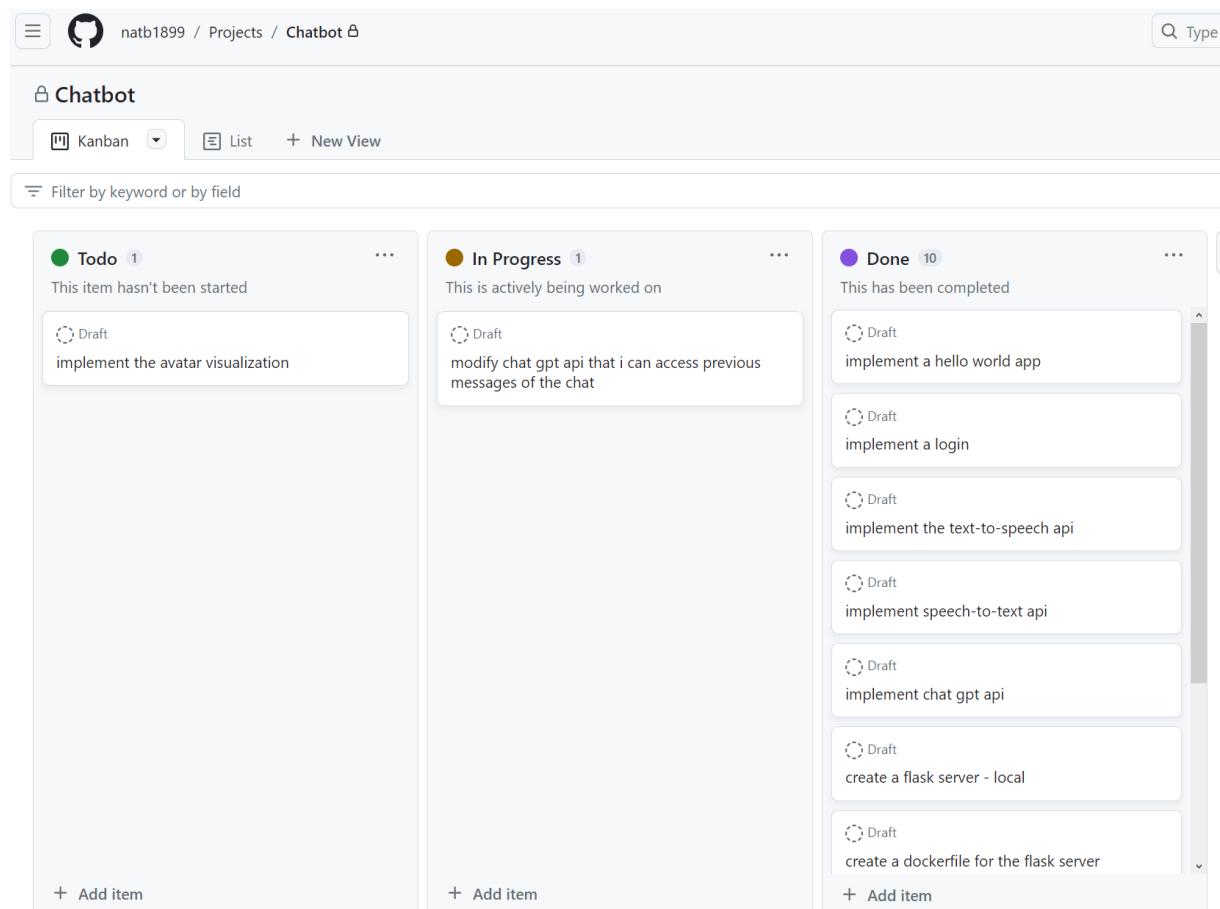
Source: Own illustration.

## Appendix 18: Product Backlog

ID	Topic	As a...	I want to be able to...	User Story		MosCoW
				So that...		
0	Research	writer	conduct research on the latest speech-to-text	I can evaluate their suitability for integration into the chatbot	Must have	
0	Research	writer	conduct research on the latest text-to-speech	I can evaluate their suitability for integration into the chatbot	Must have	
0	Research	writer	make a decision which speech-to-text api to implement	I can implement this api	Must have	
0	Research	writer	make a decision which text-to-speech api to implement	I can implement this api	Must have	
1	Implementation	user	use a mobile app to interact with the avatar and the chatbot	I can easily access it from my smartphone or tablet	Must have	
2	Implementation	user	speak into the speech-to-text app	it can transcribe my speech into text format	Must have	
3	Implementation	developer	integrate the speech-to-text app with the OpenAI GPT API	the chatbot can generate natural language responses to user queries and requests	Must have	
4	Implementation	user	hear the chatbot's response in speech format	I can interact with it without looking at the device	Must have	
5	Implementation	developer	integrate the speech-to-text app with a animated avatar api	the avatar can provide a more engaging and interactive user experience	Must have	
6	Implementation	developer	keep the full conversation with the avatar	I can use the chat capabilities to enhance the dialog and provide more personalized responses to the user	Must have	
7	Implementation	user	provide feedback on the chatbot through manual testing	I can help improve the product and make it more user-friendly	Must have	
8	Implementation	user	have access to documentation on how to interact with and manage the chatbot	I can use the chatbot effectively	Must have	
9	Implementation	developer	review the feedback gathered through manual testing and make an necessary changes or improvements	it meets the needs of the users	Must have	
10	Implementation	user	communicate with a chatbot that is trained with specific data from the company	I can get accurate and personalized responses to my questions and receive a better customer experience overall	Could have	
11	Implementation	developer	train the chat GPT model using the data	I can improve its accuracy and relevance to the company's specific use case	Could have	
12	Implementation	user	adjust the chatbot's speech rate and volume	I can customize my listening experience.	Could have	
13	Implementation	user	initiate a conversation with the chatbot using voice commands	I can interact with the system hands-free.	Could have	
14	Implementation	user	specify my preferred language for interacting with the chatbot	I can communicate more effectively.	Could have	
15	Implementation	user	easily access previous conversations with the chatbot	I can review information or pick up where I left off	Could have	

## Appendix 19: Extract from the Kanban board

Figure 18



Source: Own representation. Screenshot GitHub - natb1899

## Appendix 20: Versions of the used flutter packages

Table 10

Name	Version
record	^4.4.4
http	^0.13.6
http_parser	^4.0.0
path_provider	^2.0.15
sliding_up_panel	^2.0.0+1
firebase_core	^2.13.1
firebase_auth	^4.6.2
cloud_firestore	^4.8.1
just_audio	^0.9.34
provider	^6.0.5
rive	^0.11.3
shared_preferences	^2.2.0
rename	^2.1.1
equatable	^2.0.5
dartz	^0.10.1
cupertino_icons	^1.0.2
permission_handler	^10.4.2
flutter_lints	^2.0.0
build_runner	^2.4.6

## Appendix 21: User Guide

This user guide is intended for installation on an Android smartphone only.

### 1 Installation

Click on the following link to download the app:

<https://play.google.com/apps/internaltest/4701374153575769247>

If you receive an error message that the link cannot be accessed, please get in touch with us at the following email address: [nbumann97@gmail.com](mailto:nbumann97@gmail.com).

Otherwise, download the app by clicking "Download on Google Play" on the web page shown in Figure 1. This will open the Google Play Store, as shown in Figure 2, where you can click "Install". After installation, the app should be available on your Android smartphone. See Figure 3 white circle.

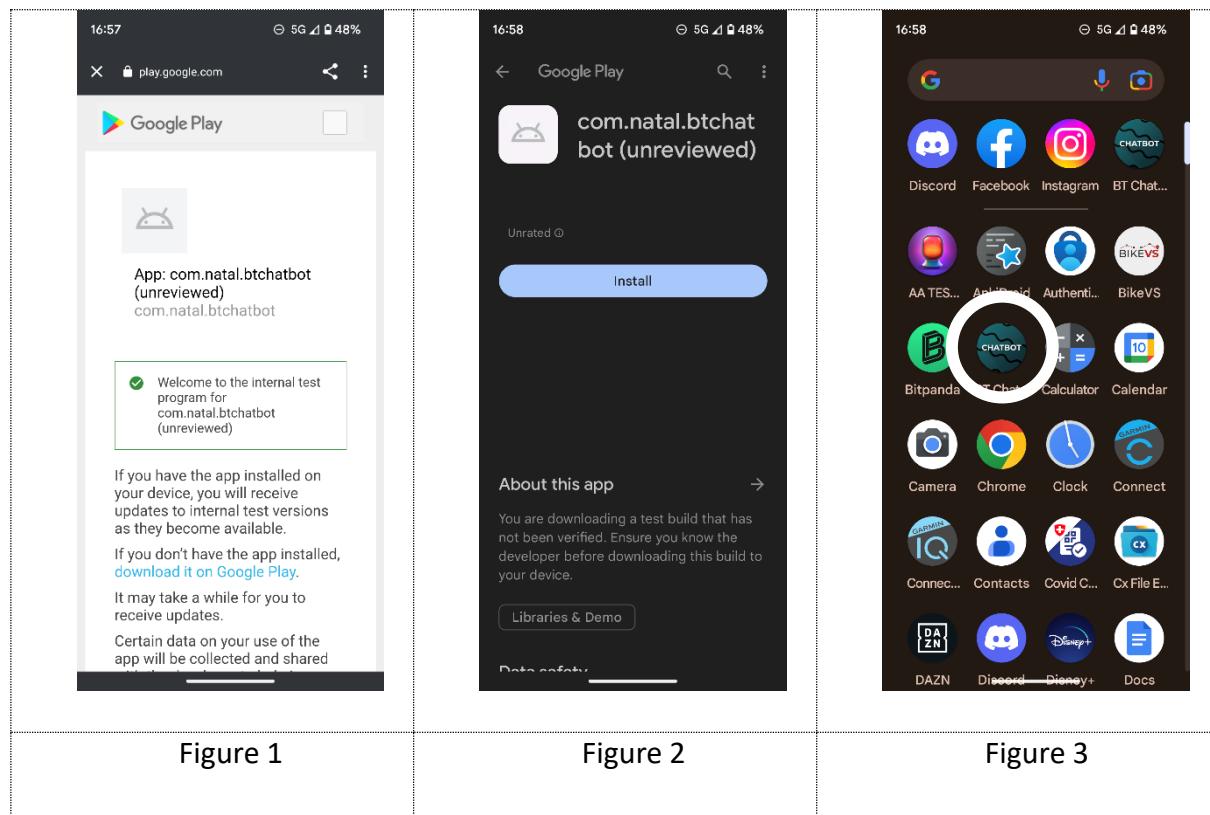


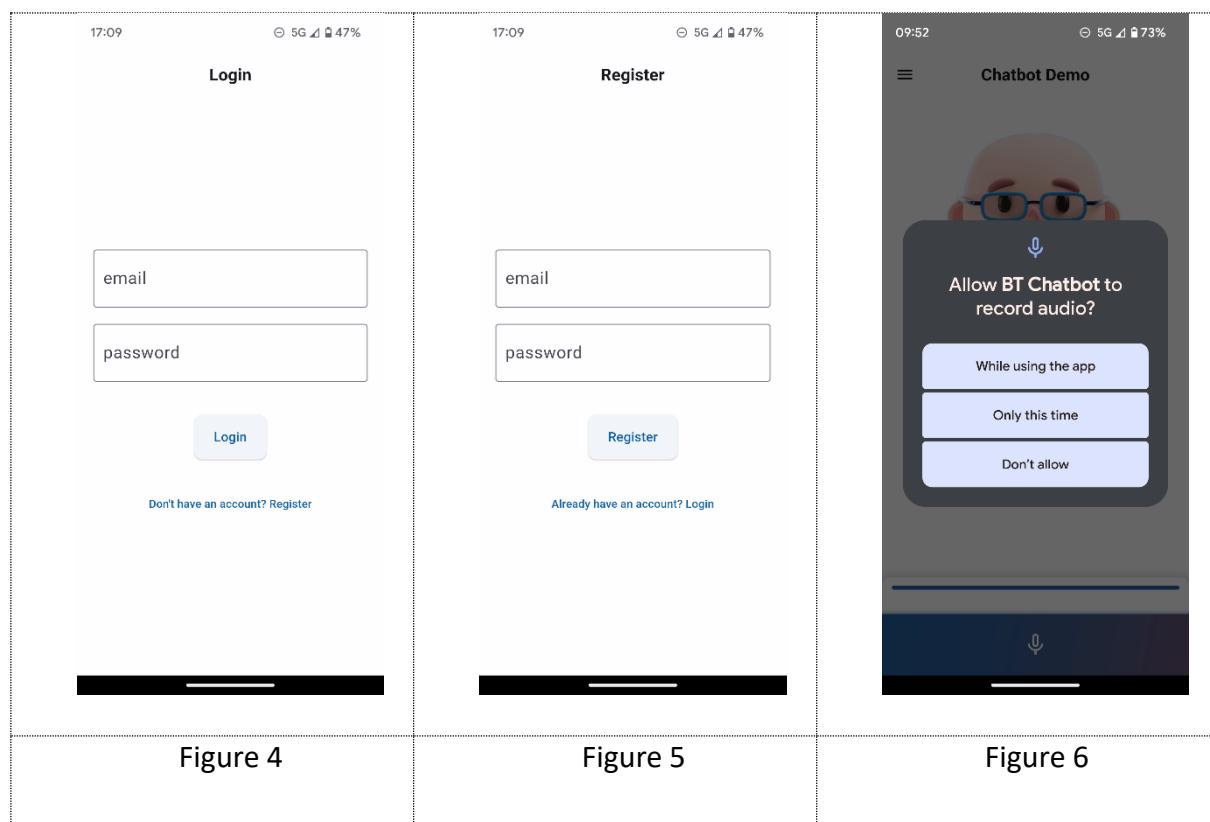
Figure 1

Figure 2

Figure 3

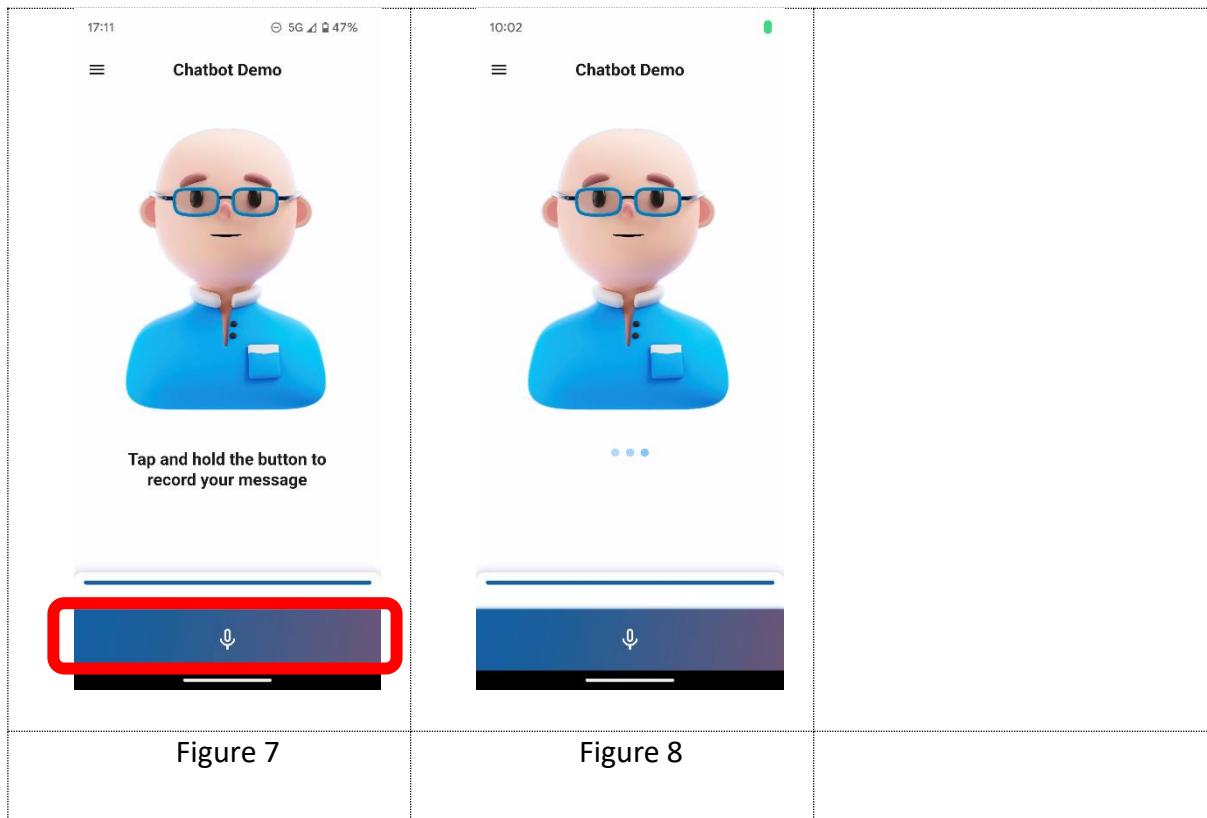
## 2 Login / Registration

The login page appears when you open the app (see Figure 4). If you don't have an account yet, you can create a new one by clicking "Don't have an account? Register". This will take you to the page shown in Figure 5. Here you can enter an e-mail address and a password with at least 6 characters. Your account will be created if you click the "Register" button. Shortly, a pop-up window will appear asking for your permission, as shown in Figure 6. To use the application, you must agree to this permission.



## 3 Interaction with the chatbot

Now there is a possibility to interact with the chatbot. To do this, press and hold the button at the bottom of the screen (marked in Figure 7) and speak your instructions/questions into the microphone. When you want to stop recording, release the button. After the recording has started, three small dots are displayed. These remain visible until the avatar has completed its voice output. This view can be seen in Figure 8.

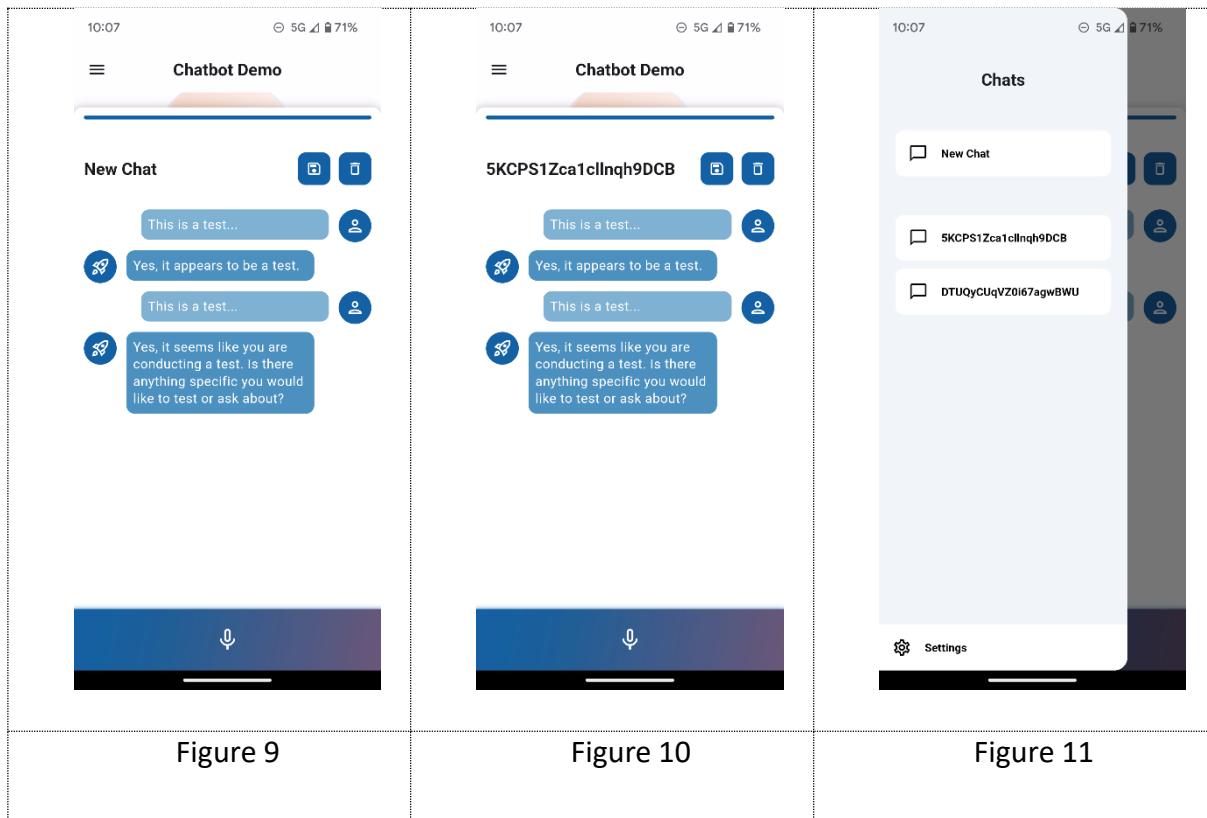


## 4 See the Chat

If you swipe the blue line upwards, you can view the entire chat with the avatar. You can see this view in image 9.

There is also a possibility to delete the chat. To do this, click the "Delete" icon in the upper right corner. This will delete the chat completely.

To view and retrieve all saved chats, open the menu on the left by clicking the "Bar" icon. This will open the menu, as shown in Figure 11. To start a new chat, click on "New Chat". Clicking on the "IDs" will open the saved chat.

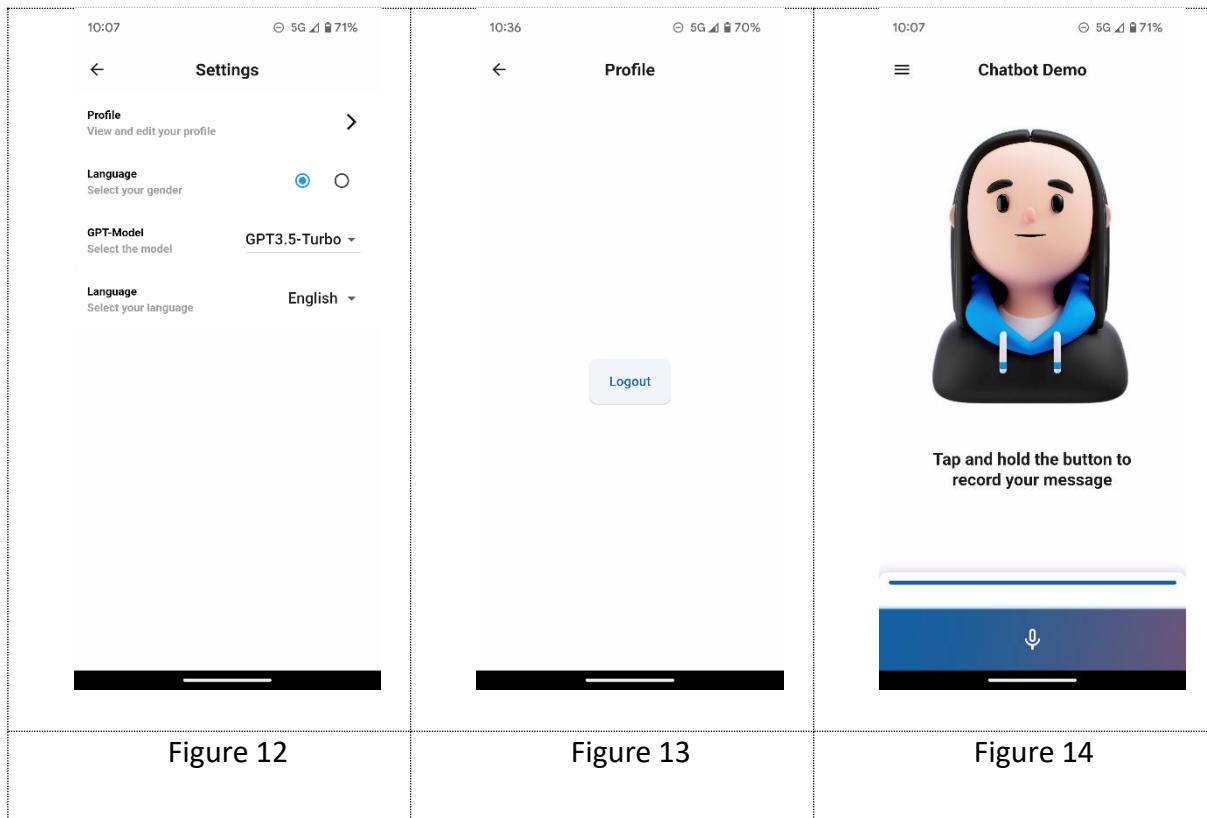


## 5 Settings

To open the settings, please open the menu on the left side. You can see the corresponding view in Figure 11. Then look for the "Settings" option at the bottom of the menu and click on it. This will open the settings page, as shown in Figure 12.

In the settings, you can make various adjustments. You can find the settings options as follows:

- You can log out to disconnect under the "Profiles" menu item. The profile page is shown in Figure 13.
- Under the menu item "Gender," you can set the gender of your avatar. You can choose between male and female. The female avatar can be seen in Figure 14.
- Under the menu item "GPT model," you can select the desired GPT model. There are two options available: 3.5-Turbo or 4.



## 6 Info's

All text was paraphrased with ChatGPT. The following prompt was applied for it:

*rewrite this text (here would be the text to rewrite) for me to use in a user guide. in text form. no listing.*

All images used are own screenshots.

## Appendix 22: User Feedback

**Table 11**

Test person	Feedback
1	The app looks good. Speech input and output facilitate communication. Some slightly long processing times. Understanding of spoken language was mostly good, but some slight comprehension issues. Sometimes difficult to identify the foldable chat, clearer indication would be helpful. Distinction between login and registration page not recognizable
2	Impressive speech recognition, hardly any errors in converting to text. Text-to-Speech is pleasant and easily understandable. Design is nice. Sometimes not clear what is happening now. Speech-to-text part is practical, but TTS playback is a bit time-consuming.
3	Great app. Had difficulty finding the chat directly. Desirable to be able to interrupt the chatbot during its responses. Sometimes long waiting times. Text-to-Speech output is clear but somewhat robotic. Option to interrupt the chatbot during responses is desired.
4	Surprisingly accurate and error-free speech input. A bit sluggish at times. Text-to-Speech voice could sound more natural. Avatar should provide visual or audiovisual feedback. Visual feedback from the avatar would be helpful.
5	Good speech recognition, but occasional comprehension issues. Text-to-Speech voice is a bit too fast and monotonous. Avatar should be more responsive and provide visual feedback. Make the foldable chat more visible.
6	Heeee, I was able to discuss with your app!!!! Excellent! First thing I noticed is the lack of feedback indicating that you are recording speech and receiving the questions. It might be useful to display the text that is pronounced for clarity. The app should validate that it understood the question before sending it to chat and show that it's waiting for the answer (e.g., with a thinking avatar). Additionally, a stop/reset button would be helpful to restart/cancel the chat. After a while, I'm not able to dialog with the app...

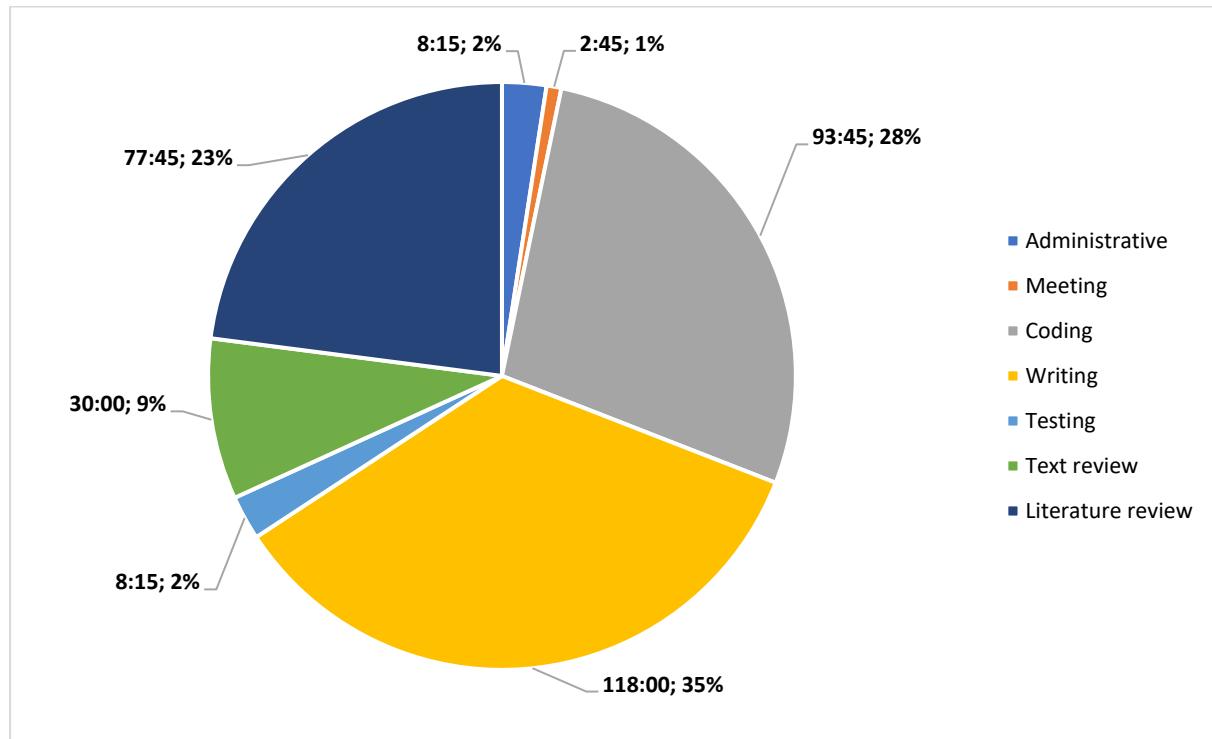
Note: Some feedback has been translated from English to German

## Appendix 23: Project Management

### Appendix 23.1: Timetable

Woche	Timetable Bachelor Thesis												
	Mai				Juni				Juli				
1	2	3	4	5	6	7	8	9	10	11	12	13	
	01.05 - 07.05	08.05 - 14.05	15.05 - 21.05	22.05 - 28.05	29.05 - 04.06	05.06 - 11.06	12.06 - 18.06	19.06 - 25.06	26.06 - 02.07	03.07 - 09.07	10.07 - 16.07	17.07 - 23.07	24.07 - 30.07
<b>Planning phase</b>													1.5 weeks
Preparations (workplan, formatting document)													
Create the product backlog													
Create an initial document structure													
<b>State of the art and justification</b>													3.5 weeks
Literature review of existing speechtotext and texttospeech technologies													
Evaluate and decide on the best speechtotext technology for the project													
Evaluate and decide on the best natural language processing API for the project													
Evaluate and decide on the best texttospeech technology for the project													
Evaluate and decide on the best avatar animation APIs for the project													
Write the justification for your choices													
<b>Implementation &amp; Testing part 1</b>													5.5 weeks
Develop a mobile app (good architecture etc.)													
Develop and test the speech-to-text app and integrate it with the natural language													
Develop and test the texttospeech app and integrate it with the avatar API													
<b>Implementation &amp; Testing part 2</b>													
If necessary: Train models for better results (ts, st, gpt, avatar)													
If time: Add capabilities of starting and exploiting the dialogs to generate better answers.													
<b>Implementation &amp; Testing part 3</b>													
Conduct feedback to evaluate and make some changes to the system													
<b>Results &amp; Evaluation</b>													1.5 weeks
Analyze the results of the user testing and feedback													
Write the results and evaluation section of your thesis													
<b>Conclusion and Abstract</b>													1.5 weeks
Summarize the key findings and contributions in the conclusion													
Write the abstract of the thesis													
<b>Editing and Finalizing</b>													2.5 weeks
Edit and revise the document for grammar, spelling, and formatting													
Finalize the thesis and prepare it for submission													

### Appendix 23.2: Invested hours per activity.



## Appendix 24: Code of the backend

hello-world.js

```
1 from flask import Flask, request, jsonify
2 import time
3 import azure.cognitiveservices.speech as speechsdk
4 from werkzeug.utils import secure_filename
5 import openai
6 import json
7 import base64
8
9 app = Flask(__name__)
10
11 #Configuration
12 OPENAI_API_KEY = "sk-ioAJ6kyTScAbjmFkyuCT3B1bkFJpNeQdtSX5uNXjYR9zwq6"
13
14 @app.route("/")
15 def hello():
16     return "Hello World!"
17
18 @app.route("/chatgpt", methods=['POST'])
19 def chatgpt():
20     model = request.json['model']
21     data = request.json['messages']
22
23     # First, you need to set up your API key
24     openai.api_key = OPENAI_API_KEY
25
26     # Then, you can call the "gpt-3.5-turbo" model
27     model_engine = model
28
29     # Send an API request and get a response, note that the interface and parameters have changed compared to
30     # the old model
31     response = openai.ChatCompletion.create(
32         model=model_engine,
33         messages=data,
34         max_tokens=100,
35         temperature=0
36     )
37
38     return json.dumps(response)
39
40 @app.route("/whisper", methods=['POST'])
41 def whisper():
42
43     model_id = 'whisper-1'
44
45     openai.api_key = OPENAI_API_KEY
46
47     file = request.files['file']
48     filename = secure_filename(file.filename)
49     file.save(filename)
50
51     media_file = open(file.filename, 'rb')
52
53     start_time = time.time()
54
55     response = openai.Audio.transcribe(
56         model=model_id,
57         file=media_file
58     )
59
60     end_time = time.time()
61
62     elapsed_time = end_time - start_time
63
64     #print(f"API call took {elapsed_time:.2f} seconds")
65
66     return json.dumps(response)
```

```

66
67 @app.route('/azure', methods=['POST'])
68 def synthesize_text():
69
70     # Get the text from the request payload
71     text = request.json['text']
72     isMan = request.json['isMan']
73     language = request.json['language']
74
75     # Set up the speech configuration based on language and gender
76     speech_config = get_speech_config(language, isMan)
77
78     # Synthesize speech
79     speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config, audio_config=None)
80
81     viseme_data = []
82
83     def viseme_cb(evt):
84         audio_offset = evt.audio_offset / 10000
85         viseme_id = evt.viseme_id
86
87         viseme_data.append((audio_offset, viseme_id))
88
89         #print(viseme_data)
90
91     # Subscribes to viseme received event
92     speech_synthesizer.viseme_received.connect(viseme_cb)
93
94     result = speech_synthesizer.speak_text_async(text).get()
95     stream = speechsdk.AudioDataStream(result)
96
97     # Save audio to file
98     file_path = 'file.wav'
99     stream.save_to_wav_file(file_path)
100
101    with open(file_path, 'rb') as f:
102        audio_bytes = f.read()
103
104        audio_base64 = base64.b64encode(audio_bytes).decode('utf-8')
105
106        response = {
107            'audio_file': audio_base64,
108            'viseme_data': viseme_data
109        }
110
111    return jsonify(response)
112
113 def get_speech_config(language, isMan):
114
115     # Set up the speech configuration based on language and gender
116     if language.lower() == 'english':
117         if isMan:
118             speech = "en-US-JacobNeural"
119         else:
120             speech = "en-US-DennyNeural"
121     elif language.lower() == 'french':
122         if isMan:
123             speech = "fr-FR-HenriNeural"
124         else:
125             speech = "fr-FR-DeniseNeural"
126     elif language.lower() == 'german':
127         if isMan:
128             speech = "de-DE-BerndNeural"
129         else:
130             speech = "de-DE-KatjaNeural"
131
132     speech_key = "d1878de63c0a46bfb59da88516ef2a3f"
133     service_region = "switzerlandnorth"
134     speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)
135     speech_config.speech_synthesis_voice_name = speech
136
137     speech_config.set_speech_synthesis_output_format(speechsdk.SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm
138 )
139
140 if __name__ == '__main__':
141     app.run(debug=True, host='0.0.0.0')

```

## **Author's declaration**

I hereby declare that I have carried out this final research project on my own without any help other than the references listed in the list of references and that I have only used the sources mentioned.

Sierre, 28. July 2023

Natal Bumann