



hochschule mannheim

Development and evaluation of a voice assistant system based on open source components for Raspberry Pi

Stella Naser

Bachelor Thesis

for the acquisition of the academic degree Bachelor of Science (B.Sc.)

Course of Studies: Computer Science

Department of Computer Science

University of Applied Sciences Mannheim

25.06.2021

Tutors

Prof. Dr. Oliver Hummel, Hochschule Mannheim

Theresa Sick, SPACE; StartupLab@HSMA

Neser, Stella:

Development and evaluation of a voice assistant system based on open source components for Raspberry Pi / Stella Neser. –

Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2021. 55 pages.

Neser, Stella:

Entwicklung und Evaluation eines Sprachassistentensystem auf Basis von Open-Source-Komponente für Raspberry Pi / Stella Neser. –

Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2021. 55 Seiten.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 25.06.2021

Stella Neser

Abstract

Development and evaluation of a voice assistant system based on open source components for Raspberry Pi

When the novelty of voice assistant software was introduced with Apple's Siri in 2011, no one could have predicted the mass amount of adoption this class of software would garner. Mass adoption of these applications drive the innovation in the space. But this innovation comes at the cost of the users privacy, as most of these voice commands must be processed off device, in order to give an accurate analysis. This raises many privacy concerns, as it is not clear, as to what data exactly is being sent back to the corporations for processing and in what way. This leads to the goal of this paper. Named after Bertha Benz, the first woman who dared to drive a car from Mannheim to Pforzheim, "Berta", a practical voice assistant application with privacy by design. Berta is designed to process all commands on device and in some cases doesn't need any network connectivity at all. Design considerations and implementation steps are given, as well as a comparison to the different network activities to more popular voice assistant applications.

Contents

1. Introduction	1
1.1. Motivation	1
1.2. The goal of the work	3
1.3. Structure of the work	3
2. State of the Art	5
2.1. Voice assistants	5
2.2. Related work	6
3. Technical basics	7
3.1. Components	7
3.1.1. Raspberry Pi 4 model B	7
3.2. Applications	8
3.2.1. Picovoice Porcupine Wake Word Engine	8
3.2.2. Speech-to-text Engine DeepSpeech Mozilla	10
3.2.3. Text-to-speech engine SVOX Pico	14
3.3. Web Application	15
3.3.1. Flask	15
3.3.2. gunicorn and supervisor	16
3.3.3. NGINX	16
3.4. Database	17
3.4.1. SQLite3	17
3.4.2. SQLAlchemy	18
3.5. Other basics	18
3.5.1. Duck typing	18
4. System concept	19
4.1. Product environment	19
4.2. Requirements	20
4.2.1. Functional requirements	20
4.2.2. Non-functional requirements	23
4.3. User interface	24
4.4. Architectural solutions	27
4.4.1. Berta architecture	27

4.4.2. Database model	29
4.4.3. Voice user interface	29
4.4.4. Berta flowchart for voice command	31
4.4.5. Berta plugin class diagram	33
4.5. Technical product environment and technologies	34
4.6. Acceptance and installation	36
4.7. Deployment on other computer systems	36
5. Implementing the concept	37
5.1. Methodology	37
5.2. Wake word and Speech-To-Text (STT) implementation	39
5.3. Plugin implementation	40
5.4. TTS implementation	41
5.5. Web application extensions and implementations	41
5.6. Configuring Unicorn, Supervisor, NGINX, SSL certificate	43
5.7. Database implementation	43
5.8. Connection of components	43
5.9. The attempt to build a Berta service	44
6. Evaluation measures	45
6.1. Usability evaluation measure	45
6.2. Offline ability evaluation measure	45
7. Experimental Setup and Evaluation	47
7.1. Intended experiment setup	47
7.1.1. Experiment setup usability	47
7.1.2. Experiment setup offline ability	48
7.2. Intended evaluation	49
7.3. Implemented experiment setup and evaluation	49
7.3.1. Evaluating the usability	49
7.3.2. Evaluating the offline ability	52
8. Conclusion and future work	53
List of Abbreviations	ix
List of Tables	xi
List of Figures	xiii
Listings	xv
Bibliography	xvii
Index	xxv

A. Usability test by voice communication	xxv
A.1. Entry scenario	xxv
A.2. Results:	xxvi
A.2.1. Male	xxvi
A.2.2. Female	xxxiv
B. Evaluation diagram in numbers	xliii

Chapter 1

Introduction

This chapter introduces the motivation for this thesis, as well as its goal. It also presents the two research questions that will be addressed in the course of the work. Finally, the structure of the project is presented.

1.1. Motivation

Smart home devices like 'Amazon Echo' or Apple's 'HomePod' offer the user the opportunity to make life easier. With the help of speech recognition, these devices can, for example, read to-do lists aloud, play selected music, switch lights on and off, or dim them. To make these functions and services available to end users, the voice recordings need to be forwarded to the respective service provider for analysis and evaluation. This is happening constantly, as the device must always be ready in case a command comes in. How this data is processed, the duration of how long the data is stored, and even the exact contents of the data is not made transparent to the user. This can lead to significant privacy issues that not every end user may agree to, nor want to deal with. Therefore, privacy is the main motivation for this project.

Accessing and collecting data is not a bad concept, it can serve to improve algorithms and systems and make them more user-friendly if the data would be used anonymously. A problem arises when the data collection contains personal data and these are used, for example, for manipulation purposes, such as personalized, tailored advertising. A lot of money is earned by companies to which private, personal information was not willingly given. [1]

1. Introduction

To protect the privacy and personal data of each individual when dealing with the Internet and other categories is not a new topic. In Germany there is the Federal Data Protection Act (BDSG), which serves to protect personal rights. The BDSG takes precedence over special legal data protection regulations and it implements the requirements of the EG data protection guidelines. [2]

The redesigned BDSG was supplemented in 2018 by the European General Data Protection Regulation, which has been in force since 2016, in the area that the regulations have been left binding to the European member states. [2]

A supplementary provision including that consumer law has been strengthened. The legal basis for data processing agencies has been specified, and stricter regulations have been drawn up for the handling of personal data. Transparency for the processing and further processing of personal data, which was not regulated and should be created. [2]

Further information on this is provided by the Federal Ministry of the Interior, Building and Home Affairs on their website [2].

Despite the efforts of the state, people should also be able to protect themselves when it comes to their data processing. By providing them with the means to protect privacy, e.g. in the smart home area. Therefore, a privacy by design approach is another motivation for this work.

Building a self-hosted voice assistant comes with many advantages. Through the use of different software components, privacy, flexibility, and scalability can be fit to meet any users needs. The project is designed with all processing taking place locally on the device, so that any reliance on cloud service providers can be avoided. This circumvents all the privacy issues mentioned earlier and allows the device to continue working, as opposed to connected devices that stop functioning after their respective cloud service is retired. A simple setup to achieve this is a piece of hardware that can record audio and a speech engine. For a more sophisticated scenario, a Text-To-Speech (TTS) engine can be incorporated with a speaker, in order for the device to deliver answers to the user. This project demonstrates the latter.

1.2. The goal of the work

The goal of this bachelor thesis is to develop and evaluate a voice assistant system using the DeepSpeech open source Speech-To-Text (STT) engine from Mozilla based on a Raspberry Pi. This project will be able to evaluate spoken commands without a connection to a specific cloud service. Furthermore, the functionalities are compared against some of the common proprietary voice assistants, such as Google's 'Assistant'. Finally, a conceptual proof of the functionalities is created. This evidence will provide a basic framework in order to expand the platform via a "plugin" system. Thus, the end user can teach the voice assistant system new functionalities. The research questions (RQ) this paper seeks to answer are the following:

RQ1: Can a voice assistant be built with open source components and work in a similar way to known systems?

RQ2: Can a voice assistant be built with open source components to argue offline without using personal data?

1.3. Structure of the work

The work begins in this chapter and gives a brief overview of the motivation and aim of this thesis. The research questions are also presented here. Chapter 2 deals with popular known voice assistants systems and the currently available related works for open source voice assistants systems. Chapter 3 begins with the technical basics, which should help to understand the used components and techniques for this project. Chapter 4 presents the system concept for developing the speech assistant. It presents the architecture and interfaces between the used engines and components. Chapter 5 describes the methodology for finding the individual components, as well as further details on implementation. Chapter 6 presents the evaluation measurements. Chapter 7 is presents the experimental setup which is needed for evaluating the evaluation measurements. Also chapter 7 evaluates and rates the evaluation of the experiment. Finally, Chapter 8 then ends with a summary, conclusion and assessment of the essay and gives a possible insight into the future of the Berta voice assistant.

Chapter 2

State of the Art

This chapter takes a closer look at existing voice assistants. At first, the most well-known online voice assistants are discussed and the general use of these assistants are presented. A transition to the current related work of offline voice assistants is given.

2.1. Voice assistants

A voice assistant is a program that can recognize spoken words, process the input, and evaluate the given command, if the assistant has a function programmed behind the command. This procedure can further process other interactions. [3]

A Smart Speaker is a speaker that has an integrated voice assistant. It can output sound and the difference to a normal speaker is that it can be connected to the internet and can process voice commands [4]. In addition to a smart speaker, other devices can also be used for voice assistants, like smartphones, in the car or smart-watches. [5] [6]

There are some types of voice assistants, that can be used with corresponding hardware. The most popular and used voice assistants are [7] [8] [6] [5]:

- Amazon with its voice assistant Alexa.
 - Smart Speaker 'Amazon Echo.'
- Apple with its voice assistant Siri
 - Smart Speaker 'HomePod'

- Google with its voice assistant Google Assistant
 - Smart Speaker 'Google Home'

Different functions can be used with these assistants like playing music, integration with other popular Services, like different online calendars, or even switching on and off lights.

2.2. Related work

Voice assistant technology has progressed far. A user wanting a personal, usable, offline voice assistant can now create a service with open-source components at home.

Jasper makes a complete guide for creating your own voice-controlled, always-on applications [9]. It provides the user with a list of hardware components [10], installation manual [11] and other useful facts. Like in the configuration of Jasper [12] you can choose between one of five different STT, but only one of them can be used completely offline. In turn, it is different with the TTS engine. Here the user can choose between eight engines, four of which are usable offline. The user can choose for themselves, which of the engines best fits their needs. They can view Jasper's posted research and test results, which include data on how the speech synthesis is done and how it sounds.

The main reason why this project does not use Jasper is that Jasper only has one option for an offline STT engine. "The recognition rate is not the best" on the supported engine, Pocketsphinx, according to the Jasper documentation [12]. There are also other alternatives to open source voice assistants like Rhasspy, Mycroft, Almond and others. These are not dealt with in more detail in this work. For more information, the Make magazine website [13] offers a good overview of them. Various companies make individual software components open-source for building and customizing a voice assistant, as with Mozilla's DeepSpeech STT, the main component of this project. Further parts used are SVOX Pico TTS and Pico Porcupine for wake word detection. Combined with Mozilla's DeepSpeech STT engine this project illustrates a new approach to building an offline voice assistant focusing on using less to no personal data To verify the research questions mentioned above, all components used in this project are offline engines, which are installed and tested on a Raspberry Pi 4.

Chapter 3

Technical basics

This chapter presents the technical basics used for setting up the voice assistant. At first, the chosen components for the project are presented. Afterwards, an overview of the Raspberry Pi 4 is given. This is followed by an outline of applications used. After that, the implemented web application is presented followed by the implemented database.

3.1. Components

The project uses the following components in order to get the voice assistant up and running.

- Raspberry Pi Components:
 - Raspberry Pi 4 Computer Model B 4GB RAM
 - Raspberry Pi OS (Raspbian)
- Speaker elari NanoBeate Portable Bluetooth Speaker
- ReSpeaker 4-Mic Array for Raspberry Pi

3.1.1. Raspberry Pi 4 model B

This subsection introduces the Raspberry Pi 4 model B. The Raspberry Pi 4 model B is a complete computer in a small size. It is a silent, energy-efficient machine. It uses gigabit networking and comes with USB 3 and USB 2 ports and a selection

of RAM sizes, which vary from 2GB to 8GB. The operating system for Raspberry Pi is Raspberry Pi OS (Raspbian). With a Raspberry Pi 4 for approximately \$35 to \$77, depending on RAM size, is inexpensive. It can be used as a desktop computer, a media center, or a network Artificial Intelligence core.

Due to the open-source nature of the operating system, software, and hardware, projects can easily be backported to use older revisions of the Raspberry. [14] [15] [16]

3.2. Applications

After careful consideration and testing processes to achieve a user friendly, lean, final system which acts as a strong baseline project for future work the following described applications are the finale used ones and will be described in their terms. The main criteria for selecting the components were the attached licence, the ability to run the given product on the Raspberry Pi, and if the software component has the ability to do its processing on device. Further clarification on why the individual application were chosen and possible alternative considerations is also give. How these applications are used together is clarified in chapter 4.

3.2.1. Picovoice Porcupine Wake Word Engine

The decision to use Picovoice Porcupine wake word engine was supported by the Rhasspy wake word documentation [17]. There were five engines summerized by their key characteristic of there system these are: listed and rated in performance, training to customize and online signing up. The performance of Porcupine was the best with a performance rating “excellent” [17].

Snowboy would have been an option because of it’s rating "good", but a registration would have been necessary for this [17]. Since the support for the Snowboy engine was discontinued in December 2020 [18], Porcupine was chosen as the wake word engine. Porcupine Wake Word Engine is made by Picovoice Inc.. Picovoice is an end-to-end platform where you can build voice products on your terms. The user is provided with Porcupine Wake Word Engine, Rhino Speech-to-Intent Engine and Picovoice Shepherd [19]. Also, Picovoice released Leopard, a speech-to-text offline

transcription engine and Cheetah a speech-to-text engine for real-time applications [20] [21]. This software stack is run entirely on-device, instead of in the cloud [19].

“Porcupine is a highly-accurate and lightweight wake word engine. It enables building always-listening voice-enabled applications.” [22]. Porcupine uses a “trained in real-world environments” deep neural network, can be used on different platforms, and brings other good characteristics, which can be found in the Picovoice documentation [22].

The Porcupine Wake Word Engine allows for different words or phrases to be used as a way to activate the system. Many pretrained words are provided by Picovoice, but also it also allows for creating a custom wake word command. This project implements the former.

The engine is continuously listening for key phrases or commands. The pretrained word phrase chosen is "bumbleblee" and "computer".

The Picovoice Console, the online application provided by Picovoice, allows for creating custom wake words and phrases. Depending on the use case of the application, there are different version of the application to choose from. The personal console, which is a self service console for the creation of the aforementioned wake words and phrases, but it does not include a commercial licence for the created model. The Enterprise console allows for creation of phrases and words with a commercial licence, which allows these models to be distributed in commercial applications. Each console allows for creating phrases and words limited to 30 days for evaluation purposes, but these can be extended if necessary by re-running the training application. The console allows for phrase and word creation in four different languages: English (en), German (de), Spanish (es), and French (fr). Apart from the language, it is also possible to choose a variety of platform on which the model should be run on. The personal console only allows for word and phrase creation on Linux(x86_64), macOS(x86_64), and Windows(x86_64) [23]. For this reason, in order to test a custom wake word, an Enterprise console account was selected to try out the "hey_berta" wake word. It comes with the same 30-day trial, and allows for wake words and phrases to be created for the Raspberry Pi 4. The wake word "hey_berta" was supposed to be used for the voice assistant, but because of the commercial license this could not be used. This decision however limits the possibilities of redistribution. For distribution and corresponding costs a request to the company has to be made [19] [23]. The developers provide a set of pre-trained

porcupine wake word models for the languages mentioned above for use with the Raspberry Pi version of the software [24]. These keywords come with an Apache 2.0 License [25] and can be used for demonstration purposes or commercial usages. The following wake words are available as English keywords [26]: "americano", "bumblebee", "grasshopper", "porcupine", "blueberry", "grapefruit", "picovoice", "terminator", "computer". During consultation, "bumblebee" and "computer" were chosen as the new wake words.

3.2.2. Speech-to-text Engine DeepSpeech Mozilla

"DeepSpeech is an open-source Speech-To-Text engine, using a model trained by machine learning techniques based on Baidu's Deep Speech research paper. Project DeepSpeech uses Google's TensorFlow to make the implementation easier." [27]. Google's TensorFlow is an open-source-platform for machine learning where machine learning models can be develop and trained [28]. On the 19th December 2014, the "end-to-end speech system called "Deep Speech" " was presented [29]. The processing stages are processed with deep learning "combined with a language model". "By training a large recurrent neural network (RNN) using multiple GPUs and thousands of hours of data" the developers achieved "higher performance than traditional methods on hard speech recognition tasks." [29]. Deep learning is not used as frequently in "traditional speech pipelines". "Top speech recognition systems rely on sophisticated pipelines composed of multiple algorithms and hand-engineered processing stages." [29]. This makes Deep Speech more special in it's approach, as they took "advantage of the capacity provided by deep learning systems to learn from large datasets to improve" their "overall performance" [29]. The "model is trained end-to-end to produce transcriptions and thus, with sufficient data and computing power, can learn robustness to noise or speaker variation on its own." [29]. This robustness results from the use of a "combination of collected and synthesized data" [29]. On the eight of December 2015, a new Baidu Deep Speech research paper was released, where they presented the second generation of their speech system [30]. Within a year the DeepSpeech 2 (DS2) end-to-end deep learning system was born. They achieved more performance in DS2 by focusing on three components: "the model architecture, large labeled training datasets, and computation scale" [30]. With better hardware, better algorithms, and better data sets, the DS2 models architecture "have nearly 8 times the amount of computation per data

example as the models in DeepSpeech 1, making fast optimization and computation critical” [30]. This project uses the 0.9.3 release of DeepSpeech, it is a bugfix release and retains compatibility with the 0.9.0, 0.9.2 models, but is not backwards compatible with earlier versions [31]. For deploying in a production setting, the system requirements of DeepSpeech show that different platforms are supported. Among other things Windows 8+, Ubuntu 14.04+, and Raspberry Buster on Raspberry Pi3 and Pi4 [32]. Next to the acoustic American English trained models, the release includes experimental Mandarin Chinese acoustic models. This project uses the American English models, which are trained with synthetic noise augmentation. The manufacturer notes that this performs “best in low-noise environments with clear recordings and has a bias towards US male accent” [27] [30]. While working with these models, it turned out that the models really lack the ability to recognize female, non-native English speakers, who speak English with a foreign accent, despite the English and Mandarin models which used a numerous amount of tagged training data [30]. More on this in the chapter Trained DeepSpeech English models.

Short digression to the problem

In order to shed light on the above-mentioned problem, a brief digression on the ethical problems arising in data is needed. According to the scientific paper by Hovy and Spruit [33], a demographic bias is increasingly evident for natural language processing. The paper shows that working with natural language processing can lead to social influences such as concepts as minorities, namely the concepts of exclusion, over-generalization, and bias confirmation. With regard to the problem mentioned above, a gender data bias is evident in practice. Hovy and Spruit address the fact that most natural language processing software uses a certain type of subject. This type was defined in advance as “western, educated, industrialized, rich, and democratic research participants” [33]. As a result, men of this type may be able to act more easily with standard language technologies, as opposed to “women or citizens of Latino or Arabic descent” [33]. A second paper by Gonen and Goldberg [34] focuses more on gender data bias. In doing so, the problem is examined from existing solution approaches, as this study is from 2019 [34] and since the 2016 study by Hovy and Spruit [33], a few solution approaches have been developed. The 2019 paper [34] shows that previous approaches to solving the problem

have not been effective in the desired way. So there are debiasing methods which cover up systematic gender biases in word embeddings [34]. In short, biases in processing speech keep occurring. This was also witnessed in the development of the Berta voice assistant application.

Here the impression was created that if you were to train DeepSpeech with a data set where the demographic bias for women was adjusted, that the speech recognition of DeepSpeech might be better for female voices. A possible solution: Maybe it is possible to take German-Speaking Youtuber Videos who speak English and create language files. This acquired data set can be used as training data for DeepSpeech. This approach could help alliviate the problem. However, this will not be investigated further in this thesis.

The following section shows a brief insight into the effort the developers took to train the English models and how one can take advantage of this approach to solve the female voice recognition problem [30].

Trained DeepSpeech English models

The developers have improved the difficulty of recognizing speech in environments that often have both loud noise and a lot of acoustic and electronic noise with their method. DeepSpeech as a deep leaning system at this large scale needs a lot of training data. Even the developer say that their “speech training set is substantially larger than the size of commonly used speech datasets”. For training the English and Mandarin DeepSpeech models they used numerous amounts of tagged training data [30]. DS1 formed the basis with the largest dataset alone included “5000hours of read speech from 9600 speakers” [29]. Back then, DS1 training data set consists of two speech types, read and conversational speech [29].

The DS2 collected training data set consists of a number of extensive data, including the DS1 data. The English data set consists of “11,949 hours of labeled speech data containing 8 million utterances”. The Mandarin data set consists of “9,400 hours of labeled audio containing 11 million utterances”. [30]. In the following, only the English DS2 model is considered. The extensive training data set of the English models consists out of three speech types, read, conversational, and mixed speech. The difference to commonly used speech datasets is that the developers of DeepSpeech used an English speech training set augmented with noise synthesis

data. They did this to reduce the error rate of the system. Besides the noisy speech, the data set consists of read speech and accent speech which is divided in “four categories”. The four categories are: “American-Canadian”, “Indian”, “Commonwealth” and “European group”.

The System was tested with three kinds of test sets. These sets consisted of read speech, accent speech, and noisy speech. During testing, the developers compared the results of DS2 against those of DS1, as well as “human level performance on read speech”. Benchmarking the system consisted of using read speech by “reading news articles”. The accent test set included “1024 examples from each accent group for a total of 4096 examples”. The performance on the accent speech testset showed how good the training data is. Furthermore, the performance on the noisy speech was tested with three testsets. The first dataset “has 1320 utterances” and includes several real life “noisy environments”. The second “includes 1320 utterances with simulated noise from the same environments”. Lastly, the third is the control set and has the same utterances and speakers except that it has a “noise-free environment” [30]. This shows that in training the DeepSpeech deep learning system, thousands of hours of training data is needed in a particular language, with millions of utterances.

Next, a short insight is given on how to train a model locally.

Training a DeepSpeech Model

It is possible to train a model locally, thanks to the highly generic approach given by the DeepSpeech developers [35]. A high performing recognizer can quickly be applied to a new language [30]. The prerequisites for training a model are [35]:

- Python 3.6
- Mac or Linux environment
- CUDA 10.0 / CuDNN v7.6 per Dockerfile.

Further detailed instructions on how to train a DeepSpeech model, with all the caveats, can be found on Mozilla’s DeepSpeech Website [35]. Future improvements could include a training a special model to better recognize female language patterns or even another language, such as German.

3.2.3. Text-to-speech engine SVOX Pico

The focus of finding the right Text-to-Speech engine was to find an offline engine, in order to process everything on device and without having to rely on a cloud service that does not comply with the privacy concerns outlined before. A Text-to-speech engine is the opposite of a Speech-to-text engine, in that it takes a written word or sentence, and generates sounds and “speaks” the input. There were several TTS engines that could be used, but only one was convincing in terms of quality. When choosing the TTS, close attention was also paid to the license. There were four Text-to-speech engines installed and tested and compared in voice sound, language support and installation effort. The Jasper documentation [12] provided the starting point for choosing the offline TTS. The four installed and tested Text-to-speech engines were:

- Festival
- Flite
- eSpeak
- SVOX Pico TTS

Ultimately, SVOX Pico TTS was able to convince.

SVOX Pico TTS

SVOX Pico TTS is “an open-source small footprint application” [12]. It meets the requirement of working offline. Also it “was the Text-to-Speech engine used in Android 1.6 “Donut” ”. [12] “The Pico Text-to-Speech (TTS) service uses the TTS binary from SVOX for producing spoken text.” [36]. “The Pico service produces audio streams using WAV containers and PCM (signed) codec with 16bit depth.” [36]. SVOX Pico TTS is licensed under an Apache 2.0 License [37] [38] [39]. Pico TTS supports the following languages [36]:

- German (de-DE)
- English, US (en-US)
- English, GB (en-GB)
- Spanish (es-ES)
- French (fr-FR)

- Italian (it-IT)

The following assumptions were developed during testing. The quality, in comparison to the other three TTS engines is quite good. The voice that is deployed is not as robotic as the others and is clearer to understand. The installation is easy and the language support is sufficient for now with six languages.

3.3. Web Application

This part of the documentation begins with some background information about Flask, then focuses on step-by-step instructions for the other web application's components.

3.3.1. Flask

Flask is a BSD-Licensed python web framework. [40]

Flask defines itself as a so called “microframework”. It comes with the bare minimum to get a web application started and does not require any specific database software to be run with it. “Flask supports extensions” to add, for example, a database abstraction layer to the application “as if it was implemented in Flask itself” [41]. The extensions it does ship with, for example the templating engine, can easily be changed if desired. Detailed installation instructions for Flask can be found on the projects website [42]. A very active community has formed around the use of Flask with free tutorials and videos that aide in development. Flask is also supported by the community, which offer a “variety of extension” that can be integrated into personal projects. Mature extensions, that have been approved by the Flask core team, make their way in to the future releases [41]. The framework has no opinions on how an application should be structured, giving the developer full creative freedom. It is recommended by the developers to use a more established webserver such as NGINX or Apache, because “Flask’s built-in server is not suitable for production” [43]. A more battle tested web server will help insure that the application can run more robustly against a high amount of incoming traffic, otherwise Flask “doesn’t scale well” [43]. To deploy the Flask application to Web Server Gateway Interface (WSGI) server, a detailed documentation is given by the developers [43].

The reason why Flask is used, is that the documentation is clean and understandable for beginners. A detailed overview of extensions and deployment options are also available. As Flask is a web framework written entirely in python, it supports the ability to keep all components in one programming language. The project does not have the requirements that call for a full featured Model-View-Control framework like Django, another popular python framework[44]. Keeping simplicity in mind, the Berta web application is a Single-page application (SPA), a single page that dynamically updates parts of its view.

3.3.2. gunicorn and supervisor

As mentioned above in 3.3.1, it is recommended to run a more robust web server than the one flask ships with. As a production web server, the decision came down to two options, gunicorn and uWSGI. Both were implemented and tested. The documentation for both is good, but as uWSGI is more demanding and has more complex configuration options, gunicorn was deployed. Gunicorn [45] is also python based. “Gunicorn ‘Green Unicorn’ is a Python WSGI HTTP Server for UNIX. It’s a pre-fork worker model.” [45]. Gunicorn is compatible with various web frameworks including Flask. Gunicorn can be downloaded at the gunicorn website [45], which provides clear documentation [46] and deployment [47] instructions. The Supervisor tool is a process control client/server system. It allows for monitoring and controlling various processes on UNIX-like operating system [48]. Supervisor, again written in python, is used to start the gunicorn server on system boot up and a restart in case the application should crash.

3.3.3. NGINX

NGINX is a BSD-Licensed published Webserver-software [49]. NGINX Open Source is an established open source web server [50]. The NGINX Application Platform is a “high-performance application delivery for microservices” [51]. NGINX Inc. has many products, for example: NGINX Plus, NGINX Controller, NGINX App Protect, NGINX Unit, and NGINX Amplify [51]. Detailed installation instructions for a Prebuilt Debian Package from an OS Repository can be found on the NGINX documentation [52]. The documentation also includes different installation manuals for different operation systems. The approach to use NGINX has

been included for possible further work and expansion on the project and the main reason is that it is recommended by the developers of Flask 3.3.1 to use a more established webserver like NGINX with it [43].

3.4. Database

In the following the SQLite 3 database is introduced as well as SQLAlchemy Object Relational Mapper (ORM)

3.4.1. SQLite3

For persisting the data created by the application, a relational database was chosen. The choice came down to SQLite and MySQL database management system [53]. Ultimately, SQLite was chosen for its serverless design and ease of use. Furthermore, SQLAlchemy, explained below, allows for database independent development, allowing for a more sophisticated database management system to be used in the future, if desired. “SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.” [54]. SQLite acts as part of the code and not as an outside resource. SQLite can be used on many machines without software specific to it. It can be used in small or medium-sized applications. The developers say that the more memory it gets SQLite runs faster but the performance can be “usually quite good even in low-memory environments” [54]. SQLite is written in C but there are numerous extensions that makes it useful with other languages, for example, Python [55]. SQLite source code is according to the developers “absolutely free to anybody who wants it” [54]. SQLite is released under a Public Domain License [56]. Extension products and enterprise support can be purchased if needed [57]. The manufacturer offers a vast amount of documentation [58] and instructions for downloading SQLite [59] on their website. This project uses the current version of SQLite, “SQLite3”. The latest version can be found at the SQLite website [60] “Version 3.35.5(2021-04-19)”. The python documentation [55] for installing SQLite3 is used. The reason for using SQLite3 was to provide a self contained database that was easy to use, could travel with the application using it, and run on a Raspberry Pi 4 without any additional required software.

3.4.2. SQLAlchemy

SQLAlchemy is an ORM [61]. “ORMs allow applications to manage a database using high-level entities such as classes, objects and methods instead of tables and SQL. The job of the ORM is to translate the high-level operations into database commands.” [62]. In reverse terms it is used for working with data as Python object.

SQLAlchemy supports the following database engines SQLite, MySQL and PostgreSQL, amongst others [63].

3.5. Other basics

In the following section the duck typing concept is presented. It is used for the implementation of the plugin system of the application.

3.5.1. Duck typing

“If it walks like a duck, and it quacks like a duck, then it must be a duck.” or also in other terms based on the book “Exploring Indiana Highways: Trip Trivia” from Michael Heim [64] is a common phrase for the duck typing concept. It is a programming concept that is used by object creation. In this context, a class with methods is given, objects created by this class are less important as the methods that the class has implemented. The keypoint of using this concept is that only methods or attributed will be checked whether these are present or not. The type of the actual object is secondary. [65]

Chapter 4

System concept

The following chapter presents the system concept, which consists of the product environment, product requirements, and product structures and architectures. The user interface and its functionalities are also described here. Furthermore, the acceptance criteria and installation procedure are shown. Also, the deployment of the application is addressed. Finally, the evaluation measures are discussed in relation to the research questions from 1.2

4.1. Product environment

The developed system runs on a Raspberry Pi 4 with Raspberry Pi OS. The web client should be usable via modern HTML5-capable web browsers. This includes most mobile browsers as well.

Software: The Berta voice assistant application is used via a standalone device, here, a Raspberry Pi4 as mentioned above. Attached to the raspberry is a microphone and speaker. The user can log in to the application via a web browser and interact with the software. The business logic is run as an additional Python application on the Raspberry. The data is stored using an SQLite database with the help of an ORM, which is also located locally on the device. An existing internet connection is required to use the Berta application.

4.2. Requirements

In the following section the functional and non-functional requirements are listed. The requirements are presented in lists and are intended to be decisive for a common understanding of the product requirements.

4.2.1. Functional requirements

The functional requirements in the criteria listed below have been established during the development of the project. The table 4.1 shows the functional requirements.

Table 4.1.: Table with functional requirements

Functional requirements			
Number	Requirement	Description	Keyword
FR01	Simple function	The application must have a function that is simple to execute and show the basic functionality of the system	"simple"
FR02	Time announcement	The application must be able to announce the time after a voice prompt.	"time"
FR03	Date announcement	The application must reproduce the date in response to a voice prompt.	"date"
FR04	Default function	The application must have a default function, if the spoken command is not understood, the user must be informed verbatim that she was not understood.	"incomprehensible spoken words"
FR05	Timer	The application should be able to set a timer after a voice prompt.	

Table 4.1.: Table with functional requirements

Functional requirements			
Number	Requirement	Description	Keyword
FR06	Interrupt function	The application should be able to interrupt actions, e.g. interrupt playing music after a voice prompt.	
FR07	Play music	The application should be able to play music when prompted to do so.	
FR08	Weather query	The application should be able to announce the current weather after a voice prompt.	
FR09	Weather forecast	The application should be able to make a weather forecast after a voice prompt.	
FR10	Listen to the radio	The application should be able to play the radio after a voice prompt.	
FR11	Calculator	The application should be able to perform simple mathematical calculations after a voice prompt.	
FR12	News summary	The application should be able to announce collected news reports after a voice prompt.	
FR13	Gimmicks	The application should be able to perform gimmicks like jokes after a voice announcement.	"joke"
FR14	Read a book aloud	The application should be able to read a book aloud after a voice prompt.	
FR15	Repeat function	The application should announce and execute the last command after a voice prompt.	

4. System concept

Table 4.1.: Table with functional requirements

Functional requirements			
Number	Requirement	Description	Keyword
FR16	Train/bus timetable	It would be nice if the application could announce train and bus timetables after a voice prompt.	
FR17	Calendar	It would be nice if the application could make calendar entries after a voice prompt.	
FR18	Write an email	It would be nice if the application could write emails after a voice prompt.	
FR19	Write notes	It would be nice if the application could write notes and shopping lists (text files) when prompted by voice.	
FR20	Undo function	It would be nice if the application could withdraw the last performed command, if applicable.	
FR21	Lighting control	It would be nice if the application could turn lights on and off, or dim them in response to voice prompts.	
FR22	Play music streaming service	It would be nice if the application could play a music streaming service, like YouTube or Spotify after a voice prompt.	

The requirements came about during the elaboration of the project, have been expanded upon and leave room for further development. These are the requirements of the Berta system itself. Apart from the "must have" requirements mentioned above,

there is no implementation obligation. With this in mind, a functional requirement is the evaluation of the voice command with the corresponding result.

The table is divided into four columns. A number has been assigned to the first column, so that reference can be made to the individual functional requirements in the later in the work. The actual name of the functional requirement is in the second column. The third column gives a description of the desired functionality. The last column contains the keywords for the functional requirements that has been implemented. A keyword for a functional requirement is the command that must be spoken, in a full sentence or alone, into the microphone of the Raspberry Pi, so that a functional requirement can be processed and executed internally.

4.2.2. Non-functional requirements

The non-functional requirements have been self specified and applied to parts of the quality model of ISO 25010 / ISO 9126. [66] [67] The table 4.2 shows the self chosen non-functional requirements.

Table 4.2.: Table with non-functional requirements

Non-functional requirements			
Number	Name	Description	Method
NF01	Security / privacy	The software should be developed and designed in such a way that help protect the security and privacy of the user	<ul style="list-style-type: none"> · Local running application · Passwords are not stored in plain text, but are hashed · SSL certificate for Hypertext Transfer Protocol Secure (HTTPS)
NF02	Mature software quality	The software code should be written in a way that is understandable / traceable. The software quality should be simple.	Use a code style guide and Code comments.

4. System concept

Table 4.2.: Table with non-functional requirements

Non-functional requirements			
Number	Name	Description	Method
NF03	Expandability	The application should be able to be further developed by others.	Sufficient / detailed documentation.
NF04	Installability	The application should be easy to install.	GitHub repository to clone the project with installation manual.
NF05	Portability	The application should also be usable on other Raspberry versions.	
NF06	Usability	The application should have a user interface to see the voice recognition results and to be able to visualize and give the possibility for later adjustment.	A web interface that can be accessed.

4.3. User interface

The following figures show the Berta applications Web view. The wireframes are part of the concept for the user interface and can be further developed.

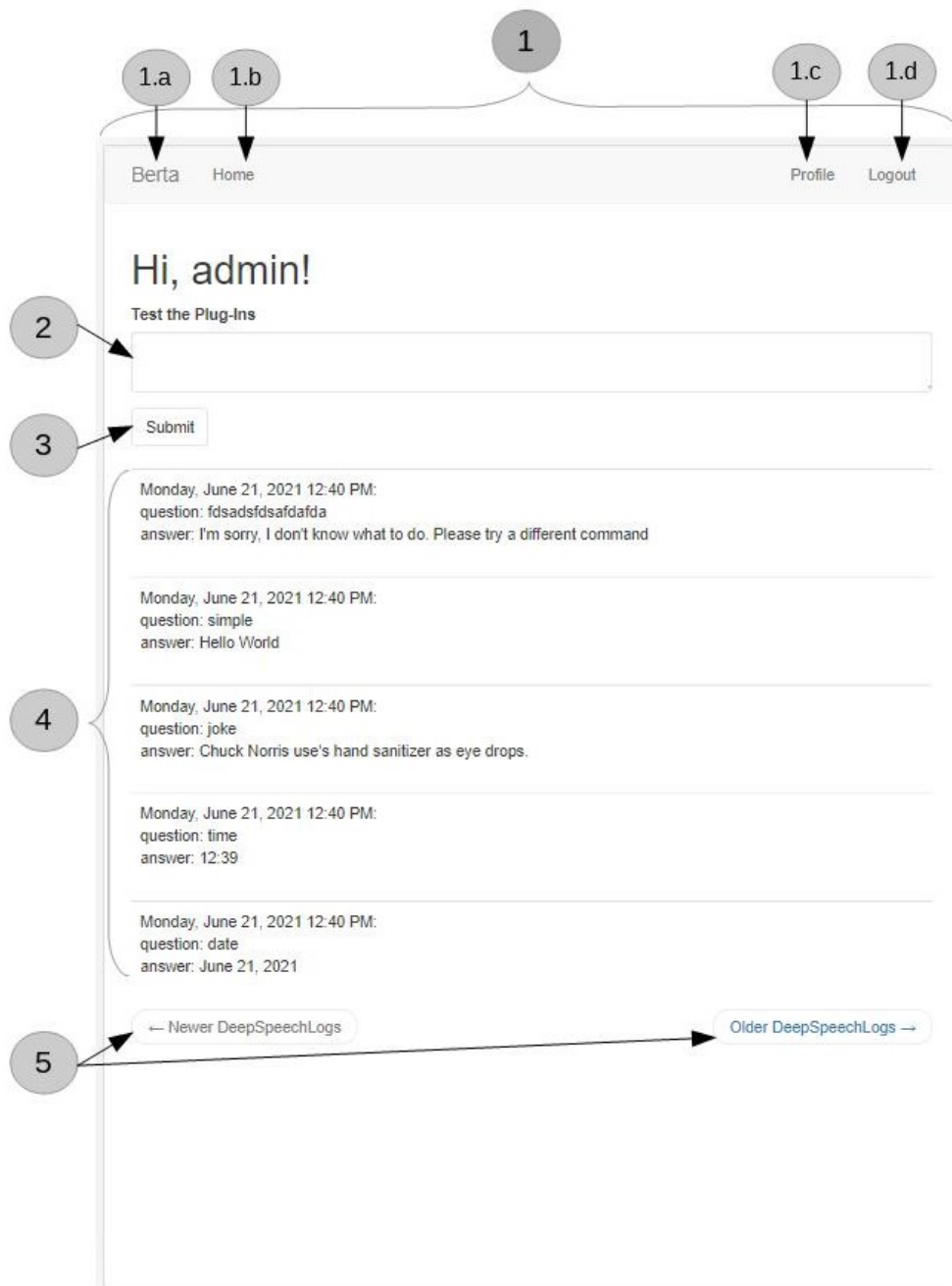
Berta home**Figure 4.1.:** Berta Home

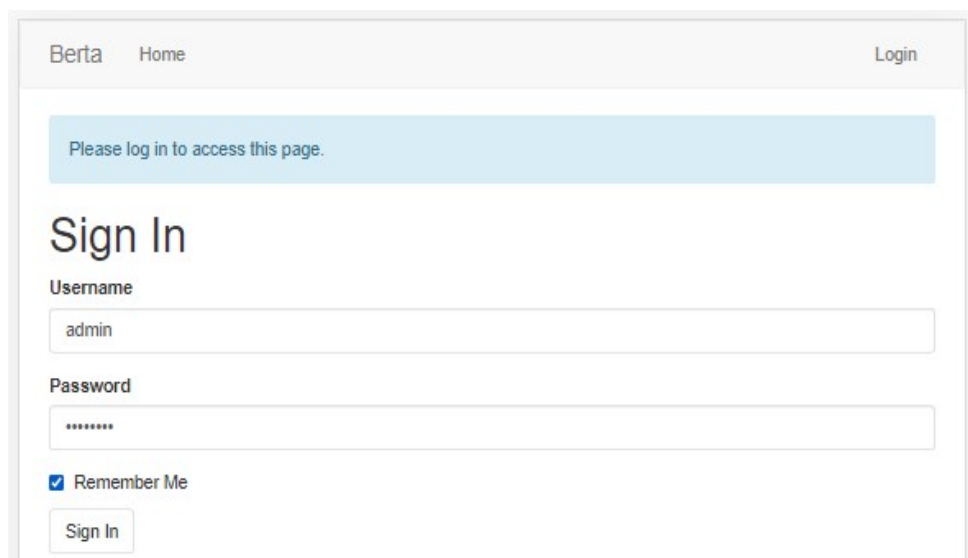
Figure 4.1 shows the Berta user interface in the logged-in state. The numbers are for describing the components. What follows is a detailed description:

1. Navigation bar:

4. System concept

- a) Placeholder for a logo of the application.
 - b) Home button to redirect to the main page
 - c) Profile button forwarding to the user profile
 - d) Logout button
2. Entry bar: This feature enables the user to test the plugins by hand rather than by voice. At this point, only the keyword can be found in 4.2.1 or a whole sentence with the keyword can be written.
 3. Submit button: The user sends his request to the application, the question and the answer is then converted to a “DeepSpeechLog”, which is saved in the database, and displayed in the feed below.
 4. Feed: The user can see all “DeepSpeechLog”s through this feature
 5. Navigation: The user can view 5 “DeepSpeechLog”s at once. Older logs can be reached through navigation.

Berta login page



Berta Home Login

Please log in to access this page.

Sign In

Username

admin

Password

☒ Remember Me

Sign In

Figure 4.2.: Berta login page

The figure 4.2 shows the Berta user interface when not logged in.

Berta user page

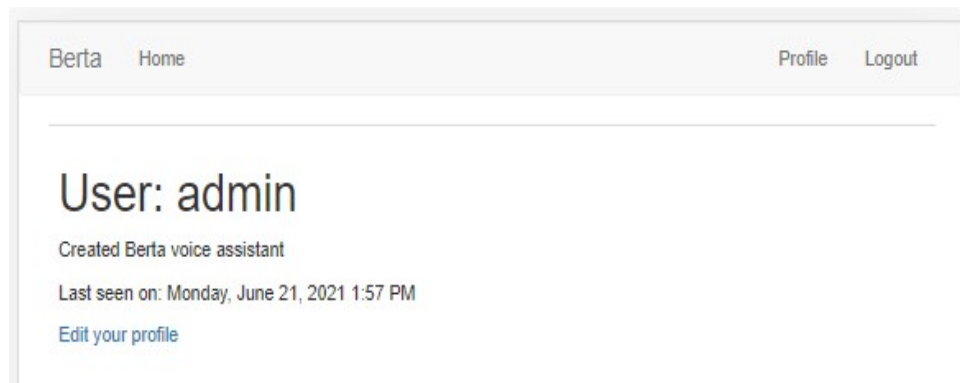


Figure 4.3.: Berta user page

The figure 4.3 shows the Berta user interface user page view. A short description of the user can be set. A “last seen on” text is viewed and an edit profile function is given.

4.4. Architectural solutions

This section describes the separate technologies used in the berta application.

4.4.1. Berta architecture

The following figure 4.4 shows the architectural design of the Berta voice assistant. It is used to better understand how the technologies are connected to each other.

4. System concept

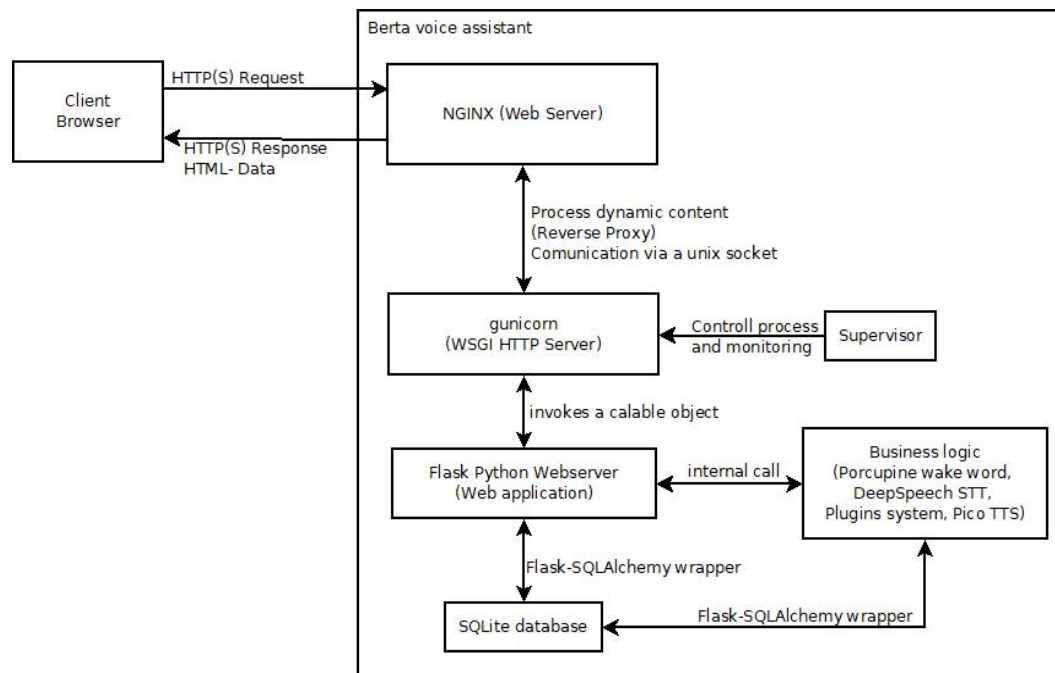


Figure 4.4.: Berta architecture

This Application uses a Python Flask Web application on Raspberry Pi 4 with NGINX, as well as gunicorn with supervisor. For the database, a SQLite database is used. The business logic implements Porcupine wake word engine, DeepSpeech STT engine, Pico TTS engine and a rudimentary plugin system.

Although berta deploys a web application, no hosting provider is needed, as it is all hosted locally on device. The operating system used is Raspbian, a Debian based linux distribution, specifically made for the raspberry. NGINX is used as a reverse proxy and handles the incoming requests. The aforementioned requests will then be passed on to the application. Gunicorn is used as the applications server. It is for executing Python code in relation to the HTTP request. The gunicorn production server is monitored by the supervisor tool. If gunicorn should crash, the supervisor tool will automatically restart it. Supervisor also starts the gunicorn process in case the system needs to be rebooted. The Flask application is WSGI compatible and can receive and forward web requests it receives through the application server. The Flask-SQLAlchemy is used as a wrapper to access and use the relational database SQLite. It also transfers the database data into python object. The business logic and Flask application have full access to the database. These can write to, read from, delete, and change data in the database. The Web application and business logic communicate between each other with internal calls.

4.4.2. Database model

The Berta source Database consists of the following tables:

users	
*id	integer auto
*username	varchar(64)
*email	varchar(120)
*password_hash	varchar(128)
*about_me	varchar(140)
*last_seen	datetime

deep_speech_logs	
*id	integer auto
*question	varchar(140)
*answer	varchar(140)
*timestamp	datetime

Figure 4.5.: Database tables

The following describes the “users” table 4.5. Each user has an id, which is used as the primary key and is therefore uniquely stored and automatically assigned by the database. A user also has a username, an email, and a password_hash field. For security reasons, the password is not written directly in to the database. It is secured as a password hash. This is implemented with the package Werkzeug which is one of Flask's dependencies [68] [69] [70]. A user can optionally write some information in the about_me field. The last_seen field saves the last time the user accessed the web application. It is displayed in the user profile. This field is automatically set and updated by the database.

Now, the “deepspeechlogs” table 4.5 is described. Each deepspeechlog has a unique, automatically assigned id and it is the primary key. The question field saves the spoken words the user gave after triggering the wake word. The answer field saves the evaluated question result. The timestamp field is automatically assigned by the database. It is displayed with the question and answer fields on the home site. This is used for checking the accuracy of berta voice assistant.

4.4.3. Voice user interface

The figure 4.6 shows the voice user interface. It should clarify the interaction with the voice assistant and its internal process when using a voice command.

4. System concept

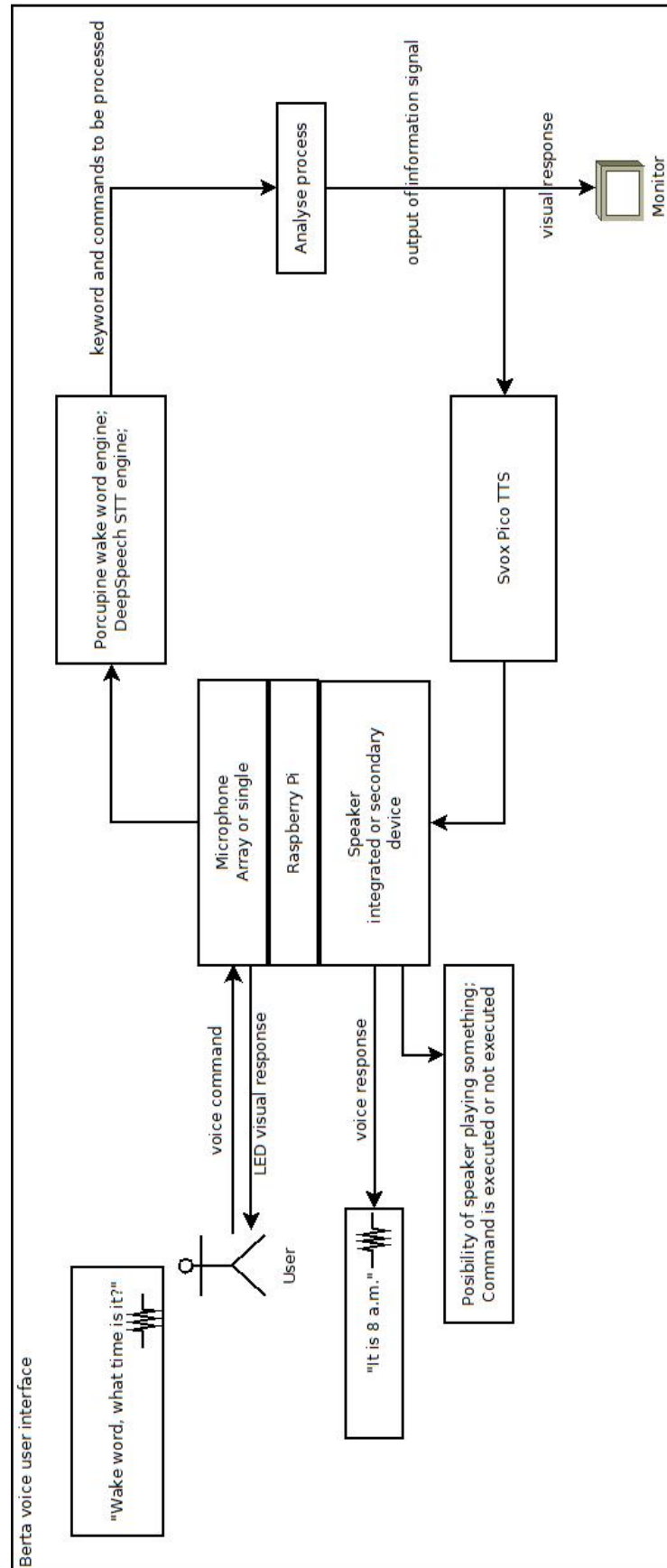


Figure 4.6.: Berta voice user interface

The User activates the Berta voice assistant via wake word through a Microphone that is connected with the Raspberry Pi. After a visual LED response, the user speaks a voice command. The LED of the microphone lights up during recognition phase. After the wake word recognition with Porcupine and the DeepSpeech speech processing, the speech to text transfer is done and the keywords inside the spoken command will be analysed. Succeeding the analysis process, the output of information is signaled in two ways. First, in a visual manner to the web application. Second, through a speaker, output by the SVOX Pico Text-To-Speech. No matter if the spoken command can be acted upon or not, an audible response is given back to the user to confirm processing has completed. The flow of the user voice interface is shown in 4.4.4

4.4.4. Berta flowchart for voice command

The following flowchart 4.7 for the process flow of using the voice assistant has been developed.

4. System concept

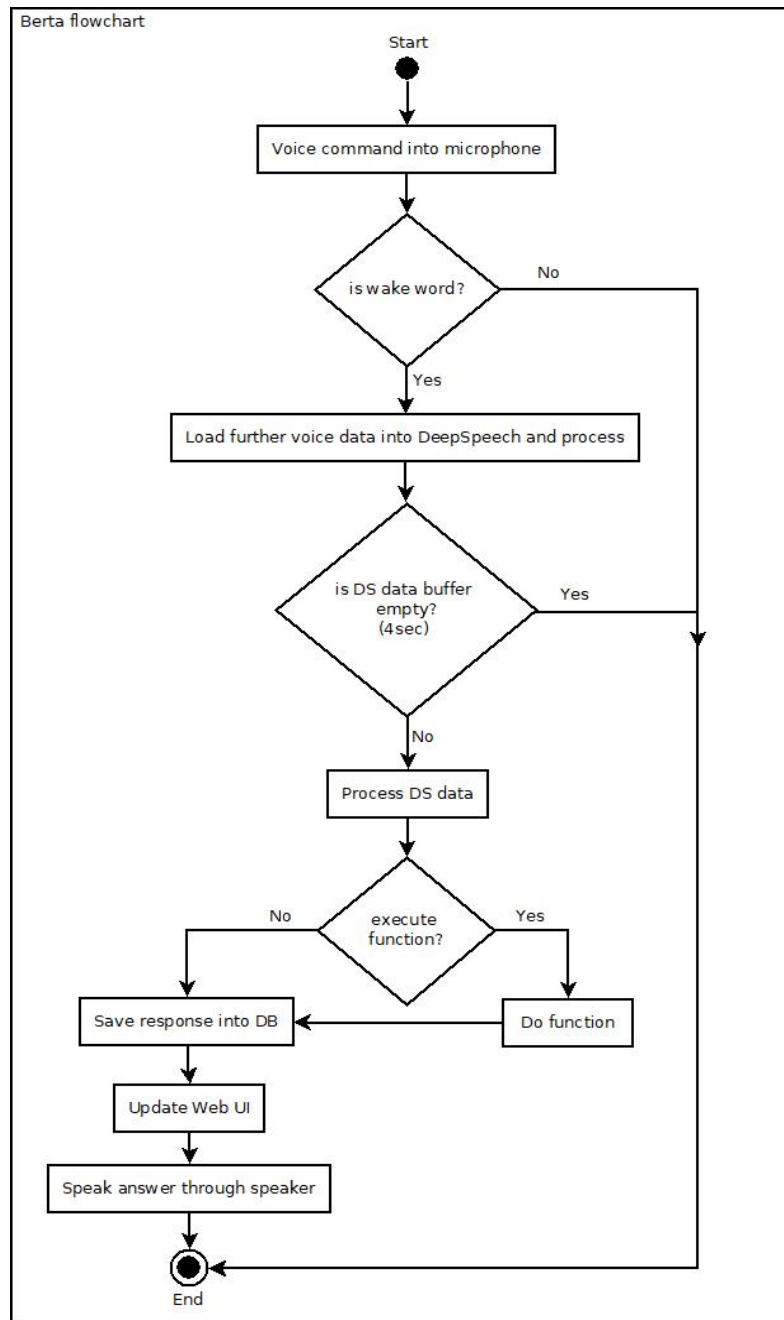


Figure 4.7.: Berta flowchart for voice command

The first step in the process is to trigger the wake word event. At first, the system checks if the wake word is spoken. If the wake word is not uttered, the system continues to wait for further input, i.e. continues to scan for the wake word. After the wake word is processed, system then continues to gather voice data, until it senses that no further speech data is put into the system. This data is then sent to DeepSpeech for further processing. Afterwards, the generated text is analyzed.

If the system has a function attached to the spoken set of words, the function is executed, and the response that is generated is returned, synthesized through the TTS engine, and spoken through the connected speaker. If the system does not have a function attached to any of the recognized words, i.e. the system did not understand what it should perform, it returns a generic response, asking the user to try repeating or try a different voice command. In both cases, the analyzed voice command and its response is persisted into the supplied database. These entries are then visually displayed in the web interface.

4.4.5. Berta plugin class diagram

To understand the structure of the plugin structure, the following figure was created.

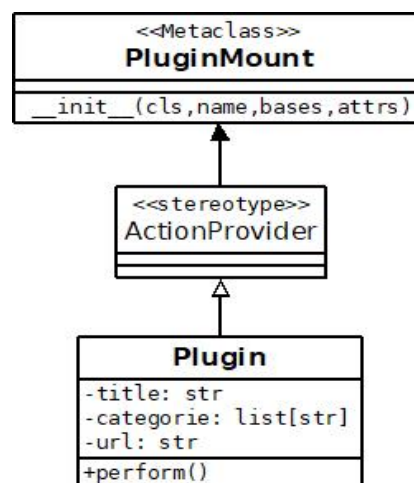


Figure 4.8.: Berta plugin class diagram

As mentioned in 3.5.1, the concept of ducktyping was applied to the plugin system. A metaclass “PluginMount” is defined, which creates a list of classes, which all plugins can register with upon creation. The “ActionProvider” class is of type Pluginmount, and serves as the basis for all further plugins. All common settings to all plugins can be set here, if needed. Every plugin that is created, needs to be a subclass from ActionProvider, so that it is automatically appended to the list of available plugins.

4.5. Technical product environment and technologies

The table 4.3 shows the used Technical product environment and technologies.

Table 4.3.: Table with technical product environment and technologies

Technical product environment			
Number	Purpose of use	Tool	Reason
TP01	Development environment	Visual Studio Code and Command line	The decision is left to the developer.
TP02	Versioning/ Installation manual	GitHub	Well-known version management tool with a wide range of functions.
TP03	Programming language	Python3	Uniform programming language can be used in every area of the project.
TP04	Python Package Manager	PIP	Conventional practice.
TP05	PIP Virtual Environment	virtualenv	Conventional practice
TP06	Engines	Procupine wake word engine, Deep Speech STT, SVOX Pico TTS	Evaluated, tested and found to meet requirements.

Table 4.3.: Table with technical product environment and technologies

Technical product environment			
Number	Purpose of use	Tool	Reason
TP07	Webframework	Flask, gunicorn with supervisor and Nginx	<ul style="list-style-type: none"> · Flask: lightweight, easy to learn, Python based, protect the form against CSRF attacks · gunicorn with supervisor: “compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy” [45]; supervisor for monitoring gunicorn · nginx: robust, SSL support, Reverse Proxy support
TP08	Databases	Python and SQLite with Flask-SQLAlchemy	<ul style="list-style-type: none"> · User friendly concept to use another database · Flask-SQLAlchemy extension wrapper to the SQLAlchemy package, which is an ORM · SQLite does not require a server
TP09	Deployment	Raspberry Pi4	Requirement for the thesis.

4.6. Acceptance and installation

The delivery of the bachelor thesis to the supervising professor takes place in the form of the written bachelor thesis and a GitHub repository, as well as a .zip file with the developed system. This repository contains all components of the system that are necessary to operate the software on a Raspberry Pi 4.

In addition to the executable source code, the GitHub repository contains more detailed installation instructions. System documentation is also included, in the sense of the system requirements and required software. The installation instructions explain exactly which steps are necessary to get the system up and running on a Raspberry Pi 4. In addition to the executable product, the file also contains the source code with documentation.

4.7. Deployment on other computer systems

As this project was specifically designed with the Raspberry Pi 4 and the given components in mind. Currently there is no way to deploy Berta to a different type of computer system, although adapting Berta to be deployed on different systems is trivial. The main difficulty would be to separate the daughter board specific code out into it's own python package and only include it if the system requirements are met. Additionally, the installation script provided is specifically tailored to debian derived linux systems, as other linux distributions use different package managers and might not even contain the required dependencies.

Chapter 5

Implementing the concept

This chapter describes an implementation to the concept outlined in chapter 4. The methodology of the approach to implement the voice assistant is presented. This is followed by an overview of the open source components used and the adaptations made for the project. Finally, the connection between the different components on how they are connected and used is illustrated.

5.1. Methodology

The given condition is that an engine that recognizes language and outputs it textually is given. The state of affairs is that spoken commands should call up various functions. The desired end result as to how the application should run was written down in simple terms. A depiction of the simplified sequence of events is provided below:

5. Implementing the concept

Process flow sketch:

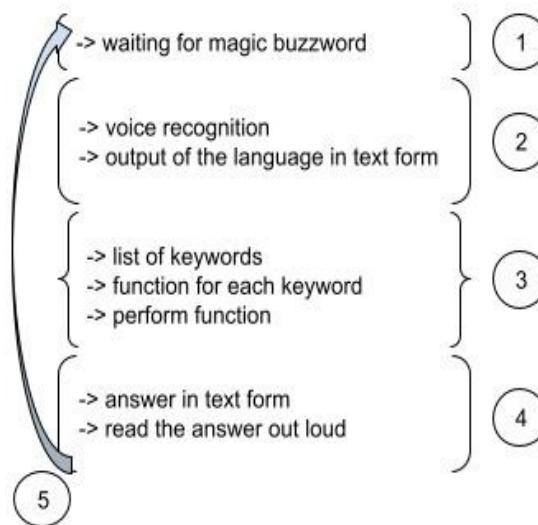


Figure 5.1.: Berta simplified process

The sequence could be divided into components and processes in an uncomplicated manner. The numbers are for describing the components/processes below. From this, components that are needed could be determined.

1. Wake word engine
2. Speech-To-Text engine
3. Analysis process for processing the text
4. Text-To-Speech engine
5. The process should be able to start all over again

Pico Voice is the wake word engine that was chosen for this project. Mozillas DeepSpeech, the Speech-To-Text, was given as the main component. The voice assistant should be built on the basis of this engineering. The analysis process had to be implemented by manually. SVOX Pico engine was selected as the Text-To-Speech engine.

While researching for “related work” in 2.2, different projects documentation affirmed the structure outlined above.

During development it turned out to be sensible to first set up the components of the voice assistant individually. The reasoning for this is that it makes it much easier in evaluating if the components are even suitable in the proposed set up. With this

approach, the voice assistant could be implemented step by step. Early development on the Raspberry Pi required a fresh install of the operating system, Raspbian OS, a mouse, keyboard, monitor, and cable network connection. After initial set up was completed, Virtual Network Computing (VNC) and Secure Shell (SSH) were activated and used for further development. The engines were each installed and tested on the Raspberry Pi4. For usability reasons a web interface to visualize the voice commands and the resulting responses was created. A web framework based on python that is easy to learn was used to accomplish this task. Flask is the web framework that was deployed. After testing the different components separately, all components were combined while keeping the licenses, python compliance, raspberry pi compliance, and resource management in mind. To continue with the “Keep It Simple and Straightforward (KISS)” [71] principle, the installation file was written in bash. An installation manual is given in the GitHub repository on how to set up and run Berta home devices. As the provided guides, documentation, and tutorials contribute a strong foundation to the project, it is not enough to use these as is without additional modifications and additions.

What follows is a methodical description of the different components on how they were used and in what context.

5.2. Wake word and STT implementation

The project `DeepSpeech_RaspberryPi4_Hotword` by Dmitry Maslov [72] was the basis for the wake word and STT components. It was supplemented with the following additions. The wake word detection for the words "bumblebee" and "computer" was added. The wake word "hey_berta" only works if a suitable .pnn file is inserted in porcupines keyword folder. The LED control on the microphone board with the colors blue, white and red. The LEDs will go on when the wake word is spoken, then they will light up brighter during the voice command capture. When the recognition is stopped, the LEDs will slowly start blinking, signaling that the system is currently processing the given command. After the response is spoken, the LEDs will turn them self off. Voice commands are saved in a Database for further processing and displaying in the web application. The speech recognition can be stop by issuing the command “stop recording” or after 4 seconds of silence.

5.3. Plugin implementation

The tutorial from Marty Alchin[73] was used to set up the plugin system. This tutorial is based on Python2 and was used to understand and build the plugin structure. The actual implemented plugin system is modified to be python3 compliant and implements ducktyping, as mentioned in 3.5.1 and designed in 4.8.

A metaclass is implemented called “PluginMount” in “pluginmount.py”. This class serves two functions. First, when instantiated, it sets up a list for all plugin to register with. Second, after a plugin is instantiated, it ensures that the plugin is added to the list, in order to keep track of it. A second class called “ActionProvider” uses PluginMount as it’s type and is the basis for all other plugins. Common settings and a shared constructor can be set up here if desired. All plugins must be inherited from ActionProvider to ensure proper addition to the plugin list. Further more, all plugins should provide the following attributes and functions:

- “title” [string] : The text describing the plugin
- “url” [string] : Uniform Resource Locator (URL) for the Application Programming Interface (API) endpoint from which any data should be extracted from
- “categories” [list(string)] : Words which classify the plugin and allow the system to react upon
- “perform()” [returns string] : The actual function the plugin should perform

Below is an implementation example of a Generic plugin:

```
class GenericPlugin(ActionProvider):
    """ Class Documentation String """
    # Every plugin must have a title that describes it
    title = "My generic plugin"
    # Every plugin must define at least one key word it should react upon
    categories = ['generic']
    #if necessary, provide an URL of a foreign API
    url = 'http://some.api.com/news'

    def perform(self):
        """ Action to be perfomed """
        # ...
        # API Processing here
        # ...
        return answer
```

Listing 5.1: class GenericPlugin

During development it turned out not all API's easily accessible. Some need an additional registration on the API's operator's website in order to receive and ultimately be able to use the given URL. Theoretically, a link to the login page of the API operator would be conceivable and a field for entering the authentication data API key and other needed functionalities, but this is a function that should be dealt with in later work.

5.4. TTS implementation

The connection to the TTS engine is given with a simple python wrapper package, as the actual engine is written in C. The python packages allows for passing in a string of text and receiving a wav audio object back for further processing. The python package used is "py-picotts"

5.5. Web application extensions and implementations

The Python Flask Web Application is based on "The Flask Mega-Tutorial" of Miguel Grinberg, which is based on the book "Flask Web Development: Developing Web Applications with Python", from the same author [74]. The tutorial and accompanying code is released under an MIT-Licence [75]. A Github repository with further information on developing web applications with flask is also available [76].

Flask has the ability to deploy custom error pages for Hypertext Transfer Protocol (HTTP) errors. This feature is used for the 404 and 505 errors pages respectively. Also a Rotating FileHandler [77] is used for receiving and writing errors in a log folder when the application is not in debug mode. The log entries also show when the server was restarted and when the application runs on a production server.

The web application ships with:

- a User login / Logout
- a Profile Page with Edit Function
- the ability to show the operations of the voice assistant

The Flask extensions used to develop Berta's web application is described below.

5. Implementing the concept

To avoid writing static, bare Hypertext Markup Language (HTML) content, a templating engine is used. Jinja2 [78] is the default templating engine provided by the Flask framework. Web forms are constructed using the Flask-WTF extension [79]. Python classes can be used to represent different web form using this extension. The extension also provides protection against Cross-Site-Request-Forgery (CSRF) attacks, by deploying a configurable `SECRET_KEY` variable [80]. In production, the `SECRET_KEY` should be changed to a unique value in the provided configuration file. All HTML files must then include the `form.hidden_tag()` function to provide maximum security. This “template argument generates a hidden field that includes a token that is used to protect the form against CSRF attacks” [81].

Flask-Migrate [82] created by Miguel Grinberg is also used. “This extension is a Flask wrapper for Alembic, a database migration framework for SQLAlchemy.” [62]. This extension facilitates working with the database. When modifying the schema of the database, it supports in making the changes as easy as possible. This allows for tracking changes in the schema in source control, as every time an update is made, an update script is produced with the delta changes from one version to the next. Rollbacks to previous versions in case of introduced bugs or mistakes can effortlessly be executed. More information about Flask-Migrate can be found on the extensions website [62] [82].

The Flask extension Flask-Login is used for managing the users logged-in state. It allows for navigating the website, while keeping a user session open. The extension also supports a “remember me” functionality, allowing the user to navigate away from the Berta web application, return, and still be logged in to the system. [83].

Flask-Moment extension uses the Moment.js library for formatting dates and time. All Jinja2 templates must include this library in order to function properly [84]. Moment.js library is an open-source JavaScript library that makes it possible to render dates and times in different formatting options.

The Flask-Bootstrap [85] extension has the Bootstrap [86] framework installed and is used to style the web page.

5.6. Configuring Gunicorn, Supervisor, NGINX, SSL certificate

Gunicorn configuration is provided in the `berta.conf` file, found in “deployment/supervisor”. By default, gunicorn listens on localhost port 8000 for incoming requests. In this scenario, as this is currently a single user application, only one worker thread is deployed. This can be changed in the future to accommodate scaling issues that might arise. Supervisor uses the same configuration file, in order to monitor the application and restart if needed. NGINX is then used as the public facing web server that accepts requests on the default ports 80 and 443. Unencrypted http requests on port 90 are automatically redirected to port 443 to enforce SSL encryption. During installation, a set of self-signed certificates are created, in order to enable SSL functionality. The configuration file “berta” can be found at “deployment/nginx” if defaults need to be changed for any reason. NGINX is not a strict requirement for this project and can be exchanged with any webserver of choice [87]

5.7. Database implementation

Flask-SQLAlchemy extension [88] is used as a wrapper for SQLAlchemy [61]. An SQLite database is used in this project. If another supported database is intended to be deployed in the future, this can easily be achieved in the SQLAlchemy settings. Because Flask-SQLAlchemy configuration offers a `SQLALCHEMY_DATABASE_URI` variable. This can be set additionally to the configured database named `app.db` in the `config.py` file. If the `SQLALCHEMY_DATABASE_URI` is set, Flask will use the configured database instead of the default one [89].

5.8. Connection of components

All necessary system components are shipped inside the provided in the accompanying zip file and GitHub repository respectively. To ensure proper installation, an install bash script is included. This bash script downloads the appropriate deep-speech model, makes sure the operating system is up to date with the required kernel version, and install all operating system level dependencies. Furthermore, it downloads the correct drivers for the Raspberry Pi to interface with the microphone

5. Implementing the concept

daughter board, as well as the correct deepspeech model required for operation. Lastly, the script then generates the needed public and private keys needed for the webserver to provide an SSL connection, as well as setting up both the application and web server.

5.9. The attempt to build a Berta service

The basic idea of starting the both the deepspeech application and the web application came while configuring the supervisor tool. At first a naive approach of just starting the deepspeech application along side the web application with supervisor was implemented. This in turn did not allow the deepspeech application to access the operating systems sound channels. Berta could still accept speech data, but would get stuck processing the as the data could not be written to an audio output stream. Two additional monitoring / initializing programs were also trialed during this process. Runit, the default init system of Void and Artix linux, provided no additional benefit. Lastly, systemd, the default init system of many modern linux systems, was thoroughly tested, but to no success. A deeper understanding of the order of system unit initialization under linux, in addition to system permissions were lacking to accomplish this goal.

Chapter 6

Evaluation measures

What follows are the descriptions of the evaluation measures that validate the research questions introduced in 1.2. An assessment of the evaluation measurements and their experiments with their results are given in 7.3. The evaluation of the established system is grounded in functionality and user tests with 10 test subjects. The evaluation measures focuses on the behavior of the voice assistant in the following criteria and their valuation method is described.

6.1. Usability evaluation measure

The usability is tested with a usability test by voice communication with voice commands. Test subjects test the Berta voice assistant with these voice commands. RQ01 mentioned in 1.2, is evaluated during the dedicated experiment. After the Berta application has been tested by test subjects with voice commands, the same questions are asked and evaluated to Google Assistant and Siri introduced in 2.1. The experiment setup is described in 7.1.1 and the associated results are attached in the appendix A.

6.2. Offline ability evaluation measure

This setup also focuses on open source components and the ease of mind of not utilizing a users private data. RQ02 mentioned in 1.2 is evaluated during the dedicated experiment. To test and evaluate the offline ability, a software that analyses

6. Evaluation measures

data protocols is needed. This project uses for this evaluation Wireshark for this purpose. Wireshark is a network protocol analyzer that can be used to detect network traffic [90]. During the usability test 6.1, the offline abilities of the Berta voice assistant is tested at the same time.

Chapter 7

Experimental Setup and Evaluation

In this chapter the experimental setup and evaluation of it is presented. All evaluation measurements mentioned in 6 for the Berta application are analyzed and an initial experimental setup with its challenges is discussed. Moreover an evaluation described in 7.3 will lead to a new design for future work of the setup including all the problems, solution and optimizations that took place.

7.1. Intended experiment setup

7.1.1. Experiment setup usability

The experiment focuses on the behavior of the voice assistant. The setup of the experiment is very effective and efficient, in theory, as the different voice assistants tested receive the input with the identical pronunciation, nuance, and background noise every time. The experiment is structured as follows: There are 10 test subjects of which one female and one male sit directly in front of the application and speak into the device. The remaining eight test subjects, four female and four male, sent five commands listed below, as well as the wake words, individually, via voice message in WhatsApp and iMessage. These voice messages were then played back to the voice assistants. Due to the circumstances, no further notes on possible anomalies or suitability could be recorded. The following voice assistants are tested on the following devices.

- Berta voice assistant
 - Device: Raspberry Pi4

7. Experimental Setup and Evaluation

- Google assistant
 - Device: OnePlus 3T

To clarify the process, the scenario structure is described below. A list of the questions that were asked is also given. The experiment is started with the devices lying on a table and the voice messages are played to the voice assistants individually. The stored question and answer are then written down for later evaluation.

The following wake words are used:

1. For Berta is the wake word: "computer"
2. For Google assistant no wake word is needed

The following commands that are executed correspond to the functional requirement numbers described in 4.2.1:

1. FR01: "Do something simple!"
2. FR02: "What time is it?"
3. FR03: "What is today's date?"
4. FR13: "Tell me a joke"
5. FR04: "This is a Test!"

Google assistant was tested first, Berta afterwards. A set of attributes that describe the test subject, which include gender, mother tongue, and age group is documented as well. After each round, the current voice assistant, the answer, and device is recorded. The amount of tries it took to activate the wake word detection is also documented.

The results of the individual experiments with their results can be found in the appendix under A.

7.1.2. Experiment setup offline ability

The offline ability is tested during the usability testes with Wireshark introduced in 6. Wireshark is run in the background during these tests 7.1.1. The offline test consists of creating an isolated network in order to only capture relevant network activity. After the test is concluded, network traffic is then filtered by the associated IP address in Wireshark in order to analyze the data exchange.

7.2. Intended evaluation

The following evaluation questions can be answered about the results collected.

1. Was the wake word recognized when the voice message was executed once or did the tester have to help it out himself?
2. How many times did the wake word have to be spoken until the recognition phase?
3. Did it come to differences between female and male voice wake word recognition?
4. How good was the speech recognition of Berta after the 5 commands were spoken once each in comparison to google assistant?
5. Does the Berta voice assistant work offline?

7.3. Implemented experiment setup and evaluation

Due to the current pandemic situation, the planned experiment in which test subjects were to sit directly in front of the voice assistant was not possible. The attempt to test the Berta voice assistant via voice messages, described in 7.1, was inconclusive. The experiment was aborted with a total of 10 test subjects, five female and five male voices. Among them, two desk tests were performed with one female and one male test subjects.

7.3.1. Evaluating the usability

The results shown in 7.1 lead to the following conclusions: Googles voice assistant was almost completely able to understand all voice messages and process the command with little deviation. The Berta voice assistant could not cope with the German-English accent of most of the voice messages. It did not have the same problem with the two test subjects that performed the tests in front of the device. Occasionally, Berta would only understand the keywords but not the rest of the sentence. It can be concluded that a desk test with all participants would yield better test results for Berta. Development tests were aided through voice utterances of an American English speaking male. The speaker is understood optimally during the

7. Experimental Setup and Evaluation

desk test, which is reflected in the test figures. There is no noticeable differences between female and male voices while testing Berta. This however, is deceptive, as testing with voice messages was not very successful in general. Desk tests that were performed with other female speakers during development, revealed that Berta does not recognize them very well. 7.1 does not show an evaluation of spoken wake words, for the reason that if the test subjects had recorded a faulty wake word message, manual utterance of the wake word had to occur in order for the test to continue. A retrained DeepSpeech model trained with the German language, or at the minimum with a German accent would be highly recommended for future work.

Appendix B, 7.1 shows the resulting numbers.

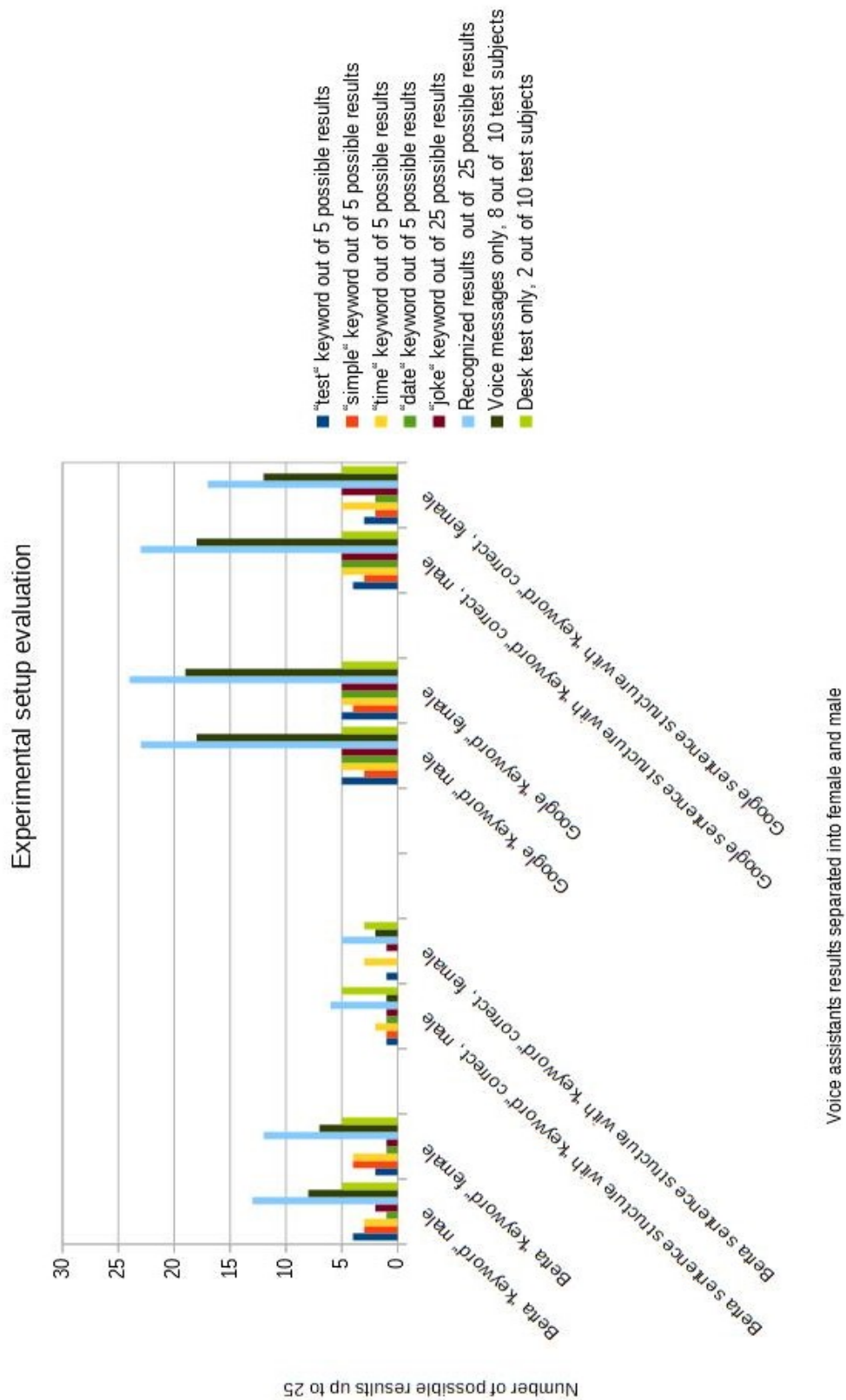


Figure 7.1.: Berta evaluation result diagram

7.3.2. Evaluating the offline ability

While testing the offline ability, nothing visible happens with Google, nor with Berta, but for Berta that was clear from the start. Google seems to be establishing a single connection and only sending the data it needs through it. Using wireshark to dissect the collected data to properly inspect the individual packets went beyond the scope of this thesis.

Chapter 8

Conclusion and future work

The goal of this bachelor thesis was to develop and evaluate a voice assistant system using the DeepSpeech open source speech-to-text engine from Mozilla based on a Raspberry Pi. For this purpose, this work first goes into chapter 2 by presenting existing popular known voice assistants and existing offline voice assistants. However, these are not based on DeepSpeech speech-to-text engine by Mozilla and were outside the scope of this work.

In chapter 4, this work presented the system concept for developing the speech assistant, which uses the Deep Speech speech-to-text engine by Mozilla and runs on Raspberry Pi4. The architecture and the interfaces between the other engines, components, and technologies used are shown here. In chapter 5 the methodology for finding the individual components was presented, as well as further details on implementation. Then in chapter 6 the experiment and the associated evaluation were presented. Which ultimately brings the work to an end here.

The conclusion is, an offline language assistant like Berta can play an important role in the everyday life of consumers in the future. The results of the present work show the potential of the technology. Possible risks with regard to data protection and the security of the systems can be eliminated. Of course, the Berta voice assistant still needs processing, but the approach alone shows that it is also possible to build an offline voice assistant with open-source components. This affirms the research question RQ01 in 1.2 that a voice assistant can be build with open source components. Concerns about the collection and use of data are not possible with the current system structure. It is designed with privacy by design. At the moment, the voice commands with their response are saved in a database after the activation of the wake word. However, these are not forwarded to any external cloud service, let

alone processed. This data is currently stored locally in the SQLite database used on the Raspberry Pi4, which is preferably connected to a network behind a firewall. This also affirms the research question RQ02 in 1.2 that a voice assistant can be built with open source components and function offline without using personal data.

Due to the use of open source components, the system cannot yet keep up with the large end devices from larger companies. The experiment and evaluation show that the Berta voice assistant can behave similar to already known systems. As the results show, there are problems with speech recognition, especially with female voices, which may be improved in future work. This thesis has also proven that the speech recognition of the DeepSpeech speech-to-text engine does not have a good recognition rate for English with a German speaking accent, which also may be improved in future work.

Perhaps with this thesis approach the basis has been created for further future thesis work. This work is freely available at the GitHub repository [91] and is intended to encourage further work.

Future Work

For future works there are several possibilities. The following ideas can be used to continue working on the Berta voice assistant. Since the Speech-To-Text engine is only represented with a pre-trained English and Mandarin speaking model, it would be possible to train and implement a model with a different, preferred language. For the future, it is recommended to retrain the DeepSpeech speech-to-text engine in German or English with a German accent.

The analyze area could be improved to process more difficult and complex commands. As well as new functional requirements in the plugin area could be implemented. Other languages except the given English one of the Text-To-Speech could be incorporated.

The user could be given the option of using the user interface in her preferred language. A registration for individual users is possible if the application is to be made accessible to multiple people living in the same household. Accordingly, it might need a database adjustment. A distribution of roles with admin and user areas would be conceivable. If changes had been made to the STT and TTS, the user could be

given the option to give commands to the STT in his preferred language and to receive an answer from the TTS. These could be set via the user interface.

After adapting the DeepSpeech engine, a new usability experiment with offline ability would be conceivable. In this case, someone should have specifically familiarized themselves with Wireshark and have a good understanding of network technology. The analysis of the network data collected with Wireshark turned out to be quite difficult without prior knowledge.

Another suggestion would be to create a Berta service. As mentioned in 5.9 it was not possible to create a Berta service because of a lack of Linux knowledge. To have an application that simply continues to work after power is restored would be desirable for the future.

Future challenges lie in making the application competitive, affordable in price, and making more functions available, if it is to be successfully launched in the marketplace.

List of Abbreviations

API	Application Programming Interface
BDSG	Federal Data Protection Act
CSRF	Cross-Site-Request-Forgery
de	German
DS2	DeepSpeech 2
en	English
es	Spanish
fr	French
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ORM	Object Relational Mapper
SPA	Single-page application
SSH	Secure Shell
STT	Speech-To-Text
TTS	Text-To-Speech
URL	Uniform Resource Locator
KISS	Keep It Simple and Straightforward
VNC	Virtual Network Computing
WSGI	Web Server Gateway Interface

List of Tables

4.1.	Table with functional requirements	20
4.1.	Table with functional requirements	21
4.1.	Table with functional requirements	22
4.2.	Table with non-functional requirements	23
4.2.	Table with non-functional requirements	24
4.3.	Table with technical product environment and technologies	34
4.3.	Table with technical product environment and technologies	35

List of Figures

4.1. Berta Home	25
4.2. Berta login page	26
4.3. Berta user page	27
4.4. Berta architecture	28
4.5. Database tables	29
4.6. Berta voice user interface	30
4.7. Berta flowchart for voice command	32
4.8. Berta plugin class diagram	33
5.1. Berta simplified process	38
7.1. Berta evaluation result diagram	51
B.1. Berta evaluation result in numbers	xliv

Listings

5.1. class GenericPlugin	40
------------------------------------	----

Bibliography

- [1] *DSGVO für Dummies: Das musst du in 2021 wissen*, de-DE, Jan. 2020. [Online]. Available: <https://www.privacytutor.de/blog/dsgvo/> (visited on 06/12/2021).
- [2] *Neukonzeption des Bundesdatenschutzgesetzes*, de. [Online]. Available: http://www.bmi.bund.de/DE/themen/it-und-digitalpolitik/datenpolitik/bundesdatenschutzgesetz/bundesdatenschutzgesetz-artikel.html;jsessionid=718CD92B24A75AA9D7D681586E7D9DA1.1_cid295?nn=9393752 (visited on 06/06/2021).
- [3] *Was ist ein Sprachassistent?*, de. [Online]. Available: <https://www.it-business.de/was-ist-ein-sprachassistent-a-660785/> (visited on 03/04/2021).
- [4] *Smarte Lautsprecher im Test (2020): Das ist der beste für euren Einsatzzweck*, de. [Online]. Available: <https://www.netzwelt.de/vergleich/smart-lautsprecher-test-2020.html> (visited on 03/04/2021).
- [5] Google, *Google Assistant – dein persönlicher Helfer*, de-DE. [Online]. Available: https://assistant.google.com/intl/de_de/ (visited on 03/04/2021).
- [6] Apple, *Siri*, de-DE. [Online]. Available: <https://www.apple.com/de/siri/> (visited on 03/04/2021).
- [7] CarbonTrackAU, *How Voice Assisted Technology is Being Used*, en. [Online]. Available: <https://www.carbontrack.com.au/blog/voice-assisted-technology/> (visited on 03/04/2021).
- [8] Amazon, *Amazon Alexa*. [Online]. Available: <https://alexa.amazon.de/spa/index.html#cards> (visited on 03/04/2021).
- [9] *Jasper | Documentation*. [Online]. Available: <https://jasperproject.github.io/documentation/> (visited on 03/18/2021).

- [10] *Jasper | Documentation*. [Online]. Available: <https://jasperproject.github.io/documentation/hardware/> (visited on 03/18/2021).
- [11] *Jasper | Documentation*. [Online]. Available: <https://jasperproject.github.io/documentation/installation/> (visited on 03/18/2021).
- [12] *Jasper | Documentation*. [Online]. Available: <https://jasperproject.github.io/documentation/configuration/> (visited on 03/18/2021).
- [13] *Private By Design: Free and Private Voice Assistants | Make*: en-US, Mar. 2020. [Online]. Available: <https://makezine.com/2020/03/17/private-by-design-free-and-private-voice-assistants/> (visited on 06/04/2021).
- [14] The Raspberry Pi Foundation, *Teach, Learn, and Make with Raspberry Pi*, en-GB. [Online]. Available: <https://www.raspberrypi.org/> (visited on 04/26/2021).
- [15] ———, *Buy a Raspberry Pi 4 Model B*, en-GB. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/> (visited on 04/26/2021).
- [16] ———, *Raspberry Pi OS*, en-GB. [Online]. Available: <https://www.raspberrypi.org/software/> (visited on 04/26/2021).
- [17] *Wake Word - Rhasspy*. [Online]. Available: <https://rhasspy.readthedocs.io/en/latest/wake-word/> (visited on 04/28/2021).
- [18] *Kitt-AI/snowboy*, original-date: 2016-05-10T00:54:30Z, Jun. 2021. [Online]. Available: <https://github.com/Kitt-AI/snowboy> (visited on 06/10/2021).
- [19] *Introduction*, en. [Online]. Available: <https://picovoice.ai/docs/> (visited on 03/20/2021).
- [20] *Picovoice/leopard*, original-date: 2020-01-14T03:37:44Z, Mar. 2021. [Online]. Available: <https://github.com/Picovoice/leopard> (visited on 04/28/2021).
- [21] *Picovoice/cheetah*, original-date: 2018-10-28T05:34:24Z, Apr. 2021. [Online]. Available: <https://github.com/Picovoice/cheetah> (visited on 04/28/2021).

-
- [22] *Porcupine SDK Introduction*, en. [Online]. Available: <https://picovoice.ai/docs/porcupine/> (visited on 03/20/2021).
- [23] *Picovoice Console*. [Online]. Available: <https://console.picovoice.ai/ppn> (visited on 03/20/2021).
- [24] *Picovoice/porcupine*, original-date: 2018-03-08T01:55:25Z, Jun. 2021. [Online]. Available: <https://github.com/Picovoice/porcupine/tree/master/resources> (visited on 06/09/2021).
- [25] *Picovoice/porcupine*, original-date: 2018-03-08T01:55:25Z, Jun. 2021. [Online]. Available: <https://github.com/Picovoice/porcupine/blob/8ac4c7fd7e1e4846fee26573776e114ebd992424/LICENSE> (visited on 06/09/2021).
- [26] *Picovoice/porcupine*, original-date: 2018-03-08T01:55:25Z, Jun. 2021. [Online]. Available: https://github.com/Picovoice/porcupine/tree/master/resources/keyword_files/raspberry-pi (visited on 06/09/2021).
- [27] *Mozilla/DeepSpeech*, original-date: 2016-06-02T15:04:53Z, Apr. 2021. [Online]. Available: <https://github.com/mozilla/DeepSpeech> (visited on 04/15/2021).
- [28] *TensorFlow*, de-x-mtfrom-en. [Online]. Available: <https://www.tensorflow.org/?hl=de> (visited on 04/15/2021).
- [29] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng, “Deep Speech: Scaling up end-to-end speech recognition”, *arXiv:1412.5567 [cs]*, Dec. 2014, arXiv: 1412.5567. [Online]. Available: <http://arxiv.org/abs/1412.5567> (visited on 11/19/2020).
- [30] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu, “Deep Speech 2: End-to-End Speech Recognition in English and Mandarin”, *arXiv:1512.02595 [cs]*, Dec. 2015, arXiv: 1512.02595. [Online]. Available: <http://arxiv.org/abs/1512.02595> (visited on 04/15/2021).
- [31] *Welcome to DeepSpeech’s documentation! — DeepSpeech 0.9.3 documentation*, en, 2020. [Online]. Available: <https://deepspeech.readthedocs.io/en/v0.9.3/> (visited on 03/01/2021).

- [32] *Releases · mozilla/DeepSpeech*, en, 2020. [Online]. Available: <https://github.com/mozilla/DeepSpeech/releases> (visited on 02/26/2021).
- [33] Dirk Hovy and Shannon L. Spruit, “The Social Impact of Natural Language Processing”, in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 591–598. DOI: 10.18653/v1/P16-2096. [Online]. Available: <https://www.aclweb.org/anthology/P16-2096> (visited on 04/16/2021).
- [34] Hila Gonen and Yoav Goldberg, “Lipstick on a Pig: Debiasing Methods Cover up Systematic Gender Biases in Word Embeddings But do not Remove Them”, *arXiv:1903.03862 [cs]*, Sep. 2019, arXiv: 1903.03862. [Online]. Available: <http://arxiv.org/abs/1903.03862> (visited on 04/16/2021).
- [35] *Training Your Own Model — DeepSpeech 0.9.3 documentation*. [Online]. Available: <https://deepspeech.readthedocs.io/en/r0.9/TRAINING.html> (visited on 04/15/2021).
- [36] *Pico Text-to-Speech - Voices*, en-US. [Online]. Available: <https://www.openhab.org/addons/voice/picotts/> (visited on 04/28/2021).
- [37] *Pico/tts - platform/external/svox - Git at Google*. [Online]. Available: <https://android.googlesource.com/platform/external/svox/+/master/pico/tts/> (visited on 04/28/2021).
- [38] *Pico/tts/NOTICE - platform/external/svox - Git at Google*. [Online]. Available: <https://android.googlesource.com/platform/external/svox/+/master/pico/tts/NOTICE> (visited on 04/28/2021).
- [39] *Pico/tts/MODULE_license_apache2 - platform/external/svox - Git at Google*. [Online]. Available: https://android.googlesource.com/platform/external/svox/+/master/pico/tts/MODULE_LICENSE_APACHE2 (visited on 04/28/2021).
- [40] *License — Flask Documentation (1.1.x)*. [Online]. Available: <https://flask.palletsprojects.com/en/1.1.x/license/> (visited on 05/06/2021).
- [41] *Foreword — Flask Documentation (1.1.x)*. [Online]. Available: <https://flask.palletsprojects.com/en/1.1.x/foreword/#what-does-micro-mean> (visited on 05/06/2021).
- [42] *Welcome to Flask — Flask Documentation (1.1.x)*. [Online]. Available: <https://flask.palletsprojects.com/en/1.1.x/> (visited on 05/06/2021).

- [43] *Deployment Options — Flask Documentation (1.1.x)*. [Online]. Available: <https://flask.palletsprojects.com/en/1.1.x/deploying/> (visited on 05/06/2021).
- [44] *Django overview | Django*. [Online]. Available: <https://www.djangoproject.com/start/overview/> (visited on 05/06/2021).
- [45] *Gunicorn - Python WSGI HTTP Server for UNIX*. [Online]. Available: <https://gunicorn.org/> (visited on 06/07/2021).
- [46] *Gunicorn - Python WSGI HTTP Server for UNIX*. [Online]. Available: <https://gunicorn.org/#docs> (visited on 06/07/2021).
- [47] *Gunicorn - Python WSGI HTTP Server for UNIX*. [Online]. Available: <https://gunicorn.org/#deployment> (visited on 06/07/2021).
- [48] *Supervisor: A Process Control System — Supervisor 4.2.2 documentation*. [Online]. Available: <http://supervisord.org/> (visited on 06/07/2021).
- [49] [Online]. Available: <http://nginx.org/LICENSE> (visited on 05/05/2021).
- [50] *NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy, en-US*. [Online]. Available: <https://www.nginx.com/> (visited on 05/05/2021).
- [51] *NGINX Application Platform, en-US*. [Online]. Available: <https://www.nginx.com/products/> (visited on 05/05/2021).
- [52] *NGINX Docs | Installing NGINX Open Source, en*. [Online]. Available: <https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/> (visited on 05/05/2021).
- [53] *MySQL*. [Online]. Available: <https://www.mysql.com/de/> (visited on 06/12/2021).
- [54] *About SQLite*. [Online]. Available: <https://sqlite.org/about.html> (visited on 05/07/2021).
- [55] *SQLite3 — DB-API 2.0 interface for SQLite databases — Python 3.9.5 documentation*. [Online]. Available: <https://docs.python.org/3/library/sqlite3.html> (visited on 05/07/2021).
- [56] *SQLite Copyright*. [Online]. Available: <https://sqlite.org/copyright.html> (visited on 05/07/2021).
- [57] *SQLite Pro Support*. [Online]. Available: <https://sqlite.org/prosupport.html> (visited on 05/07/2021).

- [58] *SQLite Documentation*. [Online]. Available: <https://sqlite.org/docs.html> (visited on 05/07/2021).
- [59] *SQLite Download Page*. [Online]. Available: <https://sqlite.org/download.html> (visited on 05/07/2021).
- [60] *SQLite Home Page*. [Online]. Available: <https://sqlite.org/index.html> (visited on 05/07/2021).
- [61] *SQLAlchemy - The Database Toolkit for Python*. [Online]. Available: <https://www.sqlalchemy.org/> (visited on 06/09/2021).
- [62] Miguel Grinberg, *The Flask Mega-Tutorial Part IV: Database*. [Online]. Available: <http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database> (visited on 06/09/2021).
- [63] *Engine Configuration — SQLAlchemy 1.4 Documentation*. [Online]. Available: <https://docs.sqlalchemy.org/en/14/core/engines.html#database-urls> (visited on 06/09/2021).
- [64] Michael Heim, *Exploring Indiana Highways: Trip Trivia*. Exploring America's Highway, 2007.
- [65] *Glossary — Python 3.9.5 documentation*. [Online]. Available: <https://docs.python.org/3/glossary.html#term-duck-typing> (visited on 06/14/2021).
- [66] 14:00-17:00, *ISO/IEC 25010:2011*, en. [Online]. Available: <https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html> (visited on 06/23/2021).
- [67] —, *ISO/IEC 9126-1:2001*, en. [Online]. Available: <https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/27/22749.html> (visited on 06/23/2021).
- [68] *Utilities — Werkzeug Documentation (2.0.x)*. [Online]. Available: <https://werkzeug.palletsprojects.com/en/2.0.x/utils/> (visited on 06/03/2021).
- [69] *Werkzeug*. [Online]. Available: <https://palletsprojects.com/p/werkzeug/> (visited on 06/03/2021).
- [70] Miguel Grinberg, *The Flask Mega-Tutorial Part V: User Logins*. [Online]. Available: <http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins> (visited on 06/03/2021).

- [71] Damodar Periwal, “The kiss (keep it simple and straightforward) principles for or-mapping products from software tree, inc”, 2005.
- [72] Dmitry Maslov, *Offline Speech Recognition on Raspberry Pi 4 with Respeaker*, en, 2020. [Online]. Available: <https://www.hackster.io/dmitrywat/offline-speech-recognition-on-raspberry-pi-4-with-respeaker-c537e7%7D> (visited on 02/26/2021).
- [73] *A Simple Plugin Framework*. [Online]. Available: <http://martyalchin.com/2008/jan/10/simple-plugin-framework/> (visited on 06/22/2021).
- [74] Miguel Grinberg, *The Flask Mega-Tutorial Part I: Hello, World!* [Online]. Available: <http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world> (visited on 05/16/2021).
- [75] *Miguelgrinberg/microblog*, en. [Online]. Available: <https://github.com/miguelgrinberg/microblog> (visited on 05/16/2021).
- [76] ———, *Miguelgrinberg/microblog*, original-date: 2012-12-17T06:01:19Z, May 2021. [Online]. Available: <https://github.com/miguelgrinberg/microblog> (visited on 05/16/2021).
- [77] *16.8. logging.handlers — Logging handlers — Python 3.6.13 documentation*. [Online]. Available: <https://docs.python.org/3.6/library/logging.handlers.html#rotatingfilehandler> (visited on 06/09/2021).
- [78] *Jinja — Jinja Documentation (3.0.x)*. [Online]. Available: <https://jinja.palletsprojects.com/en/3.0.x/> (visited on 06/09/2021).
- [79] *Flask-WTF — Flask-WTF Documentation (0.15.x)*. [Online]. Available: <https://flask-wtf.readthedocs.io/en/0.15.x/> (visited on 06/09/2021).
- [80] *Creating Forms — Flask-WTF Documentation (0.15.x)*. [Online]. Available: <https://flask-wtf.readthedocs.io/en/0.15.x/form/#secure-form> (visited on 06/09/2021).
- [81] ———, *The Flask Mega-Tutorial Part III: Web Forms*. [Online]. Available: <http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iii-web-forms> (visited on 06/09/2021).
- [82] *Flask-Migrate — Flask-Migrate documentation*. [Online]. Available: <https://flask-migrate.readthedocs.io/en/latest/> (visited on 06/09/2021).
- [83] *Flask-Login — Flask-Login 0.4.1 documentation*. [Online]. Available: <https://flask-login.readthedocs.io/en/latest/> (visited on 06/09/2021).

- [84] ———, *Miguelgrinberg/Flask-Moment*, original-date: 2013-10-24T04:17:57Z, Jun. 2021. [Online]. Available: <https://github.com/miguelgrinberg/Flask-Moment> (visited on 06/09/2021).
- [85] *Flask-Bootstrap — Flask-Bootstrap 3.3.7.1 documentation*. [Online]. Available: <https://pythonhosted.org/Flask-Bootstrap/> (visited on 06/09/2021).
- [86] *Bootstrap introduction*, en. [Online]. Available: <https://getbootstrap.com/docs/5.0/getting-started/introduction/> (visited on 06/09/2021).
- [87] ———, *The Flask Mega-Tutorial Part XVII: Deployment on Linux*. [Online]. Available: <http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xvii-deployment-on-linux> (visited on 06/09/2021).
- [88] *Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.x)*. [Online]. Available: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/> (visited on 06/09/2021).
- [89] *Configuration — Flask-SQLAlchemy Documentation (2.x)*. [Online]. Available: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/> (visited on 06/14/2021).
- [90] *Wireshark · Go Deep*. [Online]. Available: <https://www.wireshark.org/> (visited on 06/23/2021).
- [91] Stella Naser, *PassionPlus/berta*, original-date: 2021-04-11T11:45:43Z, Jun. 2021. [Online]. Available: <https://github.com/PassionPlus/berta> (visited on 06/22/2021).

Appendix A

Usability test by voice communication

A.1. Entry scenario

On a table are the Raspberry Pi4 with the Berta voice assistant as well as an OnePlus 3T cell phone with Google assistant. The exact same voice messages are played to these voice assistants via an iPhone12. The question will be played once. The question stored as text is written down with its answer. The following lists the scenario sequence, wake words for the voice assistants and the questions and requests that will be asked to the voice assistants.

Scenario sequence:

- Play the wake word if one is required
- Wait for Berta LED Signal wake up signal
- Play the question during LEDs brighten up during recognition:
- Write down whether a response was received.
- Start again with the process.

Wake words and their devices:

Berta voice assistant:

- Wake word: "computer"
- Device: Raspberry Pi4

Google assistant:

- Wake word: No wake word
- Device: OnePlus 3T cell phone

Questions that will be asked to the voice assistants:

1. Request: This is a test.
2. Request: Do something simple!
3. Question: What time is it?
4. Question: What is today's date?
5. Request: Tell me a joke!

A.2. Results:

A.2.1. Male

Test subject one:

Gender: male Mother tongue: German Age group: 20-30 Evaluated Application: Berta voice assistant

Test device: Raspberry Pi4

Question received on speaker(yes/no): Right "keyword" recognition received from Berta:

1. yes
2. yes
3. yes
4. no
5. no

Results from Berta:

:

To question 1:

- How many times speaking wake word till recognition phase: 1
- question: at as a test
- answer: Default sentence

To question 2:

- How many times speaking wake word till recognition phase: 2
- question: two something simple
- answer: Hello World

To question 3:

- How many times speaking wake word till recognition phase: 1
- question: What time is it

- answer: time announcement

To question 4:

- How many times speaking wake word till recognition phase: 1
- question: What as the day dates
- answer: Default sentence

To question 5:

- How many times speaking wake word till recognition phase: 1
- question: tel me at shoke
- answer: Default sentence

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: I can assure you, this thing is on

To request 2:

- question: do something simple
- answer: good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

A. Usability test by voice communication

Test subject two:

Gender: male Mother tongue: German Age group: 20-30 Evaluated Application:

Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. yes
2. yes
3. yes
4. no
5. yes

Results from Berta:

:

To question 1:

- question: thes as a test
- answer: Default sentence

To question 2:

- question:to was something simple
- answer: Hello World

To question 3:

- question: ot time is it
- answer: Time announcement

To question 4:

- question: what is os that tay
- answer: Default sentence

To request 5:

- question: cel me a joke
- answer: tells joke

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. no
3. yes

4. yes

5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: Is this thing on?

To request 2:

- question: to something simple
- answer: tells a joke

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject three:

Gender: male Mother tongue: German Age group: 20-30 Evaluated Application:
Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. yes

2. no

3. no

4. no

5. no

Results from Berta:

:

To question 1:

- question: sis a test

A. Usability test by voice communication

- answer: default announcement

To question 2:

- question: rer something some por
- answer: default announcement

To question 3:

- question: timeasons
- answer: default announcement

To question 4:

- question: or tetas tat
- answer: default announcement

To question 4:

- question: and he adult
- answer: default announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. no
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: No one told me there would be a test

To request 2:

- question: something something
- answer: Who knows?

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date

- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject four:

Gender: male Mother tongue: Czech,German Age group: 20-30 Evaluated Application: Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. no
2. no
3. no
4. no
5. no

Results from Berta:

:

To question 1:

- question: and he adult
- answer: default announcement

To question 2:

- question: os unting impe
- answer: default announcement

To question 3:

- question: at imen i what
- answer: default announcement

To question 4:

- question: at his pat the
- answer: default announcement

To question 5:

- question: an meantol
- answer: default announcement

A. Usability test by voice communication

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: Is this thing on?

To request 2:

- question: do something simple
- answer: Good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject five:

Gender: male (cody) Mother tongue: English(US), German Age group: 20-30 Note:

Sat in front of the system Evaluated Application: Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. yes
2. yes

3. yes

4. yes

5. yes

Results from Berta:

To question 1:

- question: this is a test
- answer: default message

To question 2:

- question: do something simple
- answer: Hello World

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To question 5:

- question: tell me a joke
- answer: joke announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes

2. yes

3. yes

4. yes

5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: Is this thing on?

To request 2:

- question: do something simple

A. Usability test by voice communication

- answer: Good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

A.2.2. Female

Test subject six:

Gender: female Mother tongue: German Age group: 20-30 Note: Sat in front of the system Evaluated Application: Berta voice assistant
Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. yes
2. yes
3. yes
4. yes
5. yes

Results from Berta:

To question 1:

- question: this is a test
- answer: default message

To question 2:

- question: do some think simple
- answer: Hello World

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To question 5:

- question: tell me a joke
- answer: joke announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: debug OK, 209489812638, That was weird

To request 2:

- question: do something simple
- answer: Good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

A. Usability test by voice communication

Test subject seven:

Gender: female Mother tongue: German Age group: 20-30 Evaluated Application:

Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. yes
2. no
3. yes
4. no
5. no

Results from Berta:

To question 1:

- question: tes as ha test
- answer: default announcement

To question 2:

- question: os unting impe
- answer: default announcement

To question 3:

- question: w time
- answer: time announcement

To question 4:

- question: what as to day's tat
- answer: default announcement

To question 5:

- question: an meantol
- answer: default announcement

default announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes

3. yes

4. yes

5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: I can assure you, this thing's on

To request 2:

- question: do something simple
- answer: Good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: what is today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject eight:

Gender: female Mother tongue: Russian, German Age group: 20-30 Evaluated Application: Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. no

2. yes

3. yes

4. no

5. no

Results from Berta:

To question 1:

A. Usability test by voice communication

- question: is is cateste
- answer: default message

To question 2:

- question: wo something simple
- answer: Hello World

To question 3:

- question: what time is thit
- answer: time announcement

To question 4:

- question: s to days sdate
- answer: default message

To question 5:

- question: ol me an tom
- answer: default announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: this is a test
- answer: I can assure you, this thing's on

To request 2:

- question: something simple
- answer: Good

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject nine:

Gender: female Mother tongue: German Age group: 20-30 Evaluated Application:
Berta voice assistant
Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. no
2. yes
3. yes
4. no
5. no

Results from Berta:

To question 1:

- question: e tect
- answer: default announcement

To question 2:

- question: oso thin simple
- answer: Hello world

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: totas tate
- answer: default announcement

To question 5:

- question: om me ad ton't
- answer: default announcement

A. Usability test by voice communication

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes
2. yes
3. yes
4. yes
5. yes

Saved results from Google assistant:

To request 1:

- question: test
- answer: No one told me there would be a test

To request 2:

- question: play something simple
- answer: Ok, which app do you want to use?

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Test subject ten:

Gender: female Mother tongue: German Age group: 20-30 Evaluated Application:

Berta voice assistant

Test device: Raspberry Pi4

Right "keyword" recognition received from Berta:

1. no
2. yes

3. no

4. no

5. no

Results from Berta:

To question 1:

- question: e tect
- answer: default announcement

To question 2:

- question: oso thin simple
- answer: default announcement

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: h
- answer: default announcement

To question 5:

- question: totas tate
- answer: default announcement

Evaluated Application: Google assistant

Test device: OnePlus 3T

Right "keyword" recognition received from Google (yes/no):

1. yes

2. no

3. yes

4. yes

5. yes

Saved results from Google assistant:

To request 1:

- question: the test
- answer: Film presenting

To request 2:

- question: something something

A. Usability test by voice communication

- answer: Who knows?

To question 3:

- question: what time is it
- answer: time announcement

To question 4:

- question: today's date
- answer: date announcement

To request 5:

- question: tell me a joke
- answer: tells joke

Appendix B

Evaluation diagram in numbers

The following figure B.1 shows the figure 7.1 of represented in numbers. With this table the diagram for figure 7.1 was created. This has been included to clarify the diagram further in case of possible misunderstandings when understanding the evaluation.

B. Evaluation diagram in numbers

	"test" keyword out of 5 possible results	"simple" keyword out of 5 possible results	"time" keyword out of 5 possible results	"date" keyword out of 5 possible results	"joke" keyword out of 25 possible results	Recognized results out of 25 possible results	Voice messages only, 8 out of 10 test subjects	Desk test only, 2 out of 10 test subjects
Berta "keyword" male	4	3	3	1	2	13	8	5
Berta "keyword" female	2	4	4	1	1	12	7	5
Berta sentence structure with "keyword" correct, male	1	1	2	1	1	6	1	5
Berta sentence structure with "keyword" correct, female	1	0	3	0	1	5	2	3
Google "keyword" male	5	3	5	5	5	23	18	5
Google "keyword" female	5	4	5	5	5	24	19	5
Google sentence structure with "keyword" correct, male	4	3	5	5	5	23	18	5
Google sentence structure with "keyword" correct, female	3	2	5	2	5	17	12	5

Figure B.1.: Berta evaluation result in numbers