**Vocal Assistant**

Diana Cernăzanu-Glăvan

Mario-Mihai Mateaș

1. **Project Idea**

The project resembles a Vocal Assistant, similar, but not as optimal and practical as Amazon Alexa[[1]](#footnote-1) or Google Home[[2]](#footnote-2). The assistant was named “Mariana”, as a fusion between the first names of the students who worked on developing it (Mario, Diana).

Until this moment, Mariana is able to listen and provide answers to some basic commands such as “What is your name?”, “Tell me a joke / pun” or “What’s the room temperature?”.

If Mariana doesn’t know the answer to a question, she will respond accordingly, asking the user to repeat the question.

1. **Main features of the project**

Mariana offers the following features:

* voice recognition
* execution of certain event-based actions (jokes, temperature)
* room temperature monitoring
* gathering all of the data resulted from user interaction in a relational database

1. **A detailed explanation of the system’s functionality**

The vocal assistant allows user input commands through their voice.

Whenever the user wants to say something, they must manually run a Python script. The script will then start listening the board’s audio input (USB-connected microphone). The user will further proceed by saying a certain command and will press the CTRL + C combination, telling the script to stop listening. The vocals will then be further processed through various cloud services and the vocal assistant will output a proper answer to the user’s question.

The questions and answers are inserted in a database, whose entries are output in both a cloud-hosted web application and a mobile app, for the users to keep track of Mariana’s answers.

Mariana is able to tell jokes, based on an external API that returns a different joke (hopefully) on every call.

The vocal assistant is also able to output the room temperature, but only if it is properly connected to a DHT-11 humidity sensor.

**System Architecture**

1. External architecture of the system

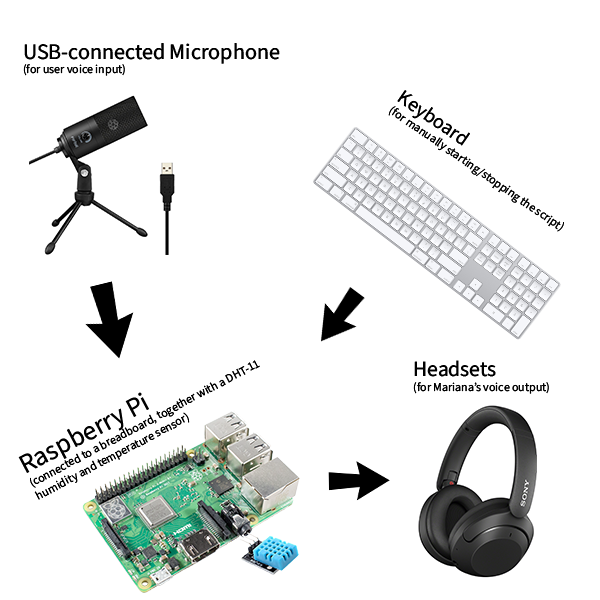


Figure 1: Vocal assistant overview with external components used: microphone, keyboard, Raspberry Pi, DHT-11 sensor and headsets

1. Software architecture

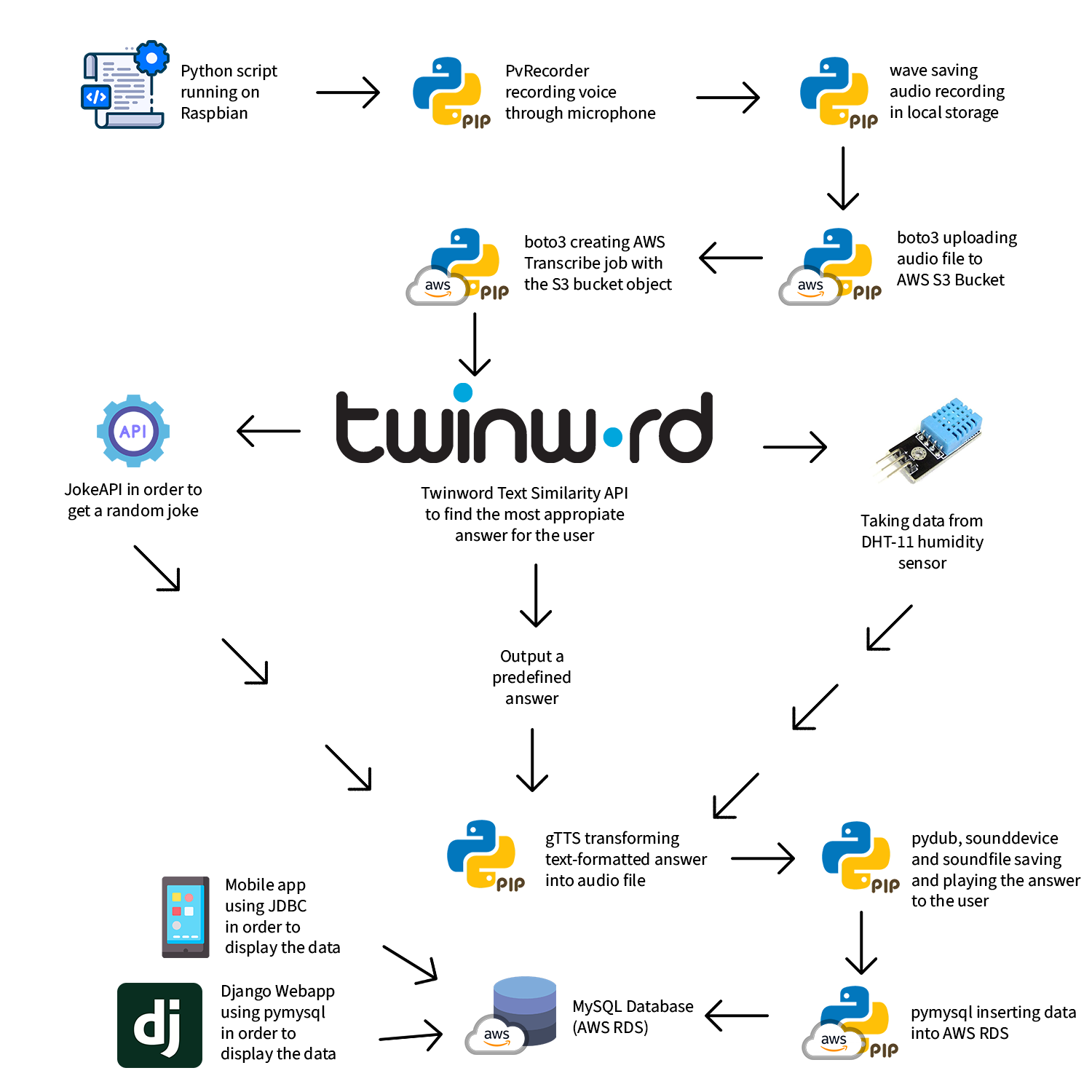
****

Figure 2: The Software & Cloud architecture of the system, describing the Python modules used, together with the external APis and Cloud Services called during this process

1. Implementation of the system

3.1.) Connecting the Raspberry Pi to a DHT-11 sensor

* a 10K Ohm resistor will be connected between the Vcc and signal lines
* one of the DHT-11 pins will be connected to the Vcc
* the second pin will be connected to one of the Raspberry PI’s data pins (GPIO)
* the third pin will be connected to the ground (GND)
* one pin will be left disconnected

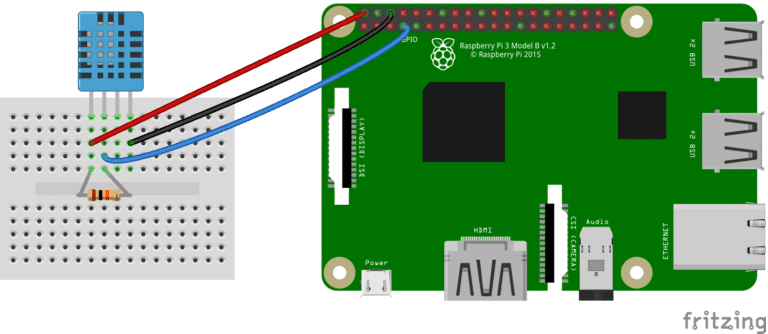


Figure 3: Connecting DHT-11 sensor to Raspberry Pi[[3]](#footnote-3)

3.3.) Software implementation of the script, web and mobile application

As the Raspberry Pi runs on a Raspberry Pi OS (a Linux distribution), there were initially some issues in testing the Python modules and other dependencies first on a local Windows machine, and then on the boards from the university.

We needed to change some of the dependencies, in order to make the system run properly on the board’s default operating system, and some of the modules needed to be manually compiled, for specific processor. The manual compilation part was not required on the Raspberry Pi boards.

On the software level, the main logic of our vocal assistant lays inside the *vocal-assistant*[[4]](#footnote-4) GitHub repository.

This repository provides a Python script (script.py), that is run manually by the user inside the Raspberry Pi OS environment. The script will not run properly, until all the dependencies (specified in the commands file) are installed. Some of those dependencies may not be required, because they depend on a specific processor architecture, and Raspberry Pi’s architecture did not seem to be a problem until this point.

After the script starts, it waits for the user input, recorded through an external Python module called PvRecorder[[5]](#footnote-5). The module records until the KeyboardInterrupt (CTRL + C) combination is pressed.



Figure 4: Beginning logic of script.py - starting the recorder and waiting for user voice input until KeyboardInterrupt is triggered

The output audio file, containing the user’s voice is then locally saved inside a file called audio.wav, which sent further for processing.

Now, the whole process is similar to a “black-box”. Everything is passed to third-party modules/APIs, while our script only processes the data and outputs information relevant to the user.

We want our system to understand the user’s command, and it would be very hard for it to directly analyze an audio file. We made use of AWS’s Transcribe[[6]](#footnote-6) service, that transforms audio files into text. Transcribe is called through the Boto3 SDK for Python[[7]](#footnote-7).

As Amazon Transcribe works only with Amazon S3[[8]](#footnote-8) bucket objects, we created a bucket, *dianamariobucketnew* (was initially *dianamariobucket*, but forgot to add policies and needed to create a new one), where the audio.wav (which was earlier locally saved) would be uploaded.

After the audio file has been uploaded to the S3 bucket, the script will automatically create a Transcribe job, which will process the object **always** called audio.wav (this means, each time the script is called, that file will be overwritten – the bucket **won’t** be filled with multiple files).

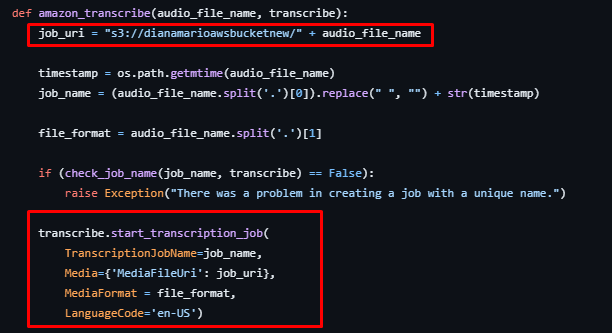


Figure 5: The script is launching a new Amazon Transcribe job on the already-uploaded audio file, available on the Amazon S3 storage

Now, the question has been transformed from audio to text. We are getting the returned text and we are trying to find out if the answer is similar to some of our predefined questions.

For this, we have used Twinword’s Text Similarity API[[9]](#footnote-9), which works together with AWS to provide similarity percentages between two texts. It provides a free version that allows a quota of thousands of calls per month (which was more than enough for testing).

The get\_answer() method takes the result returned by the Transcribe job and tries to compare it with the following predefined questions:

* What’s the room temperature?
* What’s your name?
* Tell me a joke.
* Tell me a pun.
* Do a backflip.
* How many degrees are in the room?

Obviously, not asking the questions in the same manner may still output the right answer, thanks to the performance of the API.

After doing 6 API calls for each question, we check if there is at least one similarity of 60%. If not, Mariana will say that she doesn’t understand the question, and will ask the user for clarification.

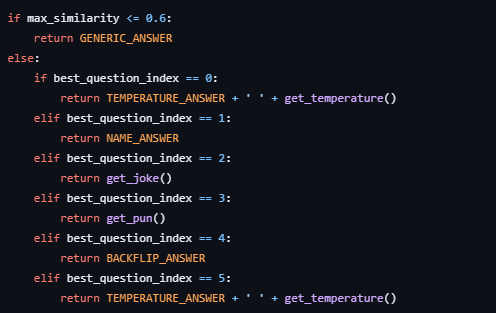


Figure 6: The script will return the index of the pre-defined question with the highest similarity percentage. It will return a specific answer for each of these indexes.

If there is a percent of at least 60%, the greatest value is chosen, and a proper answer is returned by the bot. Some of the answers are predefined and hard-coded (NAME\_ANSWER will always return “My name is Mariana”), but some of them are only based on an API output (such as get\_joke() and get\_pun() calls).

The joke and pun questions will make use of another public API, called JokeApi[[10]](#footnote-10), which is going to return a joke in the JSON format.



Figure 7: The script's procedure that calls the external Joke API and returns the response in a text format

If the user asks for the room temperature, the system may try to use the DHT-11 humidity sensor in order to gather an answer. The algorithm used for this is based on Adafruit’s dht\_simpletest.py[[11]](#footnote-11) script.

No matter the question, the answers are **never** returned in a text format only. The answer text will be transformed in another audio file, which is going to be saved locally and played for the user. We could have used another AWS feature, such as Amazon Polly[[12]](#footnote-12), but Transcribe already made the vocal assistant slow enough, so we decided to use an existent Python module for it: gTTS[[13]](#footnote-13).

The last thing done by our script is saving the questions and the answers in an RDS[[14]](#footnote-14) MySQL database, which is going to be called both by a web application (hosted on an EC2-instance) and a locally-installed mobile app.

The web application, written using the Django framework, can be found at the following GitHub repository: <https://github.com/mateasmario/vocal-assistant-webapp>.

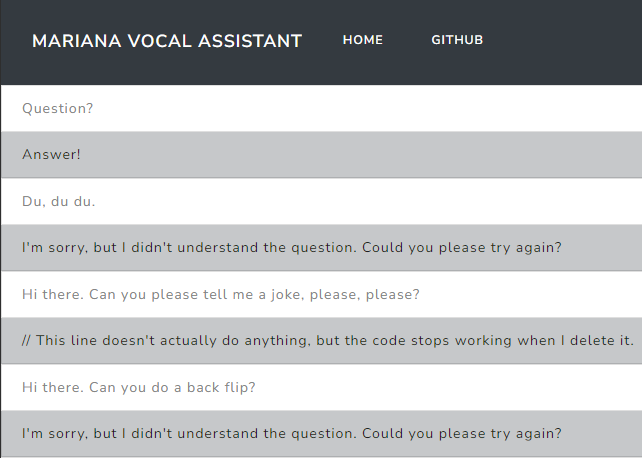


Figure 8: The UI of the Django web application, displaying information about all questions and their corresponding answers, stored in the MySQL database

The mobile application, written in Kotlin, with the help of Android Studio, can be found here: <https://github.com/mateasmario/vocal-assistant-mobileapp>.

The Database INSERT (for the Python script running on the board) and SELECT (for the webapp) statements are all sent through the PyMySQL[[15]](#footnote-15) Python module. Mobile’s SELECT statements are sent using JDBC[[16]](#footnote-16).

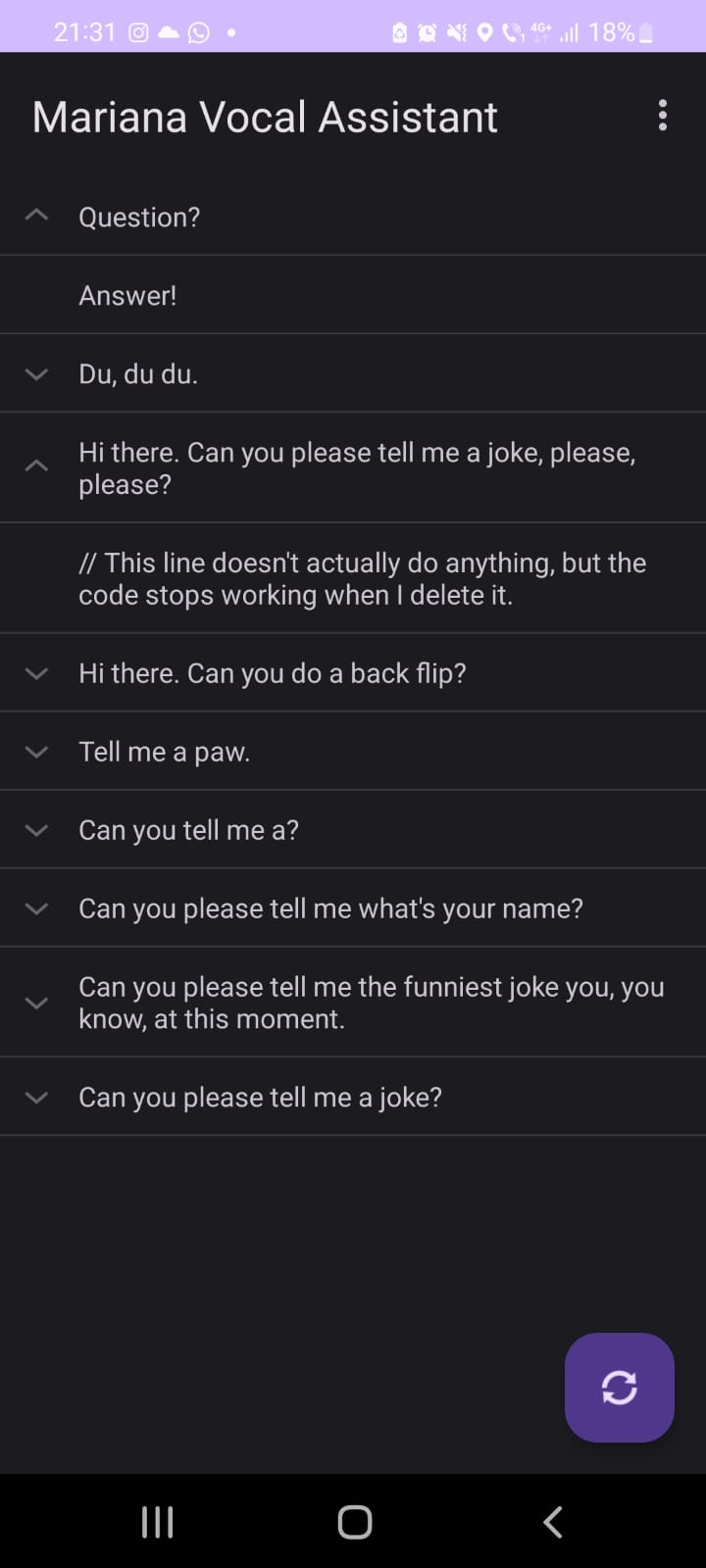
****

Figure 9: Kotlin-based mobile application, that also displays questions & answers from Mariana's MySQL database using an ExtendableListView

**Future Work**

At this moment, users need to manually start the main script by running a Python command. This can be later improved by letting the user press a button in order for the assistant to start listening, or by having a script that is constantly listening until it receives a certain keyword from the user – then it processes the command.

The vocal assistant is **not directly learning** during the processing of the commands. Mariana uses several external APIs which might get improved during the time. This means, the vocal assistant might return the same answer, every time you ask her a question (such as “What’s your name?”). This be also later improved.

Mariana only answers to a limited number of commands. Further improvements can be done in the future.

**Conclusions**

The GitHub repositories for our applications:

* The base repository, containing Mariana’s main files (<https://github.com/mateasmario/vocal-assistant>)
* Web application built on Django, that displays all the interactions with Mariana (<https://github.com/mateasmario/vocal-assistant-webapp>)
* Mobile application built using Kotlin and Android Studio, that does the same thing as the web application (<https://github.com/mateasmario/vocal-assistant-mobileapp>)
* Adafruit DHT-11 Python example (<https://github.com/adafruit/Adafruit_CircuitPython_DHT/blob/main/examples/dht_simpletest.py>)

Documentation and tutorials used in order to make the project fully functional:

* Boto3 Official Documentation – Amazon Web Services (<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>)
* Speech to text using AWS Transcribe – Intelipaat (<https://intellipaat.com/blog/speech-to-text-using-aws-transcribe/?US>)
* How to connect AWS RDS with Python using PyMySQL – KGP Talkie, YouTube (<https://www.youtube.com/watch?v=RerDL93sBdY>)
* Deploy Django Web App on AWS EC2 - Code with Clinton, YouTube (<https://www.youtube.com/watch?v=hMLyhl8hIag>)
* How to Connect to MySQL Database from Kotlin using JDBC? (<https://www.tutorialkart.com/kotlin/connect-to-mysql-database-from-kotlin-using-jdbc/>)

Cloud services and APIs used for our project:

* AWS S3
* AWS Transcribe
* AWS EC2
* Amazon RDS
* Twinword’s Text Similarity API (through AWS)
* JokeAPI

Python Modules used for our project: boto3, wave, pvrecorder, gtts, sounddevice, soundfile, pydub, requests, adafruit\_dht, pymysql

1. “Amazon’s cloud-based voice service available on more than 100 million devices from Amazon and third-party device manufacturers” – Amazon.com (<https://developer.amazon.com/en-GB/alexa>) [↑](#footnote-ref-1)
2. “Google Home unites devices and services from Google and your favorite brands to build a personalized smart home to help with your tasks” – Google.com (<https://home.google.com/welcome/>). By comparing our system with Google Home, we only refer to their capability to receive voice commands. [↑](#footnote-ref-2)
3. Scott Campbell, “How to Set Up the DHT11 Humidity Sensor on the Raspberry Pi”, CircuitBasics.com, 2016, URL: <https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/> [↑](#footnote-ref-3)
4. “Vocal Assistant”, mateasmario (GitHub), URL: <https://github.com/mateasmario/vocal-assistant/> [↑](#footnote-ref-4)
5. “pvrecorder”, Picovoice, URLs: <https://pypi.org/project/pvrecorder/>, <https://github.com/Picovoice/pvrecorder> [↑](#footnote-ref-5)
6. “Automatically convert speech to text” – Amazon.com, URL: <https://aws.amazon.com/transcribe/> [↑](#footnote-ref-6)
7. “The SDK provides an object-oriented API as well as low-level access to AWS services.”, Amazon.com, URLs: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>, <https://pypi.org/project/boto3/> [↑](#footnote-ref-7)
8. Cloud object storage, Amazon.com, URL: <https://aws.amazon.com/s3/> [↑](#footnote-ref-8)
9. “Evaluate the similarity of two words, sentences, or paragraphs.”, Twinword, URL: <https://www.twinword.com/api/text-similarity.php> [↑](#footnote-ref-9)
10. “A REST API that serves uniformly and well formatted jokes and offers a great variety of filtering methods and response customization”, Sv443, URLs: <https://v2.jokeapi.dev/>, <https://github.com/Sv443/JokeAPI> [↑](#footnote-ref-10)
11. “CircuitPython support for the DHT11 and DHT22 temperature and humidity devices”, Adafruit, URL: <https://github.com/adafruit/Adafruit_CircuitPython_DHT/blob/main/examples/dht_simpletest.py> [↑](#footnote-ref-11)
12. “Amazon Polly uses deep learning technologies to synthesize natural-sounding human speech, so you can convert articles to speech.”, Amazon.com, URL: <https://aws.amazon.com/polly/> [↑](#footnote-ref-12)
13. “A Python library and CLI tool to interface with Google Translate text-to-speech API”, Pierre Nicolas Durette, URL: <https://pypi.org/project/gTTS/> [↑](#footnote-ref-13)
14. “Set up, operate, and scale a relational database in the cloud with just a few clicks.”, Amazon.com, URL: <https://aws.amazon.com/rds/> [↑](#footnote-ref-14)
15. A “pure-Python MySQL client library, based on PEP 249”, PyMySQL, URL: <https://pypi.org/project/pymysql/> [↑](#footnote-ref-15)
16. Java Database Connectivity – “an application programming interface (API) for the Java programming language, which defines how a client may access a database.” (Wikipedia), Connectivity tutorial URL: <https://www.tutorialkart.com/kotlin/connect-to-mysql-database-from-kotlin-using-jdbc/> [↑](#footnote-ref-16)