**VOCAL ASSISTANT**

Diana Cernazanu-Glavan

Mario-Mihai Mateas

**Project Idea**

The project resembles a Vocal Assistant, similar, but not as optimal and practical as Amazon Alexa or Google Home. 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.

**Main features of the project**

The vocal assistant allows user input commands through their voice, and whenever the user wants to say something, they must manually run a Python script. This can be later improved to pressing a button / having a script that is constantly listening until the assistant receives a certain keyword.

After saying the command, the user is going to press the CTRL + C combination, to stop the script. But before the script will stop its execution, the vocals will be further processed through various cloud services and the vocal assistant will output a proper answer.

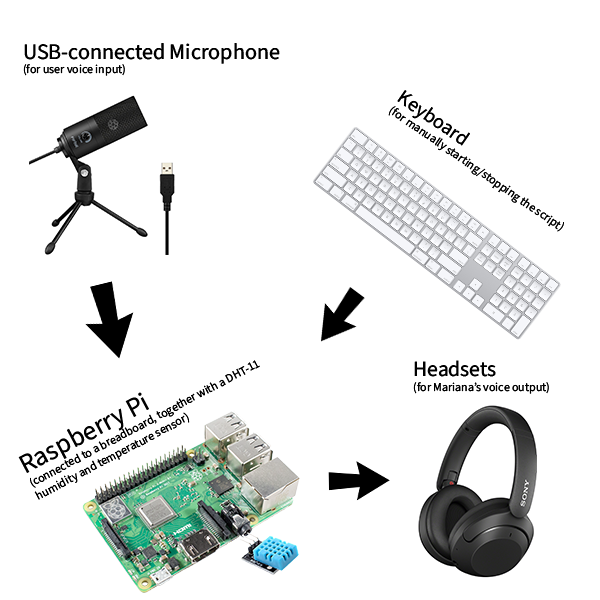
The questions and answers are inserted in a database, whose entries are output in a cloud-hosted web application, for the users to keep track of Mariana’s answers.

The vocal assistant is **not directly learning** during this process. It makes use of 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?”). “Tell me a joke” will probably always return a different answer, since the Joke API returns a random joke on each 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 (physical components used):**



**2.) Raspberry Pi’s connection to the 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

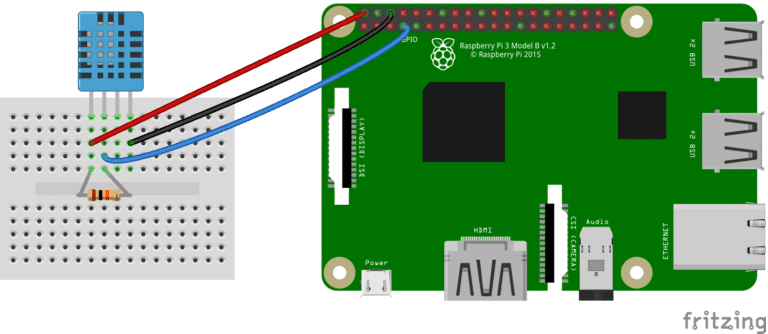


Image source: <https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/>

**3.) What happens at the software level?**

As the Raspberry Pi runs on a Raspbian (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 Pi’s OS, and some of the modules needed to be manually compiled, for specific processor architectures (we also simulated a version of Raspbian that ran on a Virtual Machine, and some of those dependencies didn’t work properly and needed to be manually compiled for a different architecture). The manual compilation part was not required on the Pi boards.

On the software level, the main logic of our vocal assistant lays inside the vocal-assistant (<https://github.com/mateasmario/vocal-assistant/>) GitHub repository.

This repository provides a Python script (script.py), that is run manually by the user inside the Raspbian environment. The script won’t run properly, until all the dependencies (specified in the commands file) are installed. Some of those dependencies won’t be required, because they depend on a specific processor architecture, and Raspberry Pi’s architecture didn’t seem to be a problem until this point. The commands file was built while testing everything on a Windows Virtual Machine (with a Raspbian image). Probably there were some problems with the laptop’s processor architecture, combined with the Raspbian.

After the script starts, it waits for user input, gained through an external Python module called PvRecorder (<https://pypi.org/project/pvrecorder/>). The module records until the KeyboardInterrupt (CTRL + C) combination is pressed.



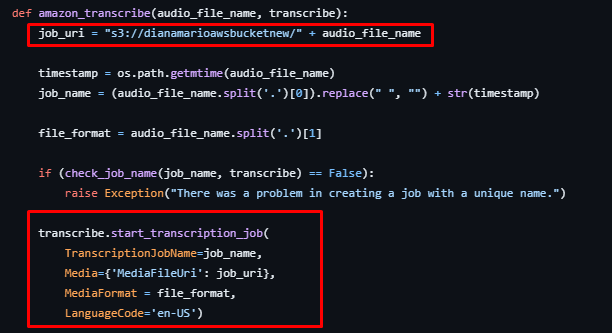
The output audio file, containing the user’s voice is then locally saved inside a file called audio.wav, which sent further for processing.

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 (<https://aws.amazon.com/transcribe/>) service, that transforms audio files into text. Transcribe is called through the Boto3 (<https://pypi.org/project/boto3/>) SDK for Python.

As AWS Transcribe works only with Amazon S3 (<https://aws.amazon.com/s3/>) 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).



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.

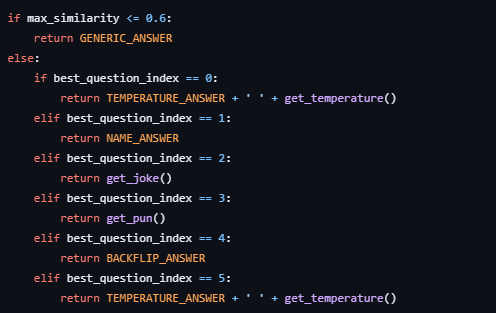
For this, we have used Twinword’s Text Similarity API (<https://www.twinword.com/api/text-similarity.php>), which works together with AWS to provide similarity percentages between two texts. It provides a free version that allows a quota of thousands of calls per month (which was more than enough for 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 ask the user for clarification.



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 (<https://v2.jokeapi.dev/>), which is going to return a joke in the JSON 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 (<https://github.com/adafruit/Adafruit_CircuitPython_DHT/blob/main/examples/dht_simpletest.py>) script.

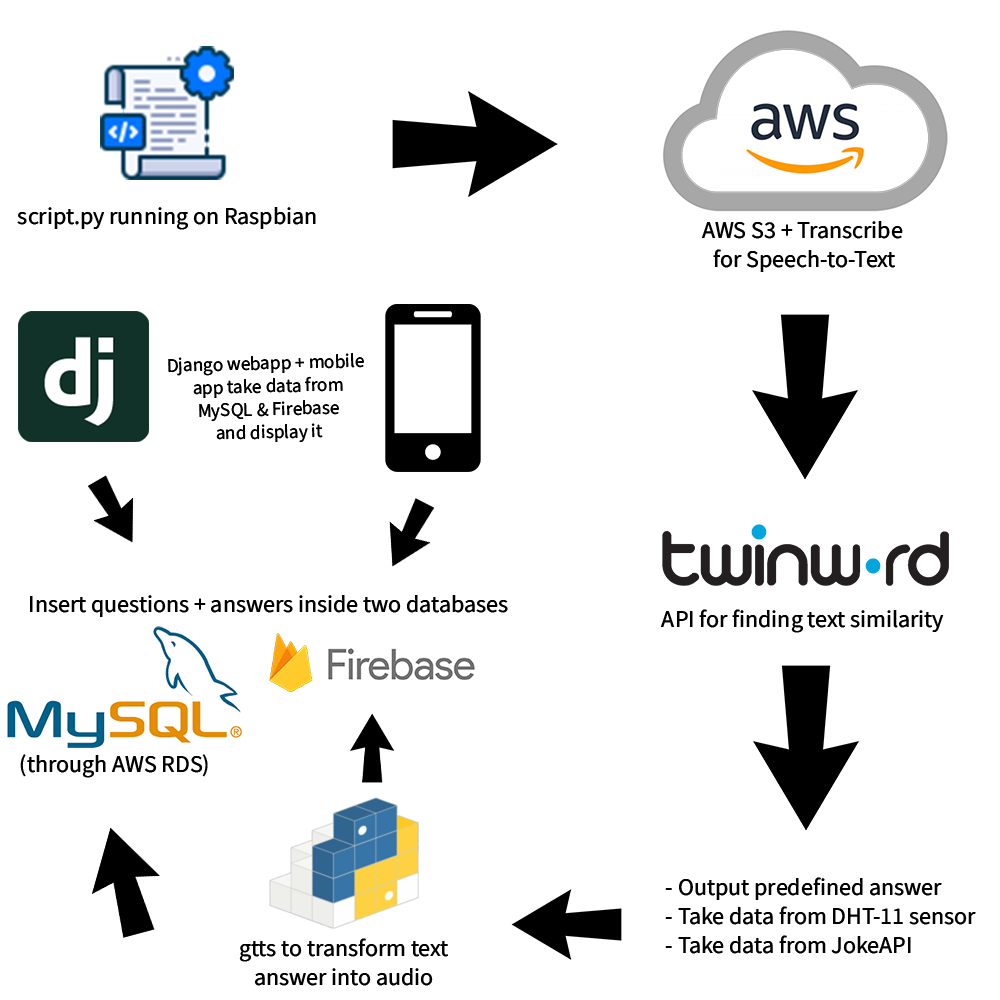
No matter the question, the answers are **never** returned in a text format only. The answer text will be transformed in another audio file, which is going to be saved locally and played for the user. We could have used another AWS feature, such as Amazon Polly (<https://aws.amazon.com/polly/>), but Transcribe already made the vocal assistant slow enough, so we decided to use an existent Python module for it: gTTS (<https://pypi.org/project/gTTS/>).

The last thing done by our script is saving the questions and the answers in an RDS (<https://aws.amazon.com/rds/>) MySQL database, which is going to be called by a web application hosted on an EC2-instance.

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

The Database INSERT (for Mariana) and SELECT (for the webapp) statements are all sent through the pymysql (<https://pypi.org/project/pymysql/>) Python module.

The question & answer groups are also sent to a Firebase Dynamic Database, in order for an Android Phone to be able to track them through a specialized application (made in Kotlin).

**4.) Software & Cloud Architecture Diagram**

**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>)
* 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>)

Cloud services and APIs used for our project:

* AWS S3
* AWS Transcribe
* AWS EC2
* Twinword’s Text Similarity API (through AWS)
* JokeAPI
* Firebase Realtime Database