INSEA

INTERNET OF THINGS

PROJECT REPORT

HOMEEX:
AN IOT-BASED INTELLIGENT HOME ASSISTANT

Realized by:

ELAZZAOUI Mohamed
ZENIBI Abdelhakim
EL GAOUT EL Mehdi
AFRACHE Yassine
BOUKOUTAYA Oussama

Supervised by:

Mme. EL ABOUDI

Academic Year : 2022-2023

Contents

List of Figures

1 Introduction

The world we live in is evolving at an unprecedented pace, and so is the technology that shapes it. The Internet of Things (IoT) is one of the most revolutionary technologies of our time, transforming the way we interact with our surroundings. In the realm of IoT, one of the most fascinating and rapidly growing fields is that of voice assistants. These virtual assistants have become ubiquitous in our homes, offices, and personal devices, making our lives easier and more convenient than ever before.

This project focuses on the development of a voice assistant using Rhasspy[11], an open-source toolkit for building voice interfaces for the IoT. Rhasspy allows us to create a highly customizable and privacy-focused voice assistant that can run on low-power devices such as the Raspberry Pi. This makes it an ideal solution for home automation, where a simple voice command can control various devices in the house.

With this project, we aim to explore the potential of Rhasspy and the capabilities of voice assistants in the IoT ecosystem. We will develop a voice assistant that can understand natural language and respond to various commands, such as turning on/off lights, getting information about temperature, maybe connecting with chatGPT. Our goal is to create a seamless user experience that integrates with different smart devices, making it easy and intuitive to control your surroundings.

2 State of the art

Home automation is a constantly evolving field, with the advent of the Internet of Things (IoT) and the growing popularity of voice assistants, smart homes have become more accessible than ever. Home automation systems can be designed to control various aspects of the home, including lighting, heating, air conditioning, blinds, doors, locks, and even appliances. The use of voice recognition for controlling these systems is a growing research area. There are currently several tools and platforms for voice recognition that can be used for home automation. Among the most popular are Amazon Alexa, Google Home, Apple HomeKit, and Microsoft Cortana. These tools allow users to control their home using simple voice commands, such as "turn on the living room light" or "close the blinds". Voice assistants can also be integrated with third-party devices, such as smart thermostats, security cameras, and sound systems. For the creation of a voice-controlled home automation system, several elements must be taken into account. Voice recognition must be accurate and reliable, so that the user's commands are correctly interpreted and executed. Connected objects must be compatible with the control system, with standardized communication protocols such as Wi-Fi or Bluetooth.

3 System Design and Modeling

This section will provide an overview of the hardware components, architecture, and software design of the voice assistant system, highlighting the key features and challenges involved in its development.

3.1 UML Diagrams

3.1.1 Use Case Diagram

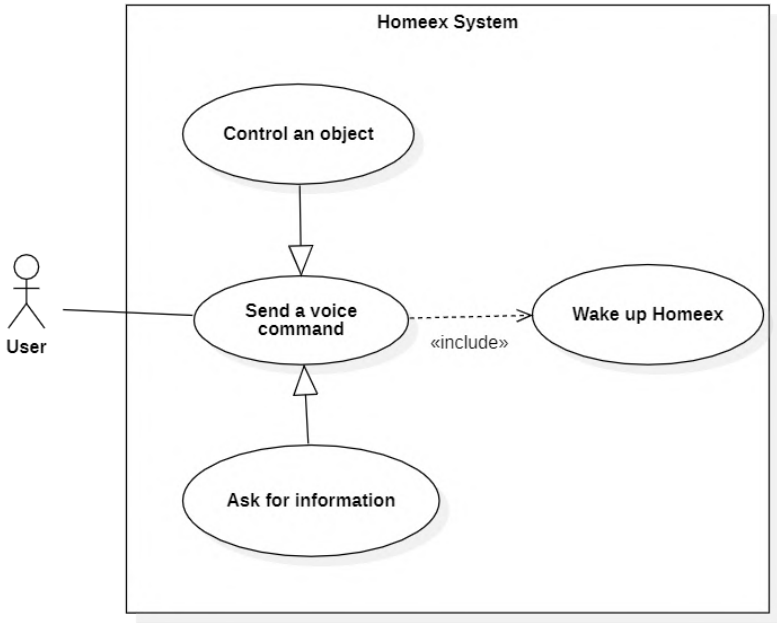

Figure 1: Homeex Use Case Diagram

The first use case is "Send Voice Command," which includes the "Wake Up System" sub-use case. This use case represents the ability of the system to recognize and interpret voice commands from the user.

Two other use cases, "Control Object" and "Ask for Information," inherit from the "Send Voice Command" use case. These use cases represent the specific tasks that the system can perform based on the voice command given by the user.

The "Control Object" use case represents the ability of the system to perform actions on objects in the home, such as turning on lights, adjusting thermostats, or controlling a smart lock. The "Ask for Information" use case represents the ability of the system to retrieve and provide information to the user, such as the weather forecast, traffic conditions, or the status of a specific sensor in the home.

3.1.2 Activity diagram

The activity diagram for a home voice assistant system involves a series of automated tasks triggered by voice commands from the user. The system can handle functions such as turning on/off lights and retrieving environmental data like temperature and humidity.

The diagram depicts a flowchart of these activities and how they are executed by the system without human intervention, making it convenient for the user.

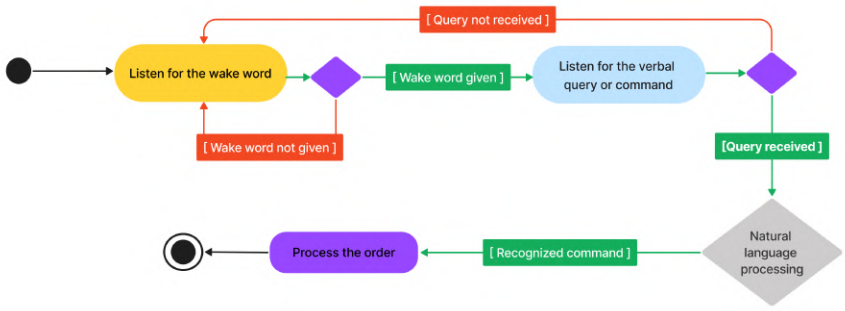

Figure 2: Homeex Activity Diagram

3.2 Hardware Software Components

Homeex project involves the integration of various hardware and software components to create a smart home assistant capable of handling various tasks. The following sections describe the used hardware and software components.

3.2.1 Hardware Components

The following hardware components were used in the Homeex project:

- Raspberry Pi 3 B+: The main processing unit for our project. A small single-board computer that can run various operating systems, including Raspbian. It is commonly used for various applications, including IoT projects, media centers, and retro gaming systems, due to its small size, low cost, and high performance.

Figure 3: Raspberry Pi 3 B+

- NodeMCU: The NodeMCU is a low-cost IoT development microcontroller that is based on the ESP8266 WiFi module. It is used to control the lamp and dht-11 components in our project.

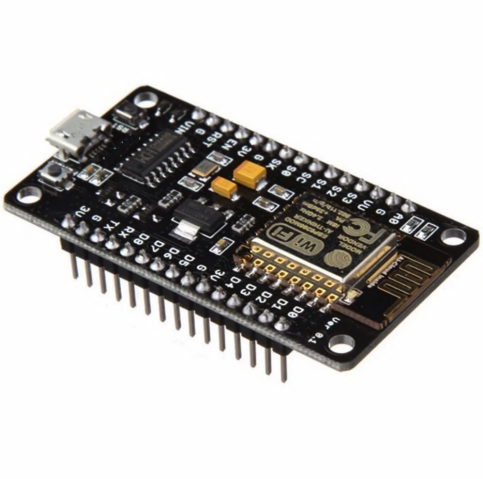

Figure 4: NodeMCU

- Relay SRD-05VDC-SL-C: The relay component is used to switch the power supply to the lamp. The relay is controlled by the NodeMCU.

Figure 5: Relay SRD-05VDC-SL-C

- DHT-11: Digital temperature and humidity sensor connected to the NodeMCU.

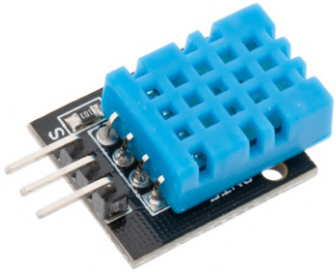

Figure 6: DHT-11 Sensor

- Microphone: An USB Microphone is connected to the Raspberry Pi to capture voice commands from the user.
- Speaker: The speaker is connected to the Raspberry Pi to provide audible feedback to the user.
- Wires: The wires are used to connect the various components with the nodeMCU.

- Router: The router is used to connect the Raspberry Pi and NodeMCU in the same network.
- Lamp, socket and Power Plug.

3.2.2 Software Components

The following software components were used in the Homeex project:

- Raspbian OS: Raspbian is a free operating system based on Debian, optimized for the Raspberry Pi hardware. [7]

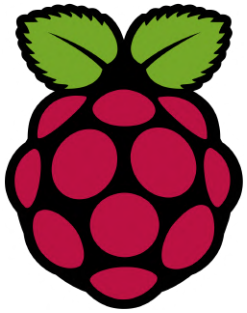

Figure 7: Raspbian OS

- Docker: Docker is a platform for developing, shipping, and running applications. Docker is used to create a container for Mosquitto, Rhasspy and Node-RED. [2]

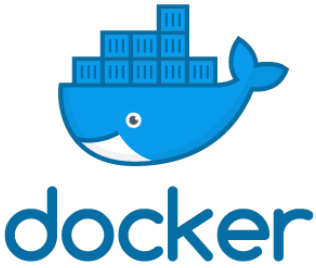

Figure 8: Docker

- Mosquitto: Open-source message broker that implements the MQTT protocol. Mosquitto is used to facilitate communication between the Raspberry Pi and NodeMCU. [3]

Figure 9: Mosquitto

- Rhasspy: Lightweight and privacy-focused voice assistant that runs entirely on the user's device. Rhasspy is used to process voice commands from the user. [11]

Figure 10: Rhasspy

- Node-RED: Programming tool for wiring together hardware devices, APIs, and online services. Node-RED is used to create the logic and flow of our project. [5]

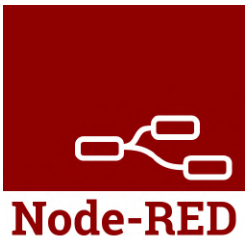

Figure 11: Node-RED

- Arduino IDE: Open-source software development environment for the Arduino platform. The Arduino IDE is used to program the NodeMCU. [1]

Figure 12: Arduino IDE

- Putty: Free and open-source terminal emulator, serial console, and network file transfer application. Used in out project to access the Raspberry Pi via SSH. [6]

3.3 System Architecture

The home voice assistant system generally consists of three main components: a microphone, speech recognition software, and a command processing device. The microphone is a crucial component of the home voice assistant system, as it is responsible for capturing the user's voice. The speech recognition software is a key component of the home voice

assistant system. This software is able to convert the user's voice commands into text or executable commands. The command processing device is the brain of the home voice assistant. It receives the commands from the speech recognition software and executes them. This device can be integrated directly into the home voice assistant, or it can be connected to another device, such as a lamp or smartphone. Together, these three elements allow the home voice assistant to capture the user's voice commands, understand them, and execute them. Our *homeex* voice assistant can provide its owner with current time and hour, temperature, humidity and even jokes.

The process begins with the user saying a "wake word" which is in our case *Homeex* to activate the voice assistant, from the *"Homeex"* system. The voice recognition system is constantly listening and detecting the command. Once the system is activated, the speech recognition software (Rhasspy) converts the user's spoken words into text. Then, the text is analyzed by natural language comprehension software which makes it possible to understand what the user has requested. Based on the identified intent, the system performs a relevant action. Finally, the system generates an audio response which is transmitted to the user to confirm that the command has been executed.

We can divide our *HOMEEX* system into two major parts:

3.3.1 Controlling part

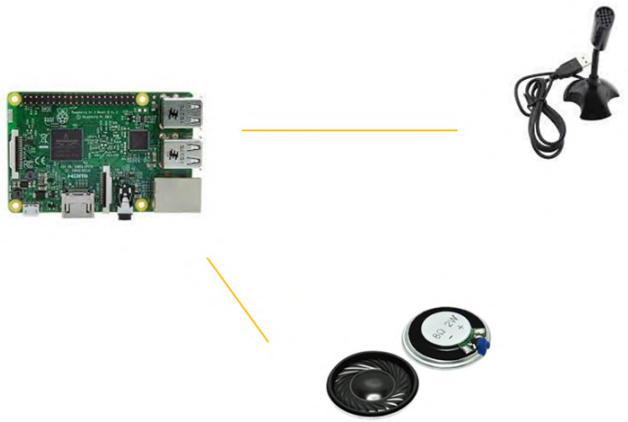

Figure 13: Controlling part

This part is responsible for listening to vocal commands emitted by user and making vocal response to him if his voice command is accepted. This part needs a microphone to receive the vocal command, a speaker to make the response and finally a micro-controller for processing the command and response either the command acceptable or rejected. The micro-controller used is Raspberry, which has its own OS, that will help us to perform high level processing like voice recognition.

3.3.2 Controlled part

In the other this part executes the commands after were processed by the micro-controller, this part contains a DHT sensor for temperature and humidity and lamp. These two

components leads our system to set light and provide user the current temperature and humidity state of its environment. Other operations which provide time and telling jokes are made within raspberry, otherwise they don't need any components.

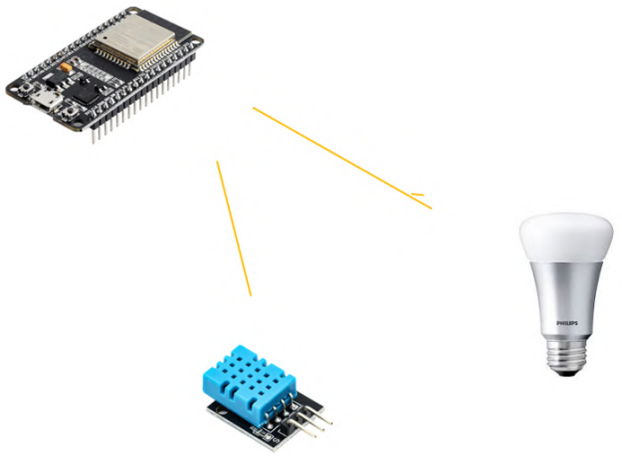

Figure 14: Controlled part

4 Implementation

In this section, we will be detailing every step we had to take to bring our project to life. First and foremost, we had to set up our environment and all the necessary tools required to execute the project. After that, we focused on configuring these tools to optimize their performance and ensure their seamless integration with our project.

4.1 Raspbian OS

To install Raspberry Pi OS on our Raspberry Pi, we need to follow a simple process that can be accomplished in just a few steps. Firstly, we need to download and install the Raspberry Pi Imager software [7] onto our computer. Next, we need to insert the SD card (using SD card reader) that we want to use as the storage device for our Raspberry Pi OS. Then, we need to launch the Raspberry Pi Imager and select the version of Raspberry Pi OS that we want to install. Once we have selected the OS, we can click on advanced options (Cogwheel icon in the lower right corner) and select "Enable SSH". Then we click on "write" to start the installation process.

Figure 15: Raspberry Pi Imager

After a few minutes, the Raspberry Pi Imager will complete the installation process, and we can eject the SD card and insert it into our Raspberry Pi. When we power up the Raspberry Pi, it will boot up into the newly installed Raspberry Pi OS.

4.2 Putty

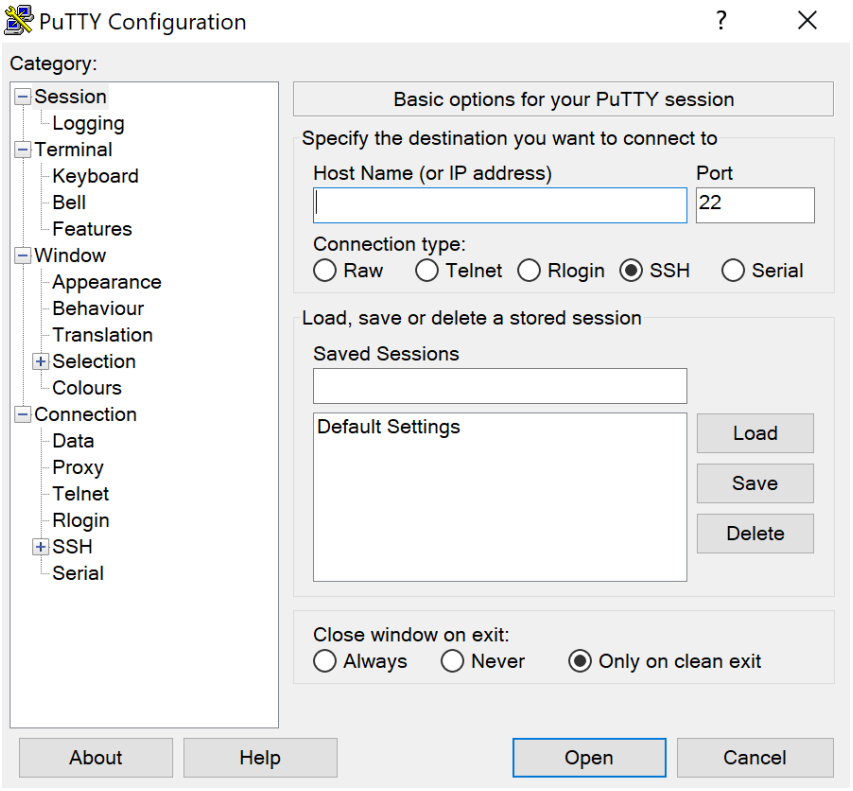

Figure 16: Putty

To access our Raspberry Pi Shell, we first need to ensure that our Raspberry Pi is connected to the same network as our computer. Once we have done that, we need to download and open PuTTY, then enter the address “raspberrypi.local” in the "Host Name" field. We should then select "SSH" as the connection type and click "Open". This will open up a terminal window where we can enter our Raspberry Pi login credentials. Once we have successfully logged in, we will have access to our Raspberry Pi through PuTTY. We can then execute commands, edit files, and perform any other tasks as needed.

4.3 Docker

In order to test our app before implementing it to raspberry we had to run it in an isolated environment called container using docker and docker compose to ensure that is working correctly.

the first thing to do is to pull the required containers from docker hub.

the required images are :

- **rhasspy/rhasspy** (the voice assistant platform) [13]
- **nodered/node-red** (the flow-based development tool) [10]
- **eclipse-mosquitto** (the MQTT message broker) [12]

```
$ docker pull rhasspy/rhasspy
$ docker pull nodered/node-red
$ docker pull eclipse-mosquitto
```

to run all those images, we will use docker-compose to simplify the configuration of all those containers in a single file named ***docker-compose.yml*** [2]

```yaml
version: "3.0"
services:
  mynodered:
    image: nodered/node-red
    container_name: nodered
    volumes:
      - node_red_data:/data
    ports:
      - 1880:1880
  mosquitto:
      image: eclipse-mosquitto:2
      container_name: mosquitto
      volumes:
        - ./config/:/mosquitto/config/
        - ./log/:/mosquitto/log/
        - data:/mosquitto/data/
      ports:
        - 1883:1883
        - 9001:9001
  rhasspy:
    image: rhasspy/rhasspy
    container_name: rhasspy
    # this container wont run unless mosquitto container is running
    depends_on:
        - mosquitto
    # Command to run this file with the current user and not as root
    # CURRENT_UID=$(id -u):$(id -g) docker-compose up
    user: ${CURRENT_UID:-1000:1000}
    environment:
      # user-writable home directory
      - HOME=/temp
    volumes:
      - /etc/localtime:/etc/localtime:ro
      # Link user configuration.
      - $HOME/.config/rhasspy/profiles:/tmp/.config/rhasspy/profiles
    ports:
      - 12101:12101 # http
      # we won't need this because we're using an external MQTT message broker
      # - 12183:12183 # mqtt
    command: --profile en


volumes:
  data:
  node_red_data:
```

Listing 1: docker-compose.yaml file

This file defines a multi-container application consisting of three services: Node-RED, Mosquitto, and Rhasspy.

The Rhasspy service depends on the Mosquitto service to be running first, which is specified with the ***depends_on*** option. The service also specifies an environment variable

for the user, and sets up several *volumes* to link to configuration files and directories within the container. The container also exposes several ports for accessing the service, and sets a command to start the container with the "en" for the english profile.

we can now deploy our containers as services and execute them in the background using this docker-compose command:

```
$ docker-compose up -d
```

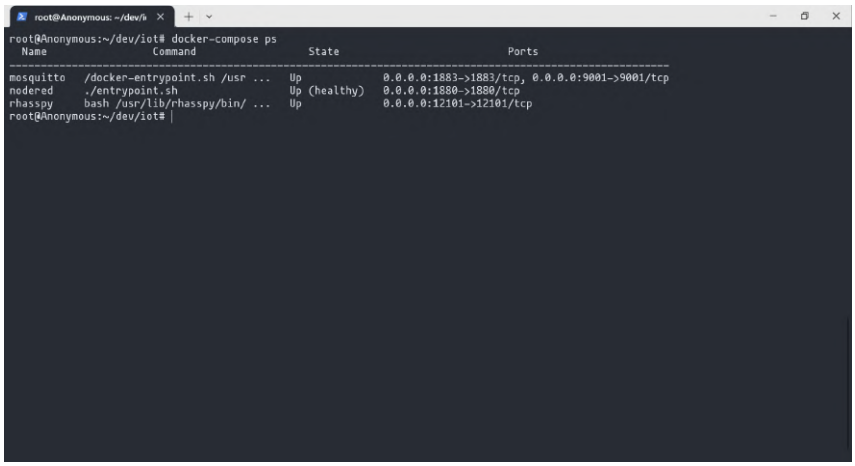

Figure 17: Containers running

after setting up the docker containers we can access each one using the corresponding url and port as configured:

- Rhasspy : ***http://localhost:12101/***
- Node-Red : ***http://localhost:1880/***

4.4 Rhasspy

To configure Rhasspy [11], we first need to make sure that it is started (Figure 17). Once it is started, we can open the Rhasspy web interface in our browser by navigating to the appropriate address and port, which is **http://localhost:12101**.

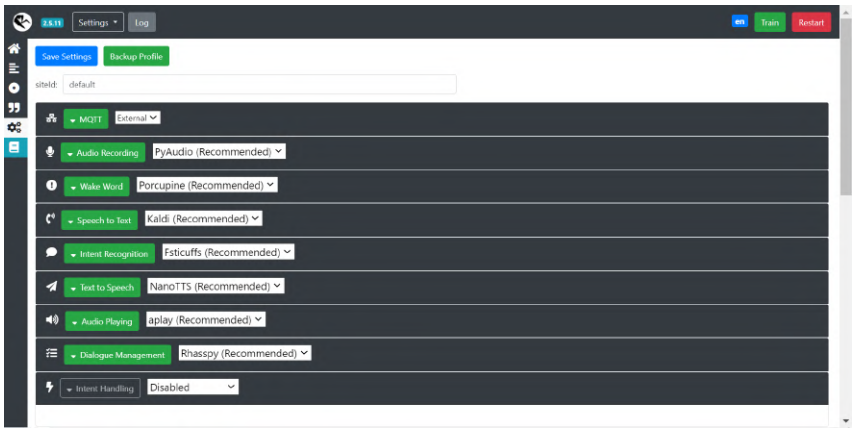

Figure 18: Configuration page

In the web interface, we can configure various settings, including the language model, the wake word, and the speech recognition engine. We can also train the system by providing it with examples of speech and text, so that it can better understand and recognize our voice.

4.4.1 MQTT Message Broker

as we're using an external MQTT message broker, we have to change the the host and port to our own host broker and to the default port value 1833. [4]

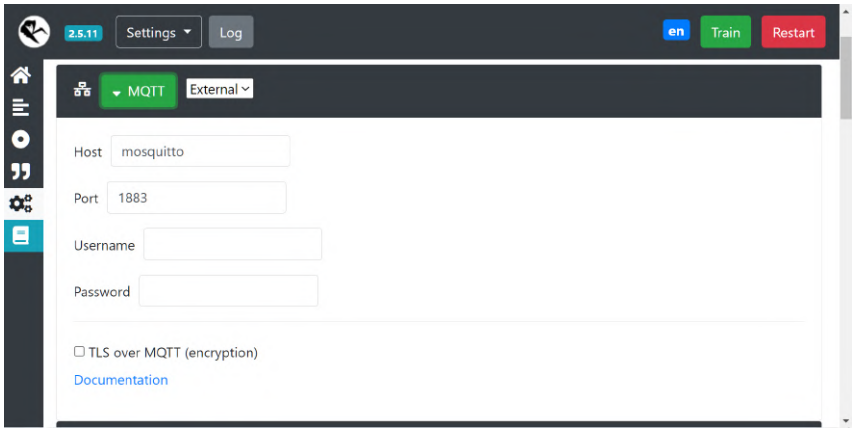

Figure 19: MQTT Messging Broker configuration

4.4.2 Wake Word

A wake word is a specific phrase or word that is used to activate the voice assistant and begin processing voice commands. For example, the wake word might be "Hey Rhasspy," "OK Google," or "Alexa." When the wake word is spoken, the voice assistant begins listening for further commands and can respond to user requests.

In this setting we choose the Raven engine that help us include our own wake words and train Rhasspy to recognise them.

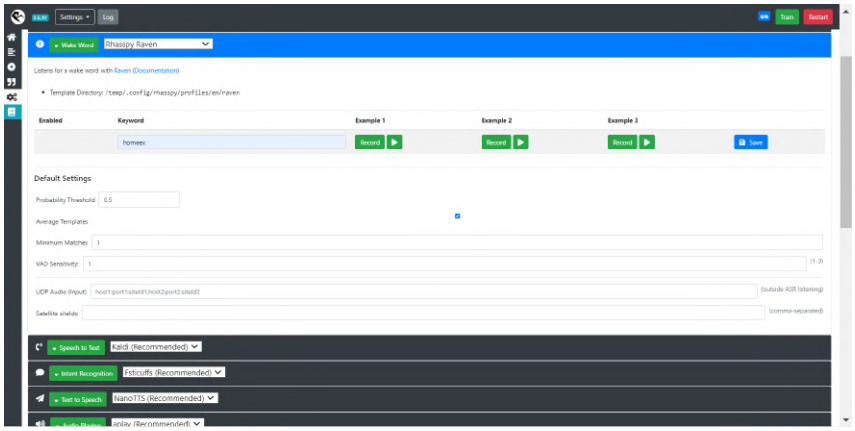

Figure 20: Wake Word Configuration

4.4.3 Audio Input

We need to make sure that Rhasspy is receiving input from the correct microphone or audio source by selecting the right input.

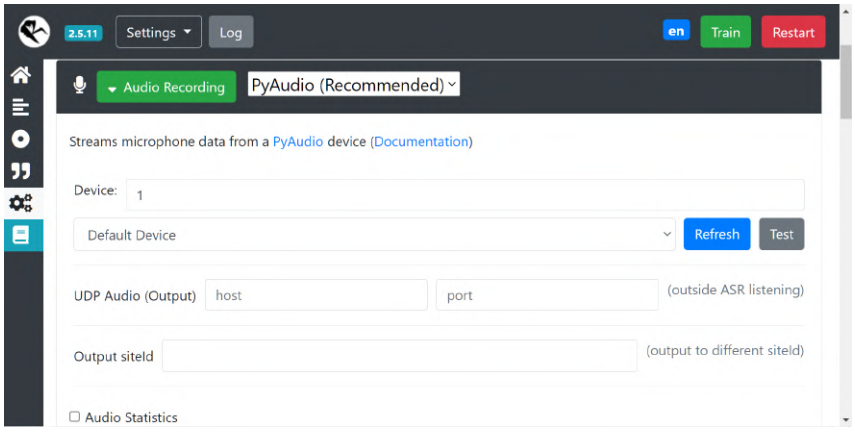

Figure 21: Audio Input Configuration

4.4.4 Speech To Text (STT)

Configuring speech-to-text (STT) is also an important part of setting up Rhasspy. STT allows Rhasspy to convert speech input into text, which can then be used to trigger specific actions or responses.

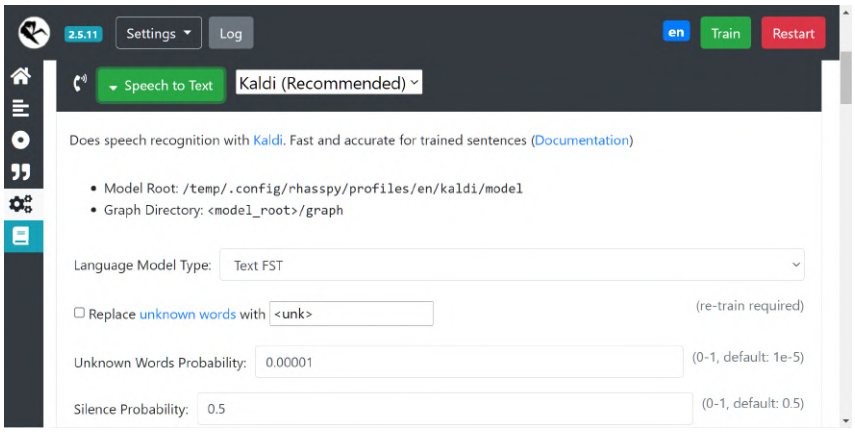

Figure 22: STT configuration

To configure STT, we need to choose an STT engine that is compatible with Rhasspy. Some popular options include Kaldi, PocketSphinx, and Google Cloud Speech-to-Text. we have chosen the recommended option.

4.4.5 Intent Recognition

Another key step in configuring Rhasspy is setting up the intent handling. This involves defining the various intents that Rhasspy should be able to recognize, and specifying how it should respond to each one.

An intent represents the user's goal or purpose in making the request. For example, an intent might be to turn on the lights, play music, or set a timer. The dialogue manager uses a set of predefined intents or custom intents created by the user to match the user's input to a specific intent.

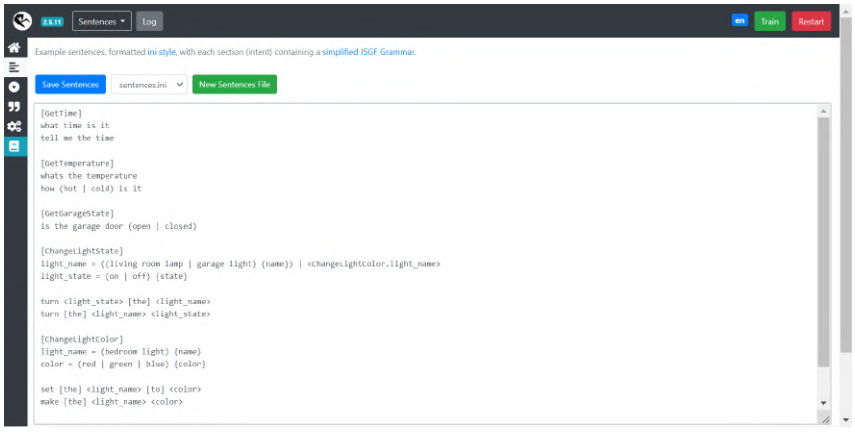

Figure 23: Intents Configuration

this figure shows the sentences written in a way that indicates the different part of a command, which include slots (Figure 24) that represent specific pieces of data that are needed to fulfill the user's request. For example, if the user says "Set a timer for 5 minutes," the dialogue manager would extract the value **"5 minutes"** as a slot.

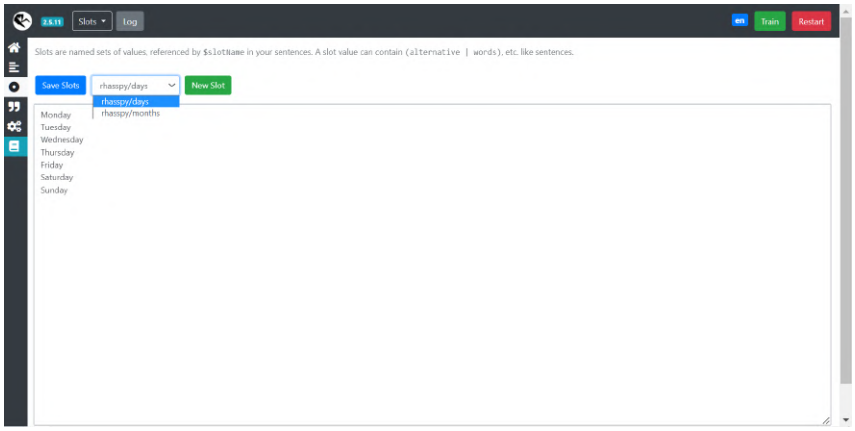

Figure 24: Slots Configuration

4.4.6 Text To Speech (TTS)

To configure TTS, we first need to choose a TTS engine that is compatible with Rhasspy. Some popular options include Google WaveNet TTS, MaryTTS, and Flite and others, Once we have chosen a TTS engine, we need to configure it in Rhasspy by specifying its settings and parameters like bellow.

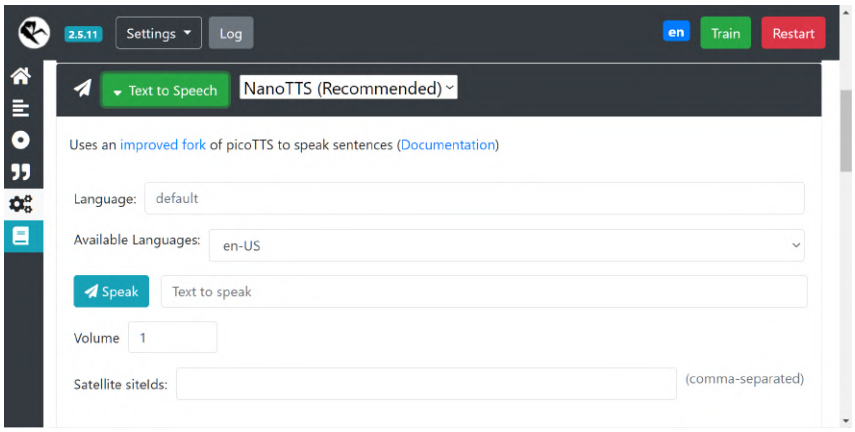

Figure 25: TTS Configuration

Once the TTS engine is configured, we can test it by entering sample text into the Rhasspy web interface and listening to the generated speech output. We can also customize the TTS settings and parameters to adjust the pitch, volume, and other aspects of the speech output.

4.4.7 Audio Output

here we have to choose the right audio output.

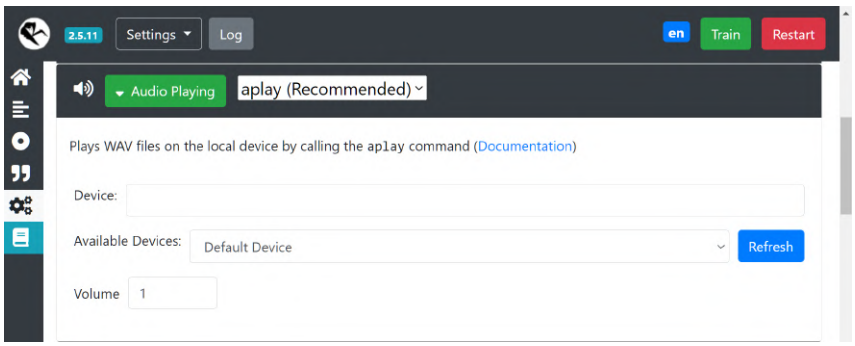

Figure 26: Audio Output Configuration

4.4.8 Dialogue Manager

The Dialogue Manager API in Rhasspy is designed to provide a flexible and customizable way to handle conversations between the user and the voice assistant. The API is based on the concept of *"intents"* and *"slots"* which allow the system to understand the user's input.

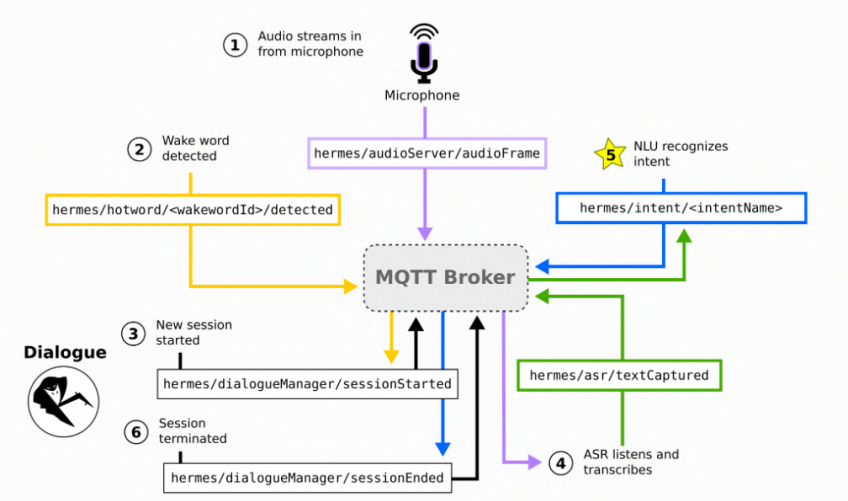

Figure 27: Rhasspy Dialogue Manager

When the user speaks to the voice assistant, the audio input is first converted to text using the configured speech-to-text (STT) engine. The resulting text is then processed by the dialogue manager, which attempts to identify the user's intent based on predefined rules and scripts in the intent configurations.

As shown in the figure 27 the MQTT API endpoints can be used to receive voice commands and send responses to external devices, as well as to trigger events and receive notifications from other devices or services.

the Rasspy's documentation has a full description of these APIs endpoints, including web-socket and REST APIs which gives the developer a great variety of techniques to fully communicate with the services.

4.5 Node-RED

Node-RED's simple yet powerful flow-based programming model allowed us to quickly and easily create our home automation system.

In this project, Node-RED was used to connect Rhasspy to Mosquitto Broker in order to control lights, ask for informations (time, date, temperature, humidity, jokes) more.

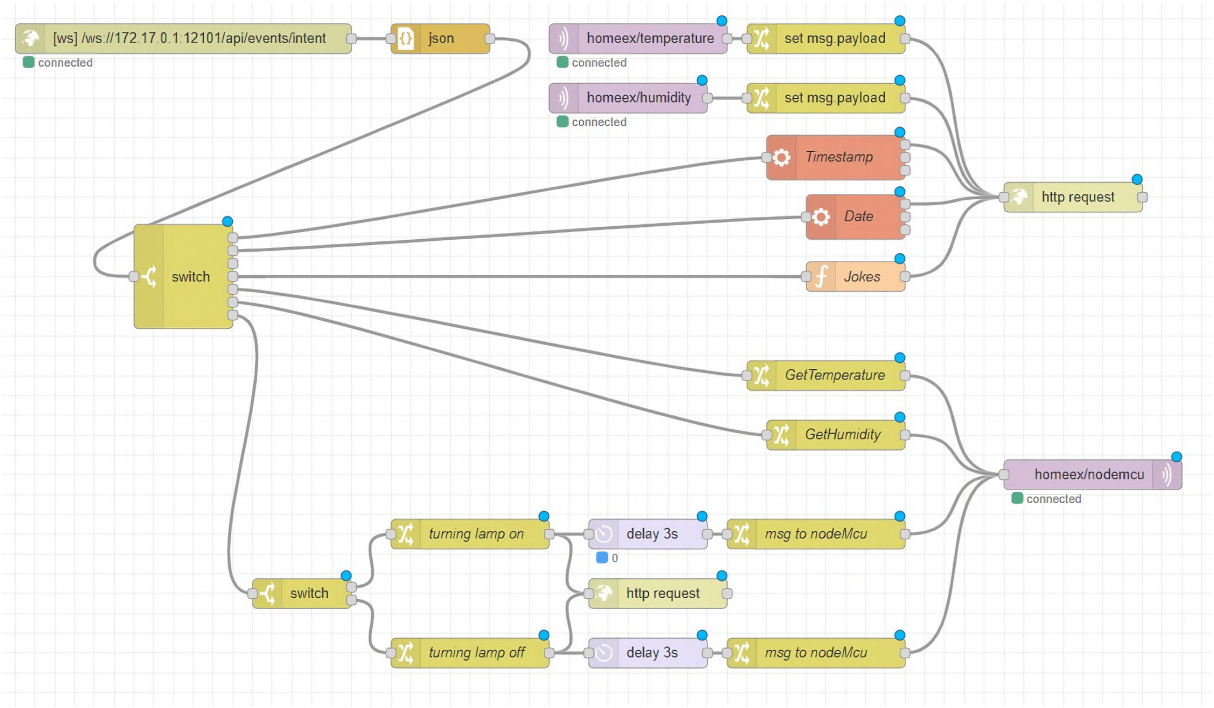

Figure 28: Homeex Node-RED Flow

When a WebSocket message is received by Node-RED from Rhasspy, a JSON node is used to parse the message, and a switch node is used to check the intent name. Depending on received intent name, Node-RED has many possibilities, based on the payload message, he determines what is required and performs all the operations that have been listed for him.

For example, to control lights:

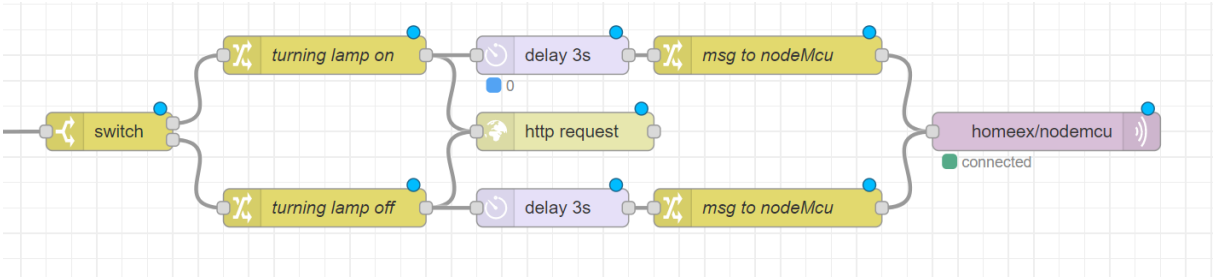

Figure 29: Control Lights Node-RED Flow

First a voice command is received, Rhasspy sends a JSON payload to Node-RED, which uses a switch node to identify the desirable state of the lamp. If the state desirable is "on," Node-RED sends an "on" message to Mosquitto MQTT Broker, which forwards it to the target device via the "homeex/nodemcu" topic. The target device receives the message and turns the light on. If the state is "off," Node-RED sends an "off" message to Mosquitto MQTT Broker, which is also forwarded to the target device via the same topic. To provide feedback to the user that the command was received and executed, we use a change node to send a message via an HTTP request to Rhasspy text-to-speech API, which is then played through the speakers.

4.6 NodeMCU

NodeMCU is an essential component for our home assistant IoT project as it acts as a microcontroller board that is specifically designed for internet of things applications. NodeMCU is based on the ESP8266 Wi-Fi chip, which provides a reliable and efficient way to connect our IoT devices to the internet.

To make NodeMCU work with our project, we used the Arduino IDE to write and upload code to the board. Before uploading the code, we must check that the necessary drivers and libraries are installed, and select the correct board and port settings.

Our code in the Arduino IDE allows NodeMCU to connect to our network, communicate with the DHT-11 (temperature and humidity sensor), and control the lamp in our home. We also added code that enables NodeMCU to send and receive requests to and from Node-RED using Mosquitto MQTT broker. This allows our home assistant to seamlessly communicate with other IoT devices and applications in our network. Let's take a look at what this code contains:

Firstly, we start by defining all dependencies, variables and also constants.

```
1   //calling librairies
2   #include <ESP8266WiFi.h>    //this librairy handle connexion with
3   //local network.
4   #include <PubSubClient.h>   //this librairy handle MQTT protocol
5   //between the nodeMCU(client) and raspberry(server).
6   #include "DHT.h"    //this librairy responsible to menage the DHT
7   //sensor.
8
9   #define MSG_BUFFER_SIZE  (50)
10
11  #define BUILTIN_LED D1  //D1 specify the pin number that is binded
12  //to lamp
13
14  DHT dht(D3, DHT11);
15  /*
16      D3 : this is the pin number on the nodeMCU board to which
17      the DHT sensor data is connected. inthis case, pin D3
18      DHT11 : this the type of DHT sensor being used.
19  */
20
21  float temp; //holds the value catched of temperature sensor.
22  float humi; //holds the value catched of humidity
23
24  // specify the name & password of the network to connect with
25  const char* ssid = "NETWORK_SSID";
26  const char* password = "NETWORK_PASSWORD";
27
28  //specify the ip adress of the MQTT broker.
29  const char* mqtt_server = "BROKER_SERVER_ADRESS";
30
31  //global variables used in our program
```

```

    /*
        creates a wifiClient object named, "espclient", which will
        be used to connect to a wifi acsses point, then it creates
        a PubSubClient object named "client" which takes the
        wifiClient object "espclient" as a parameter. this
        PubSubClient object will be used to connect an MQTT broker
        and publish or subscribe to message
    */
    WiFiClient espClient;
    PubSubClient client(espClient);

    char msg[MSG_BUFFER_SIZE];  //takes the topic from MQTT broker

```

Then, we define the Setup_wifi function which will ensure that our nodeMCU is connected to the network.

```
    //establish the connection with the network
    void setup_wifi() {
        delay(10);
        // We start by connecting to a WiFi network
        Serial.println();
        Serial.print("Connecting to ");
        Serial.println(ssid);

        WiFi.mode(WIFI_STA);
        WiFi.begin(ssid, password);

        //loop is turning while there is no connectionestablished
        while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
        }

        randomSeed(micros());
        /*
            enabling the random number generator, the randomSeed
            function is called to initialise the random number
            generator then, the random() function is used to generate
            a random number between 0 and returned value of micro()
            function which returns the number of milliseconds
            since th board began running. we will need it in
            the line 98
        */
        Serial.println("");
        Serial.println("WiFi connected");
```

```
30          Serial.println("IP address: ");
31          Serial.println(WiFi.localIP());
32          }
33
```

After defining setup_wifi, we override the function callback, which is major role organizing and ensuring that the MQTT protocol is working.

```
1       /*
2           when an event or message triggers a callback function, it is
3           executed by the MQTT client library. this allow the user
4           to perform custom actions in response to MQTT events
5           and messages.
6       */
7       void callback(char* topic, byte* payload, unsigned int length) {
8           //topic : the field that specifies which component will be
9           //triggered
10          //message : specifies the action performed by this component
11          Serial.print("Message arrived [");
12          Serial.print(topic);
13          Serial.print("] ");
14          for (int i = 0; i < length; i++) {
15          Serial.print((char)payload[i]); // retrieves the message
16          //recieved and casting it from byte type to string
17          }
18          Serial.println();
19
20          //if temperature is required
21          if ((char)payload[0] == 'T') {
22              char temperature[6];
23              temp = dht.readTemperature();   // retrieve the temperature
24              //from the sensor
25              sprintf(temperature, "%.f", temp);
26              client.publish("homeex/temperature",temperature);
27              /*
28                the value is read and sent to the MQTT broker with the
29                publish function under the topic homeex/temperature
30                and the message carries the the value read by the sensor
31              */
32
33          //if humidity is required
34          }else if((char)payload[0] == 'H'){
35              char humidity[6];
36              humi = dht.readHumidity();  // retrieve the humidity
37              //from the sensor
38              sprintf(humidity, "%.f", humi);
39              client.publish("homeex/humidity",humidity);
```

```
            /*
              the value is read and sent to the MQTT broker with the
              publish function under the topic homeex/humidity
              and the message carries the the value read by the sensor
            */
        //the message 1 for turning on the light
        //the message 0 for turning off
        }else  if ((char)payload[0] == '0') {
            digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note
            //that LOW is the voltage level
        }else if ((char)payload[0] == '1'){
            digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off
            //by making the voltage HIGH
        }
    }

```

The function reconnect came to establish the connection between the client and the broker.

```
    /*
        the reconnect function is usually implimented by the MQTT
        client librairy and can be configured to automatically attempt
        to reconnect to the broker with a specified intervall between
        each attempt.
    */
    void reconnect() {
      // Loop until we're reconnected
      while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Create a random client ID
        String clientId = "ESP8266Client-";
        clientId += String(random(0xffff), HEX);
        // Attempt to connect
        if (client.connect(clientId.c_str())) {
          Serial.println("connected");
          // Once connected, publish an announcement...
          client.subscribe("homeex/nodemcu");
        } else {
          Serial.print("failed, rc=");
          Serial.print(client.state());
          Serial.println(" try again in 5 seconds");
          // Wait 5 seconds before retrying
          delay(5000);
        }
      }
    }
```

The function setup initializes all parameters of our system, from making the connection with the network, then set the connection with the broker to finally sets the function callback which will be triggered once a message it is received from the MQTT broker.

```
1
2   void setup() {
3
4       dht.begin(); //enable the DHT sensor
5       pinMode(BUILTIN_LED, OUTPUT);   // Initialize the BUILTIN_LED
6       //pin as an output
7       Serial.begin(115200);
8       setup_wifi();
9       client.setServer(mqtt_server, 1883);
10      /*
11        sets the broker that the client will connect to it takes two
12        arguments:
13        - mqtt_server : this is a character array or string that
14        specifies the IP adress or domain name of the MQTT broker.
15        - 1883 : is the port number on which the MQTT broker is
16        listening to for connection.
17      */
18      client.setCallback(callback); //sets the callback function that
19      //will be called when a message is recieved from MQTT broker.
20  }
21
```

Finally, we define the function loop which will be running our system repeatedly.

```
1   void loop() {
2
3       if (!client.connected()) {
4           reconnect();
5       }
6       client.loop();
7       /*
8           the loop function is used to maintain the MQTT client's
9           connection with the broker and to process incoming and
10          outgoing messages.
11      */
12
13  }
```

5 Results and discussions

In this chapter, we will discuss the tests we conducted on our Homeex project and the results we obtained. Our project involved creating a home assistant system that can control lamps using voice commands, provide information about time, date, temperature and humidity, and tell jokes. However, we added some challenges to our project, including the ability to respond to commands in a noisy environment.

- Lamp Control Test:

  We tested the lamp control feature by giving voice commands to Homeex using our microphone, such as "Turn on the lamp" and "Turn off the lamp." We observed that Homeex successfully controlled the lamp using the relay module, and the lamp turned on and off according to our voice commands.

- Temperature and Humidity Test:

  We used the DHT-11 sensor to measure the temperature and humidity in the room. To test this feature, we asked Homeex, "What is the temperature?" and "What is the humidity?" We observed that Homeex accurately reported the temperature and humidity values in Celsius and percentage, respectively.

- Time and Date Test:

  To test the time and date feature, we asked Homeex, "What is the time?" and "What is the date?" We observed that Homeex correctly reported the current time and date.

- Jokes Test:

  To test the joke feature, we asked Homeex, "Tell me a joke." We observed that Homeex successfully told us a funny joke, making our day a little brighter.

- Noise Challenge Test:

  We added a challenge to our Homeex project, which was the ability to respond to voice commands in a noisy environment. We tested this feature by turning on some background noise, such as music and giving voice commands to Homeex. However, we observed that Homeex did not perform well in this scenario. The system struggled to recognize our voice commands and failed to recognize some of the them.

In conclusion, our tests showed that Homeex performed well and accurately controlled the lamp, reported temperature and humidity values, provided us with the correct time and date information, and told us some funny jokes. However, we also discovered that Homeex struggled to respond to voice commands in a noisy environment, which impacted its overall performance. This highlights the need for further improvements in the system's voice recognition and noise reduction capabilities to enhance its performance in noisy environments.

6 Conclusion

Iot objects have been used widely in our daily life because of its ability to access information from anywhere at any time on any device, improved communication between connected electronic devices, transferring data packets over a connected network saving time and money and automating tasks helping to improve the quality of a business's services and reducing the need for human intervention. In our voice assistant project, we had the opportunity to live this experience and make a voice assistant system that automatizes simple tasks like handling lights and getting some environmental data (temperature and humidity). All these tasks don't rely directly on human, just voice command from user is enough. Over our project we have faced some issues related to command detection like:

- Voices in the Background: in a quiet setting, the software will pick up the user's voice with ease. That's often not the case in a loud or crowded area. Multiple voices in the background will interfere with a user's voice inputs.
- Similar-Sounding Words: for speech recognition software, similar-sounding words pose a problem. Such software doesn't always process and discern between these types of words.

Additionally, we have also faced some problems related to budget that prevent us making easily our project, otherwise we substitute some components by other less performant.

We are planning in future to expand Homeex to be more interactive with its users in order to be able to speak. This feature makes Homeex answering to any question by using the "tell me" Intent word.

This well be possible with ChatGPT API due to its ability to understand and respond to natural language input. This makes it ideal for use in chatbot applications, as it can understand and respond to user queries in a way that feels natural and human-like. This API is not free, it requires to pay $0.002 / 1K tokens. Tokens in this API are pieces of words, where 1,000 tokens are about 750 words. [9] [8]

References

[1] Arduino. https://www.arduino.cc/. Accessed: March 19, 2023.

[2] Docker. https://www.docker.com/. Accessed: March 19, 2023.

[3] Eclipse mosquitto. https://mosquitto.org/. Accessed: March 19, 2023.

[4] Mqtt. https://mqtt.org/. Accessed: March 19, 2023.

[5] Node-red. https://nodered.org/. Accessed: March 19, 2023.

[6] Putty. https://www.putty.org/. Accessed: March 19, 2023.

[7] Raspbian. https://www.raspbian.org/. Accessed: March 19, 2023.

[8] Exploring the capabilities of the chatgpt api: A beginner's guide, 2021.

[9] OpenAI Pricing. https://openai.com/pricing, 2021.

[10] Node-RED Contributors. Node-red docker image. https://hub.docker.com/r/nodered/node-red. Accessed: March 19, 2023.

[11] Michael Hansen and contributors. Rhasspy documentation, 2021.

[12] Eclipse Mosquitto Project. Mosquitto docker image. https://hub.docker.com/_/eclipse-mosquitto. Accessed: March 19, 2023.

[13] Koen Vervloesem. Rhasspy docker image. https://hub.docker.com/r/rhasspy/rhasspy. Accessed: March 19, 2023.