**Voice assistance documentation**

# Summary

As part of our project, we need to set up an ESP32 with a microphone that will record our commands and return answers from OpenAI API.

The steps we need to overcome are:

# Challenges:

1. [Setup the ESP32 board](#_Setup_the_ESP32) and upload Basic code to it (Blink)
2. [Set up A microphone](#_Setup_A_microphone) on the ESP32 Board and get the sound from the environment.
3. [Convert the sound](#_Covert_to_WAV) into a WAV file and save it.
4. [Hear the WAV](#_Ensure_the_WAV) file that was saved and ensure it sounds good.
5. [Train a classify model](#_Train_Wake-up_model) that we will use for our Wake-up word(side-work)
6. Convert the Wav with a speech-to-text tool into a Base64 variable.
7. Create an OpenAI user and use the token to send requests and get answers.
8. Listening to the microphone and changing to a listening situation whenever we hear the wake-up word.
9. When the user finishes recording the command, convert it to Base64 and send it to OpenAI.
10. Get the Open AI answer.
11. Return the answer to the user.

# Documentation

## Setup the ESP32

We use the acknowledge bank to configure the ESP32. In the beginning, we used the Adi computer and it worked fine But with Yagel computer we struggled too much with it because the computer is an IOS operating system so we replaced with Windows and it worked smoothly. The Blink code was uploaded perfectly as well and when we changed the delay variable the light blinked faster and slower depending on the value we sent to the delay function.

## Setup A microphone

Tom gave us a microphone component (INMP441) and with the help of the Bank we tested the microphone:  
<https://github.com/0015/ThatProject/blob/master/ESP32_MICROPHONE/ESP32_INMP441_SETUP_ESP-2.X/ESP32_INMP441_SETUP_ESP-2.X.ino>

**The main challenge:**

When we connected the microphone to the board it was received short every time.

**The reason that caused that and the solution we found**

the INMP441 component can't setup to the board we used because The shape of the component does not allow connection to the matrix that we are used to due to the fact that the matrix connects rows to the ESP while the component is built so that there is no connection in a line that does not pass between two inputs of the component itself, therefore this caused a short and the solution was to connect wires directly to the ESP instead of to the board that the ESP connects to.

## Convert to WAV

In the repository there is a branch Yoav created that should be good start to create Wav file and understand how to convert the microphone data into Wav file:

<https://github.com/adiyosef9387/IOT-Project---Voice-assistant/tree/platformIO2>

in this branch the main files are:

1. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/platformio.ini> - the only thing that we should take care of is to change *upload\_port* and *monitor\_port* to the port we connect the esp component.
2. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/src/main.ino> - the main loop that will execute the listening to the microphone.
3. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/Scripts/record_audio.py> - after execute the *main.ino* file we should change *COM\_PORT* to the port that the ESP is connected to (like the ini file).

The flow should be like so:

1. Download the *PlatformIo* extension.
2. Pip install ion
3. Pip install pyserial
4. Ctrl+p then platformio:upload
5. Run the python script *record\_audio.py*
6. After finish record interrupt in the terminal the python script running with ctl+c
7. The wav file will be created in the root folder

The main challenge:

We couldn’t find some organized documentation that helped us use the microphone data in WAV.

## Ensure the WAV sound

**The main challenge**

Still didn’t succeed with the previous task…

## Train Wake-up model

At first, we thought to use the trained model from the reference that was already trained for the word “go” but Tom told us that he expected us to train our own model.

**Main challenge:**

So we tried to figure out how to do that from the next repo:

<https://github.com/ICST-Technion/S22_VoiceControl/tree/main/SplitNetwork_ESP32>

we tried to start with this and struggled a little bit with it.

**The first attempt to train the model:**

Tom said that considering the circumstances we can use:

<https://docs.edgeimpulse.com/docs/edge-ai-hardware/mcu/seeed-xiao-esp32s3-sense>  
Which is an online service that trains split networks and supports specific microphones we got from Tom.

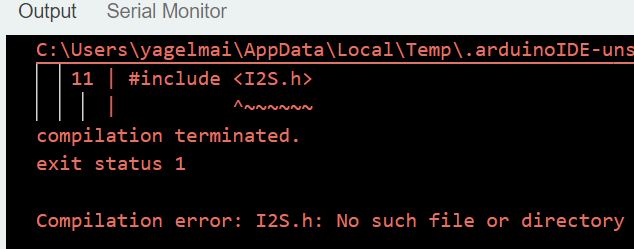
**The main challenge with the train model via online service**

We connected the microphone with the ESP board to the computer but when we got to the compile and uploaded the program to the ESP board we had to choose if our ESP32 version is 2.0.x or 3.0.x

A screenshot of a computer

Description automatically generated

So as we configured at the esp32 setup it is 2.0.x so we tried to upload this code but got an error that the I2S.h file did not exist:



And when we tried to upload the 3.0.x code we got the esp\_i2s is not declared:

