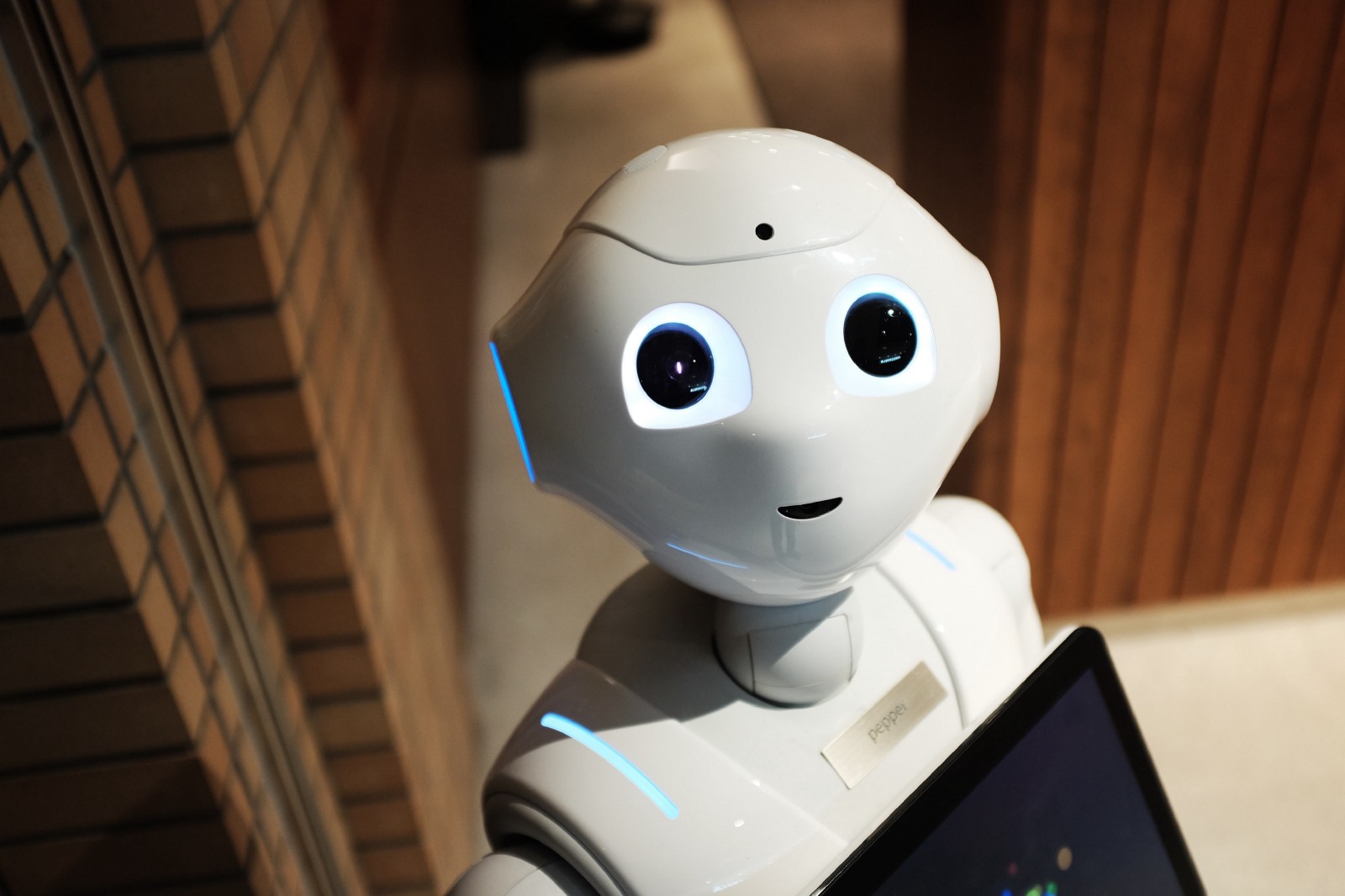
# Building a bot that speaks like you

## In this article, we will build a bot using Python and the WhatsApp chats that will speak like you

Have you ever seen those Artificial Intelligence bots on websites that answer your questions? And wondered how are they able to understand my questions and answer them correctly? These are called chatbots which make use of Natural Language Processing to converse in a human manner



Natural Language Processing is the branch of Machine Learning which makes use of various ML algorithms in order to facilitate humans converse with machines in their own language.

In this article, we will build a chatbot using Python (mainly TensorFlow) using chats from Whatsapp. The chatbot will generate messages coming from you and the person you download the chat with. The article is divided into multiple sections as follows:

1. Cleaning Chats
2. Preprocess Data
3. Using the model to speak like you
4. Building the API
5. Building the UI
6. Containerize the application

If you want to jump to the code, here’s a link to the code repository: [me\_bot](https://github.com/fenago/blog/tree/main/Me%20Bot" \t "_blank)

**Cleaning Chats**

First of all, you need to download the data from WhatsApp

From WhatsApp on your phone, go to any chat and export it by going into the settings. Move the txt file that you receive inside the main folder.

Let’s clean these chats before pre-processing



**clean\_whatsapp\_chats.py**

Let’s dissect this file:

* We start by importing the necessary files and then defining the YOUR\_NAME and OTHER\_NAME variables (replace these with the names from your WhatsApp chat)
* We then make use of the read\_file function that reads the file whose name we pass through the command line
* Our main function, extract\_text , generates arrays for all the texts, texts sent by you, texts sent by the other person. It also keeps tracks of whether the last text was sent by you or the other person, in case the messages are multi-line, so they can be attributed to the correct person
* make\_directory creates a folder for saving these arrays
* Finally, write\_to\_files is called on all these arrays to write them to disk using pickle

We can call the file using the following command where you replace “<name\_of\_text\_file.txt>” with the name of your chats file:

python clean\_whatsapp\_chats.py <name\_of\_text\_file.txt>

**Preprocess Data:**

Let’s prepare the data before building our WhatsApp chatbot. We will start by importing the necessary libraries and loading the sentence model:



Let’s explain what’s happening in this block of code:

* We start by importing the necessary libraries.
* Then, we define some variables, mainly our media app, i.e, WhatsApp and the link to the model URL.
* In the next few lines, we load a pre-trained sentence model. This model is a **Universal Sentence Encoder** that encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks.
* We use tensorflow\_hub to load this model and further use sentence piece library to call functions like encode(change the text to high dimensional vectors) on our chats

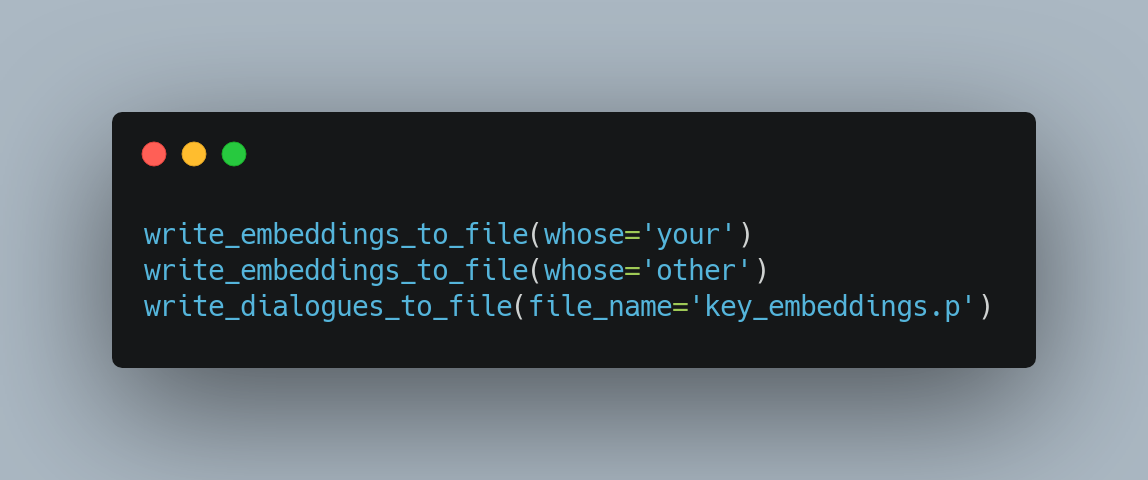
Now, let’s define some functions that will read the chats we cleaned in the last section along with generating embeddings for them:



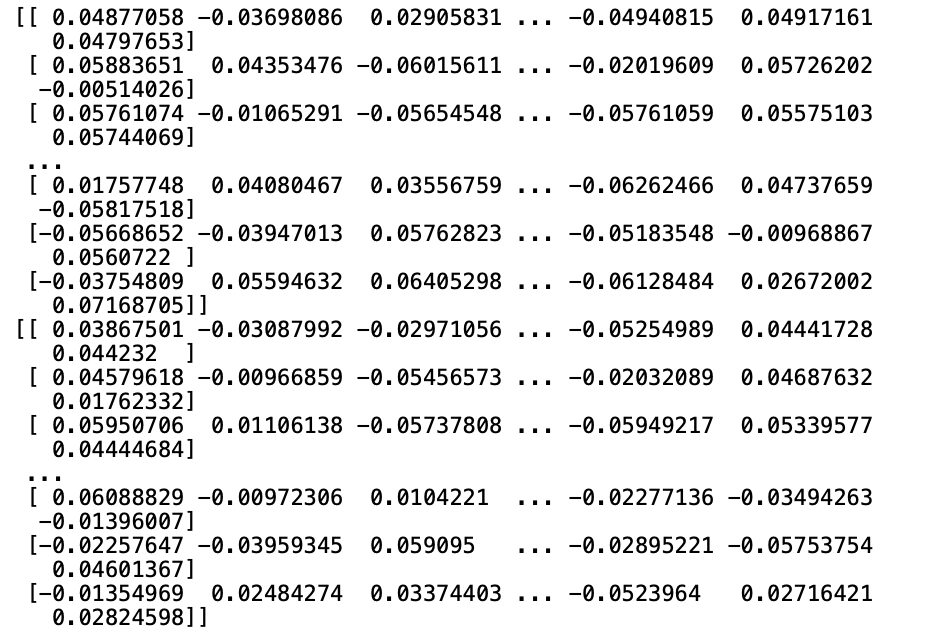
Let’s explain these functions one by one

* process\_to\_IDs\_in\_sparse\_format : This function takes in the sentences from the chats and returns three things values (IDs generated against every word by the sentence piece model), indices (list of lists containing the index of a sentence and its length) and dense\_shape (the shape of the matrix which is the number of sentences by the length of the longest sentence)
* embed\_sentence\_lite : This function takes in the sentences and calls the above message. Then passes those values to the model to generate the embeddings and returns them
* write\_embeddings\_to\_file : Takes in the whose texts to read and generates embeddings using the above function.
* write\_dialogues\_to\_file : Generates embeddings using the above functions but uses the dialogues dictionary created in the previous section

Let’s call these functions to process our chats:



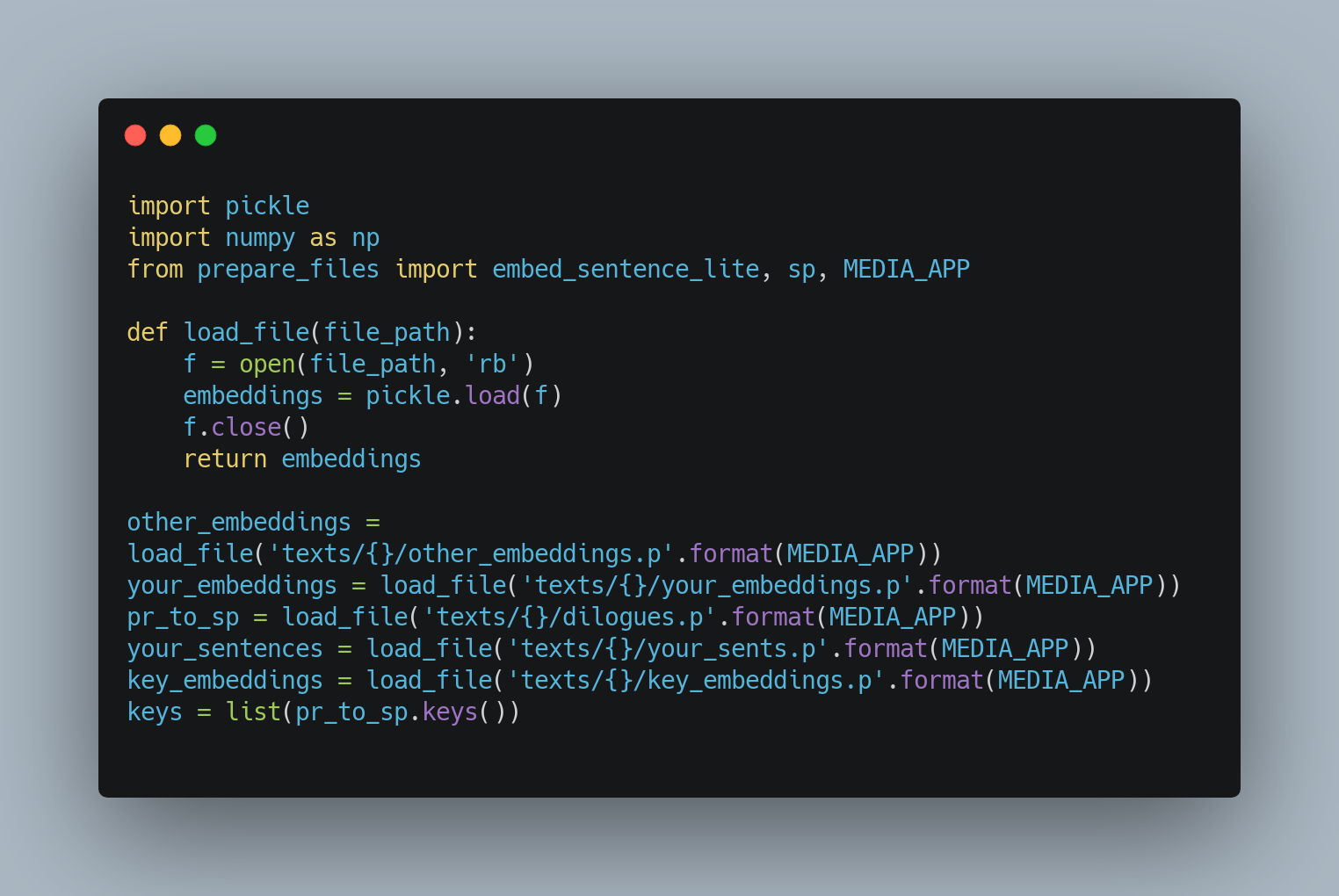
You can analyze the embeddings by printing them. They are just arrays containing the vectors representing the texts:



**Using the model to speak like you**

Let’s now get to the fun part, getting the program to speak and respond like you

Let’s start by importing the required libraries and functions defined above along with loading the chats and embeddings we created above:



your\_embeddings contain the embeddings of quotes where you have spoken and key\_embeddings contain the embeddings of quotes where you have responded

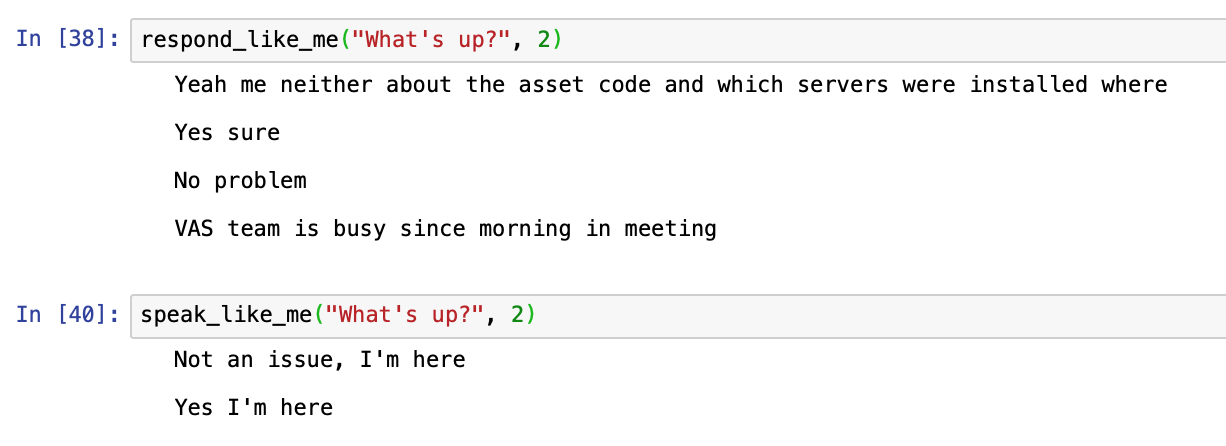
Let’s now build the functions that we will use to speak and respond like you



Let’s explain these functions here:

* find\_closest : Finds the closest embeddings with respect to the query we pass. It calculates the distance between all the embeddings and the query embedding and returns the top K indices that close to the query embedding
* speak\_like\_me : Returns the top K sentences in response to the query sentence. Calls find\_closest on your\_embeddings and the query embedding and uses the indices to return the sentences
* response\_like\_me : Same as above but uses key\_embeddings for finding the closest sentences

Let’s call these functions and see the output



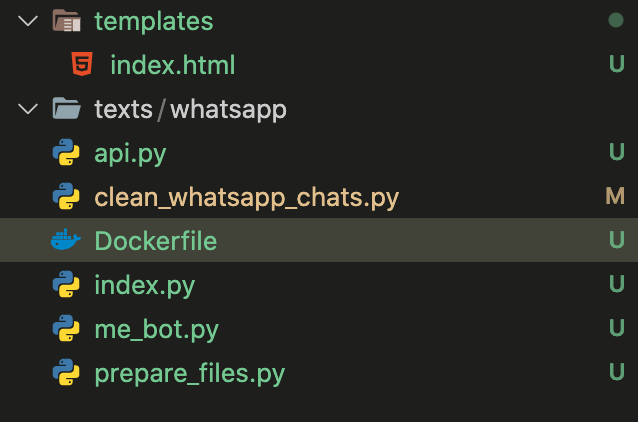
**Building the API**

There will be two endpoints for speaking and responding. Let’s first create separate files for the above modules. Here’s the name of all the files that we will be using:

1. clean\_whatsapp\_chats.py
2. prepare\_files.py
3. me\_bot.py
4. api.py
5. index.py (for UI)
6. Dockerfile (Containerize the application)
7. templates/index.html (Contains the UI)

**prepare\_files.py****me\_bot.py**

Here’s the directory structure:

****

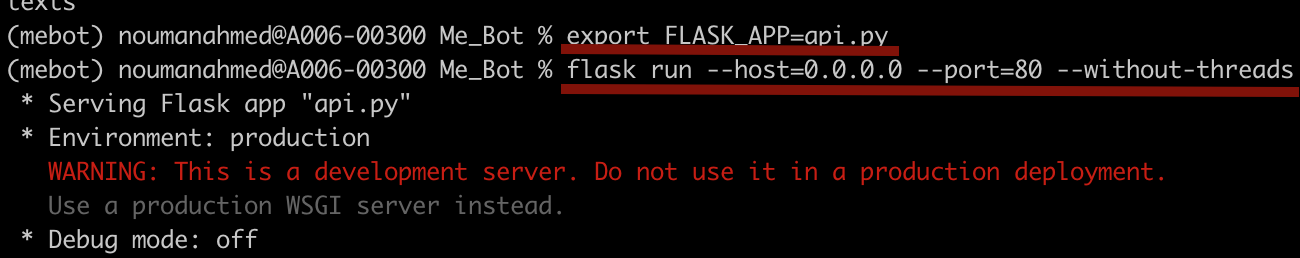
We will be using Flask for our purposes as it helps in building simple APIs really fast. I will introduce you to the whole code of API and then explain it.

**api.py**

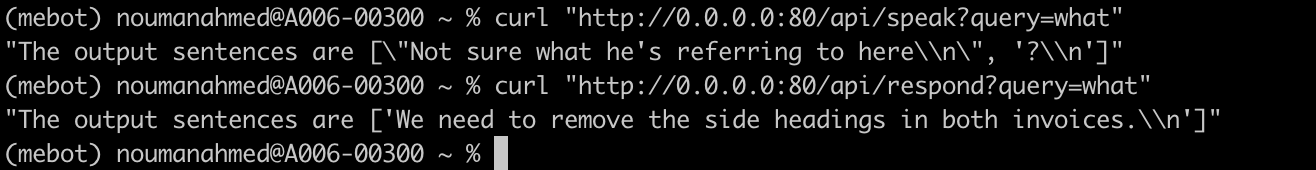
Let’s dissect this file:

1. We start by importing the required libraries. These include Flask and functions from the modules above.
2. We define the app variable, which basically represents our web app
3. We then use this syntax, @app.route . This basically is a decorator. All you need to know is whenever we hit this route, the function below that is called.
4. We have only included the method GET as we are not passing any data from a form.
5. We define the routes for speaking and responding like you.
6. Notice the last line in each of the endpoints, jsonify . JSON is the standard data format in web applications and hence we change our output to JSON before sending.
7. Lastly, we run the app on port 80.

Let’s run this file using the following commands:



Currently, there is no UI so we can’t view the app on our browser. Instead, we will use curl to access the API endpoints. We need to keep the above terminal running and open a new tab for this. We need to pass the query and make sure its named query as it’s named in the api.py , this is required for the API to function correctly



**Building the UI**

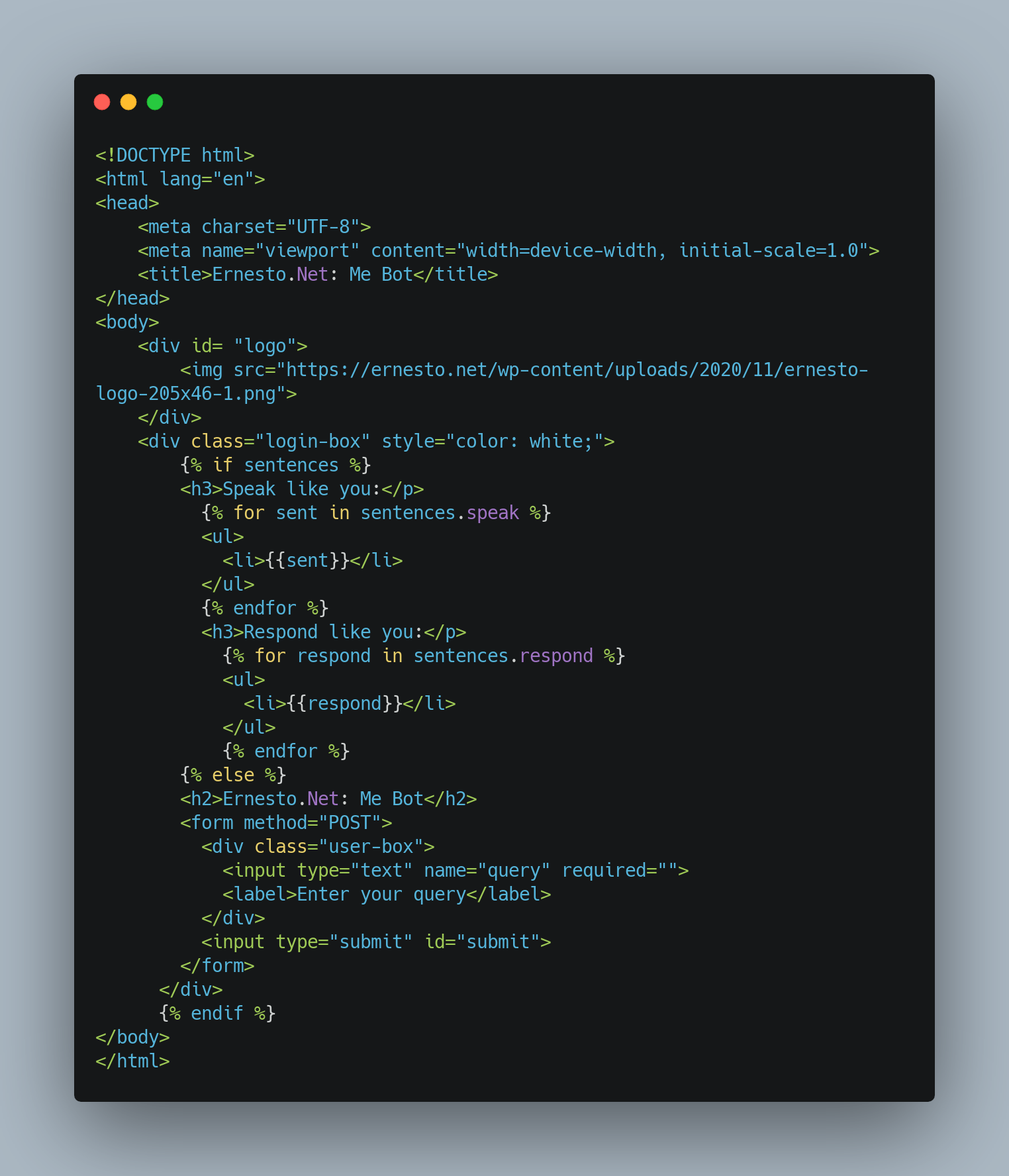
As before, let’s first see the code:

**index.py**

Let’s dissect the code:

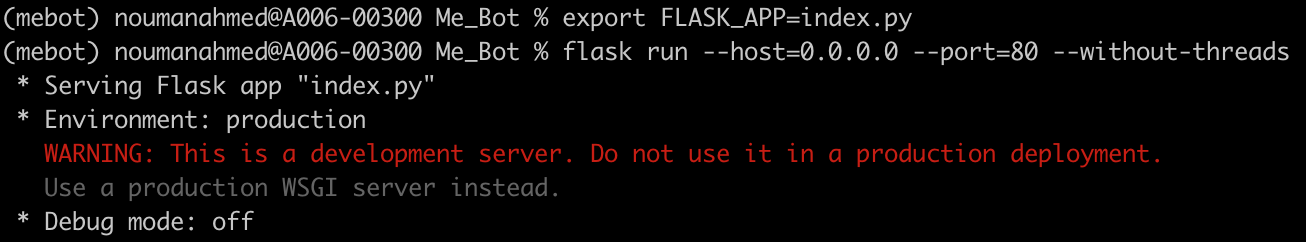
1. We import and define the app as we did in the api.py file but here we also define the Upload folder for saving the files
2. Then we define the route as before but here we are going to define two methods GET and POST as we are passing the query through a form.
3. In case of GET , we just render the form.
4. For POST method, we read the query and call the functions speak\_like\_me and respond\_like\_me
5. Finally, we send the sentences to the template.

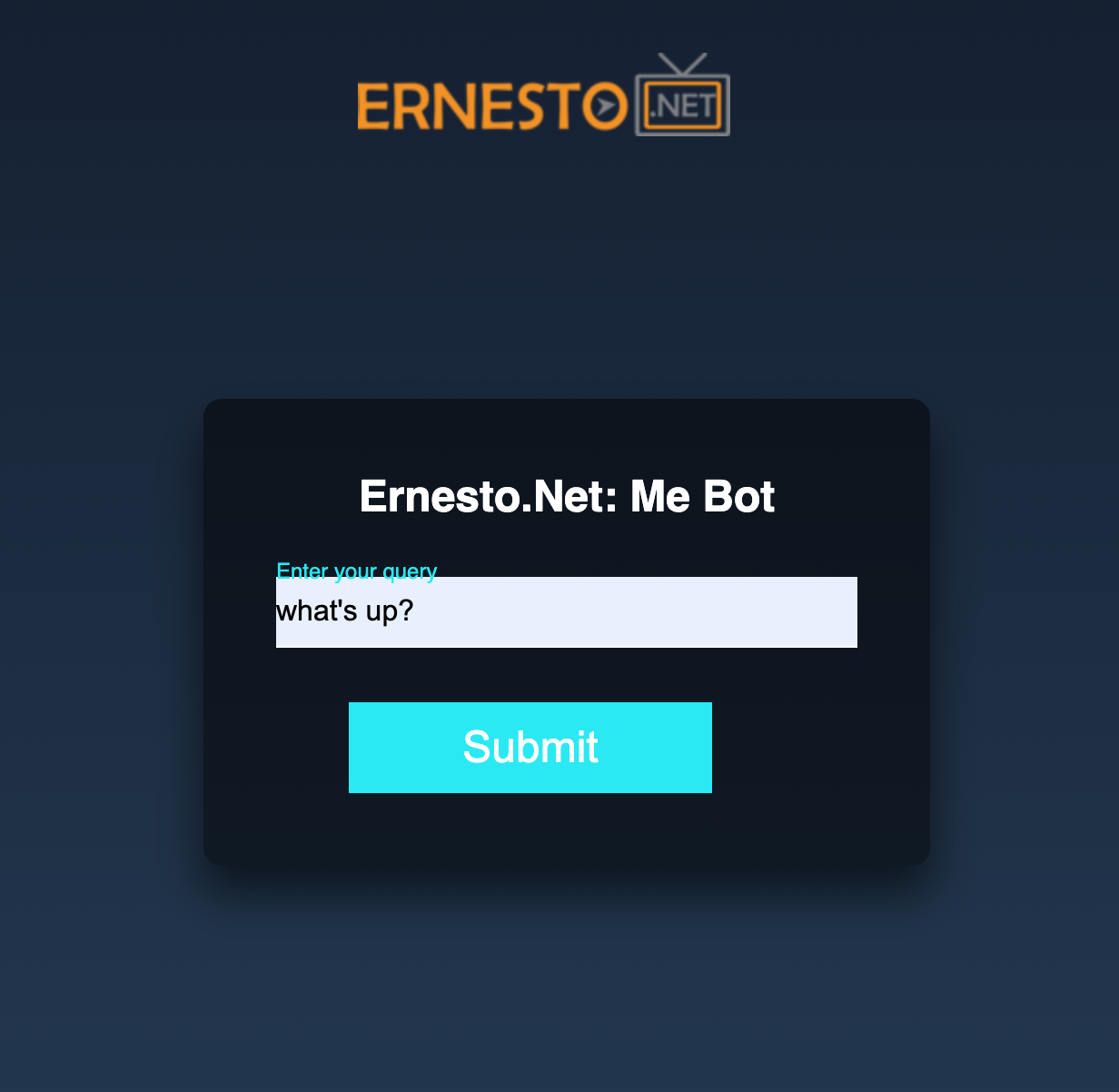
Let’s now look into the index.html file. Note that the index.html file should be defined in the templates folder otherwise Flask won’t be able to check it.

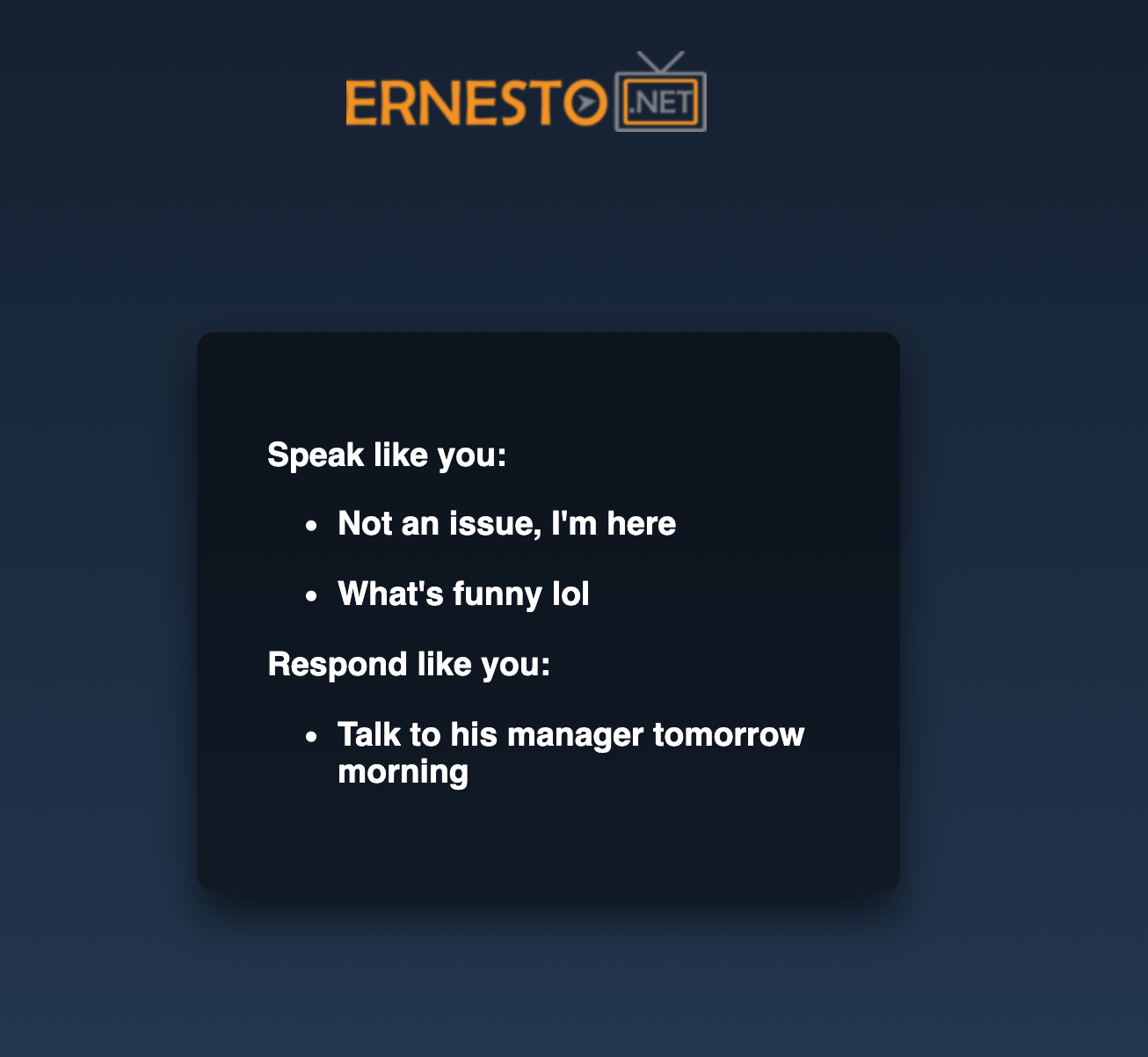
**templates/index.html**

Here, we check if we have the sentences dictionary and shows the form or the sentences respectively.

We will then run the app using the following command:



Below are the screenshots from the app: 



**Containerize the Application**

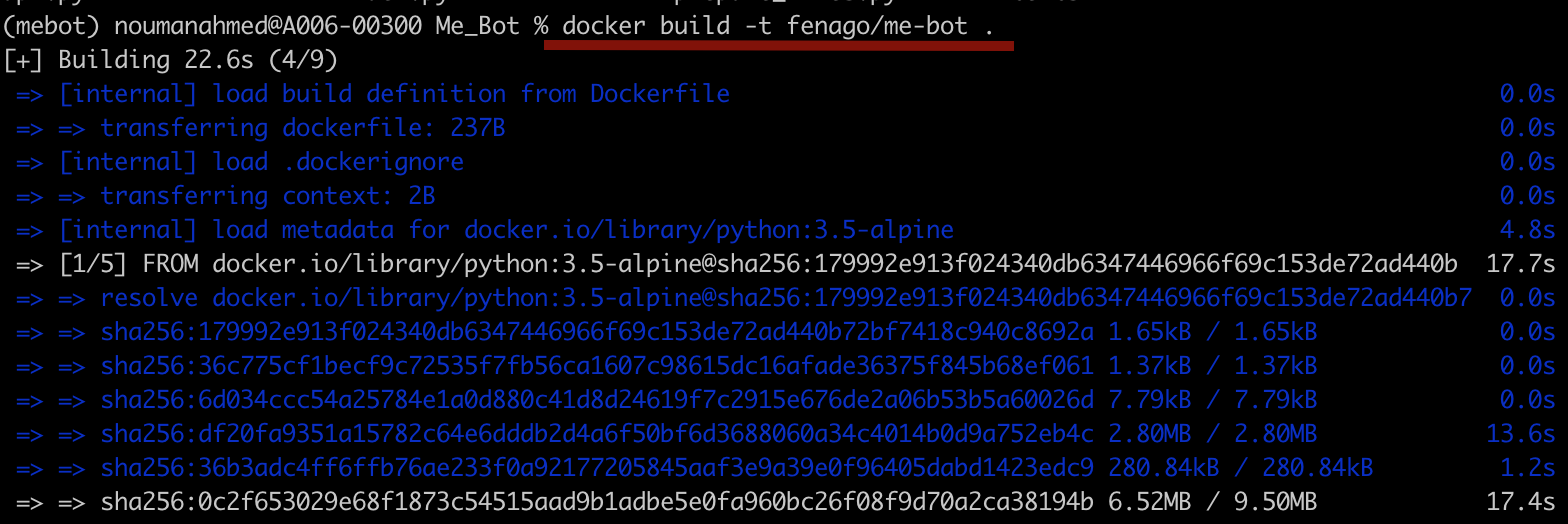
To make our app more useful and easily accessible to people, we would containerize it using Docker. We will create a Dockerfile in the same folder as the index.py. Let’s examine the Dockerfile

**Dockerfile**

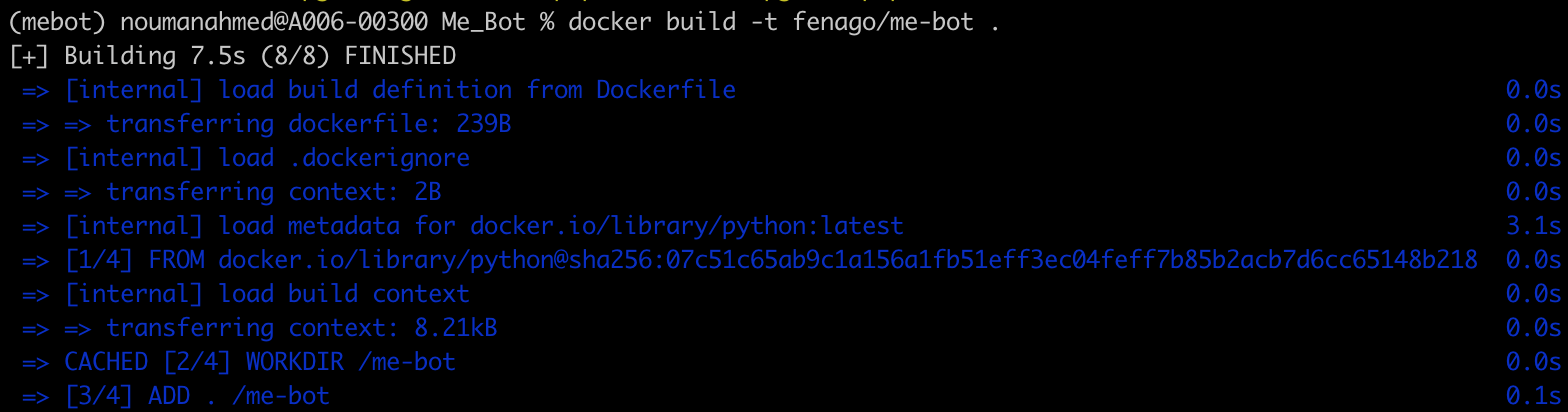
The commands are executed one by one here.

1. First, we install python and create a new working directory and copy all the contents from the current directory to this one.
2. Then, we run the commands to upgrade pip and install all the necessary libraries for this.
3. Finally, we run the index.py file which runs the server of our app.

To make an image, first, go inside the folder where your Dockerfile exists, then run the following command:



This will take some time to execute. After it’s done you can run your app as follows:



-p tag defines the port that you want the application to run on. You can view the image running on the docker desktop and also view the application in the browser.

I have also pushed this docker image to docker hub so you can download and play around with this application here: [me\_bot](https://hub.docker.com/r/fenago/me-bot" \t "_blank)

**Conclusion**

In this article, we looked into using Python to build a bot that speaks and responds like you using your WhatsApp chats:

1. We started by cleaning the chats and making separate lists for each user texts
2. We then loaded the sentence models and generated embeddings for the chats
3. Then, we used that model to speak and respond like you
4. After that, we created an API and a UI for our application using Flask
5. Finally, we built a docker image so that anyone can use this application.