**Building a voice-input and text-based AI chatbot with OpenAI and Python — Chatbot Backend Setup (Part 1)**

[[Heng Hui San (Grace)](https://medium.com/@gracehenghuisan?source=post_page-----e3a620b50d5b--------------------------------)](https://medium.com/@gracehenghuisan?source=post_page-----e3a620b50d5b--------------------------------)

[Heng Hui San (Grace)](https://medium.com/@gracehenghuisan?source=post_page-----e3a620b50d5b--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F503a15d8da8f&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40gracehenghuisan%2Fpart-1-building-a-voice-input-and-text-based-ai-chatbot-with-openai-and-python-e3a620b50d5b&user=Heng+Hui+San+%28Grace%29&userId=503a15d8da8f&source=post_page-503a15d8da8f----e3a620b50d5b---------------------post_header-----------)

5 min read

·

Sep 9

10



This blog series will guide you through ***the development of a voice-input and text-based AI chatbot using***[***OpenAI***](https://platform.openai.com/)***and Python***, with a focus on ***Flask***. The series is divided into two parts:

* **Part 1**: We will cover the application setup and delve into chatbot management, covering the process of converting speech to text and generating chatbot responses.
* **Part 2:** We will cover chatbot UI development of how to record audio from the microphone and make it into Blob before sending the data to server.

In this blog, we will focus on Part 1. If you’re keen on the content of Part 2, please visit [this blog](https://medium.com/@gracehenghuisan/part-1-building-a-voice-input-and-text-based-ai-chatbot-with-openai-and-python-19b1eec50).

**1. Prerequisites**

Before you begin, ensure you have the following requirements in place:

* **Python**: Install Python from the [Python website](https://www.python.org/downloads/).  
  In this context, I will use Flask for chatbot development., ensure you have Python installed in your device.
* **Open AI**: Have an account of [Open AI](https://openai.com/).  
  If you do not have an account of Open AI, you can enjoy a three-month gift credit of $5 which is a free trial that Open AI provides for new user.
* **Cloudinary**: Have an account of [Cloudinary](https://cloudinary.com/).  
  Cloudinary will serve as the platform for storing user voice inputs.

**2 Application Setup**

**2.1 Create Flask App**

Now, let’s get started by creating a new directory for this project. Inside this directory, you’ll need to create a directory structure like this:

/your\_directory\_name  
 /app  
 /static  
 /js  
 /script.js  
 /template  
 /index.html  
 /\_\_init\_\_.py  
 /routes.py  
 /utils.py  
 /.env  
 /requirements.txt  
 /run.py

**2.2 Create Virtual Environment**

Then, you need to create a virtual environment for Flask app by running the following commands.

* On macOS and Linux:

python -m venv your\_virtual\_environment\_name  
source your\_virtual\_environment\_name/bin/activate

* On Windows:

python -m venv your\_virtual\_environment\_name  
your\_virtual\_environment\_name\Scripts\activate

**2.3 Install Dependencies**

After that, let’s install all the required dependencies. You can copy and paste the following dependencies to your requirements.txt.

Flask==2.0.2  
Werkzeug==2.0.2  
openai==0.27.8  
cloudinary==1.33.0  
requests==2.26.0  
python-dotenv==0.19.2

Then, you can run the below command to install all the dependencies.

pip install -r requirements.txt

**2.4 Setup Environment**

First, go to your root directory and open your .env, copy and paste the following environment variables name to the file.

FLASK\_APP=app.py  
FLASK\_ENV=development  
OPENAI\_API\_KEY=  
CLOUDINARY\_CLOUD\_NAME=  
CLOUDINARY\_API\_KEY=  
CLOUDINARY\_API\_SECRET=

**2.5 Create Open AI API Key**

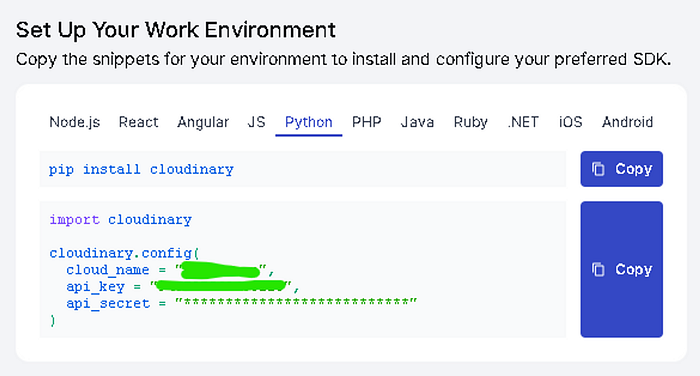
Now, let’s create a API KEY to access [Open AI](https://platform.openai.com/) APIs. Ensure that you logged into your Open AI account before going to [this page](https://platform.openai.com/account/api-keys) to create your API KEY

Once you create your API KEY, assign it to your OPENAI\_API\_KEY variable in your .env file.

**2.6 Setup Cloudinary Work Environment**

Now, let’s setup the work environment before storing audio file to Cloudinary.

Remember to log into your Cloudinary account before going to [this page](https://console.cloudinary.com/console/c-c74771f54ad46e333740e0fb85b24c). You see the following snippets of your environment on the page but remember to select **PYTHON** for your environment configuration:



After that, you can copy your snippets and assign them to the CLOUDINARY\_CLOUD\_NAME, CLOUDINARY\_API\_KEY, CLOUDINARY\_API\_SECRET variables in your .env file.

**3. Chatbot Management**

Now, we can delve into chatbot management, covering the process of converting speech to text and generating chatbot responses.

**3.1 Configure utils.py**

In utils.py, let’s create two methods:

* **generate\_ai\_chatbot\_response(messages):**To**a**ccess the OpenAI API for generating chatbot responses using the **gpt-3.5-turbo**model.
* **generate\_corrected\_transcript\_with\_cloudinary\_audio\_file(audio\_url)**:  
  To transcribe the audio URL (where the audio is stored in Cloudinary, and the URL is supplied from routes.py).

import os  
import io  
import requests  
import openai  
from openai.error import InvalidRequestError  
  
# Set the OpenAI API key from environment variables  
openai.api\_key = os.getenv("OPENAI\_API\_KEY")  
  
def generate\_ai\_chatbot\_response(messages):  
 try:  
 completion = openai.ChatCompletion.create(  
 model="gpt-3.5-turbo", messages=messages  
 )  
 reply = completion.choices[0].message.content  
 messages.append({"role": "system", "content": reply})  
 return "success", messages  
 except InvalidRequestError as e:  
 return "fail", str(e)  
  
  
def generate\_corrected\_transcript\_with\_cloudinary\_audio\_file(audio\_url):  
 try:  
 # Fetch the audio file as bytes using requests  
 response = requests.get(audio\_url)  
 audio\_data = response.content  
 # Create a io.BufferedReader object from the bytes data and set the name attribute  
 audio\_buffer = io.BytesIO(audio\_data)  
 audio\_buffer.name = "user\_voice\_input.wav" # Replace with the desired filename  
 # Pass the temporary file to the translate method  
 transcript = openai.Audio.translate("whisper-1", audio\_buffer)  
 result = transcript.text  
 return "success", result  
 except InvalidRequestError as e:  
 return "fail", str(e)

**3.2 Configure routes.py**

In routes.py, let’s setup Cloudinary configuration and define a Flask blueprint for routing and organizing related endpoints.

import os  
from flask import Blueprint, render\_template, request, jsonify, redirect, url\_for  
import uuid  
import cloudinary.uploader  
import cloudinary.api  
from .utils import (  
 generate\_ai\_chatbot\_response,  
 generate\_corrected\_transcript\_with\_cloudinary\_audio\_file,  
)  
  
cloudinary.config(  
 cloud\_name=os.getenv("CLOUDINARY\_CLOUD\_NAME"),  
 api\_key=os.getenv("CLOUDINARY\_API\_KEY"),  
 api\_secret=os.getenv("CLOUDINARY\_API\_SECRET"),  
)  
  
ai\_chatbot\_bp = Blueprint("ai\_chatbot", \_\_name\_\_)

The routes within the ai\_chatbot\_bp blueprint:

* **/**: Redirects to the /ai-chatbot route.

@ai\_chatbot\_bp.route('/', methods=("GET", "POST"))  
def index():  
 return redirect(url\_for("ai\_chatbot.ai\_chatbot"))

* /ai-chatbot: Handles AI chatbot interactions. It accepts user input, generates chatbot responses, and maintains the conversation history.

@ai\_chatbot\_bp.route('/ai-chatbot', methods=("GET", "POST"))  
def ai\_chatbot():  
 messages = (  
 request.args.get("messages")  
 if request.args.get("messages")  
 else [{"role": "system", "content": "What can I help you today?"}]  
 )  
 if request.method == "POST":  
 prompt = request.form["input"]  
 messages.append({"role": "user", "content": prompt})  
 status, messages = generate\_ai\_chatbot\_response(messages)  
 return jsonify({"status": status, "messages": messages}), 200  
 return render\_template("index.html", messages=messages)

* /speech-to-text: Handles speech-to-text transcription. It accepts users audio input, stores it in Cloudinary before using the audio data for chatbot management via generate\_corrected\_transcript\_with\_cloudinary\_audio\_file(audio\_url). Once we get the result, we need to remove the audio data from Cloudinary because we won’t need the data anymore

@ai\_chatbot\_bp.route('/speech-to-text', methods=("GET", "POST"))  
def speech\_to\_text():  
 if request.method == "POST":  
 if "audio" in request.files:  
 audio\_file = request.files["audio"]  
 if audio\_file:  
 # Generate a unique filename for the audio file  
 folder = "open-ai-audio"  
 filename = f"{str(uuid.uuid4())}.wav"  
  
 # Save the audio file to the "open-ai-audio" folder in Cloudinary  
 result = cloudinary.uploader.upload(  
 audio\_file,  
 folder=folder,  
 resource\_type="raw",  
 public\_id=filename,  
 overwrite=True,  
 )  
  
 # Get the public URL of the uploaded audio file  
 audio\_url = result["secure\_url"]  
 (status, result) = generate\_corrected\_transcript\_with\_cloudinary\_audio\_file(audio\_url)  
   
 # Delete the file  
 public\_id = f"{folder}/{filename}"  
 cloudinary.uploader.destroy(public\_id, resource\_type="raw")  
   
 return jsonify({"status": status, "result": result}), 200  
 return jsonify({"status": "error", "message": "No audio file received."})  
 return redirect(url\_for("ai\_chatbot.ai\_chatbot"))

**3.3 Configure init.py**

In \_\_init\_\_.py, let’s define a Flask application factory function named create\_app and register the ai\_chatbot\_bp blueprint with the Flask application. This means that all the routes and views defined in the ai\_chatbot\_bp blueprint become part of the application.

from flask import Flask  
  
def create\_app():  
 app = Flask(\_\_name\_\_)  
  
 from .routes import ai\_chatbot\_bp  
 app.register\_blueprint(ai\_chatbot\_bp)  
  
 return app

**3.4 Configure run.py**

In run.py, let’s initialize our application using the create\_app()

from app import create\_app  
  
app = create\_app()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 create\_app().run()

**4. Run the Development Server**

Once we completed the configuration of AI Chatbot, we can start the development server and run the app:

flask run

You should now be able to access the app using <http://localhost:5000/>

We’ve completed Part 1! Remember to proceed with the remaining configuration steps in [Part 2](https://medium.com/@gracehenghuisan/part-1-building-a-voice-input-and-text-based-ai-chatbot-with-openai-and-python-19b1eec50).

Thank you for reading, and I hope you’ve gained valuable insights.