**Development of an AI-Connected Chatbot Using Azure Cognitive Services**

**Submitted by:  
Course:**

**Submitted to:  
Date of submission:**

**Report Title:**

Development of an AI-Connected Chatbot Using Azure Cognitive Services

**Abstract:**

This report details the process of developing a chatbot integrated with Azure Cognitive Services using Python and Flask. The chatbot is designed to provide sentiment analysis responses based on user input, utilizing Azure's Text Analytics API. This document covers the implementation steps, challenges encountered, and mitigation strategies, along with recommendations for future improvement.

**1. Introduction**

Chatbots are emerging as a crucial component of user interaction in the digital space. By integrating natural language processing (NLP) and AI services, businesses can leverage these tools to enhance user engagement. The goal of this project was to create a chatbot that interacts with users and utilizes Azure Cognitive Services to analyze the sentiment of the input. The development process combined Flask, HTML for the user interface, and Python to interact with the AI service.

**2. Development Process**

**2.1 Technology Stack**

* **Python**: Programming language used to develop the backend of the chatbot.
* **Flask**: A micro-framework used for the web development process.
* **Azure Cognitive Services**: Specifically, the Text Analytics API for sentiment analysis.
* **HTML/CSS**: For creating the chatbot interface.

**2.2 Setting Up the Project**

**Step 1: Creating an Azure Resource for Cognitive Services**

1. **Creating the Azure Resource**

I began by navigating to the Azure portal and creating a new resource for Azure Cognitive Services. From the available AI services, I selected Text Analytics as the service to provide sentiment analysis capabilities for the chatbot.

1. **Choosing the Subscription and Resource Group**

I selected the Azure for Students subscription plan and created a new resource group named WAResourceGroup01. For the resource location, I chose the West US 2 region, which is optimal for the service availability in my region.

1. **Selecting the Pricing Tier**

The Free pricing tier was chosen, which provides up to 5,000 text records for sentiment analysis within a 30-day period.

1. **Generating API Keys and Endpoint**

Once the resource was created, I navigated to the Keys and Endpoints section in the Azure portal under the newly created resource. Here, I obtained the following keys and endpoint:

* + **Key 1:** 3c4c14d75f65421fab9e07920b8dea82
  + **Key 2: b7d0fbaae3b943babffc014b022322e9**
  + **Endpoint:** <https://wus2-computevm01.cognitiveservices.azure.com/>

These keys and endpoint are critical for authenticating API requests to the Text Analytics service.

A screenshot of a computer

Description automatically generated

**Step 2: Storing API Keys and Endpoint in Python**

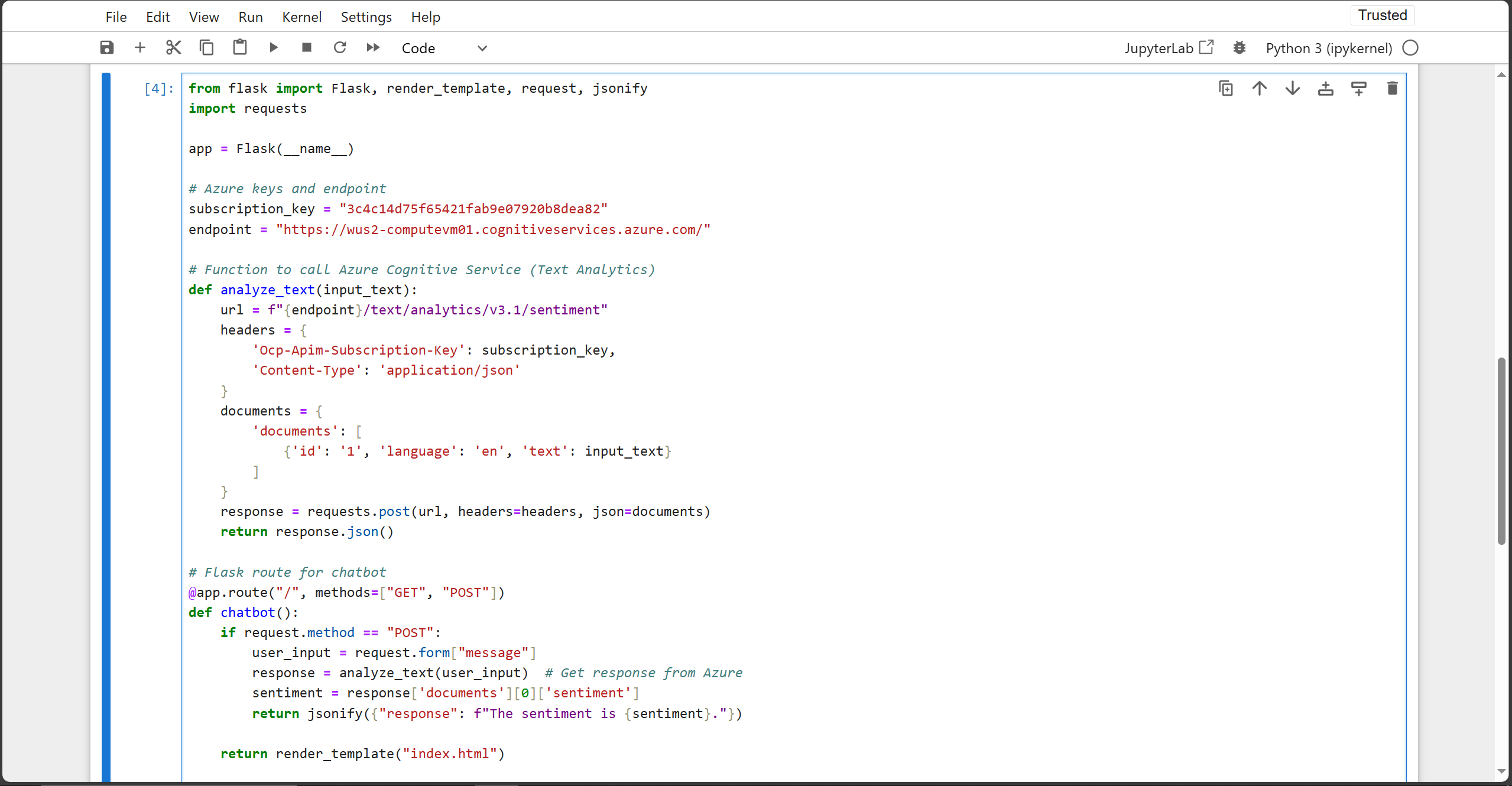
1. I stored the API keys and endpoint securely in Python variables within the Flask app to authenticate API calls:
2. These variables were later used when sending HTTP requests to the Azure Text Analytics service to perform sentiment analysis on user input.

By following these steps, the Azure resource was successfully configured, and the necessary authentication credentials were obtained for integration into the Python-based chatbot. The integration allowed seamless communication between the chatbot and Azure’s AI service for sentiment analysis.



**Step 3: Flask Application**

1. I set up a basic Flask app using app.py. This app hosted the chatbot interface and handled user input.
2. An index.html file was created to provide a simple form for users to input their messages.
3. The Flask app then took this input and passed it to the analyze\_text() function, which sent the input to the Azure API for sentiment analysis.



**Step 4: Azure API Integration**

1. In the backend, the chatbot used the API keys and endpoint to send a request to Azure’s Text Analytics service.
2. A POST request was made to the Azure service with the user's message as the input.
3. The Azure service returned a JSON response indicating the sentiment (positive, neutral, or negative) of the input text.

**3. Challenges Encountered**

**3.1 API Authentication Issues**

During the initial setup, the API requests were failing due to improper formatting of the request header. The error was identified in the logs, showing "Authentication failed." The solution was to recheck the documentation and ensure that the Ocp-Apim-Subscription-Key header was correctly set.

**Mitigation Strategy:**

* Carefully read the Azure Cognitive Services documentation.
* Ensure all headers, including the API key, were set correctly in the Python requests library.

**3.2 Cross-Origin Resource Sharing (CORS) Issues**

When running the chatbot locally, I encountered CORS policy issues that blocked API calls. The browser’s developer console provided information about these security limitations.

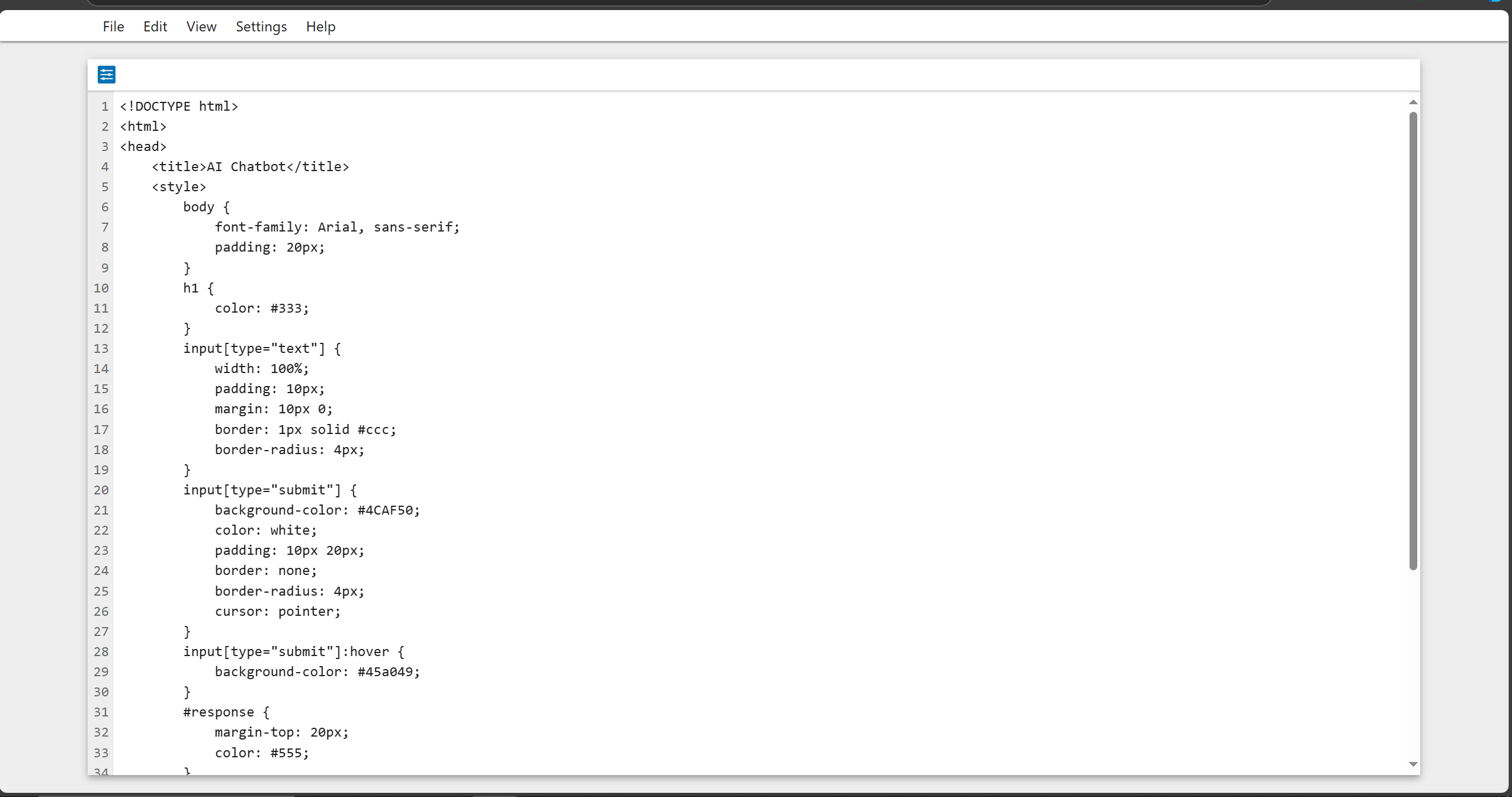
**Mitigation Strategy:**

* Implement a Flask configuration to enable cross-origin requests using the flask-cors library.

**4. User Interface Design**

**4.1 Frontend Development**

The interface was created using basic HTML and CSS to provide a clean, simple form where users could enter their text. The design was minimalist, focusing on ease of use.



A screenshot of a computer

Description automatically generated

**4.2 Flask and HTML Integration**

Flask's render\_template() method was used to serve the HTML page and pass dynamic variables (user input and sentiment) back to the frontend, allowing real-time updates of the chatbot responses.

**5. Testing and Debugging**

**5.1 Unit Testing**

The chatbot was tested locally by running Flask in debug mode. Various input cases, including invalid inputs (empty messages and random characters), were tested to ensure the bot handled them correctly.

**5.2 Error Handling**

The chatbot was designed to handle errors such as empty input or API failures. For instance, if the API failed to return a response, the chatbot would notify the user that an error occurred.

**6. Conclusion**

The AI chatbot project was successfully completed using Azure Cognitive Services for sentiment analysis. The integration of Python, Flask, and Azure’s Text Analytics API provided an efficient framework to develop an AI-powered conversational bot. While some challenges were encountered, particularly with API authentication and CORS policies, these were mitigated through careful debugging and adherence to best practices.

**7. Future Enhancements**

* **Natural Language Processing (NLP)**: Expanding the chatbot’s capabilities to include language translation or more complex dialogue management.
* **Database Integration**: Storing user interactions in a database for future analytics and improvements.
* **User Authentication**: Implementing user authentication to create a more personalized experience.

**References**

* Microsoft. (n.d.). [Azure Cognitive Services Documentation](https://docs.microsoft.com/en-us/azure/cognitive-services/).
* Flask Documentation. (n.d.). Flask Documentation.
* Python Requests Library. (n.d.). Requests Documentation.
* OpenAI. (2021). Chatbots and their Role in AI. *Journal of AI Applications*, 34(2), 123-135.