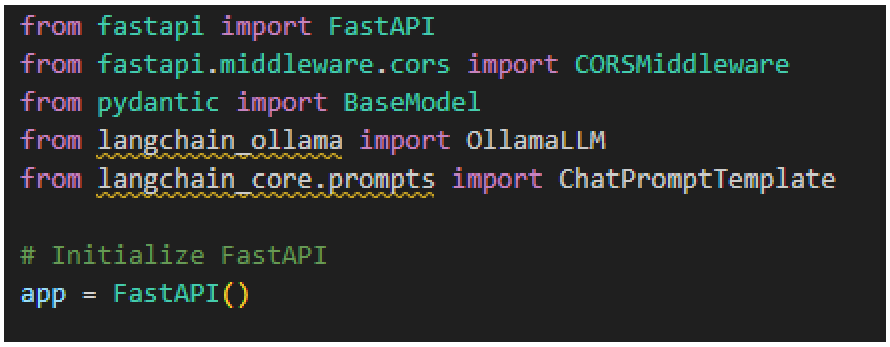
CHATBOT DOCUMENTATION

1. Project Setup and Initialization

main.py

- **Purpose:** This file sets up the **FastAPI** server, initializes the chatbot model, defines endpoints for chat and reset actions, and manages the conversation context.
- **Key Imports and Initial Setup:**

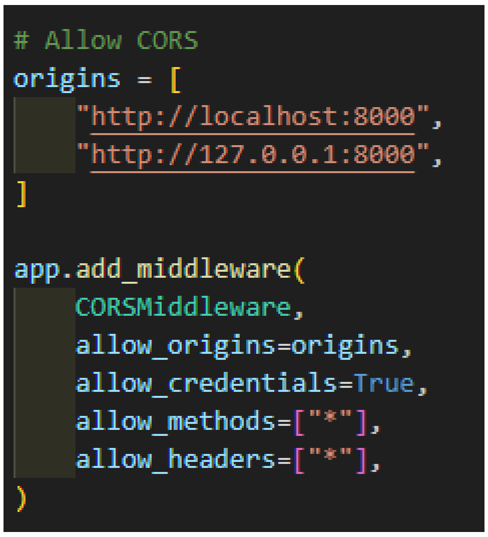


```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from langchain_ollama import OllamaLLM
from langchain_core.prompts import ChatPromptTemplate

# Initialize FastAPI
app = FastAPI()
```

Explanation:

- **FastAPI:** Main web framework.
- **CORSMiddleware:** Enables front-end communication by allowing Cross-Origin Resource Sharing.
- **BaseModel:** Defines expected input data structure.
- **OllamaLLM and ChatPromptTemplate:** Used to create and configure the chatbot model.

2. CORS Configuration

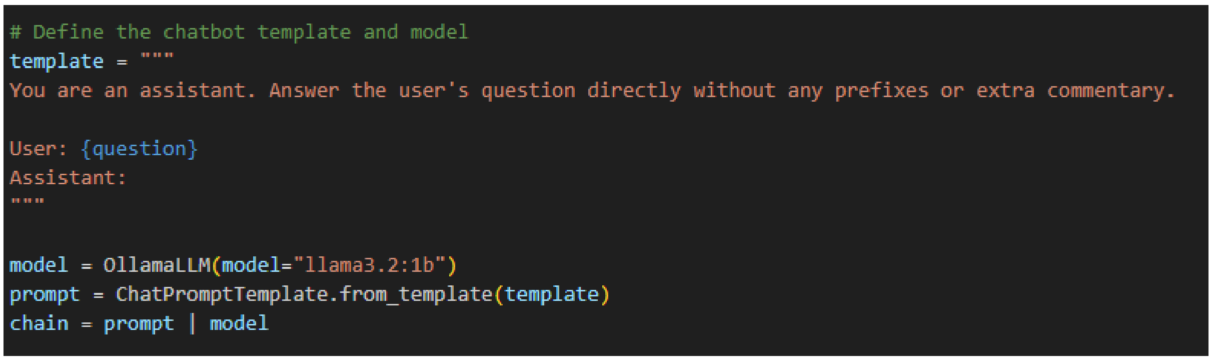


```python
# Allow CORS
origins = [
    "http://localhost:8000",
    "http://127.0.0.1:8000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Explanation:

- Defines permitted front-end origins.
- Enables credential sharing, allowing all HTTP methods and headers for full interaction between front and back end.

3. Define Chatbot Model and Prompt Template

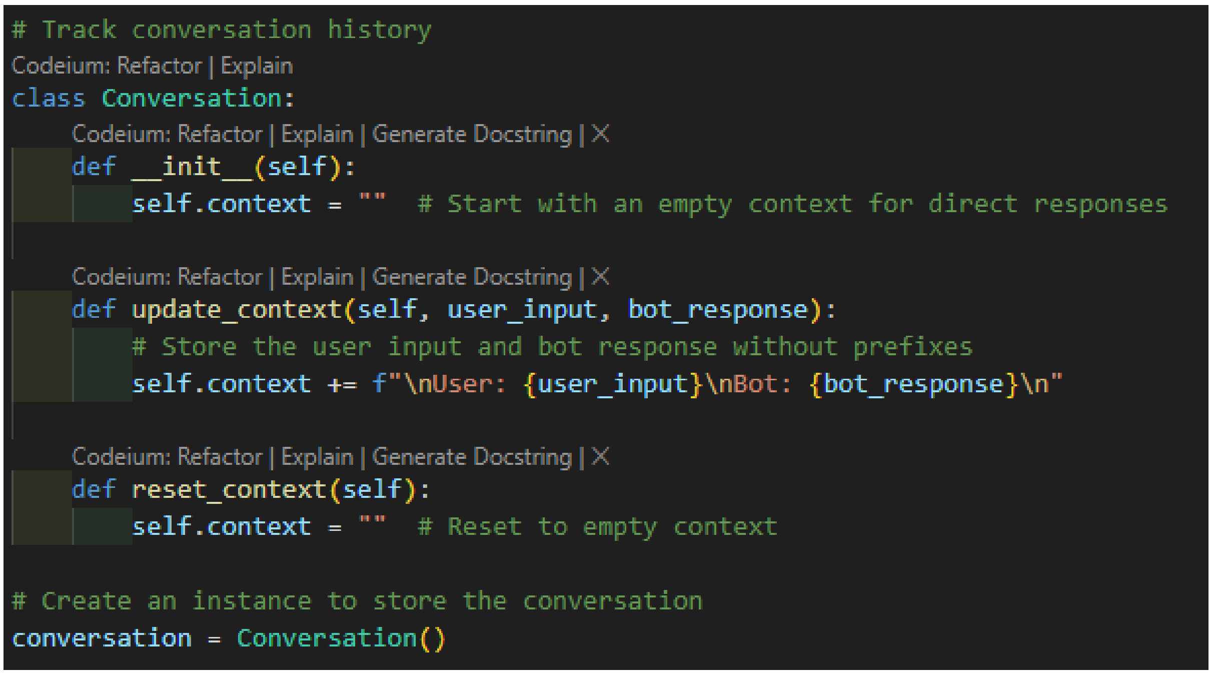


```python
# Define the chatbot template and model
template = """
You are an assistant. Answer the user's question directly without any prefixes or extra commentary.

User: {question}
Assistant:
"""

model = OllamaLLM(model="llama3.2:1b")
prompt = ChatPromptTemplate.from_template(template)
chain = prompt | model
```

Explanation:

- The template variable specifies the format for user-bot conversations.
- **OllamaLLM** loads a specific model (**llama3.2:1b**) for response generation.
- The prompt and model are combined into a chain, which processes user input and generates a bot response.

4. Conversation Context Management



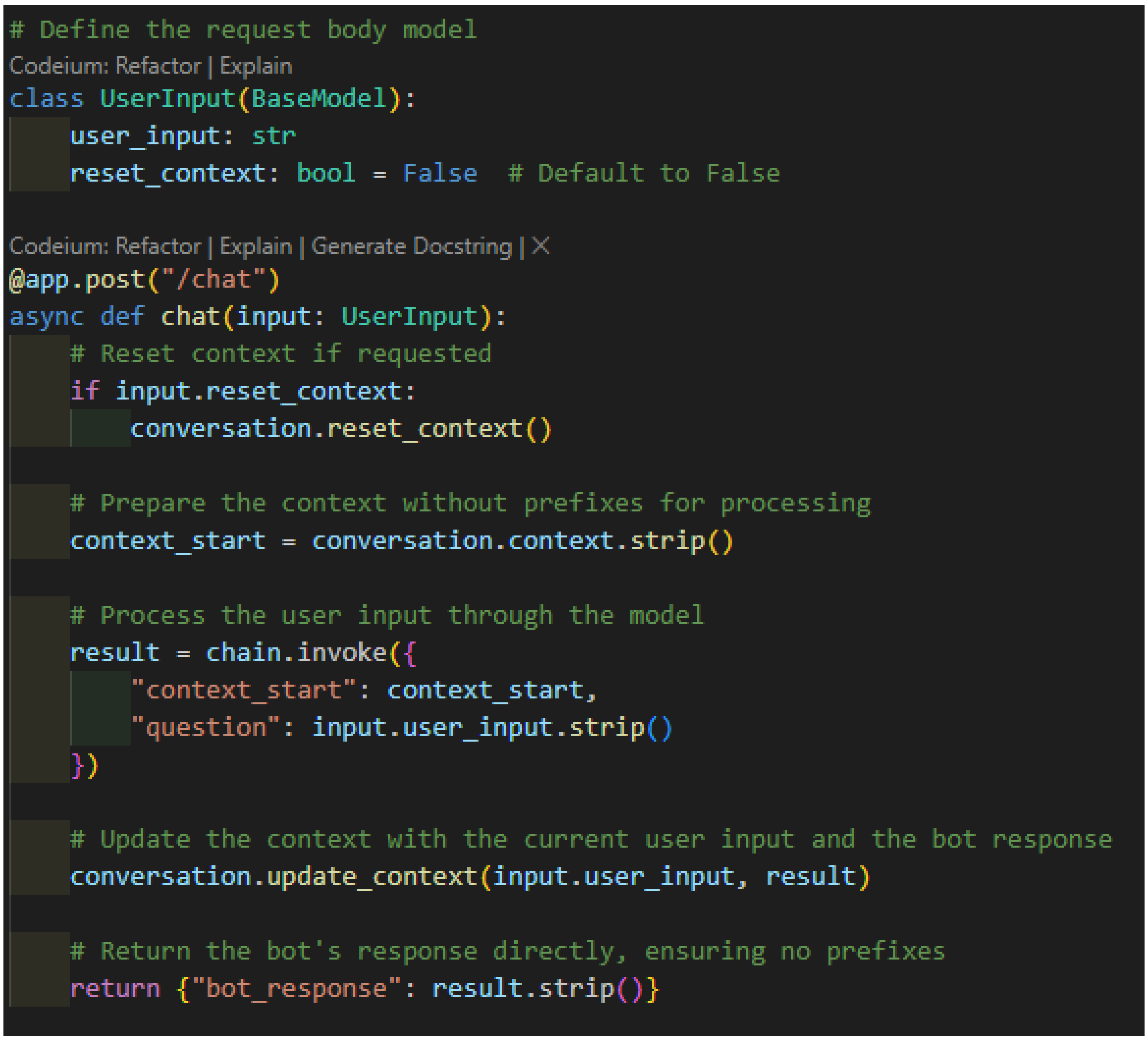
```python
# Track conversation history
Codeium: Refactor | Explain
class Conversation:
    Codeium: Refactor | Explain | Generate Docstring | X
    def __init__(self):
        self.context = ""  # Start with an empty context for direct responses

    Codeium: Refactor | Explain | Generate Docstring | X
    def update_context(self, user_input, bot_response):
        # Store the user input and bot response without prefixes
        self.context += f"\nUser: {user_input}\nBot: {bot_response}\n"

    Codeium: Refactor | Explain | Generate Docstring | X
    def reset_context(self):
        self.context = ""  # Reset to empty context

# Create an instance to store the conversation
conversation = Conversation()
```

Explanation:

- **Purpose:** To track and manage the conversation history.
- **update_context:** Appends each interaction to self.context for continuity in responses.
- **reset_context:** Clears the conversation history, allowing a fresh start.

5. Define API Endpoints

Chat Endpoint

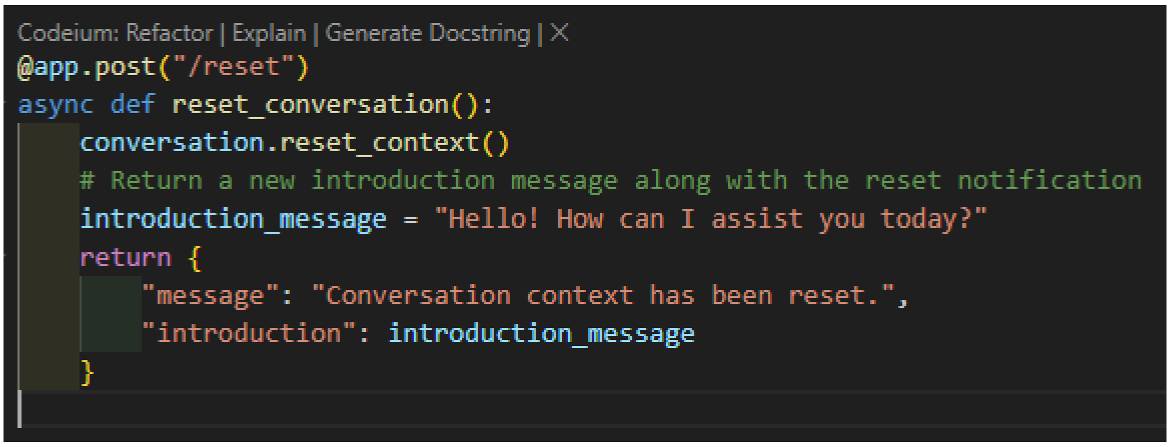


```
# Define the request body model
Codeium: Refactor | Explain
class UserInput(BaseModel):
    user_input: str
    reset_context: bool = False  # Default to False

Codeium: Refactor | Explain | Generate Docstring | X
@app.post("/chat")
async def chat(input: UserInput):
    # Reset context if requested
    if input.reset_context:
        conversation.reset_context()

    # Prepare the context without prefixes for processing
    context_start = conversation.context.strip()

    # Process the user input through the model
    result = chain.invoke({
        "context_start": context_start,
        "question": input.user_input.strip()
    })

    # Update the context with the current user input and the bot response
    conversation.update_context(input.user_input, result)

    # Return the bot's response directly, ensuring no prefixes
    return {"bot_response": result.strip()}
```

Explanation:

- **UserInput model:** Defines the JSON structure with **user_input** for user messages and **reset_context** to control context resetting.
- **chat function:**
  - If **reset_context** is True, clears the context by calling **conversation.reset_context()**.
  - Sends the current context and user input to **chain.invoke**, generating a bot response.
  - Updates the conversation with the new interaction and returns the response as JSON.

Reset Endpoint



```python
@app.post("/reset")
async def reset_conversation():
    conversation.reset_context()
    # Return a new introduction message along with the reset notification
    introduction_message = "Hello! How can I assist you today?"
    return {
        "message": "Conversation context has been reset.",
        "introduction": introduction_message
    }
```

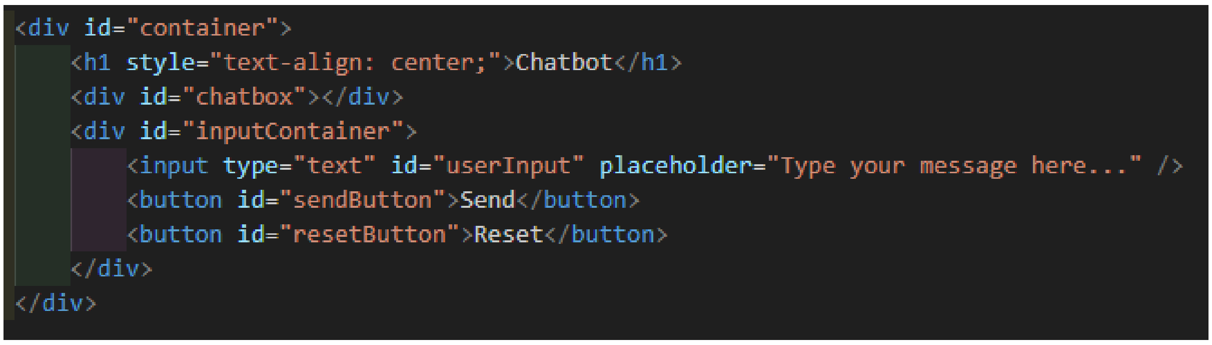
Explanation:

- Clears the conversation context.
- Returns a reset confirmation and introductory message to start the conversation afresh.

6. Frontend

Index.html

HTML Structure:

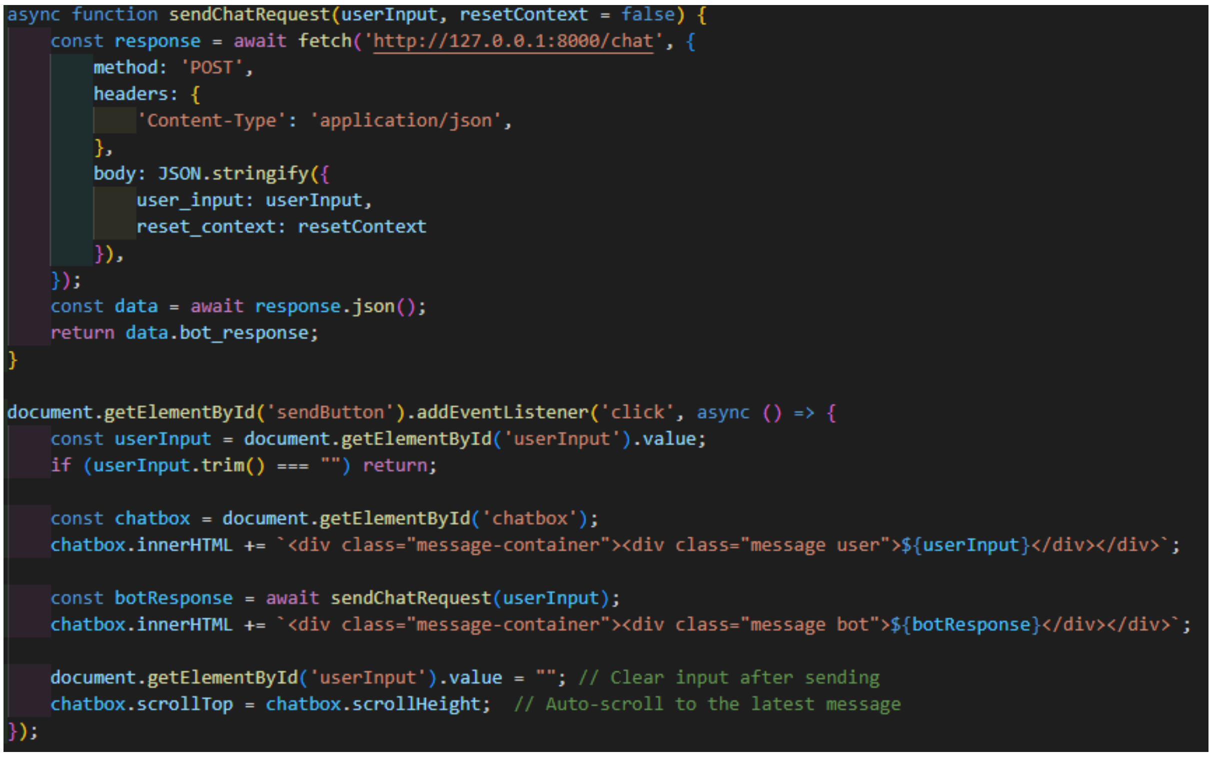


```html
<div id="container">
    <h1 style="text-align: center;">Chatbot</h1>
    <div id="chatbox"></div>
    <div id="inputContainer">
        <input type="text" id="userInput" placeholder="Type your message here..." />
        <button id="sendButton">Send</button>
        <button id="resetButton">Reset</button>
    </div>
</div>
```

Explanation:

- **container:** Main wrapper for the chatbot UI.
- **chatbox:** Displays messages exchanged between the user and bot.
- **inputContainer, userInput, sendButton, and resetButton:** Elements for message input, sending, and resetting.

JavaScript Logic: Sending and Displaying Messages

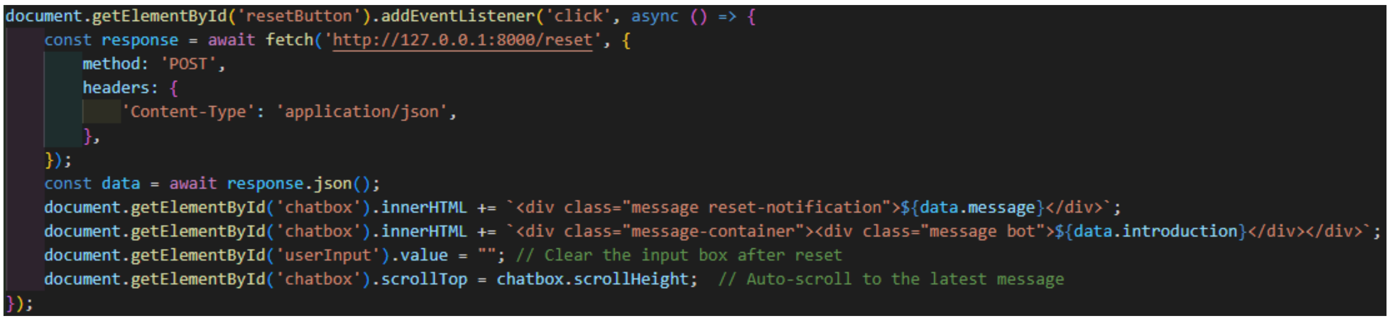


```javascript
async function sendChatRequest(userInput, resetContext = false) {
    const response = await fetch('http://127.0.0.1:8000/chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            user_input: userInput,
            reset_context: resetContext
        }),
    });
    const data = await response.json();
    return data.bot_response;
}

document.getElementById('sendButton').addEventListener('click', async () => {
    const userInput = document.getElementById('userInput').value;
    if (userInput.trim() === "") return;

    const chatbox = document.getElementById('chatbox');
    chatbox.innerHTML += `<div class="message-container"><div class="message user">${userInput}</div></div>`;

    const botResponse = await sendChatRequest(userInput);
    chatbox.innerHTML += `<div class="message-container"><div class="message bot">${botResponse}</div></div>`;

    document.getElementById('userInput').value = ""; // Clear input after sending
    chatbox.scrollTop = chatbox.scrollHeight;  // Auto-scroll to the latest message
});
```

Explanation:

- **sendChatRequest function:** Sends the user's message and the **reset_context** flag to the FastAPI server, receiving the bot response.
- **Send Button Event Listener:**
  - Gets user input and displays it in chatbox.
  - Sends the input to the backend and displays the bot's reply.
  - Clears the input field and scrolls to the latest message.

Reset Button and Reset Event



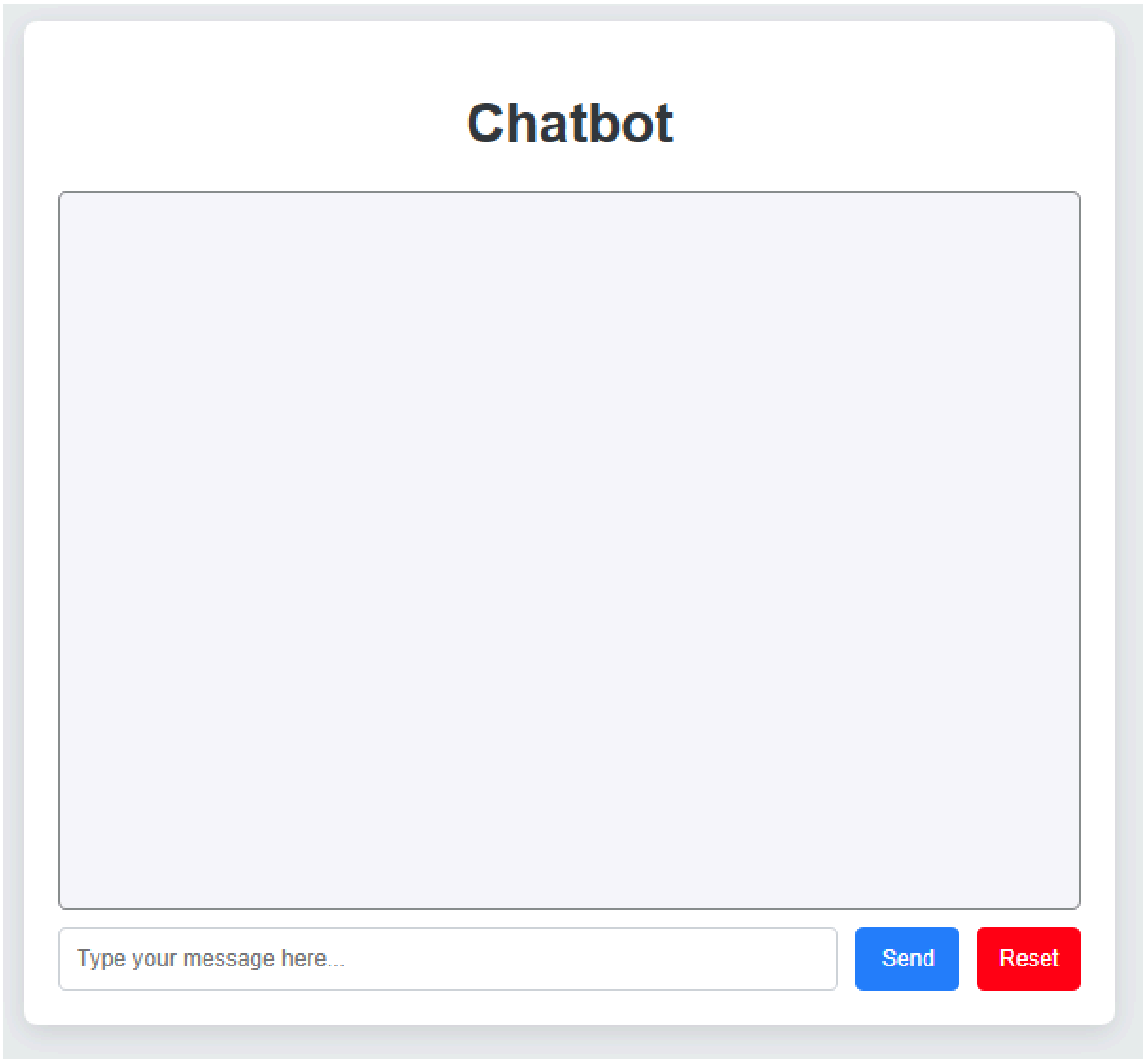
```javascript
document.getElementById('resetButton').addEventListener('click', async () => {
    const response = await fetch('http://127.0.0.1:8000/reset', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
    });
    const data = await response.json();
    document.getElementById('chatbox').innerHTML += `<div class="message reset-notification">${data.message}</div>`;
    document.getElementById('chatbox').innerHTML += `<div class="message-container"><div class="message bot">${data.introduction}</div></div>`;
    document.getElementById('userInput').value = ""; // Clear the input box after reset
    document.getElementById('chatbox').scrollTop = chatbox.scrollHeight;  // Auto-scroll to the latest message
});
```

Explanation:

- Fetches a reset message and introduction from the /reset endpoint.
- Displays reset confirmation and introductory message in chatbox.
- Clears the input and scrolls to the latest message.

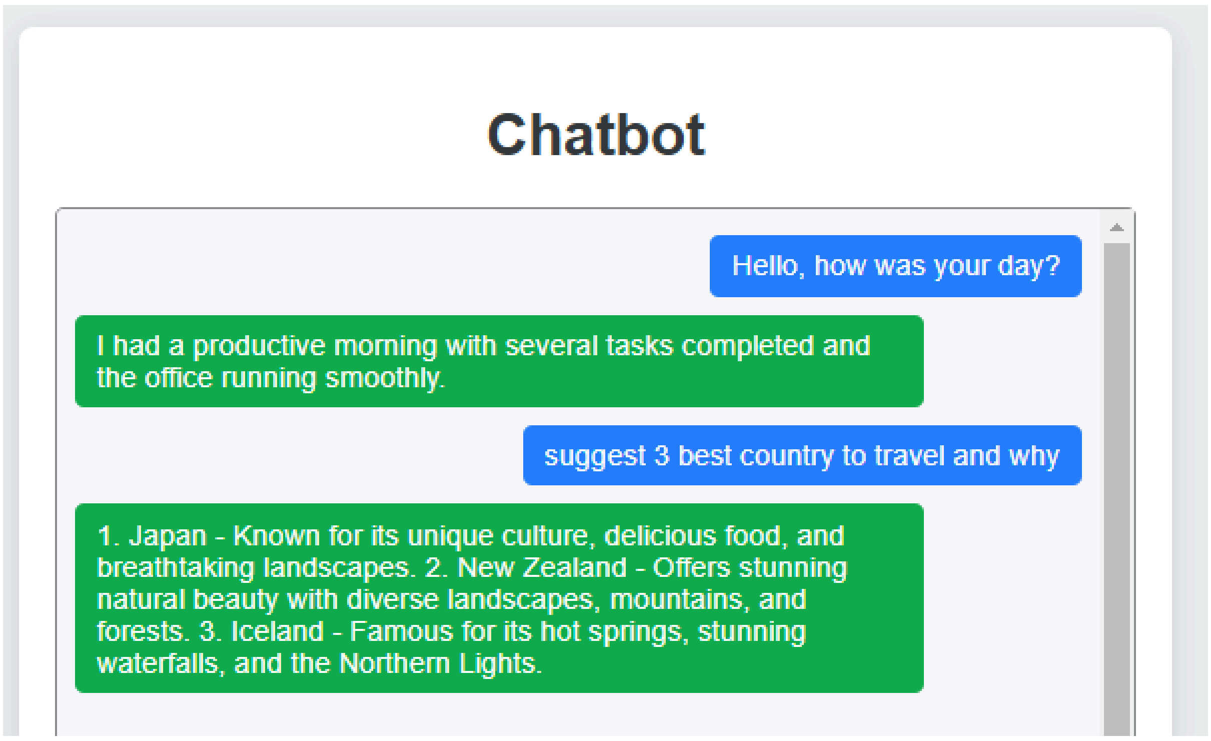
Flow Summary

User Sends Message:



- User types a message, clicks **"Send".**
- JavaScript displays the message, sends it to /chat, and displays the bot’s reply.

<u>Context and Response:</u>



Chatbot

Hello, how was your day?

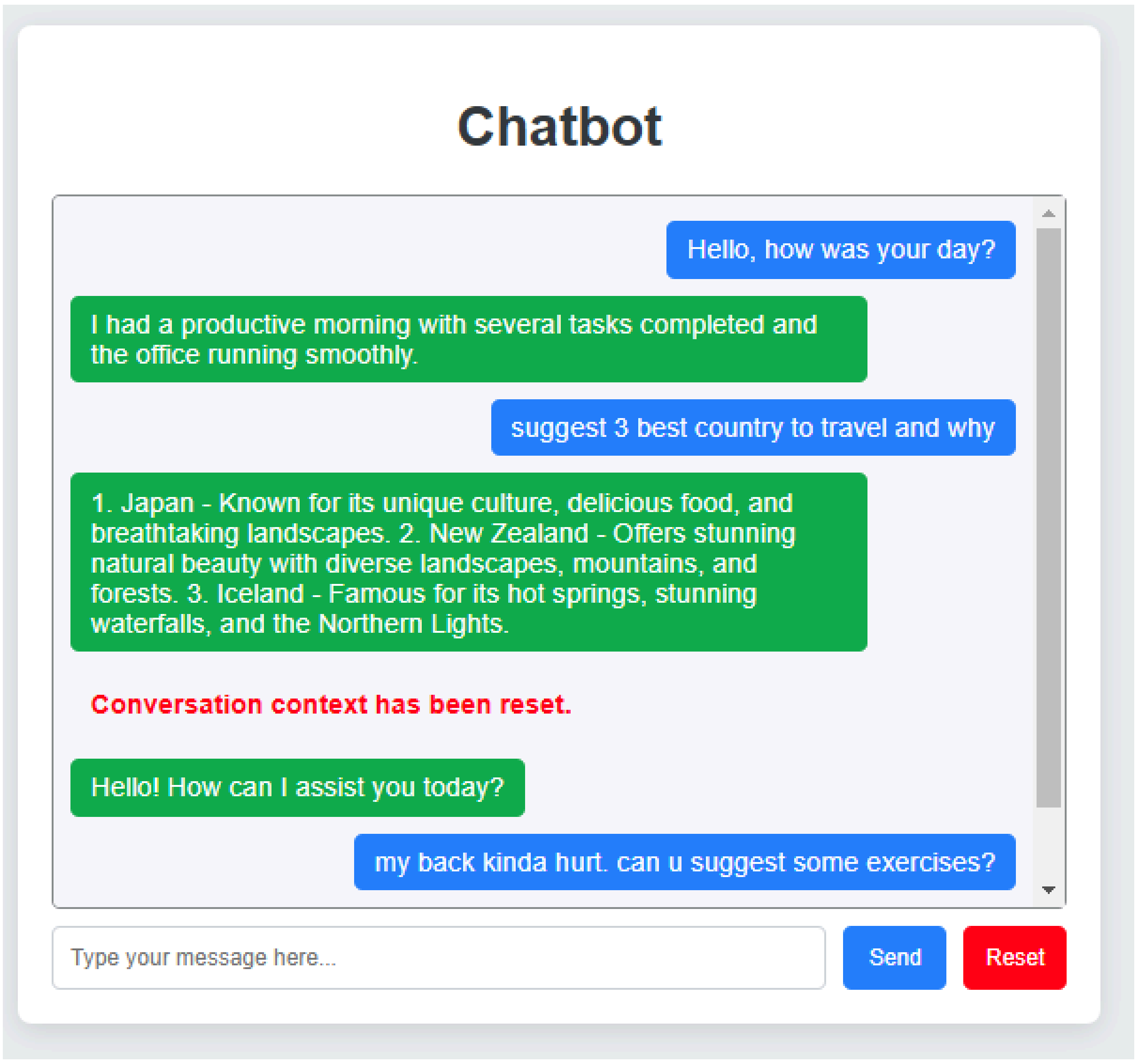
I had a productive morning with several tasks completed and the office running smoothly.

suggest 3 best country to travel and why

1. Japan - Known for its unique culture, delicious food, and breathtaking landscapes. 2. New Zealand - Offers stunning natural beauty with diverse landscapes, mountains, and forests. 3. Iceland - Famous for its hot springs, stunning waterfalls, and the Northern Lights.

- FastAPI receives the message, optionally clears context.
- Adds user input and response to context.
- Model processes input within current context, returns bot reply.

Reset Action:



Chatbot

Hello, how was your day?

I had a productive morning with several tasks completed and the office running smoothly.

suggest 3 best country to travel and why

1. Japan - Known for its unique culture, delicious food, and breathtaking landscapes. 2. New Zealand - Offers stunning natural beauty with diverse landscapes, mountains, and forests. 3. Iceland - Famous for its hot springs, stunning waterfalls, and the Northern Lights.

Conversation context has been reset.

Hello! How can I assist you today?

my back kinda hurt. can u suggest some exercises?

- Clicking the "**Reset"** button clears context on the backend.
- Displays reset confirmation and introductory message on the frontend.