Project Report: AI-Powered Chatbot Development

1. Introduction

The objective of this project was to develop an AI-powered chatbot using HTML, CSS, JavaScript, Node.js, Express, and OpenAI's GPT API. The chatbot was designed to take user input, send it to the OpenAI API, and return responses in a chat interface. The project was completed within IBM Skills Network Labs.

---

2. Project Setup & Implementation

2.1 Setting Up the Environment

- Used IBM Skills Network Labs for development.
- Installed required Node.js dependencies: `express`, `dotenv`, and `openai`.

Command to install dependencies:
npm install express dotenv openai

---

3. Implementation

3.1 Task 1: Creating the User Interface

Created an `index.html` file in the `public` directory. This file contains the chatbot's front-end UI.

Code - `index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chatbot</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="chat-container">
        <div class="header">
            <img src="chat.png" alt="Logo" class="logo">
            <h2 class="name">Chat Window</h2>
```

```
        </div>
        <div class="chat-log" id="chat-log"></div>
        <div class="input-container">
            <input type="text" id="user-input" placeholder="Ask me anything...">
            <button onclick="sendMessage()">Send</button>
        </div>
    </div>
    <script src="main.js"></script>
</body>
</html>
```

3.2 Task 2: Styling the Chatbot

A `style.css` file was created to style the chatbot interface.

Code - `style.css`

```
.chat-container {
    max-width: 600px;
    margin: 20px auto;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    overflow: hidden;
    display: flex;
    flex-direction: column;
    height: 80vh;
    background: linear-gradient(to bottom, #1e5799, #2989d8);
}
.input-container input {
    flex: 1;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
}
.input-container button {
    margin-left: 10px;
    padding: 8px 16px;
    border: none;
    border-radius: 5px;
    background-color: #4CAF50;
    color: #fff;
    font-size: 16px;
}
```

3.3 Task 3: Implementing JavaScript Functionality

A `main.js` file was created to handle user interactions.

Code - `main.js`

```
const chatLog = document.getElementById('chat-log');
const userInput = document.getElementById('user-input');

function sendMessage() {
    const message = userInput.value;
    displayMessage('user', message);
    getChatbotResponse(message);
    userInput.value = '';
}

function displayMessage(sender, message) {
    const messageElement = document.createElement('div');
    messageElement.classList.add('message', sender);
    messageElement.innerHTML = `<p>${message}</p>`;
    chatLog.appendChild(messageElement);
}

async function getChatbotResponse(userMessage) {
    const response = await fetch('/getChatbotResponse', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userMessage })
    });
    const data = await response.json();
    displayMessage('chatbot', data.chatbotResponse);
}
```

3.4 Task 4: Setting Up the Server

A Node.js server was created using Express to handle API requests.

Code - `server.js`

```
const express = require('express');
const path = require('path');
const { OpenAIAPI } = require('./openai');
const app = express();
const port = process.env.PORT || 3000;

app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());
```

```
app.post('/getChatbotResponse', async (req, res) => {
    const userMessage = req.body.userMessage;
    const chatbotResponse = await OpenAIAPI.generateResponse(userMessage);
    res.json({ chatbotResponse });
});

app.listen(port, () => console.log(`Server is running on port ${port}`));
```

3.5 Task 5: Integrating OpenAI API

A `openai.js` file was created to handle API requests to OpenAI.

Code - `openai.js`

```
const fetch = require('node-fetch');
require('dotenv').config();

class OpenAIAPI {
    static async generateResponse(userMessage) {
        const apiKey = process.env.OPENAI_API_KEY;
        const response = await fetch('https://api.openai.com/v1/chat/completions', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${apiKey}`
            },
            body: JSON.stringify({
                model: "gpt-3.5-turbo",
                messages: [{ role: "user", content: userMessage }],
                max_tokens: 150
            })
        });
        const data = await response.json();
        return data.choices[0].message.content;
    }
}
module.exports = { OpenAIAPI };
```

---

4. Challenges & Solutions

Issue 1: Module Not Found Error

Problem: When running `node server.js`, it could not find the `openai` module. **Solution:** Installed missing dependencies with `npm install dotenv openai` and ensured `server.js` was in the correct directory.

Issue 2: API Key Not Recognized

Problem: The chatbot was not returning responses from OpenAI. **Solution:** Used `dotenv` to load API keys properly and placed them in an `.env` file.

Issue 3: UI Not Updating

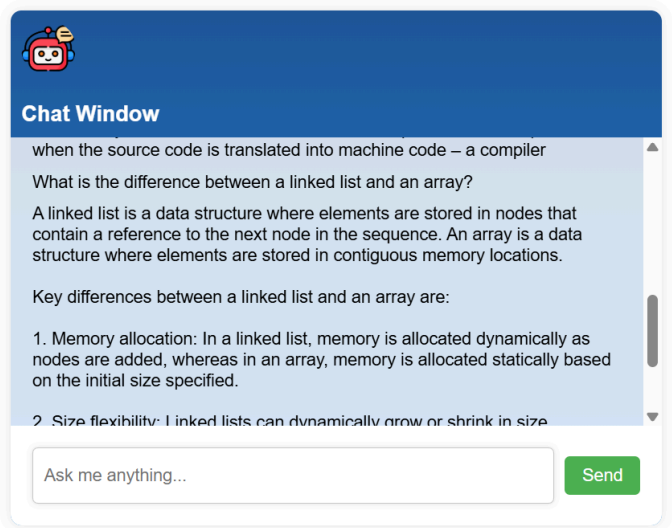
Problem: Messages were not appearing in the chat window. **Solution:** Debugged and corrected event listener logic in `main.js`.

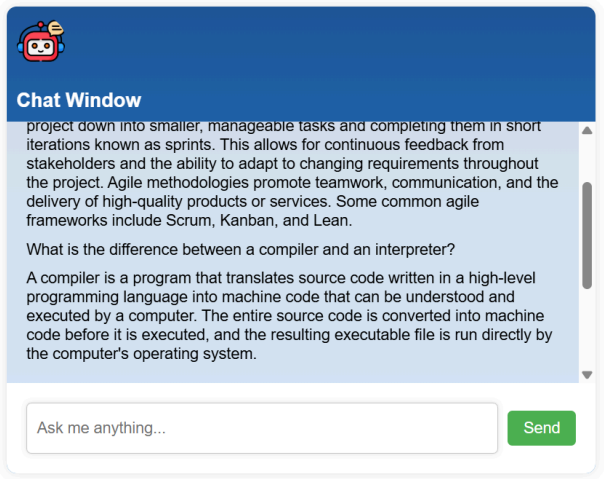
---

5. Final Output

The chatbot successfully responds to user queries. Example output:

User Input: What is the difference between a compiler and an interpreter? **Chatbot Response:** A compiler translates source code into machine code before execution, whereas an interpreter executes code line by line.





---

6. Conclusion

The project successfully implemented a chatbot that integrates OpenAI’s API, handles user input, and provides meaningful responses.

🎯 Project Completed Successfully!