Lab: Building a Basic Chatbot Using ChatGPT

Estimated Time: 90 minutes

In this lab, you will get familiar with the process of developing a basic chatbot capable of responding to software development-related questions. We'll be utilizing OpenAI's API along with Node.js using the Express.js framework to build this interactive chatbot. Let's get started!

Learning Objectives:

After completing this exercise,you should be able to perform the following tasks:

- Build a basic chatbot using the free Skills Network-provided OpenAI APIs.
- Converse with the chatbot to ask any questions related to software development and receive the appropriate response.

  Please note Generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.

Pre-requisites

Understanding the basics of Node.js and Express.js

- Node and Express are two technologies that are commonly used to create web applications.
- Node is a runtime environment that allows you to run JavaScript code on the server side, without a browser.
- Express is a framework that provides a set of features and tools to simplify the development of web applications with Node.

Some of the benefits of using Node and Express are:

- They are fast and scalable, as they use an event-driven, non-blocking I/O model that can handle many concurrent requests.
- They are flexible and modular, as they allow you to use various libraries and middleware to customize your application according to your needs.
- They are easy to learn and use, as they are based on JavaScript, which is a popular and widely used programming language.

Task 1: Set Up Your Project

- Here, we are setting up a Node.js project using the Express.js framework.
- Node.js is a JavaScript runtime that allows you to run JavaScript on the server, while Express.js is a web application framework for Node.js.
- Follow the below instructions and refer to the screenshots for creating a Node.js Project.

Step 1: Open a terminal, click on "Terminal", and then select the "New Terminal" option.

Step 2: Create a new directory for your project named `software-dev-chatbot`, navigate to the directory, and initialize a Node.js project using the commands below.

```
mkdir software-dev-chatbot
cd software-dev-chatbot
npm init -y
```

Step 3: Install express and openai dependencies by executing the following command.

```
npm install express openai
```

Step 4: Create a new directory named `public` inside a project directory using the below command.

```
mkdir public
```

Please Note: It is important to understand that the lab environment is ephemeral; it only exists for a short duration and will be destroyed afterward. Therefore, it is essential to download the project directory so that it can be reopened in a new lab environment whenever needed.

To download the project directory, right-click on the directory and choose the “Download” option, as shown in the screenshot below. This will download a zipped folder.

Additionally, please be aware that this environment is shared. Avoid storing any personal information, usernames, passwords, or access tokens in this environment for any purpose.

Task 2: Create the user interface for your chatbot using HTML and CSS

- Create an `index.html` file within the public directory, right-click on the `public` folder, and select the "New File" option, as shown in the screenshot below. This file will function as the user interface for your chatbot.

• Inside the HTML file, we are using the following elements:

a. Text input element -> To take user input

b. Submit button -> To submit the question from the user

c. Text area -> where the chatbot's responses will be displayed.

- Here are the steps to set a chatbot logo of your choice:

1. Download an image from the internet for the chatbot logo and name it `chat.png`
2. Right-click on the public folder, and select the "Upload Files" option, as shown in the screenshot below.
3. Choose the downloaded image and click "OK" to upload it.

- The styling is applied through an external CSS file (style.css)
- The implementation of chatbot logic is done through an external JavaScript file named `main.js`
- You can type a message, and click the `Send` button, to interact with the chatbot.

Insert the provided code into the newly created HTML file.

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Software Dev Chatbot</title>
</head>
<body>
    <div class="main-container">
        <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>
    </div>
    <script src="main.js"></script>
</body>
</html>
```

- Create a `style.css` file in the public directory and insert the following code into the CSS file.

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;
    justify-content: space-between;
    height: 80vh;
    position: relative;
    background: linear-gradient(to bottom, #1e5799, #2989d8);
}
.logo {
    width: auto;
    height: 50px;
    margin-left: 10px;
    margin-top: 10px;
    margin-bottom: 10px;
    border-radius: 50px;
}
.name {
    font-size: 20px;
    font-weight: bold;
    margin: 10px;
    color: #fff;
    margin-left: 10px;
    margin-right: 10px;
}
.chat-window {
    flex-grow: 1;
    display: flex;
    flex-direction: column;
    align-items: flex-end;
    padding: 20px;
    background-color: #fff;
overflow-y: auto;
```

```
        }
        .input-container {
            display: flex;
            align-items: center;
            padding: 20px;
            background-color: #fff;
        }
        input[type="text"] {
            width: 100%;
            padding: 10px;
            border: none;
            border-radius: 5px;
            outline: none;
            font-size: 16px;
            box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        }
        .chat-log {
            height: 400px;
            padding: 20px;
            overflow-y: auto;
            background-color: rgba(255, 255, 255, 0.8);
        }
        /* Added space between user prompts */
        .chat-log p {
            margin: 10px 0;
        }
        .input-container input[type="text"] {
            flex: 1;
            height: 40px;
            padding: 5px 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 16px;
            outline: none;
        }
        .input-container button {
            margin-left: 10px;
            padding: 8px 16px;
        border: none;
        border-radius: 5px;
        background-color: #4CAF50;
        color: #fff;
         font-size: 16px;
         cursor: pointer;
        }
        .input-container button:hover {
            background-color: #45a049;
        }
        html, body {
```

```
        margin: 0;
        padding: 0;
        font-family: Arial, sans-serif;
    }
```

Launch live server

You can check the user-interface of your chatbot by launching the live server.

- Please right-click on the index.html and select the "Open with Live server" option.

- Click on the "Skills Network" button on the left, which will open the "Skills Network Toolbox"
- Next, select "Launch Application" and enter the port as 5500 and launch the server. Please refer to the screenshot below.

- Select the `software-dev-chatbot` folder and then select the `public` folder to view the HTML page as shown in the screenshots below.

- You should be able to see the chat interface as below:

Task 3: Create a JavaScript file to implement the functionality of the Chatbot

- Create a `main.js` file in the public directory to implement the functionality.
- In the `main.js` file, the JavaScript code is designed to handle user input and initiate an API call to the OpenAI API.
- An event listener is attached to the submit button so that when the user clicks it, the code is executed.

- In the event listener callback function, we retrieve the user's input from the text input element and display their message in the chat log.
- Next, we make an API request to the server with the user's message using the `fetch()` function, specifying the appropriate endpoint on the server to receive this request.
- After receiving the response from the OpenAI API, we send it back to the client and display the chatbot's response in the text area of the `index.html` file.
- We then append the response to the chat log on the front end, making the conversation visible on the UI (User Interface).

Insert the following code into the newly created main.js file.

main.js:

```
const chatLog = document.getElementById('chat-log');
const userInput = document.getElementById('user-input');
function sendMessage() {
    const message = userInput.value;
    // Display user's message
    displayMessage('user', message);
    // Call OpenAI API to get chatbot's response
    getChatbotResponse(message);
    // Clear user input
    userInput.value = '';
}
function displayMessage(sender, message) {
    const messageElement = document.createElement('div');
    messageElement.classList.add('message', sender);
    // Wrap the message in a <p> tag
    const messageParagraph = document.createElement('p');
    messageParagraph.innerText = message;
    // Append the <p> tag to the message element
    messageElement.appendChild(messageParagraph);
    chatLog.appendChild(messageElement);
}
function getChatbotResponse(userMessage) {
    // Make a request to your server with the user's message
    fetch('/getChatbotResponse', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ userMessage }),
    })
    .then(response => response.json())
    .then(data => {
        // Display chatbot's response
```

```
            displayMessage('chatbot', data.chatbotResponse);
        })
        .catch(error => console.error('Error:', error));
    }
```

Task 4: Create an Express Server and integrate OpenAI API

- Create a new file named `server.js` within your project directory (software-dev-chatbot)
- This file will be responsible for managing API requests and integrating with OpenAI.
- This code sets up a basic Express.js server that serves static files from a `public` directory.
- It includes a route for the root path ('/') to serve an HTML file.
- Additionally, there is a POST endpoint `/getChatbotResponse` that receives a user message, utilizes an OpenAI API wrapper (`OpenAIAPI`), generates a chatbot response using the OpenAI API, and sends the response back to the client.
- The server listens on a specified port 3000, and a message is logged to the console when the server is successfully running.

Insert the following code into the newly created server.js file.

server.js

```
process.env["NODE_TLS_REJECT_UNAUTHORIZED"]=0;
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.get('/', (req, res) => {
```

```
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
app.post('/getChatbotResponse', async (req, res) => {
    const userMessage = req.body.userMessage;
    // Use OpenAI API to generate a response
    const chatbotResponse = await OpenAIAPI.generateResponse(userMessage);
    // Send the response back to the client
    res.json({ chatbotResponse });
});
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Task 5: Create OpenAI API Module

- Create an `openai.js` file within the project directory to encapsulate the OpenAI API logic.
- This code facilitates communication with the OpenAI API, specifically the gpt-3.5-turbo Codex engine, allowing developers to generate responses based on user input through a secure configuration using an API key.

Insert the following code into the newly created openai.js file.

openai.js:

```
class OpenAIAPI {
    static async generateResponse(userMessage, conversationHistory = []) {
        const apiKey = process.env.OPENAI_API_KEY;
        const endpoint = 'https://api.openai.com/v1/chat/completions';
        const response = await fetch(endpoint, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${apiKey}`,
            },
```

```
            body: JSON.stringify({
                model: "gpt-4o-mini",
                messages: conversationHistory.concat([{ role: 'user', content: userMessage }]),
                max_tokens: 150
            }),
        });
        const responseData = await response.json();
        // Log the entire API response for debugging
        console.log('Response from OpenAI API:', responseData.choices[0].message);
        // Check if choices array is defined and not empty
        if (responseData.choices && responseData.choices.length > 0 && responseData.choices[0].message) {
            return responseData.choices[0].message.content;
        } else {
            // Handle the case where choices array is undefined or empty
            console.error('Error: No valid response from OpenAI API');
            return 'Sorry, I couldn\'t understand that.';
        }
    }
}
module.exports = { OpenAIAPI };
```

Task 6: Run Your Application

- The Project folder structure should be organized as shown in the screenshot below.

Follow the steps below to run the application

Step 1: Run your Express server by executing the below command in the Terminal

```
node server.js
```

Step 2: Click on the "Skills Network" button on the left, which will open the "Skills Network Toolbox"
Next, select "Launch Application" and enter the port as 3000 and launch the server. Please refer the screenshot below.

You should be able to see the chat interface as below:

Step 3: When you enter a question and click `Send` the chatbot will respond to your question using the OpenAI API.

Please refer to the screenshot below, where the question "What is agile methodology?" is posed, and the chatbot has provided an appropriate response.

Step 4: You can submit the following software development-related questions one at a time to the chatbot by entering them in the `Ask me anything` text box.

Click `Send` to receive a response from the chatbot.

Q1. What is the difference between a compiler and an interpreter?

Your output may be similar to the responses shown below:

Q2. What is the difference between a stack and a queue?

Your output may be similar to the responses shown below:

Q3. What is the difference between a linked list and an array?

Your output may be similar to the responses shown below:

Note: In case you want to use your own API Key instead of the lab enviornment's free access then you need to make a few changes to the configuration and openai.js file. Also, to. run the code outside of Skills network labs you will need to add your own API key.

▶ Click here to see the configuration changes:

Summary:

This hands-on lab is designed to assist in the creation of a basic chatbot proficient in responding to queries related to software development. The project uses the free Skills Network-provided OpenAI APIs, along with Node.js and the Express.js framework.

Congratulations! You have completed the lab on Building a Basic Chatbot using ChatGPT.

Author(s)

Rajashree Patil
Ritika Joshi