AI_Phase4: Flask Web Development

This phase covers the development of the Flask web application for interacting with our chatbot. Here's how we can make it happen:

1. Directory Structure:

Let's begin by ensuring our project directory is structured as follows:

```
LearnProg chatbot/
 ├── intents.json
 ├── tokenizer/
 |   ├── tokenizer.pickle
 ├── label_encoder/
 |   ├── label_encoder.pickle
 ├── app.py
 ├── chat_model/
 ├── train_chatbot.py
 ├── simple_chatbot.py
 ├── static/
 |   ├── style.css
 |   ├── script.js
 ├── templates/
 |   ├── index.html
```

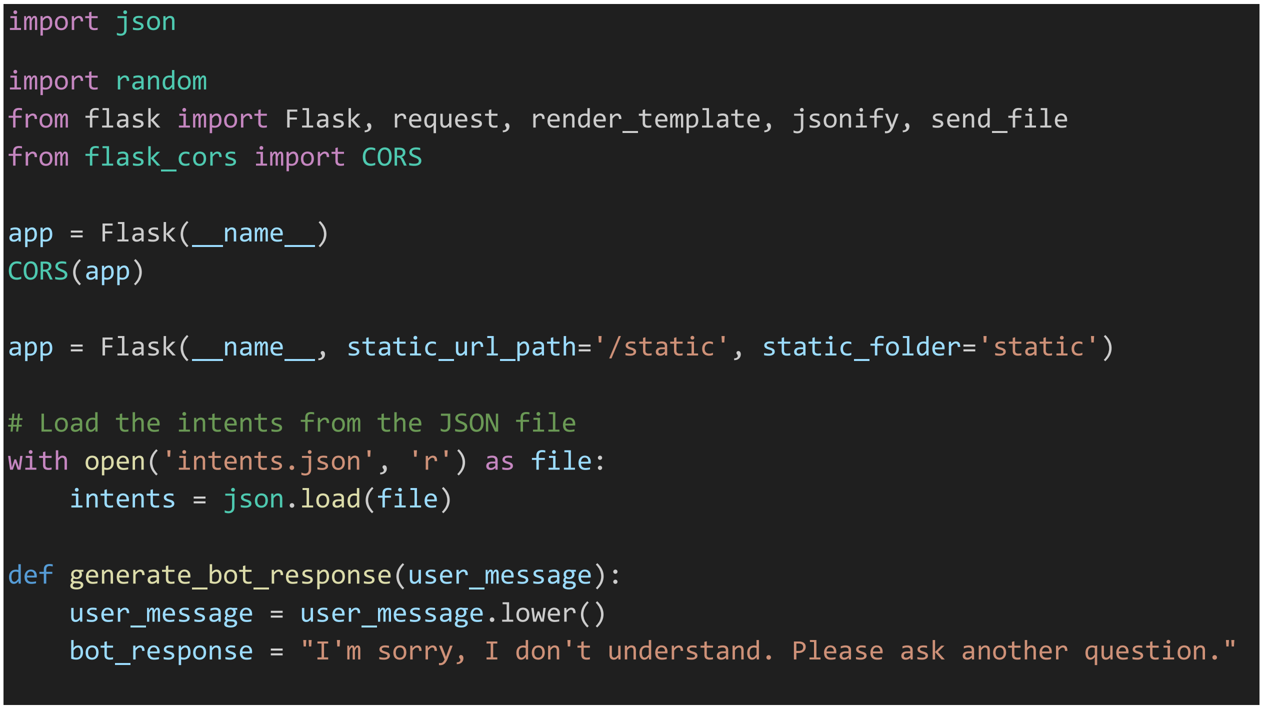
This organized structure encompasses our chatbot model, Flask application (`app.py`), HTML templates, static files (CSS), and saved tokenizer and label encoder.

2. Setting Up Flask Application:

- It all starts with creating a Python script (e.g., `app.py`) to serve as the heart of our Flask application.

- Let's import the required modules, including `Flask`, `request`, `render_template`, `jsonify`, and `send_file`.

- We'll initiate our Flask app with the following code:



```python
import json

import random
from flask import Flask, request, render_template, jsonify, send_file
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

app = Flask(__name__, static_url_path='/static', static_folder='static')

# Load the intents from the JSON file
with open('intents.json', 'r') as file:
    intents = json.load(file)

def generate_bot_response(user_message):
    user_message = user_message.lower()
    bot_response = "I'm sorry, I don't understand. Please ask another question."
    for intent in intents["intents"]:
        for pattern in intent["patterns"]:
            if pattern.lower() in user_message:
                bot_response = random.choice(intent["responses"])
                break

    return bot_response

@app.route('/static/script.js')
def serve_script():
    return send_file('static/script.js', mimetype='application/javascript')


@app.route('/')
def index():
    return render_template('index.html')

@app.route('/process_user_message', methods=['POST'])
def process_user_message():
    user_message = request.json['user_message']

    # Convert the user's message to lowercase
    user_message = user_message.lower()

    # Use the loaded model and dataset to generate a response
    bot_response = generate_bot_response(user_message)

    return jsonify({"bot_response": bot_response})

if __name__ == '__main__':
    app.debug = True
    app.run(debug=True)
```

3. HTML Templates:

- In our project directory, we'll craft a `templates` directory to house our HTML template (e.g., `index.html`). This template will be our user interface for interacting with the chatbot.

- We'll take the liberty to customize the HTML template, defining the structure and layout of our chat interface. Our aim is to ensure it includes placeholders for user input, chat history, and bot responses.

```
<!DOCTYPE html>
<html>
<head>
    <title>LearnProg</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.5.0/css/all.css" integrity="sha384-B4dIYHKNBt8Bc12p+WXckhzcICo0wtJAoU8YZTY5qE0Id1GSseTk6S+L3BlXeVIU" crossorigin="anonymous">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css')}}">
</head>
<body>
    <div class="container-fluid h-100">
        <div class="row justify-content-center h-100">
            <div class="col-md-8 col-xl-6 chat">
                <div class="card">
                    <div class="card-header msg_head">
                        <div class="d-flex bd-highlight">
                            <div class="img_cont">
                                <img src="https://pbs.twimg.com/profile_images/1430732297075003392/uJfd7-pW.jpg" class="rounded-circle user_img" width="40" height="40">
                                <span class="online_icon"></span>
                            </div>
                            <div class="user_info">
                                <span>LearnProg</span>
                                <p>Ask me anything!</p>
                            </div>
                        </div>
                    </div>
                    <div id="messageFormeight" class="card-body msg_card_body">
                        <!-- Chat messages will go here -->
                    </div>
                    <div class="card-footer">
                        <form id="messageArea" class="input-group">
                            <input type="text" id="text" name="msg" placeholder="Type your message..." autocomplete="off" class="form-control type_msg" required/>
                            <div class="input-group-append">
                                <button type="submit" id="send" class="input-group-text send_btn"><i class="fas fa-location-arrow"></i></button>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- First, load jQuery -->
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

    <!-- Then, load your custom JavaScript (script.js) -->
    <script src="{{ url_for('static', filename='script.js') }}"></script>

    <!-- Finally, include your inline JavaScript -->
    <script>
        $(document).ready(function() {
            $("#messageArea").on("submit", function(event) {
                const date = new Date();
                const hour = date.getHours();
                const minute = date.getMinutes();
                const str_time = hour + ":" + minute;
                var rawText = $("#text").val();

                // Check if the user input is empty or only contains the time
                if (rawText.trim() === "" || rawText.trim() === str_time) {
                    // Do not send empty or time-only messages
                    return;
                }

                var userHtml = '<div class="d-flex justify-content-end mb-4"><div class="msg_cotainer_send">' + rawText + '<span class="msg_time_send">' + str_time + '</span></div><div class="img_cont_msg"><img src="https://i.ibb.co/d5b84Xw/Untitled-design.png" class="rounded-circle user_img_msg" width="40" height="40"></div></div>';

                $("#text").val("");
                $("#messageFormeight").append(userHtml);

                $.ajax({
                    type: "POST",
                    url: "/process_user_message",
                    contentType: "application/json",
                    data: JSON.stringify({ user_message: rawText }), // Pass the user's message as JSON
                    success: function(data) {
                        var botHtml = '<div class="d-flex justify-content-start mb-4"><div class="img_cont_msg"><img src="https://pbs.twimg.com/profile_images/1430732297075003392/uJfd7-pW.jpg" class="rounded-circle user_img_msg" width="40" height="40"></div><div class="msg_cotainer">' + data.bot_response + '<span class="msg_time">' + str_time + '</span></div></div>';
                        $("#messageFormeight").append($.parseHTML(botHtml));
                    },
                    error: function(error) {
                        console.error("Error:", error);
                    }
                });
                event.preventDefault();
            });
        });
    </script>

</body>
</html>
```

4. Static Files (CSS):

- Let's create a `static` directory within our project directory.

- Within the `static` directory, we'll place our CSS file (e.g., `style.css`) to bestow style upon our chat interface and JavaScript file (e.g.,`script.js`) for adding interactive features.
- We'll make certain to link the CSS and JavaScript files in our HTML template.

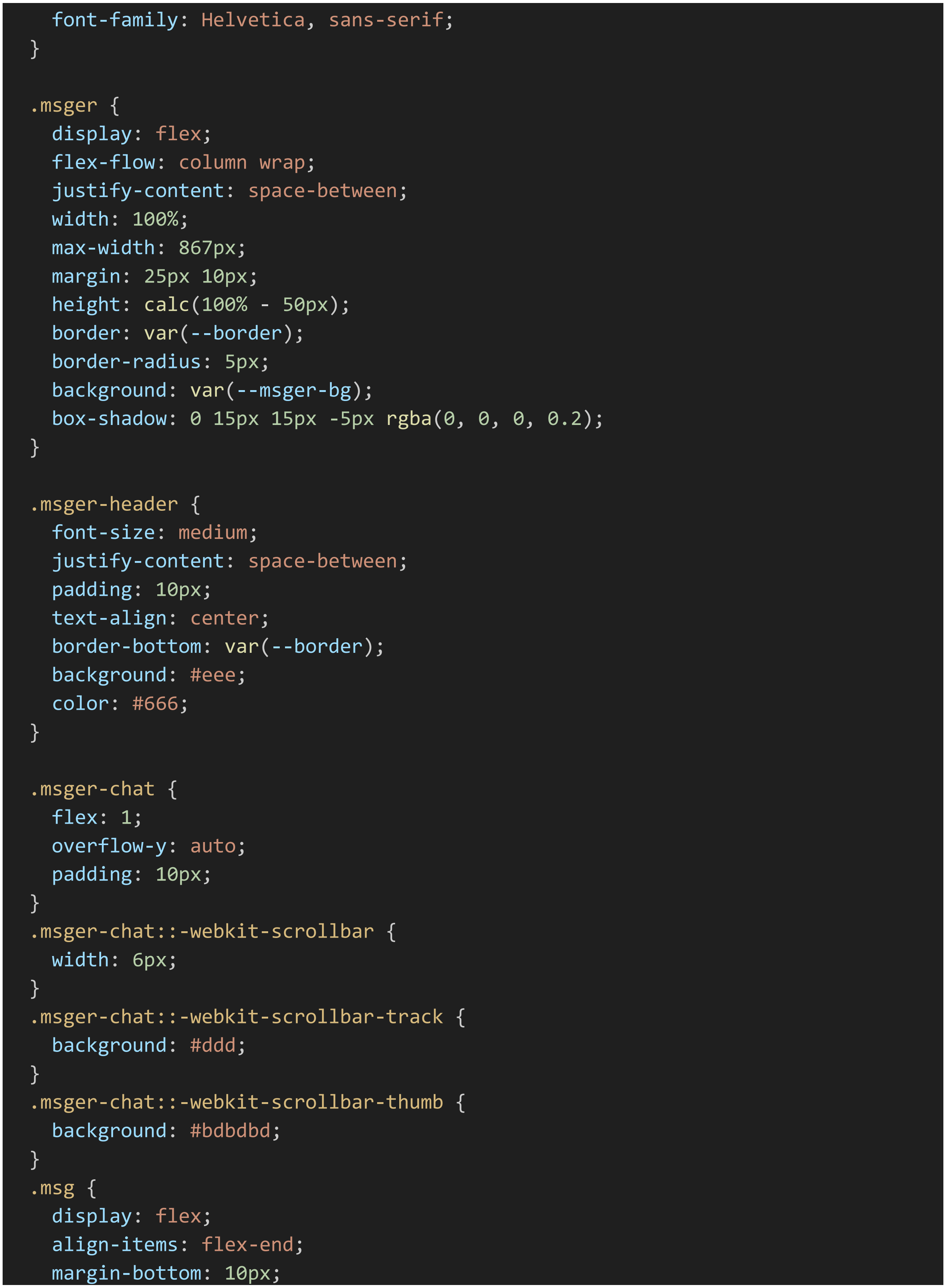
Style.css

```css
:root {
    --body-bg: linear-gradient(135deg, #f5f7fa 0%, #c3cfe2 100%);
    --msger-bg: #fff;
    --border: 2px solid #ddd;
    --left-msg-bg: #ececec;
    --right-msg-bg: #579ffb;
  }

  html {
    box-sizing: border-box;
  }

  *,
  *:before,
  *:after {
    margin: 0;
    padding: 0;
    box-sizing: inherit;
  }

  body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-image: var(--body-bg);
```

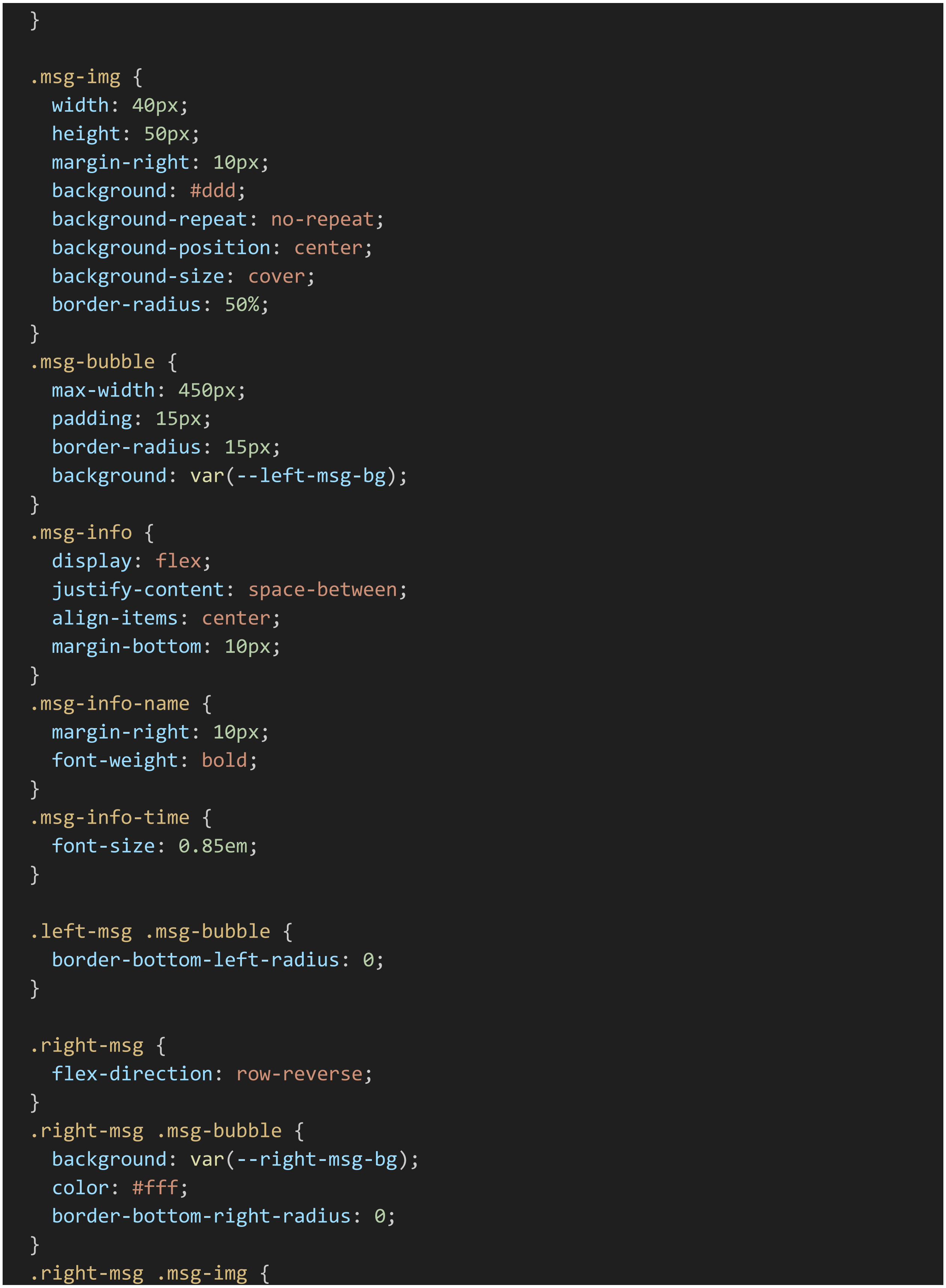


```css
  font-family: Helvetica, sans-serif;
}

.msger {
  display: flex;
  flex-flow: column wrap;
  justify-content: space-between;
  width: 100%;
  max-width: 867px;
  margin: 25px 10px;
  height: calc(100% - 50px);
  border: var(--border);
  border-radius: 5px;
  background: var(--msger-bg);
  box-shadow: 0 15px 15px -5px rgba(0, 0, 0, 0.2);
}

.msger-header {
  font-size: medium;
  justify-content: space-between;
  padding: 10px;
  text-align: center;
  border-bottom: var(--border);
  background: #eee;
  color: #666;
}

.msger-chat {
  flex: 1;
  overflow-y: auto;
  padding: 10px;
}
.msger-chat::-webkit-scrollbar {
  width: 6px;
}
.msger-chat::-webkit-scrollbar-track {
  background: #ddd;
}
.msger-chat::-webkit-scrollbar-thumb {
  background: #bdbdbd;
}
.msg {
  display: flex;
  align-items: flex-end;
  margin-bottom: 10px;
```



```css
}

.msg-img {
  width: 40px;
  height: 50px;
  margin-right: 10px;
  background: #ddd;
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  border-radius: 50%;
}
.msg-bubble {
  max-width: 450px;
  padding: 15px;
  border-radius: 15px;
  background: var(--left-msg-bg);
}
.msg-info {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 10px;
}
.msg-info-name {
  margin-right: 10px;
  font-weight: bold;
}
.msg-info-time {
  font-size: 0.85em;
}

.left-msg .msg-bubble {
  border-bottom-left-radius: 0;
}

.right-msg {
  flex-direction: row-reverse;
}
.right-msg .msg-bubble {
  background: var(--right-msg-bg);
  color: #fff;
  border-bottom-right-radius: 0;
}
.right-msg .msg-img {
  margin: 0 0 0 10px;
}

.msger-inputarea {
  display: flex;
  padding: 10px;
  border-top: var(--border);
  background: #eee;
}
.msger-inputarea * {
  padding: 10px;
  border: none;
  border-radius: 3px;
  font-size: 1em;
}
.msger-input {
  flex: 1;
  background: #ddd;
}
.msger-send-btn {
  margin-left: 10px;
  background: #579ffb;
  color: #fff;
  font-weight: bold;
  cursor: pointer;
  transition: background 0.23s;
}
.msger-send-btn:hover {
  background: rgb(0, 180, 50);
}

.msger-chat {
  background-color: #fcfcfe;
}
```

Script.js

```javascript
$(document).ready(function() {
    $("#messageArea").on("submit", function(event) {
        event.preventDefault(); // Prevent the default form submission

        const date = new Date();
        const hour = date.getHours();
        const minute = date.getMinutes();
        const str_time = hour + ":" + minute;
        var rawText = $("#text").val();


        var userHtml = '<div class="d-flex justify-content-end mb-4"><div class="msg_cotainer_send">' + rawText + '<span class="msg_time_send">' + str_time + '</span></div><div class="img_cont_msg"><img src="https://i.ibb.co/d5b84Xw/Untitled-design.png" class="rounded-circle user_img_msg" width="40" height="40"></div></div>';

        $("#text").val("");
        $("#messageFormeight").append(userHtml);

        // Send a POST request to the server
        $.ajax({
            type: "POST",
            url: "/process_user_message",
            contentType: "application/json",
            data: JSON.stringify({ user_message: rawText }), // Pass the user's message as JSON
            success: function(data) {
                var botHtml = '<div class="d-flex justify-content-start mb-4"><div class="img_cont_msg"><img src="https://pbs.twimg.com/profile_images/1430732297075003392/uJfd7-pW.jpg" class="rounded-circle user_img_msg" width="40" height="40"></div><div class="msg_cotainer">' + data.bot_response + '<span class="msg_time">' + str_time + '</span></div></div>';
                $("#messageFormeight").append($.parseHTML(botHtml));
            },
            error: function(error) {
                console.error("Error:", error);
            }
        });
    });
});
```

5. Flask Routes:

- It's time to define Flask routes within our application script. We'll establish a default route for rendering the chat interface and an API route for processing user messages.

```
@app.route('/')
 def home():
    return render_template('index.html')


 @app.route('/process_user_message', methods=['POST'])
 def process_user_message():
    # Let's retrieve user input from the request


    # We'll send the user message to our chatbot model


    # We'll obtain the bot's response


    return jsonify({'bot_response': bot_response})
```

6. Running the Flask App:

- Adding the following code to our Flask script to run the app:

```
if __name__ == '__main__':
```

```
app.run(debug=True)
```

7. Running the Chatbot:

- Include instructions for running the chatbot, which you can use in the `process_user_message` route.

```
# Send the user message to the chatbot model
bot_response = chatbot_response(user_message)
```

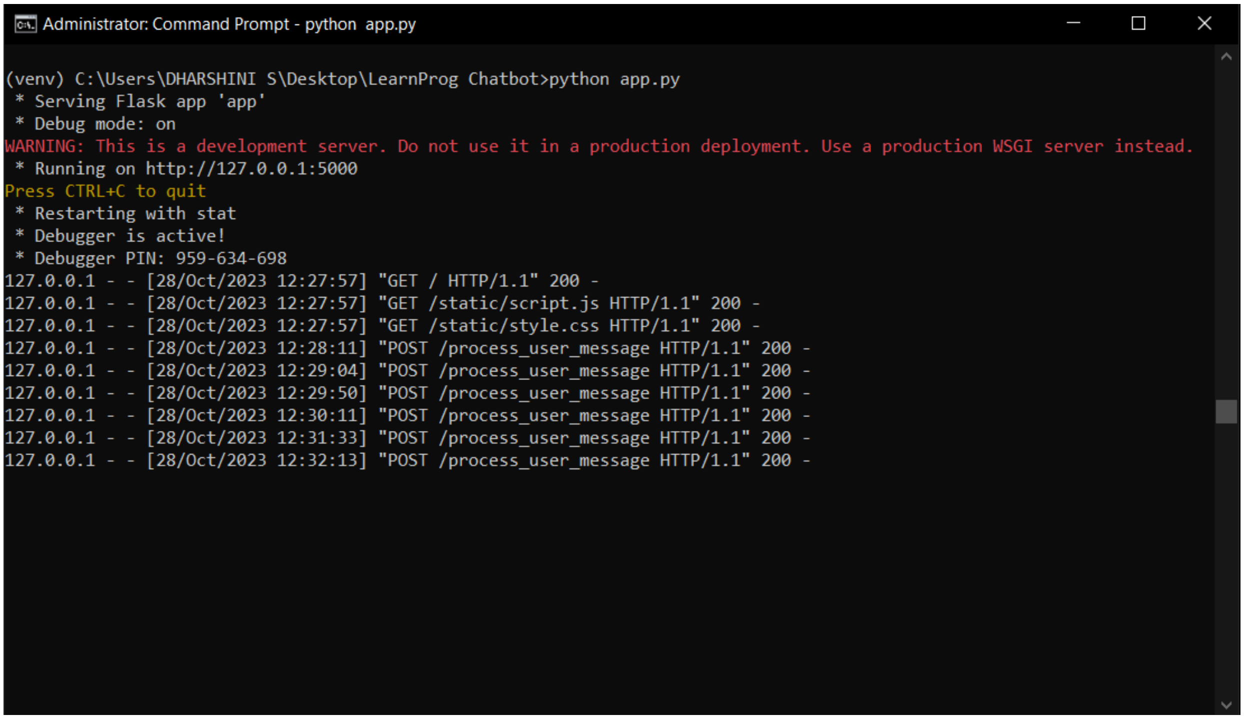
- Making sure that we load our chatbot model and preprocessing components.

8. Running the Flask App:

- To start the Flask application, we run the `app.py` script:

```
python app.py
```

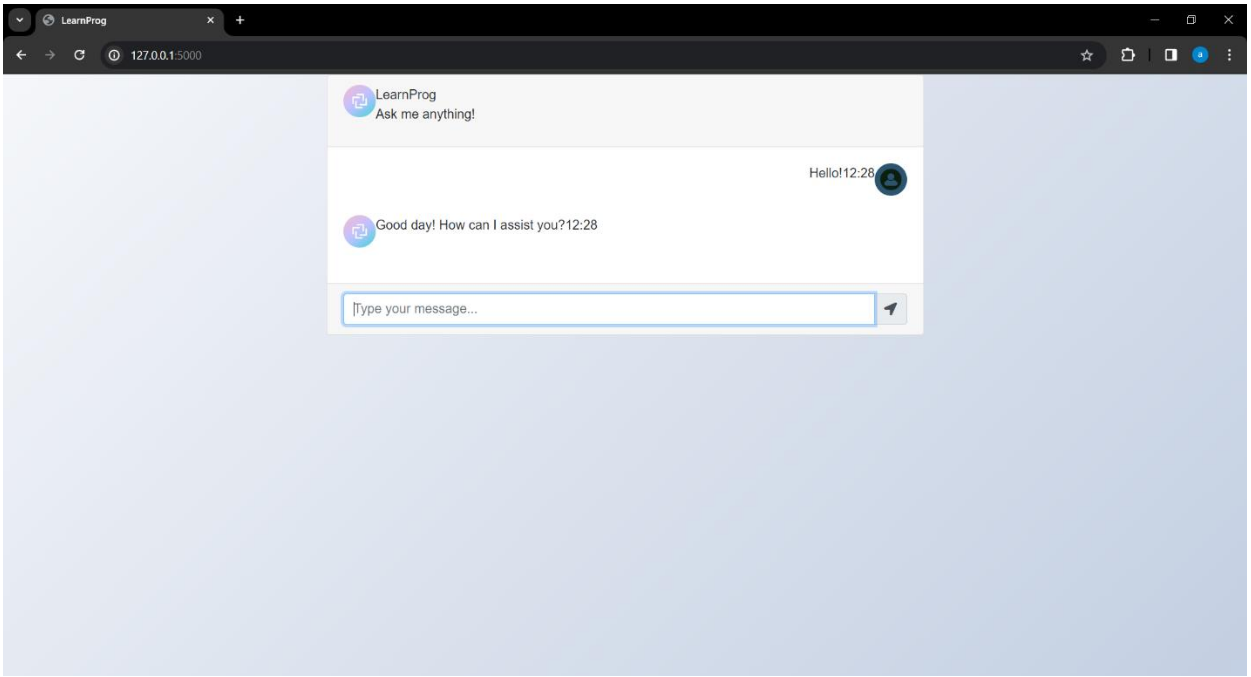
- our chatbot web application now be accessible at a URL (usually `http://127.0.0.1:5000`).



```
(venv) C:\Users\DHARSHINI S\Desktop\LearnProg Chatbot>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 959-634-698
127.0.0.1 - - [28/Oct/2023 12:27:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:27:57] "GET /static/script.js HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:27:57] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:28:11] "POST /process_user_message HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:29:04] "POST /process_user_message HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:29:50] "POST /process_user_message HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:30:11] "POST /process_user_message HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:31:33] "POST /process_user_message HTTP/1.1" 200 -
127.0.0.1 - - [28/Oct/2023 12:32:13] "POST /process_user_message HTTP/1.1" 200 -
```

9. User Interaction:

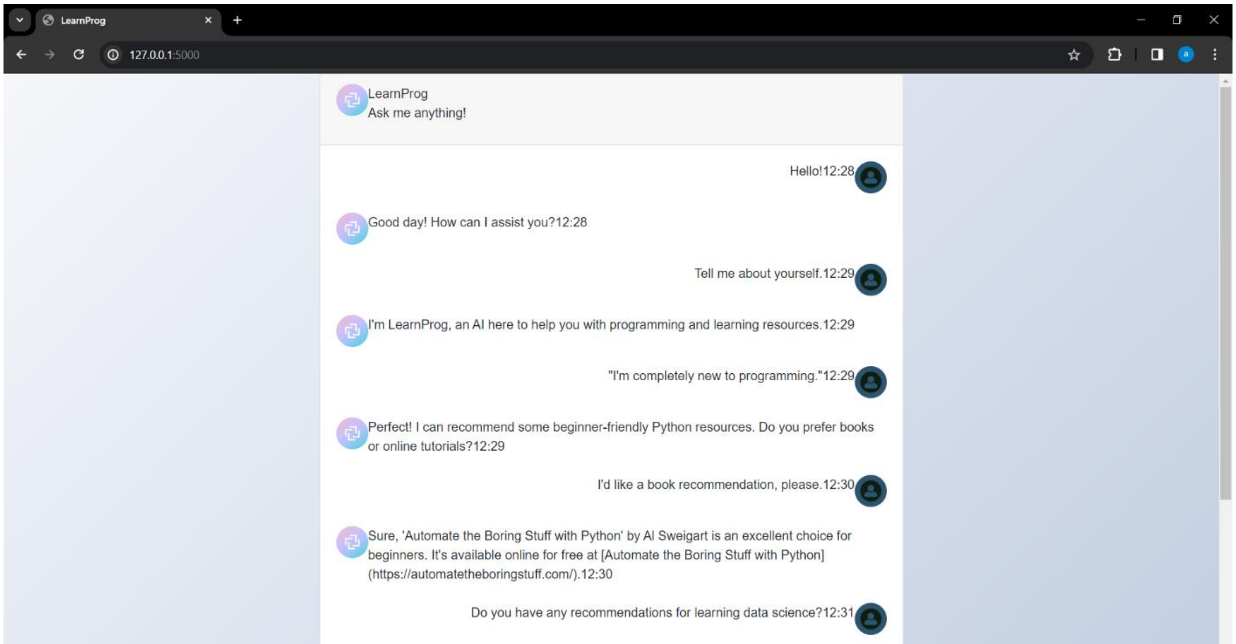
- Explaining how users can interact with your chatbot using the web interface.



LearnProg
Ask me anything!

Hello!12:28

Good day! How can I assist you?12:28



LearnProg
Ask me anything!

Hello!12:28

Good day! How can I assist you?12:28

Tell me about yourself.12:29

I'm LearnProg, an AI here to help you with programming and learning resources.12:29

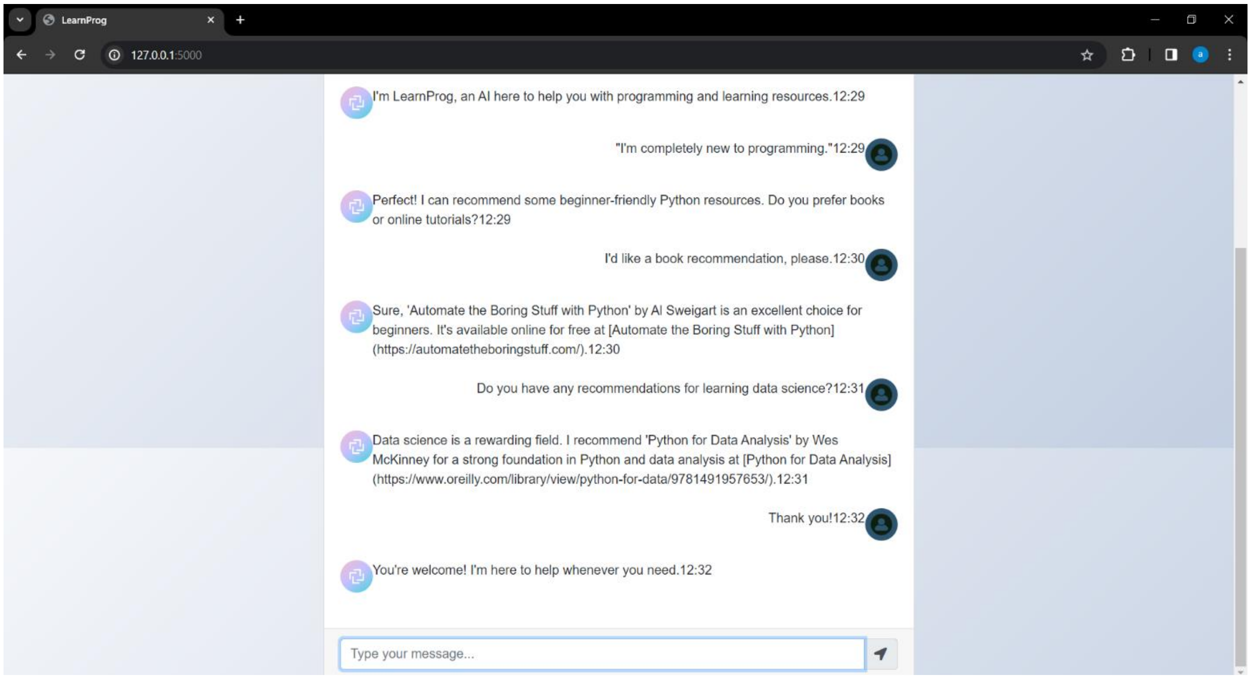
"I'm completely new to programming."12:29

Perfect! I can recommend some beginner-friendly Python resources. Do you prefer books or online tutorials?12:29

I'd like a book recommendation, please.12:30

Sure, 'Automate the Boring Stuff with Python' by Al Sweigart is an excellent choice for beginners. It's available online for free at [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/).12:30

Do you have any recommendations for learning data science?12:31



I'm LearnProg, an AI here to help you with programming and learning resources.12:29

"I'm completely new to programming."12:29

Perfect! I can recommend some beginner-friendly Python resources. Do you prefer books or online tutorials?12:29

I'd like a book recommendation, please.12:30

Sure, 'Automate the Boring Stuff with Python' by Al Sweigart is an excellent choice for beginners. It's available online for free at [Automate the Boring Stuff with Python](https://automatetheboringstuff.com/).12:30

Do you have any recommendations for learning data science?12:31

Data science is a rewarding field. I recommend 'Python for Data Analysis' by Wes McKinney for a strong foundation in Python and data analysis at [Python for Data Analysis](https://www.oreilly.com/library/view/python-for-data/9781491957653/).12:31

Thank you!12:32

You're welcome! I'm here to help whenever you need.12:32

Conclusion:

In conclusion, the development of our Flask web application marks a significant milestone in making our chatbot accessible to users. This phase focused on building the necessary infrastructure for user interaction. We've successfully set up a well-organized project directory, created HTML templates for the chat interface, styled it using CSS, JavaScript to enhance the interactive features of the web application and defined Flask routes for rendering the interface and processing user messages.

The Flask application, with its routes, serves as the bridge between the user and the chatbot. It opens up new possibilities for users to engage with our chatbot on a user-friendly platform. With this interface in place, users can now access the chatbot from a web browser, making interactions convenient and intuitive.

This phase not only enhances the user experience but also demonstrates the versatility of our chatbot model. Users can now ask questions and receive responses through a dynamic and visually appealing web interface. The Flask web application acts as a powerful tool for deploying AI models in real-world applications.

Our journey is far from over, and future phases will focus on further improving the chatbot's capabilities and integrating it into various platforms. With each phase, we get closer to our goal of creating an AI-powered assistant that can help and guide users effectively.