

**The Superior University Lahore**

**Faculty of Computer Science & Information**

**Technology**

**Name: Alishba Haroon**

**Roll No: BSAI-116-4C**

**Date: 27 March 2025**

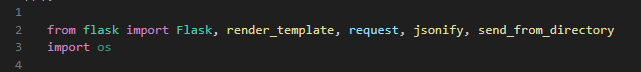
**Subject: PAI LAB**

**Task 10**

**CyberGruad Information**

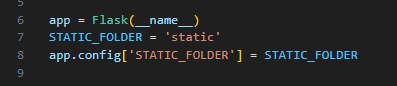
**(App.py)**

**1. Importing Libraries**

****

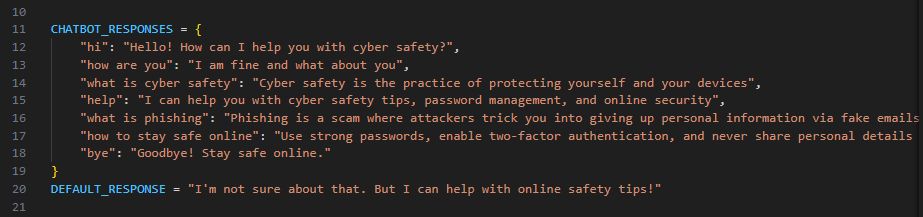
* **Flask:** Framework to build a web application.
* **render\_template:** Loads HTML templates like index.html.
* **request:** Reads data sent by users.
* **jsonify:** Returns data in JSON format.
* **send\_from\_directory:** Serves static files like CSS, JS, and images.
* **os:** helpful for file operations.

**2. Flask App Configuration**

****

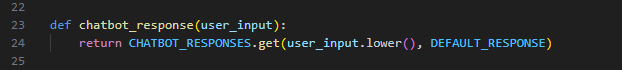
* **app = Flask(\_\_name\_\_):** Initializes the Flask app.
* **STATIC\_FOLDER = 'static':** The folder that contains static files.
* **app.config['STATIC\_FOLDER'] = STATIC\_FOLDER:** Configures Flask to know where static files live.

**3. Chatbot Responses Dictionary**



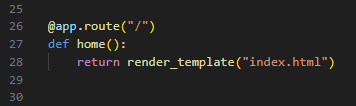
* A simple dictionary maps user inputs to fixed responses.
* If the input doesn’t match any key, it returns the DEFAULT\_RESPONSE.

**4. Chatbot Response Function**



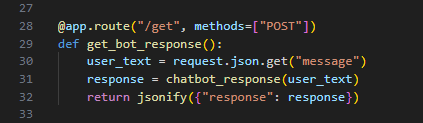
* Takes user input.
* Converts it to lowercase and looks for a matching response in the dictionary.
* If not found, returns the default message.

**5. Route for Homepage**



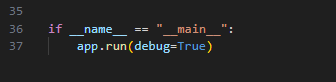
* When the user visits the root URL (/), Flask serves the index.html file.

**6. Route to Handle User Messages**

****

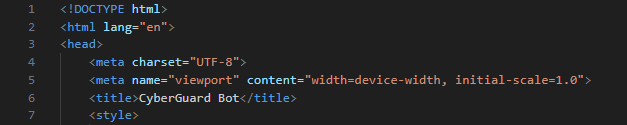
* This route receives POST requests from JavaScript into HTML.
* It gets the user's message from the request.
* Calls chatbot\_response() to generate a reply.
* Sends back a JSON object containing the bot's response.

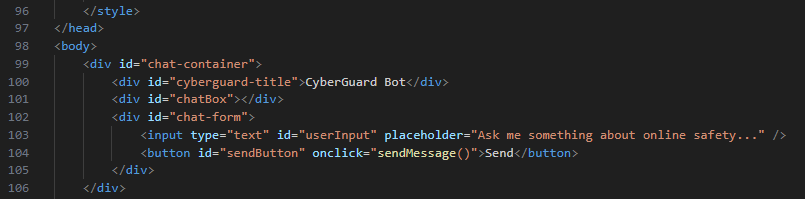
**7. Run the Flask App**

****

* This runs the Flask development server.
* debug=True enables live reload and better error messages.

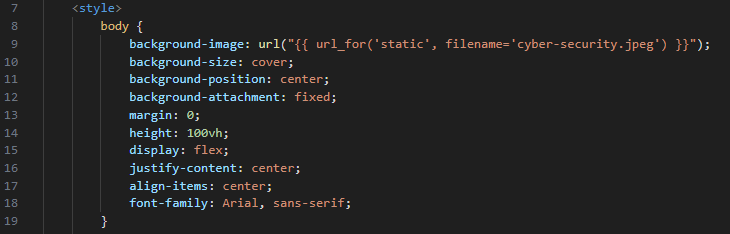
**1. HTML Structure**

****

****

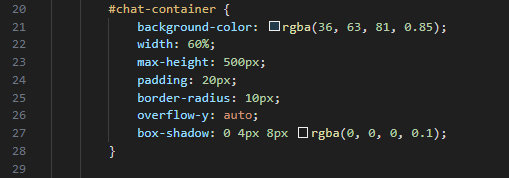
* **DOCTYPE:** This declares that the document type is HTML5.
* **<html> tag:** The root element of the HTML document. The lang="en" attribute specifies that the document is in English.
* **<head> tag:** Contains metadata about the HTML document, like character set, viewport settings, and the title of the page.
* **<body> tag:** This contains the visible content of the webpage, including the chat interface.

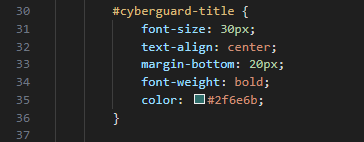
**2. CSS Styling:**

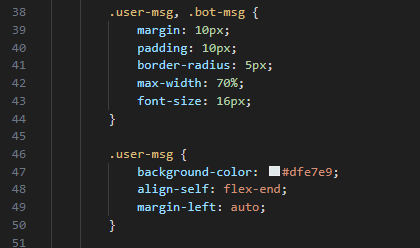
****

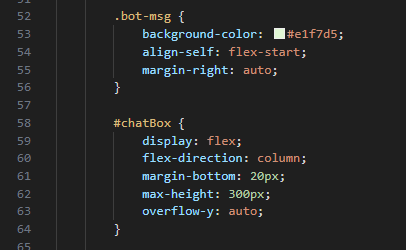
* **background-image:** This uses a relative URL to load a background image cyber-security.jpeg from the static directory of your Flask app. url\_for('static', filename='cyber-security.jpeg') is a Flask-specific function that generates the URL to static files.
* Flexbox Layout: The body uses Flexbox to center the content both vertically and horizontally.
* font-family: The page uses the Arial font.

**3. Chat Container and Messages:**

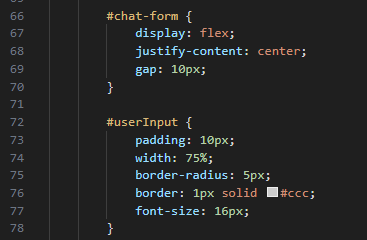
****

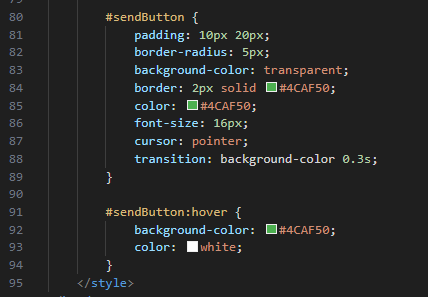
****

****

****

* **#chat-container:** The main chat window, with a semi-transparent dark background, rounded corners, and a shadow effect. The content scrolls when it exceeds 500px in height.
* **#cyberguard-title:** The title "CyberGuard Bot" at the top of the chat interface, centered with a greenish color.
* **Messages (.user-msg and .bot-msg):** User messages are aligned to the right (align-self: flex-end) with a different background color (#dfe7e9), while bot messages are aligned to the left (align-self: flex-start) with another background color (#e1f7d5). Both have padding and rounded corners.

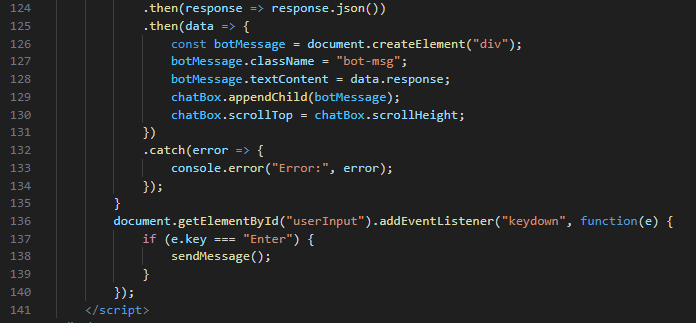
****

****

* **#chat-form:** Contains the input field and the send button. It's styled with Flexbox to center the elements and add some space between them.
* **#userInput:** The text input field for the user’s message, with padding, a border, and a rounded appearance.
* **#sendButton:** The send button, styled with a transparent background and a green border. It changes color when hovered over (via the :hover selector).

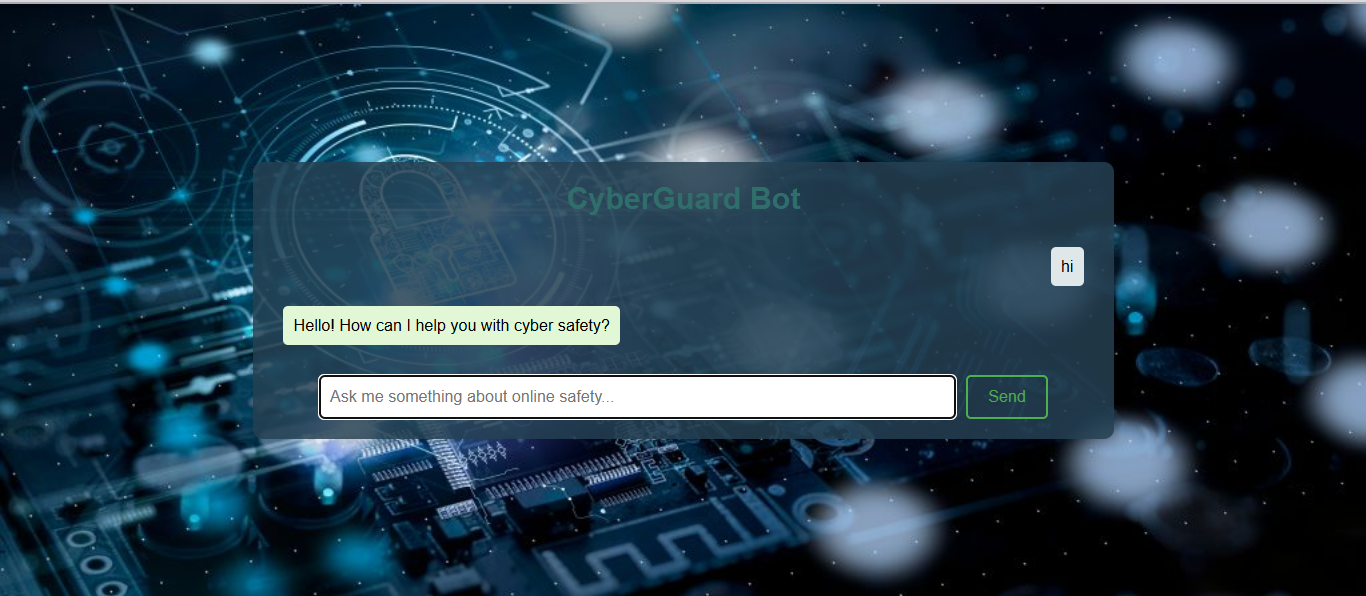
**5. JavaScript Logic:**

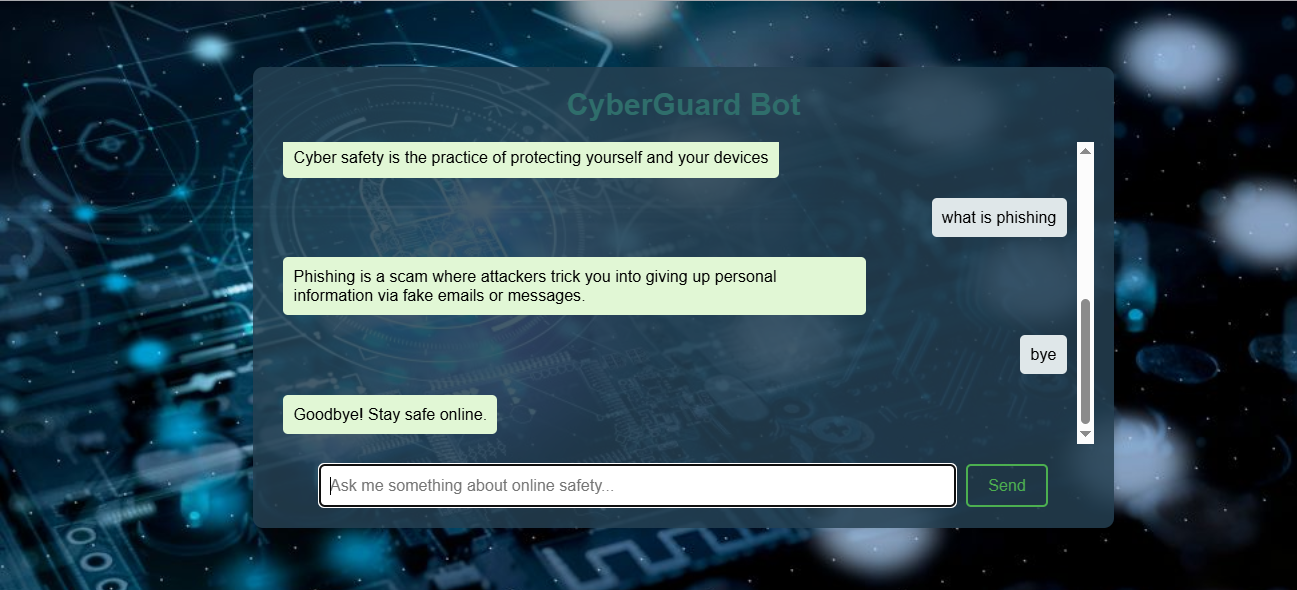
****

****

* sendMessage(): Retrieves the value from the input field (userInput.value) and trims any unnecessary spaces.
* If the message is not empty, it creates a new <div> element for the user’s message and appends it to the chat box.
* The fetch() function sends the user’s message to the server (via the /get endpoint) in JSON format.
* When the server responds with a bot message (as JSON), that message is displayed in the chat box as a bot message. The chat box scrolls automatically to show the latest message.
* The event listener listens for the Enter key (e.key === "Enter") and triggers the sendMessage() function.

**OUTPUT:**

****

****