

**The Superior University Lahore**

**Faculty of Computer Science & Information**

**Technology**

**Name: Alishba Haroon**

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

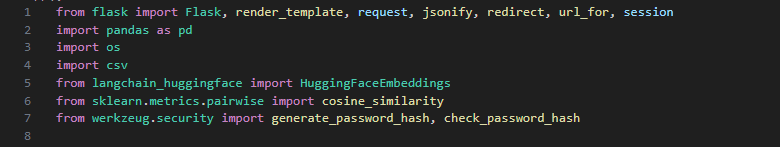
**Date: 28 April 2025**

**Subject: PAI LAB**

**LAB 13 TASK**

**Finance Chatbot**

1. **Imports**

****

1. **Flask-related imports:**

* **Flask:** The main class that is used to create the Flask app.
* **render\_template:** Renders HTML templates.
* **request:** Handles incoming HTTP requests (e.g., form submissions).
* **jsonify:** Converts Python dictionaries into JSON responses.
* **redirect:** Redirects the user to a different route.
* **url\_for:** Generates the URL for a given route.
* **session:** Used to store data between requests, e.g., storing user login status.

1. **Data-related imports:**

* **pandas:** A library for data manipulation, here used to read the dataset.
* **os:** Provides a way to interact with the operating system (e.g., checking if files exist).
* **csv:** Allows reading and writing CSV files.

1. **Langchain-related imports:**

* **HuggingFaceEmbeddings:** A class from langchain\_huggingface that provides embeddings from Hugging Face models for text queries.

1. **Machine learning-related imports:**

* **cosine\_similarity:** Computes the cosine similarity between two vectors to measure how similar they are.

1. **Password security imports:**

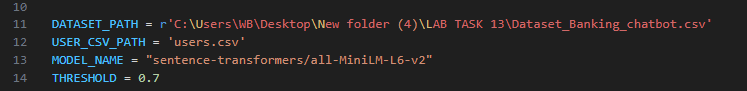
* **generate\_password\_hash:** Hashes a plain-text password into a secure hashed version for storage.
* **check\_password\_hash:** Compares the provided plain-text password with the stored hashed version

1. **Flask App Setup**

****

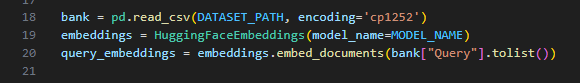
* **app = Flask(\_\_name\_\_):** Initializes the Flask application.
* **app.secret\_key = '7654':** Sets the secret key for session management. This key is used to sign session cookies to prevent tampering. It is a security measure for session management.

1. **File Paths and Constants**

****

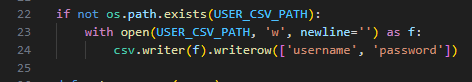
* **DATASET\_PATH:** The path to the CSV file that contains the dataset of queries and responses for the chatbot.
* **USER\_CSV\_PATH:** The path where user credentials (username and hashed password) will be stored in a CSV file.
* **MODEL\_NAME:** Specifies the name of the pre-trained model used for embedding queries (Hugging Face's sentence-transformers/all-MiniLM-L6-v2).
* **THRESHOLD:** A cosine similarity threshold (0.7). If the similarity score between the user query and dataset queries is below this threshold, a default message is returned.

1. **Load Dataset and Generate Embeddings**

****

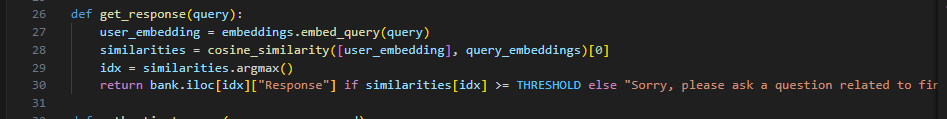
* **bank = pd.read\_csv(DATASET\_PATH, encoding='cp1252'):** Loads the CSV file into a pandas DataFrame bank. This file contains the chatbot’s queries and responses.
* **encoding='cp1252':** Specifies the encoding to handle non-ASCII characters.
* **embeddings = HuggingFaceEmbeddings(model\_name=MODEL\_NAME):** Initializes the HuggingFaceEmbeddings object with the model specified earlier (sentence-transformers/all-MiniLM-L6-v2).
* **query\_embeddings = embeddings.embed\_documents(bank["Query"].tolist()):** Embeds all the queries in the Query column of the dataset. This generates vector representations of each query to compare user queries with dataset queries.

1. **Ensure the User CSV File Exists**

****

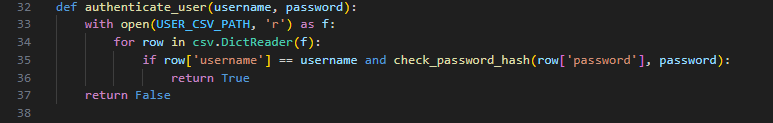
* **if not os.path.exists(USER\_CSV\_PATH):** Checks if the users.csv file exists.
* **with open(USER\_CSV\_PATH, 'w', newline='') as f:** If the file doesn’t exist, it creates a new file and opens it in write mode.
* **csv.writer(f).writerow(['username', 'password']):** Writes the header row ['username', 'password'] to the file for storing user credentials**.**

1. **Helper Functions**

****

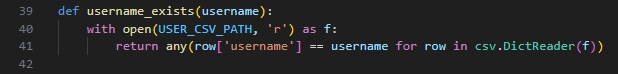
* **user\_embedding = embeddings.embed\_query(query):** Embeds the user’s query into a vector.
* **similarities = cosine\_similarity([user\_embedding], query\_embeddings)[0]:** Calculates the cosine similarity between the user query and all precomputed query embeddings from the dataset.
* **idx = similarities.argmax():** Finds the index of the query with the highest similarity score.
* **return bank.iloc[idx]["Response"]:** If the highest similarity score is above the threshold, it returns the corresponding response from the dataset. Otherwise, it returns a default message.

1. **Authenticate User Function**

****

* **with open(USER\_CSV\_PATH, 'r') as f:** Opens the users.csv file in read mode.
* **for row in csv.DictReader(f):** Reads each row of the file as a dictionary.
* **if row['username'] == username and check\_password\_hash(row['password'], password):** Compares the provided username and password (the password is hashed and checked).
* **return True:** If a matching username and correct password are found, it returns True (authenticated).
* **return False:** If no match is found, it returns False.

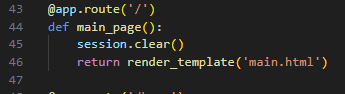
1. **Check if Username Exists**

****

* **with open(USER\_CSV\_PATH, 'r') as f:** Opens the users.csv file.
* return any(row['username'] == username for row in **csv.DictReader(f)):** Returns True if any username matches the input username, indicating the username already exists. Otherwise, it returns False.

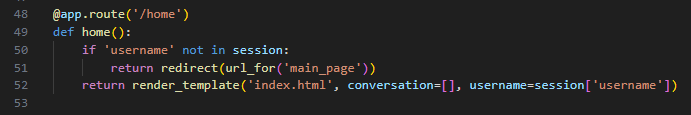
**Flask Routes**

1. **Main Page Route**

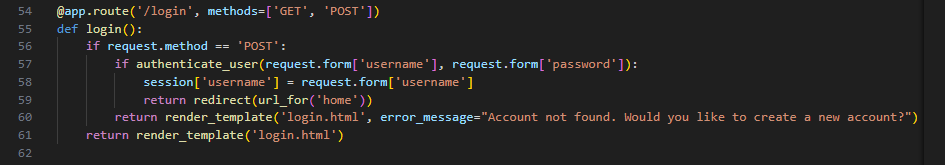
****

* **@app.route('/'):** The main route, which is the homepage of the application.
* **session.clear():** Clears any existing session data (logs the user out).
* **return render\_template('main.html'):** Renders the main HTML template, where the user can log in or sign up.

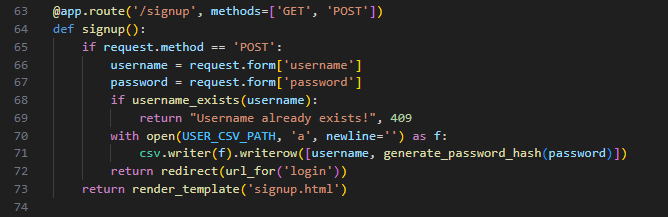
**10. Home Page Route**

****

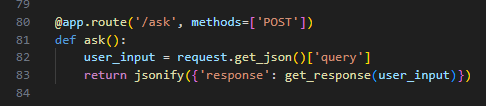
* **if 'username' not in session:** Checks if the user is logged in (i.e., if there is a username in the session).
* **return redirect(url\_for('main\_page')):** If the user is not logged in, redirects them to the main page.
* **return render\_template('index.html', conversation=[], username=session['username']):** If logged in, it renders the home page template, passing the username and an empty conversation list.
  1. **Login Route**

****

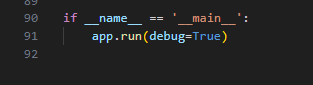
* **if request.method == 'POST':** Handles form submission (POST request).
* **if authenticate\_user(request.form['username'], request.form['password']):** Authenticates the user with the provided username and password.
* **session['username'] = request.form['username']:** Stores the username in the session to track the logged-in user.
* **return redirect(url\_for('home')):** Redirects to the home page after a successful login.
* **return render\_template('login.html', error\_message="Account not found..."):** If authentication fails, renders the login page with an error message.
  1. **Signup Route**

****

* **if request.method == 'POST':** Handles form submission (POST request) for signup.
* **if username\_exists(username):** Checks if the username already exists in the CSV file.
* **csv.writer(f).writerow([username, generate\_password\_hash(password)]):** If the username is new, it hashes the password and saves the username and hashed password to users.csv.
* **return redirect(url\_for('login')):** Redirects to the login page after successful signup.
* **return render\_template('signup.html'):** Renders the signup page.
  1. **Ask Route**

****

1. **user\_input = request.get\_json()['query']:** Retrieves the query sent by the user in the JSON body of the POST request.
2. **return jsonify({'response': get\_response(user\_input)}):** Returns the response from the get\_response function as a JSON object.
   1. **Running the Application**

****

* **app.run(debug=True):** Starts the Flask development server with debugging enabled to help with development.