

CIS 4951 U01 1251

Masoud Sadjadi, Florida International University

CAPSTONE II

AI-Based Customizable Chatbot

Team Members:

Mohammed Alfarra, Omar Soliman, Alexis Okafor, Alejandro
Alonso, David Del Valle

April 16, 2025

Introduction:

Our project for this semester is an AI-powered customizable chatbot platform developed to explore the intersection of AI and customer service. As technology continues to influence how we communicate and access information, the limitations of chatbots we use today have become increasingly apparent. Many conversational AI tools online offer generic, one-size-fits-all interactions that fail to adapt to individual user needs. These systems often lack a sense of personality, provide little to no customization, and can reset the conversation with each new session. As a result, users are frequently left repeating themselves and engaging with assistants that feel more mechanical than helpful. For this project, we aim to have a platform that enables users to interact with customizable chatbot personalities for a personalized and engaging experience.

What sets this chatbot apart from other AI solutions is its emphasis on customization and personality variety. Users can choose from multiple predefined personas—such as a pirate, detective, or other engaging archetypes—each with its own distinct behavior, language style, and tone. This approach creates a sense of fun and immersion, and also allows users to tailor the assistant to their specific needs. In addition, the platform supports persistent session memory, meaning that conversations can be continued and carry over between visits. Users can also export their chats for reference and record keeping. These features are tied together to have the user enjoy a seamless and intuitive user experience.

Key Features:

Our chatbot was built with usability, personalization, and flexibility as the main focus. Listed below are the key features that distinguish it from other chatbot systems to provide a more engaging user experience.

Flexible User Access

The platform is designed to be accessible to a wide range of users. Whether someone wants to register for a full account, log in to access their personalized settings and chat history, or simply continue as a guest, the system provides seamless functionality for all users.

Custom Chatbot Creation

Users can create their own unique chatbot by naming it, selecting a visual character, and assigning one of ten available personalities. These personalities affect how the chatbot responds and behaves, offering a wide variety of interaction styles and tones. This feature allows users to shape their experience based on their specific needs.

Multi-Personality Conversations

Each chatbot personality is designed with its own distinct language style and behavior. The chatbot adapts its responses based on the selected personality, delivering a dynamic and immersive interaction. Additionally, it remembers prior context, enabling continuity in conversations and allowing users to pick up where they left off.

Multiple Chat Sessions

Users have the option to create and manage multiple chat sessions. They can switch between ongoing conversations or start fresh ones as needed. This is especially useful for users who want to keep different topics or tasks organized, or who wish to experiment with different chatbot personalities in parallel.

Chat History & Memory

All chat sessions are saved to a centralized PostgreSQL database, allowing for persistent memory across sessions. This enables context-aware responses, improves long-term usability, and ensures that users don't need to repeat themselves every time that they use the chatbot. Past conversations can be revisited at any time, enhancing continuity and personalization.

Export Functionality

For users who want to retain a record of their conversations, this chatbot offers an export feature. Individual sessions can be downloaded for future reference, academic purposes, or personal tracking. This function adds a layer of transparency and utility, making the chatbot not just a conversation partner but a helpful digital assistant.

Implementation:

The implementation of our AI Chatbot was done using a combination of modern web and backend technologies. The project is divided into two main components: the frontend interface and the backend system, both working together to deliver a seamless user experience.

The frontend of the project was built using React, a popular JavaScript library for building user interfaces. This choice allowed us to create a dynamic and responsive web app that works across various devices. The interface was structured using HTML5 and CSS3, with interactivity handled through JavaScript. Additionally, Node.js is used to manage the dependencies, streamlining development and the deployment workflow.

On the backend, we implemented a Flask server in Python to handle application logic, process user input, and manage communication with the OpenAI API. The core functionality of the chatbot, such as generating responses, managing personalities, and handling memory, is built using Python libraries and the OpenAI API, which enables natural language understanding and generation.

For data storage, we used PostgreSQL, a powerful open source relational database system. It stores user data, chatbot personalities, message histories, and other relevant information. We accessed and managed the database through pgAdmin, a graphical interface that simplifies database interaction and administration. This setup ensures reliable data handling, supports persistent conversations, and allows users to maintain a consistent chat experience across sessions.

Together, these technologies form the base of our customizable chat platform, enabling real-time communication, memory retention, and personalized experiences, which is all done in a clean and intuitive web interface.

Verification

To ensure the reliability, accuracy, and functionality of the chatbot, we had to verify that all of the features worked as intended. The process involved systematically validating that each component of the application met the intended requirements and operated as expected under a variety of use cases.

To begin with, the registration and login system underwent testing to prevent duplicate account creation. The backend was configured to check for existing email addresses during registration and to enforce proper email formatting, ensuring a secure and clean database for users' information. This step is essential for maintaining account integrity and improving the user authentication process.

The chat history feature was carefully verified to confirm that messages were properly stored in the PostgreSQL database and were retrievable by the corresponding user. Tests were conducted across multiple sessions and user accounts to ensure that no data overlap or loss occurred, validating the persistence and isolation of each user's chat history.

Verification also covered the bot customization functionality, ensuring that users could successfully personalize their chatbot's appearance and personality. These preferences were tested for proper saving, retrieval, and consistent display throughout the application. The system was validated to retain these settings across different sessions, even when switching between chat sessions or accounts.

Finally the chatbot's capability to support multiple simultaneous sessions was verified through both automated and manual testing. This included initiating, switching between, and continuing separate conversations without loss of context or data. This functionality was crucial to delivering a smooth and intuitive user experience, especially for users engaging with multiple bots or tasks concurrently.

Requirements

To ensure smooth installation, deployment, and operation of the AI chatbot, both hardware and software requirements must be met. The requirements are listed below:

Hardware Requirements

- Memory (RAM): Minimum 2 GB
- Storage: At least 1 GB of free disk space
- Internet Connection: Stable and strong Wi-Fi connection for API interactions and cloud-based data access

Software Requirements

- Operating Environment:
 - Windows 10 or MacOS 10 or any later versions are recommended
 - Python (version 3.10 or later recommended)
 - Node.js
 - Command Line Interface (Command Prompt, Terminal, etc.)
 - Backend Libraries & Frameworks:
 - Flask==3.1.0
 - flask-cors==5.0.1
 - openai==0.28.0
 - python-dotenv==1.0.1
 - gunicorn==21.2.0
 - pyjwt==2.3.0
 - uuid==1.30
 - Flask-SQLAlchemy==3.1.0
 - psycopg2-binary
-

Installation Instructions

Before running the application, ensure the following software is installed on your system:

End users: For people who just want to run the code without editing or viewing how it works, the following is required:

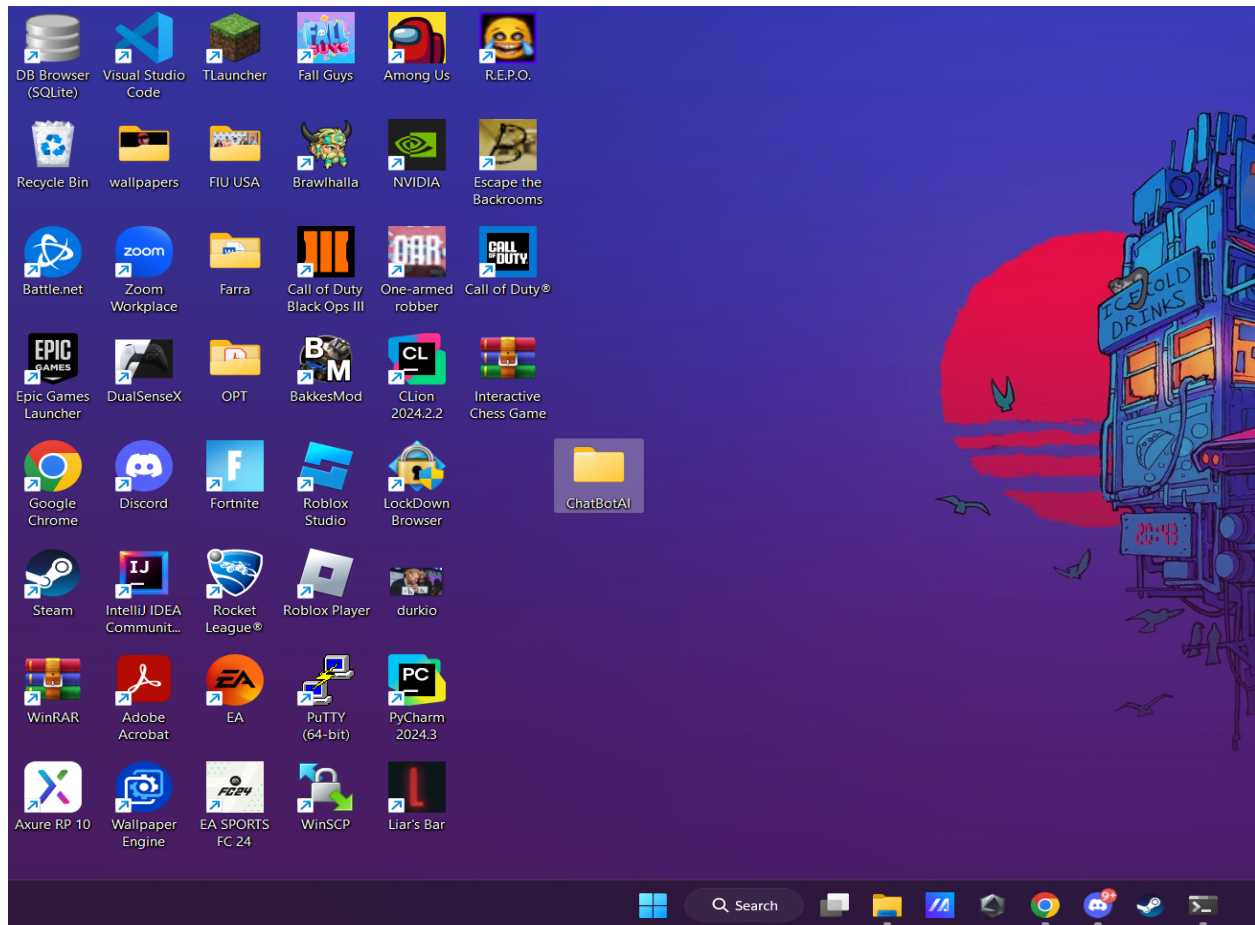
- **Google Chrome Browser or equivalent:** To open the frontend and interact with our chatbot.
- **Node.js:** Required for frontend development using React. Download from: <https://nodejs.org/>
- **Python 3.x:** Required to run the backend Flask server. Download from: <https://www.python.org/downloads/>

Developers: For developers who want to go over the code or the functionality of the project and how it works, the following is required:

- **Google Chrome Browser or equivalent:** To open the frontend and interact with our chatbot.
- **Node.js:** Required for frontend development using React. Download from: <https://nodejs.org/>
- **Python 3.x:** Required to run the backend Flask server. Download from: <https://www.python.org/downloads/>
- **Visual Studio Code (VS Code):** Recommended code editor for editing frontend and backend code. Download from: <https://code.visualstudio.com/>
- **PostgreSQL:** Used as the relational database to store user data, chatbot configurations, and conversation history. Download from: <https://www.postgresql.org/download/>
- **pgAdmin:** Optional GUI tool to manage your PostgreSQL database. Download from: <https://www.pgadmin.org/>

Download the Code

Our code is attached in the submission, you would go ahead and download it in a directory that is easily accessible by you as a user. **(Desktop is recommended as a directory):**



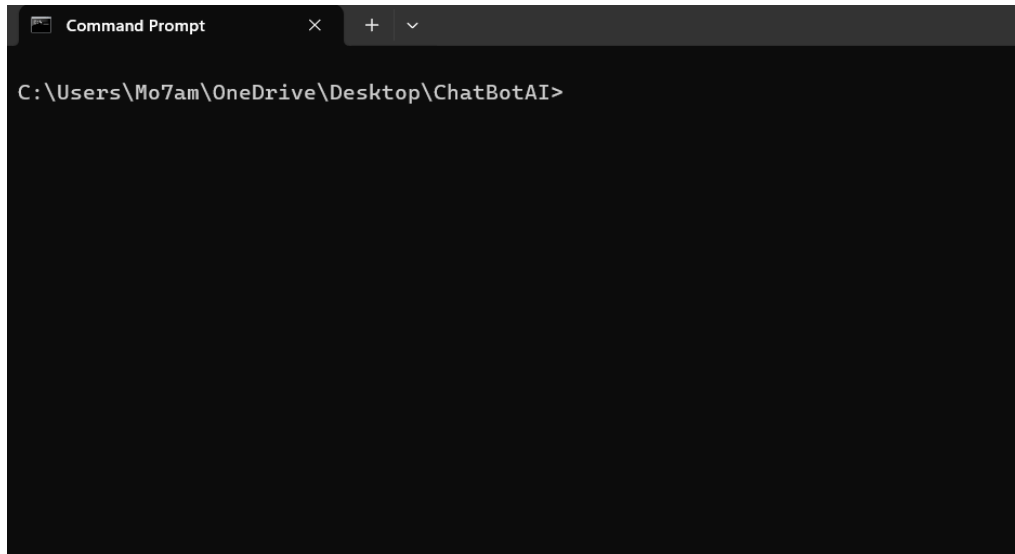
Command Prompt or VS Code Terminal Setup:

After downloading the code and downloading the requirements mentioned above, it is time to run the frontend and backend to be able to use the chatbot. You can either run it using VS Code Command Prompt terminal or Command Prompt itself.

Follow the steps below to setup your command prompt:

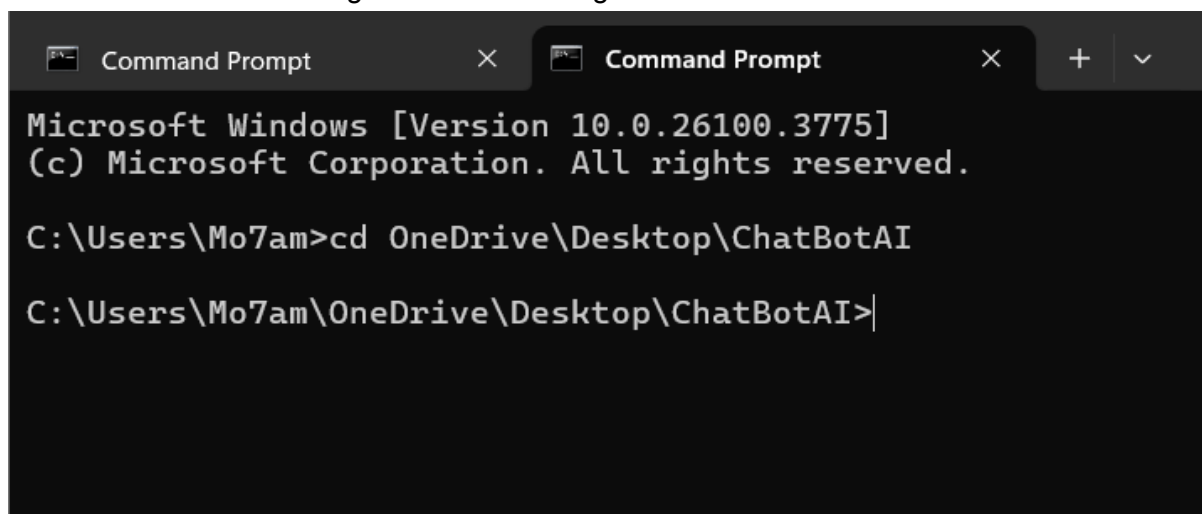
Open Command Prompt: Launch the command prompt app and navigate to the directory where the source code using the cd command

(e.g. `cd C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI`)



Repeat the process:: Open another command prompt tab and navigate to the same directory.

You should have something similar to the image below:



Frontend Setup

After setting up both of your command prompt tabs, it is time to work on the frontend setup to run the frontend. Follow these steps to set up the frontend (React.js app):

Navigate to your frontend folder: In your first terminal you would type the following to navigate to the frontend folder of the source code: **cd frontend**

Run the frontend react website: Once you are inside the frontend, you would run the following command to launch the frontend website of our chatbot locally: **npm run dev**

A link similar to the one below would generate and you would press on it using CTRL + Left Click or copying it and pasting it in your browser.:

```
C:\WINDOWS\system32\cmd. x Command Prompt
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Mo7am>cd OneDrive\Desktop\ChatBotAI

C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI>cd frontend

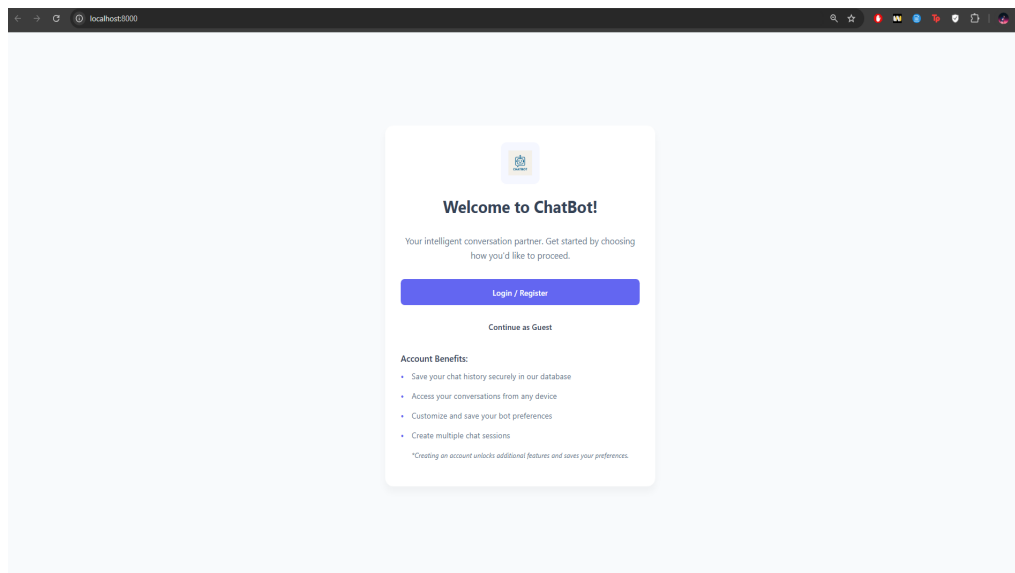
C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\frontend>npm run dev

> frontend@1.0.0 dev
> vite

VITE v6.2.4 ready in 172 ms

→ Local:    http://localhost:8000/
→ Network:  use --host to expose
→ press h + enter to show help
```

The link should take you to our chatbot website.



Backend Setup

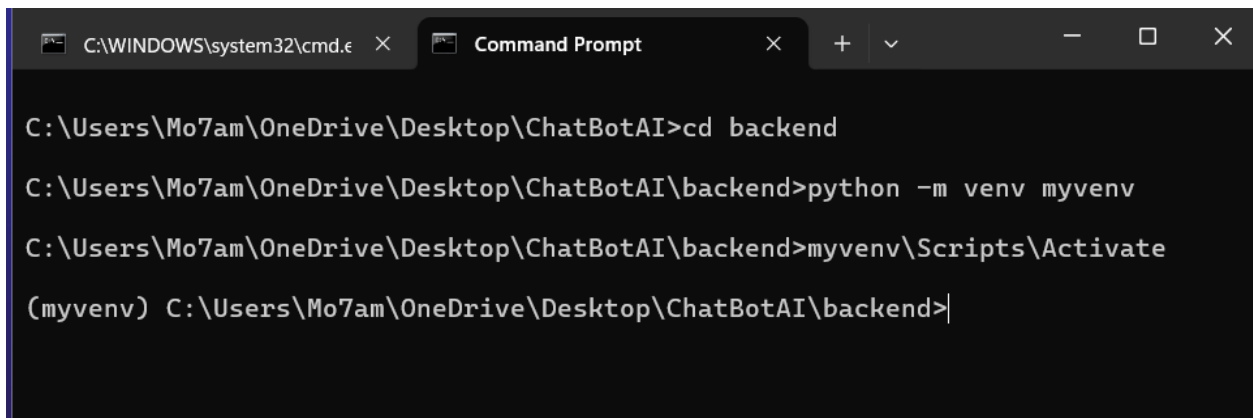
After setting up the frontend react website, it is time to work on the backend setup to be able to interact with the chatbot through the frontend website:

Navigate to your backend folder: In your second terminal you would type the following to navigate to the frontend folder of the source code: **cd backend**

Create a python virtual environment (venv): Once you are inside the backend, it is recommended to create a virtual environment (venv) to install the python libraries to isolate dependencies. Type and run the following command to create a venv:

python -m venv chatbot-venv

Activate the Virtual Environment: Type and run the following to activate the venv:
myenv\Scripts\Activate



```
C:\WINDOWS\system32\cmd.exe X Command Prompt X + v - □ X

C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI>cd backend

C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>python -m venv myenv

C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>myenv\Scripts\Activate

(myenv) C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>|
```

Install Dependencies: Inside your venv, you would Install the required Python dependencies using the following command:

pip install -r requirements.txt

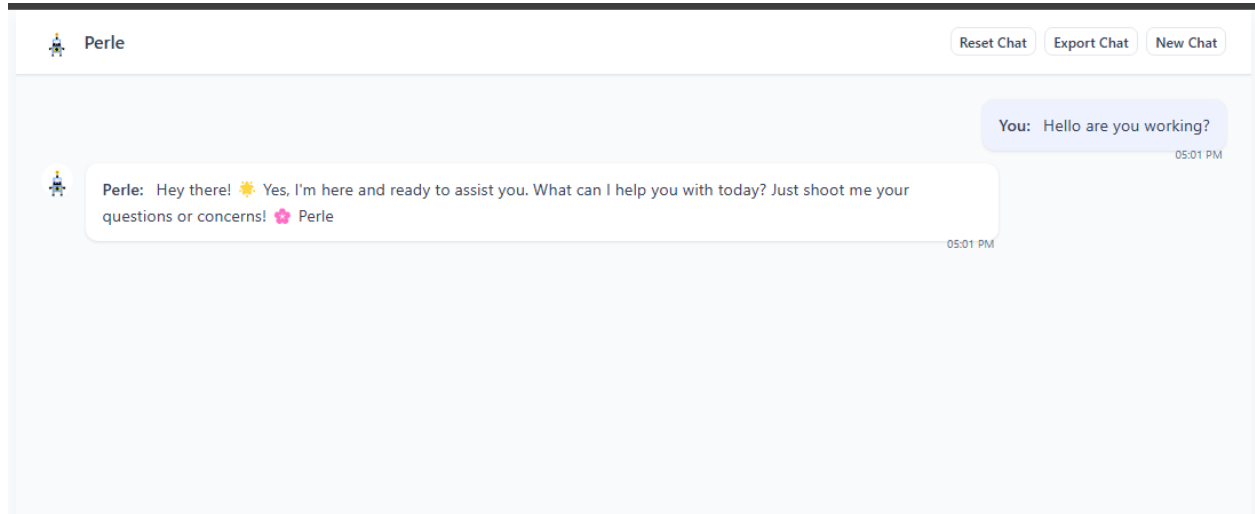
```
C:\WINDOWS\system32\cmd.exe X Command Prompt - pip insta X + v
C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI>cd backend
C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>python -m venv myenv
C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>myenv\Scripts\Activate
(myenv) C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>pip install -r requirements.txt
Collecting Flask==3.1.0 (from -r requirements.txt (line 1))
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting flask-cors==5.0.1 (from -r requirements.txt (line 2))
  Using cached flask_cors-5.0.1-py3-none-any.whl.metadata (961 bytes)
Collecting openai==0.28.0 (from -r requirements.txt (line 3))
  Using cached openai-0.28.0-py3-none-any.whl.metadata (13 kB)
Collecting python-dotenv==1.0.1 (from -r requirements.txt (line 4))
  Using cached python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Collecting gunicorn==21.2.0 (from -r requirements.txt (line 5))
  Using cached gunicorn-21.2.0-py3-none-any.whl.metadata (4.1 kB)
Collecting pyjwt==2.3.0 (from -r requirements.txt (line 6))
  Using cached PyJWT-2.3.0-py3-none-any.whl.metadata (4.0 kB)
```

Run the Python Flask Application: After installing the Python libraries required, start the Flask server by using the following command:

python app.py

```
(myenv) C:\Users\Mo7am\OneDrive\Desktop\ChatBotAI\backend>python app.py
API Key loaded: Yes
Database tables created successfully!
Default personalities initialized!
2025-04-16 16:58:51,074 - __main__ - INFO - Starting server on port 5000, debug mode: False
* Serving Flask app 'app'
* Debug mode: off
2025-04-16 16:58:51,099 - werkzeug - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.175.1.218:5000
2025-04-16 16:58:51,100 - werkzeug - INFO - Press CTRL+C to quit
2025-04-16 16:59:15,922 - werkzeug - INFO - 127.0.0.1 - - [16/Apr/2025 16:59:15] "GET / HTTP/1.1" 404 -
2025-04-16 16:59:16,129 - werkzeug - INFO - 127.0.0.1 - - [16/Apr/2025 16:59:16] "GET /favicon.ico HTTP/1.1" 404 -
```

Congratulations! Your AI-Customizable ChatBot should be ready to use.



Accessing the Database (For developers ONLY):

For our chatbot project we used a **public PostgreSQL database** that is hosted on Render. In case you are willing to check it out and see how we store user data, messages(user messages and chatbot responses), chatbot customization, etc. Follow the following steps:

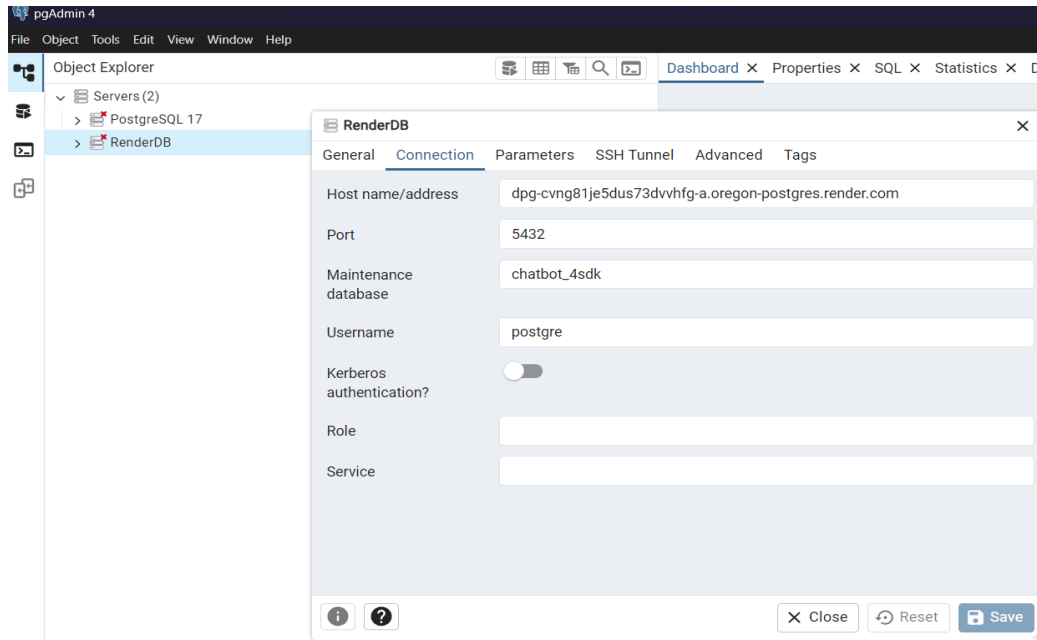
Install PostgreSQL and pgadmin4: Refer to the installation instructions at the beginning for installing pgadmin and PostgreSQL.

Launch pgadmin: Open pgadmin4 and press on **Add New Server** and then **Connections** to add the details of our public database to be able connect to it and manage it from your end.

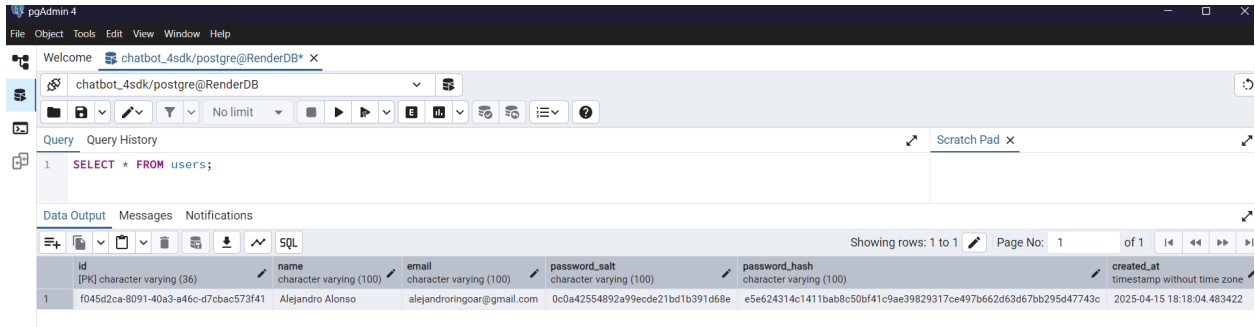
Add the following information in the Connection section:

- Host name/address: dpg-cvng81je5dus73dvvhfg-a.oregon-postgres.render.com
- Port: 5432
- Maintenance database: chatbot_4sdk
- Username: postgre

Save everything and you would be able to connect to our database and run SQL queries to view data stored in tables.



Below is an example of running an SQL query on the users table which is used to store user info:



User Manual

Creating a Chatbot

1. **Navigate to Chatbot Creation Page:** On the homepage, you'll find an option to create a new chatbot. Click "Create Chatbot" to begin.
2. **Provide a Name:** Enter a unique name for your chatbot. This name will appear in the user interface when you interact with the bot.
3. **Select a Visual Character:** Choose a character to visually represent your chatbot. Available options include various cartoon-style avatars or professional icons.
4. **Choose a Personality:** Select one of ten predefined personalities for your chatbot. The chatbot's responses will be tailored based on the selected personality, affecting its tone, language style, and behavior.
5. **Finalize the Chatbot:** After filling out the form, click "Create" to finalize the creation of your chatbot. You'll be taken to the chat interface to begin interacting with it.

Starting a Conversation

1. **Start a Chat:** Once your chatbot is created, click "Start Chat" to initiate a conversation.
2. **Interacting with the Bot:** Type your message in the input box and hit "Enter" or click the send button. The chatbot will generate a response based on the selected personality and previous interactions.
3. **Continuing a Conversation:** If you've interacted with the chatbot before, the conversation will persist across sessions. The chatbot will remember past messages and provide context-aware responses.

Exporting Conversations

1. **Export Chat History:** If you wish to save a record of your conversation, you can click the "Export" button at any time during the chat.
 2. **Choose Export Format:** The conversation will be exported as a text or JSON file, which can then be saved to your device for later reference
-