OpenAI Chatbot

Azure Services

Fawz Mehfil

Objective

I did 4 total projects, a Flask App + Azure SQL local deployment project, Azure VM with Apache Tomcat local project, and a AWS MediaConvert Workflow project. This is the main project. Deploying a full stack web application where users can chat with a GPT-3.5 assistant using a React frontend and Flask backend. The assistant is powered via Azure OpenAI.

Frontend	React (built with npm run build)
Backend	Flask (Python web framework)
AI Model	Azure OpenAI gpt-35-turbo
Hosting	Azure App Service

Required Services

App Service	Host Flask + React app (Python runtime)
Azure OpenAI	GPT API access via your resource
Resource Group	Organize App Service and related assets
App Service Plan	Compute resource backing the App Service

How the service works from the application standpoint:

1. Frontend (React)
Displays a simple chat UI (ChatBox.js)

On send:

- Pushes user message to /api/chat

- Appends user + assistant responses to UI
- Messages are stored in local state, so chat history persists during session

2. Backend (Flask)
Hosts two routes:

- /api/chat: POST endpoint to call Azure OpenAI
- Serve React app from the static/ folder

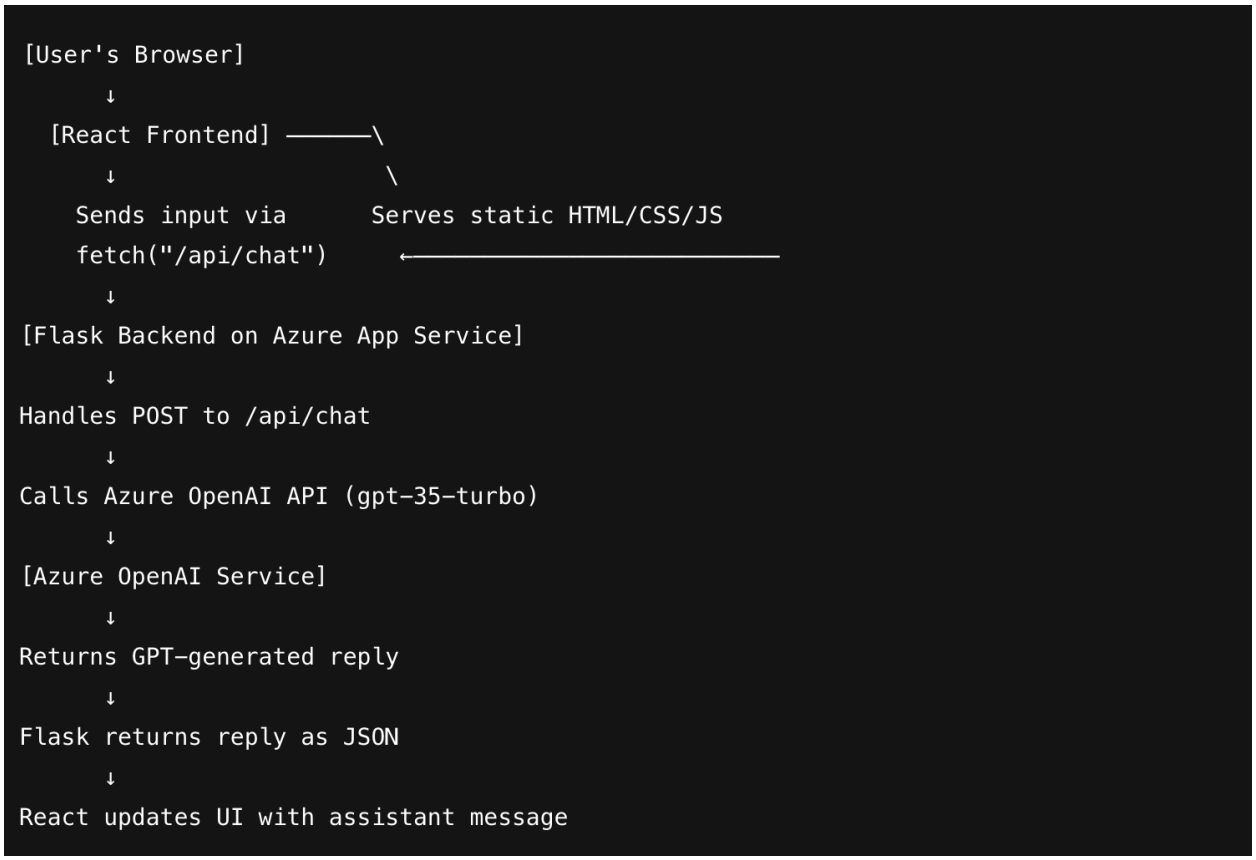
On receiving user input:

- Sends it to Azure OpenAI chat.completions.create()
- Returns the assistant's response to the frontend

3. Azure OpenAI Integration
API call includes:

- api_key from environment variable
- deployment name (gpt-35-turbo)
- api_version: e.g., 2023-05-15
- Endpoint format: https://<your-resource>.openai.azure.com/

Structure



1. Resource Group

- What it does:
  A Resource Group is a logical container that holds all the Azure resources related to your project—such as your App Service and Azure OpenAI resource.

- Role in structure:
  It acts as an organizational unit. Everything for this ChatGPT app—compute, AI, network, and diagnostics—is grouped under a single resource group.

2. Azure App Service

- What it does:
  Azure App Service hosts your full Flask backend along with the frontend (React build) under a single deployment (via zip).

- Role in structure:

- Frontend delivery: Serves your index.html, CSS, and bundled JS (React app) from the /static directory.

- API handling: Exposes the POST /api/chat endpoint for incoming messages and forwards those to Azure OpenAI.

- Binding environment variables: Azure App Service injects your OPENAI_API_KEY, AZURE_ENDPOINT, and AZURE_DEPLOYMENT_NAME into the runtime environment of the Flask app so that you don't hardcode sensitive values.

- Deployment point: You use az webapp deploy --src-path app.zip to push code here.

3. Azure OpenAI Resource

- What it does:
  Hosts the GPT model (e.g., gpt-35-turbo) and serves completions when requested.

- Role in structure:

  - Responds to API requests that your Flask backend makes using the AzureOpenAI SDK.

  - Requires a valid endpoint and API key to authenticate.

4. Environment Variables (in App Service Configuration)

- Purpose:
  To securely store and access sensitive data like API keys and deployment names without hardcoding them in source code.

- Used for:

  - OPENAI_API_KEY – Authorizes your backend with Azure OpenAI.

  - AZURE_ENDPOINT – Tells your app where to send requests.

- AZURE_DEPLOYMENT_NAME – Specifies which model deployment to use (e.g., chat-gpt).

Database Integration

1. Created Azure SQL Database:
    a. Created SQL server (e.g., chat-sql-server-fawz)
    b. Created SQL database (e.g., chatdb) inside it
2. Configured Authentication:
    a. Initially had AAD-only authentication enabled
    b. Disabled it so you could use SQL username/password
3. Reset SQL Admin Credentials:
    a. Set admin username to something like sqladmin@chat-sql-server-fawz
    b. Reset the password using Cloud Shell
4. Whitelisted Client IP:
    a. Went to the "Firewalls and virtual networks" tab on the SQL Server
    b. Added your client IP to allow access

5. Created Table for Messages:

Used the Azure Query Editor or SSMS to run:

```
CREATE TABLE chat_history (
    session_id NVARCHAR(64),
    role NVARCHAR(16),
    message NVARCHAR(MAX),
    timestamp DATETIME DEFAULT GETDATE()
     );
```

6. Set Environment Variables:

In Azure → App Service → Configuration, added:

SQL_SERVER=chat-sql-server-fawz.database.windows.net

SQL_DATABASE=chatdb

SQL_USERNAME=sqladmin@chat-sql-server-fawz

SQL_PASSWORD=YourPassword

Problems

1. App Not Loading on Browser (White Screen / No Output)

- Cause: Frontend was either not built or not placed correctly in the static/ directory.

Fix: I ran:

```
cd frontend
npm install
npm run build
```

- Then moved the contents of the frontend/build/ directory into static/ so Flask could serve them.

2. Page Loads, But Assistant Response Missing

- Cause: Flask responded to /api/chat, but the frontend didn't display the assistant's reply.

Fix: Identified that the reply wasn't being properly appended to the messages array in ChatBox.js. Updated:

```
setMessages([...newMessages, { role: "assistant", content: data.reply }]);
```

3. ModuleNotFoundError: No module named 'openai'

- Cause: Azure App Service didn't have the correct dependencies installed.
- Fix:
    - Ensured openai was included in requirements.txt.
    - Verified the correct version (openai==1.82.0) matched the usage of AzureOpenAI.

4. Error code: 401 - Invalid API Key or Endpoint

- Cause: The API key or endpoint for Azure OpenAI was incorrect or missing.
- Fix:
    - Logged into Azure Portal and copied the correct Key and Endpoint from the Azure OpenAI resource.
    - Went to App Service → Configuration → Application Settings:
        - Added:
            - OPENAI_API_KEY
            - AZURE_ENDPOINT
            - AZURE_DEPLOYMENT_NAME
    - Restarted the app.

5. Couldn't Find Deployment Name

- Cause: You were unsure where to get the GPT model deployment name from.
- Fix:
    - Navigated to the Azure OpenAI resource in the portal.
    - Went to the Deployments tab.
    - Found the name of the deployed model (e.g., gpt-35-turbo) and set it as AZURE_DEPLOYMENT_NAME.

6. Couldn’t Remember Resource Group

- Cause: You forgot which resource group you used for the App Service.

- Fix:

Ran:

bash
CopyEdit
```
az group list -o table
```

  - and reviewed names based on creation date.
  - Eventually identified the correct one used in deployment.

7. Logs Not Appearing or App Crashing Silently

- Cause: App was crashing but not showing errors.
- Fix:

Used:

```
az webapp log tail --resource-group <rg> --name <app-name>
```

  - This showed real-time startup errors, including the missing module and auth issues.

8. Static Files Not Loading

- Cause: React build was not correctly routed or missing.
- Fix:

Ensured Flask had this route:

python
CopyEdit
```
@app.route("/")
```

```
def serve():
    return send_from_directory(app.static_folder, "index.html")

@app.route("/<path:path>")
def static_files(path):
    return send_from_directory(app.static_folder, path)
```

Database Integration Problems

1. 10-Minute Timeout
    - Azure App Service timed out during deployment
    - This was due to Flask app not starting correctly

Resolved by:

- Setting correct `host="0.0.0.0"` and using `PORT` env variable
- Adding a `requirements.txt`
- Setting the startup command to `python app.py`

2. SQL Auth Failure (AAD-Only)
    - Couldn't reset your SQL password or connect
    - Reason: AAD-only authentication was enabled

Resolved by:

- Disabling "Microsoft Entra-only authentication" in SQL Server settings

3. Couldn't Reset SQL Admin Password
    - Tried resetting via CLI but failed due to AAD-only
    - Needed to disable it first (see above)

4. “Argument 1 must be a string” Error
    - This came from passing a `NoneType` or broken `pyodbc` connection

Resolved by:

- Double-checking that all SQL env variables were set
- Confirming the `pyodbc.connect(...)` connection string was correct

5. App Service Couldn’t Start
    - Continuous “Starting the site…” messages during deployment

Resolved by:

- Ensuring:
    - `requirements.txt` was present
    - App had correct folder structure
    - Startup command was defined

6. Environment Variables Not Found

- Code couldn't find `SQL_SERVER` etc.

Resolved by:

- Setting them in Azure → App Service → Configuration, redeploying after that

7. ResourceNotFound for Web App or Group

- This was caused by:
    - Wrong web app name (e.g., `your-chatgpt-app`)
    - Wrong resource group name

Resolved by:

- Verifying names in Azure Portal

Updating deployment command with:

```
--resource-group <your-correct-group>
--name <your-actual-app-name>
```

8. Missing SQL Server from List

- After creating the SQL Server, it didn't show up immediately

Resolved by:

- Waiting a minute and manually refreshing Azure Portal