Career Compass

Group 2

Here is where your Career begins!

In collaboration with ***Dr. Jim Egenrieder and Thinkabit Lab.***

Table of contents

About Us

- Anvita Sanjay Karne (Data Visualization)
- Atharva Bipin Chouthai (Backend)
- Drushya Chikkaballapur Kumar (Frontend, AWS)
- Julia Marie Gutgesell (Data Visualization)
- Mahita Maddipati (Frontend)
- Rajat Ratnadeep Belgundi (Backend)
- Shubham Laxmikant Deshmukh (Backend, Frontend)
- Siddhi Kasera (Frontend, AWS)

Introduction

1. **Career Compass,** an AI-powered web application that provides personalized, interactive career guidance through a **Retrieval-Augmented Generation (RAG) system**.

2. Most tools are static — **CareerCompass** uses **LLM + O*NET 29.1** data for dynamic, chat-based exploration.

3. **Project Goals:**
   To create an intuitive and intelligent advisor that helps users explore career paths, skill gaps, and educational needs with **real-time AI responses**.

4. **Target Users:**
   High school/college students, career switchers, and educators.

5. **Key Features:**
   - RAG system with **OpenAI embeddings** + **ChromaDB** + **LangChain**
   - Seamless **chat + data dashboard** integration
   - Interactive insights from **O*NET + BLS**
   - Scalable, open-source, and **AWS-hosted** platform

Backend

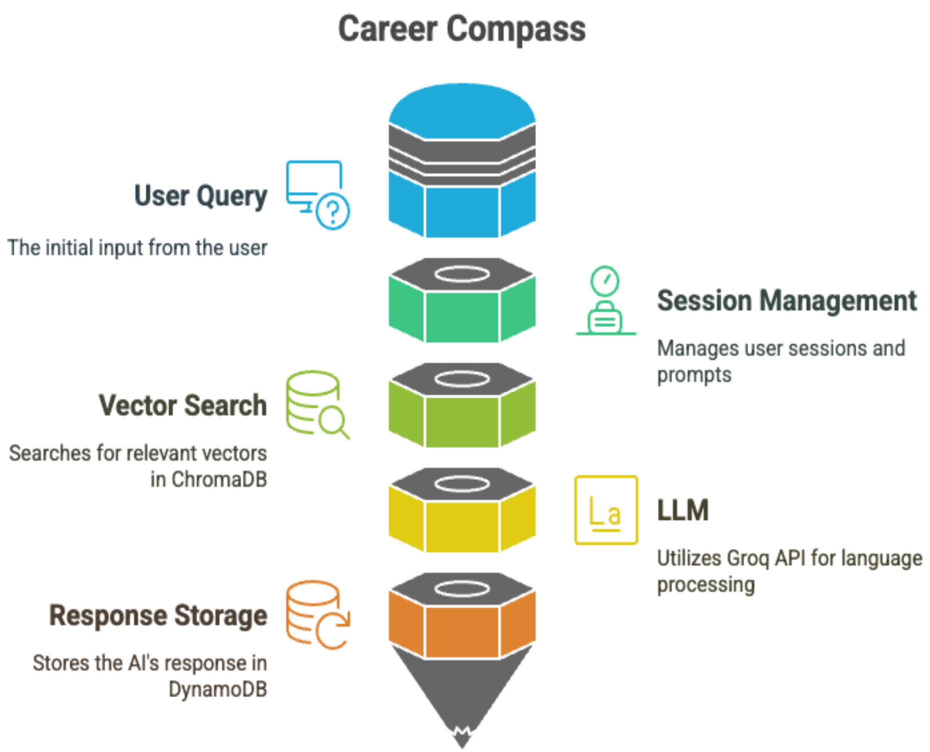
- **ONET → Embeddings + ChromaDB**
  Job data embedded via **Open AI's text-embedding-ada-002**, stored in **ChromaDB** for semantic retrieval.

- **LLaMA 3 via Groq API**
  Ultra-fast responses from **LLaMA3-70B** via **Groq's** compatible API.

- **LangChain for Context and Chat Memory**
  Structures prompts using schema messages to enable multi-turn memory.

- **User Sessions + Storage**
  Stores/resumes chat history per user using **DynamoDB**.

- **IDE's and Tools**: Jupyter Notebook, Visual Studio Code, Groq API, Open AI, LangChain, FAST API, Dynamo DB, GitLab.



Backend

Problems Faced

- **O*NET Data Extraction**
  Mapping multiple data files into semi-structured **JSONs** for embeddings was time-consuming.

- **LLM Deployment Decisions**
  Weighing local (**ollama models**) vs. **API**-based models (e.g., **OpenAI**, **Groq**) required trade-offs in speed, cost, and ease.

- **Prompt Alignment**
  Ensuring the chatbot stayed grounded in **O*NET** data—avoiding hallucinations.

- **LangChain Integration**
  Managing multi-turn memory + frontend/backend sync (via **AWS DynamoDB**) increased system complexity.

Frontend

- **Wireframing and Prototyping with Figma**
  Low-fidelity and high-fidelity wireframes were created using Figma to define user flows, chatbot layout, and dashboard structure before development.
- **Next.js (React Framework)**
  Used to build a fast, server-rendered React application with dynamic routing and modular components.
- **Tailwind CSS**
  Enabled responsive, utility-first styling for a consistent and modern user interface across all screen sizes.
- **FastAPI Integration (for Frontend API Calls)**
  Connected the frontend to the backend via asynchronous **REST API calls** for real-time chatbot responses and chat history handling.

Frontend

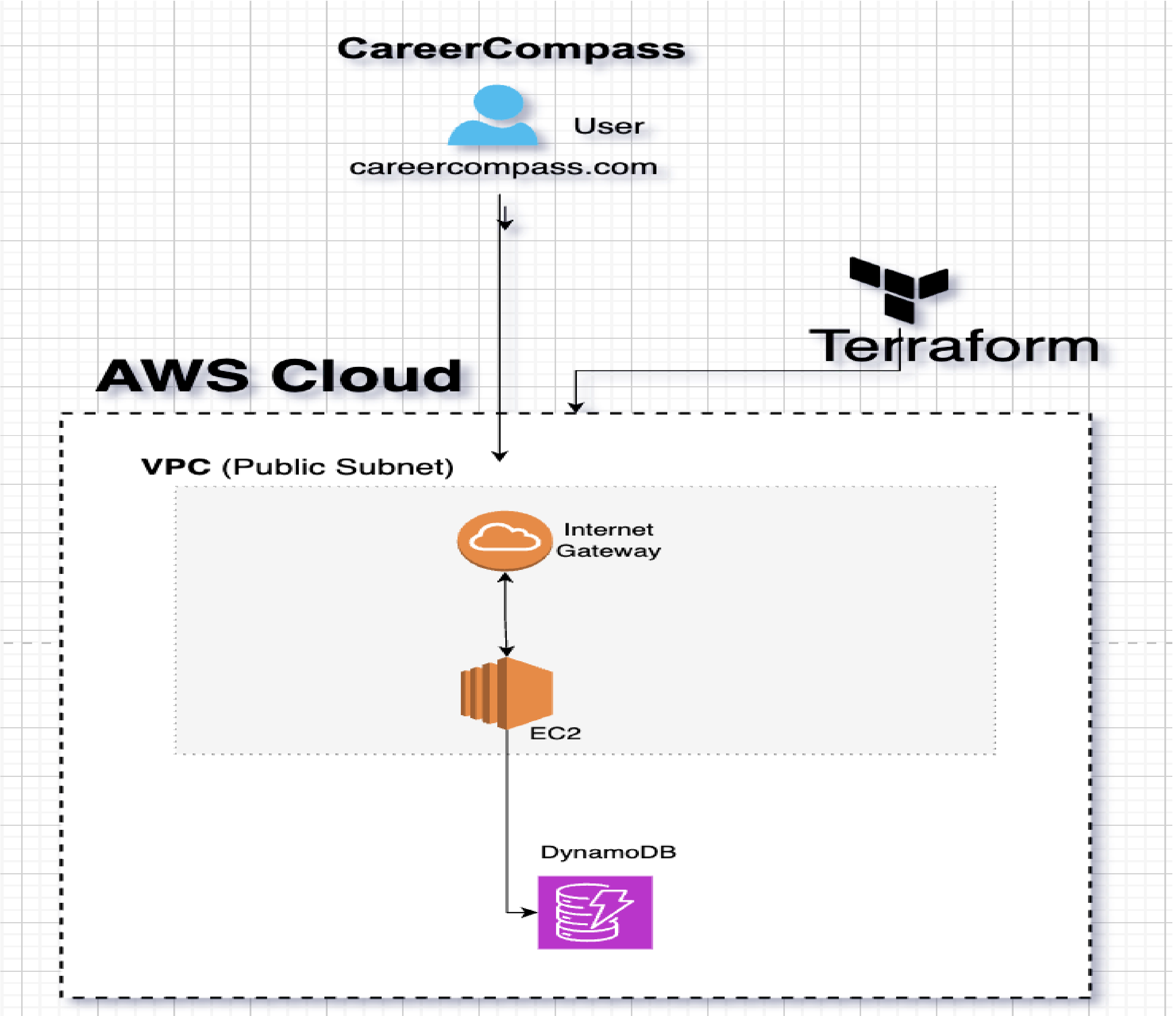
Problems Faced

1. **Responsiveness Issues**
   Early UI did not scale well on smaller screens; addressed by enhancing layout with Tailwind's responsive utilities.

2. **Unreadable Bot Responses**
   Bot replies included raw formatting symbols like `\n` and `**`; fixed using `react-markdown` for clean rendering.

3. **Chat History Display and Management**
   Initially, chat histories were not displaying correctly or were tied to the wrong session; resolved by assigning unique session IDs per user and rendering saved conversations based on timestamp and preview data.

AWS Infrastructure



Data Visualization

- **Tableau Public**
  Platform used for building visualizations, chosen for its accessibility, interactive capabilities, and ease of sharing insights.

- **Two visualization formats**:
  - A concise **overview dashboard** designed to provide users with a quick, high-level summary of key data points.

  - **Two data stories**:
    - **Salaries:** guides users through national and occupational wage trends
    - **Employment:** presents insights into occupational demand across various sectors.

- Emphasized user engagement through **interactivity, filtering options, and a narrative flow** to improve data comprehension.

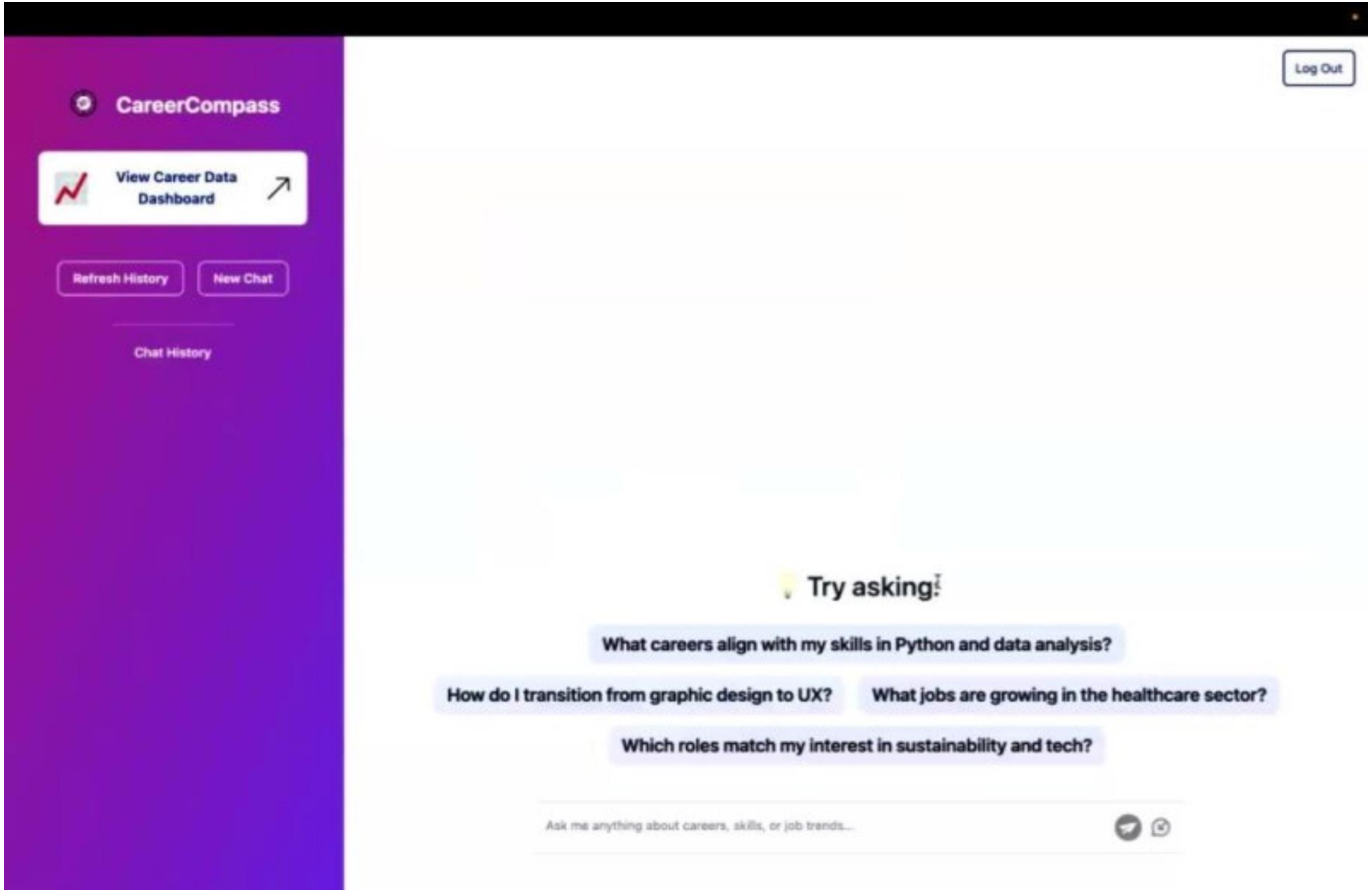
Data Visualization

Problems Faced

1. **Data**
   The O*NET database **lacked essential numerical data**, such as employment figures and wage statistics. Relevant data for visualization had to be sourced from U.S. Bureau of Labor Statistics.

2. **Comprehension**
   Initially the project featured only an interactive dashboard for a self-guided experience but **lacked guidance**. Some users – especially younger audiences – may struggle to understand key insights. In response, we developed data stories that **combined interactive visualization with narrative elements.**

Demo



Log Out
CareerCompass
View Career Data Dashboard
Refresh History
New Chat
Chat History
Try asking:
What careers align with my skills in Python and data analysis?
How do I transition from graphic design to UX?
What jobs are growing in the healthcare sector?
Which roles match my interest in sustainability and tech?
Ask me anything about careers, skills, or job trends...

Conclusion

Reflection & Retrospective

- Selecting only the most relevant O*NET data ensures efficient, accurate RAG responses
- Prompt engineering guides the LLM to deliver grounded, task-specific answers
- Sequential multi-turn memory maintains coherent, user-aware conversations
- Tight integration of context, memory & storage modules provides a consistent user experience

Future Directions

- Integrate additional career-related datasets to broaden job exploration
- Offer **resume suggestions** tailored to each selected role
- Surface **course & certification** links for skill development
- Add **skill-gap analysis** and personalized career-preparation tips

Thank You!

Do you have any questions on Career Compass?