Career Compass
Final Product Submission
Group 2

Link to Demo: https://code.vt.edu/cs-5934-spring25/group-2/career_compass/-/wikis/home

Product Description:

Career Compass is an AI-powered career guidance platform that helps students and job seekers make informed decisions by combining large language models with structured labor data like O*NET and U.S. Bureau of Labor Statistics. Unlike static career sites, it offers dynamic, personalized insights through a conversational chatbot and an interactive Career Insights Dashboard built in Tableau, all delivered through a responsive and intuitive frontend built with Next.js and Tailwind CSS. By integrating Retrieval-Augmented Generation (RAG), vector search, and prompt engineering, CareerCompass delivers real-time answers on skills, job outlooks, and educational paths—making career exploration more accessible, scalable, and effective for both early-career users and professionals navigating industry shifts.

Product functionalities:

CareerCompass: User Walkthrough

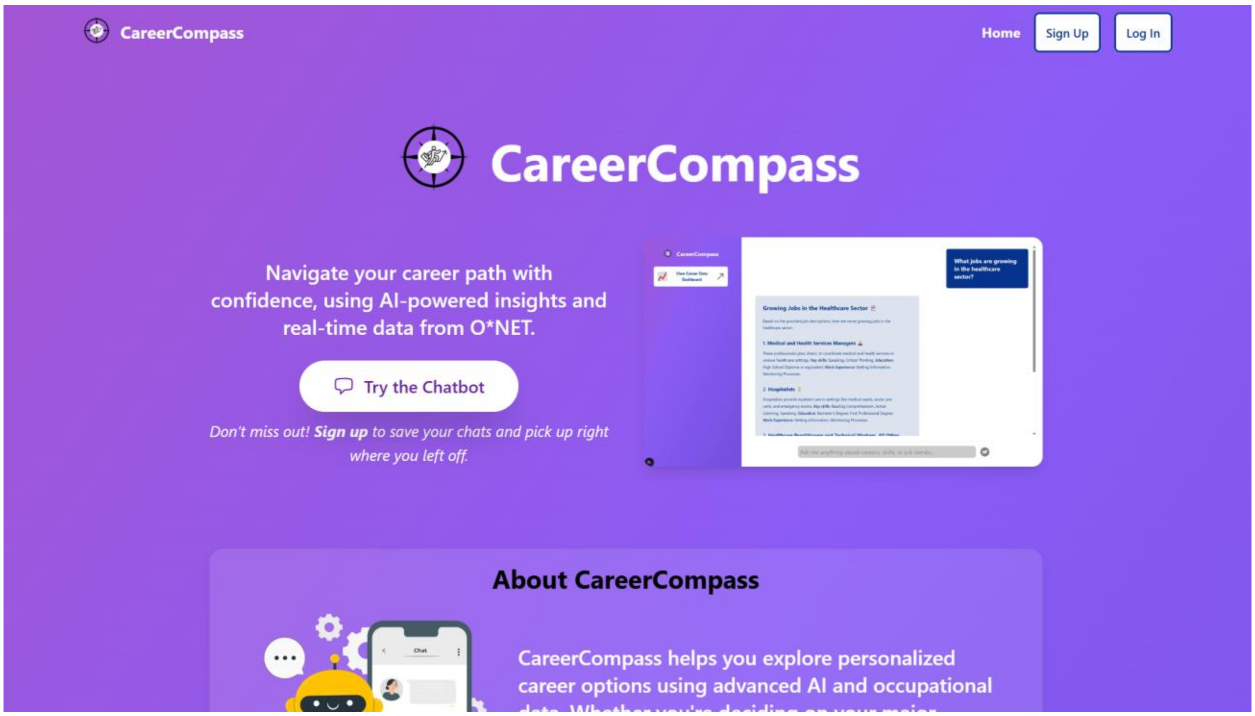
Users begin on an inviting landing page and can try the chatbot immediately. As they interact, they receive structured, explainable guidance powered by AI and real labor data. They can save chats, revisit past sessions, and explore visual dashboards that tell data stories about job opportunities, salaries, and employment growth.

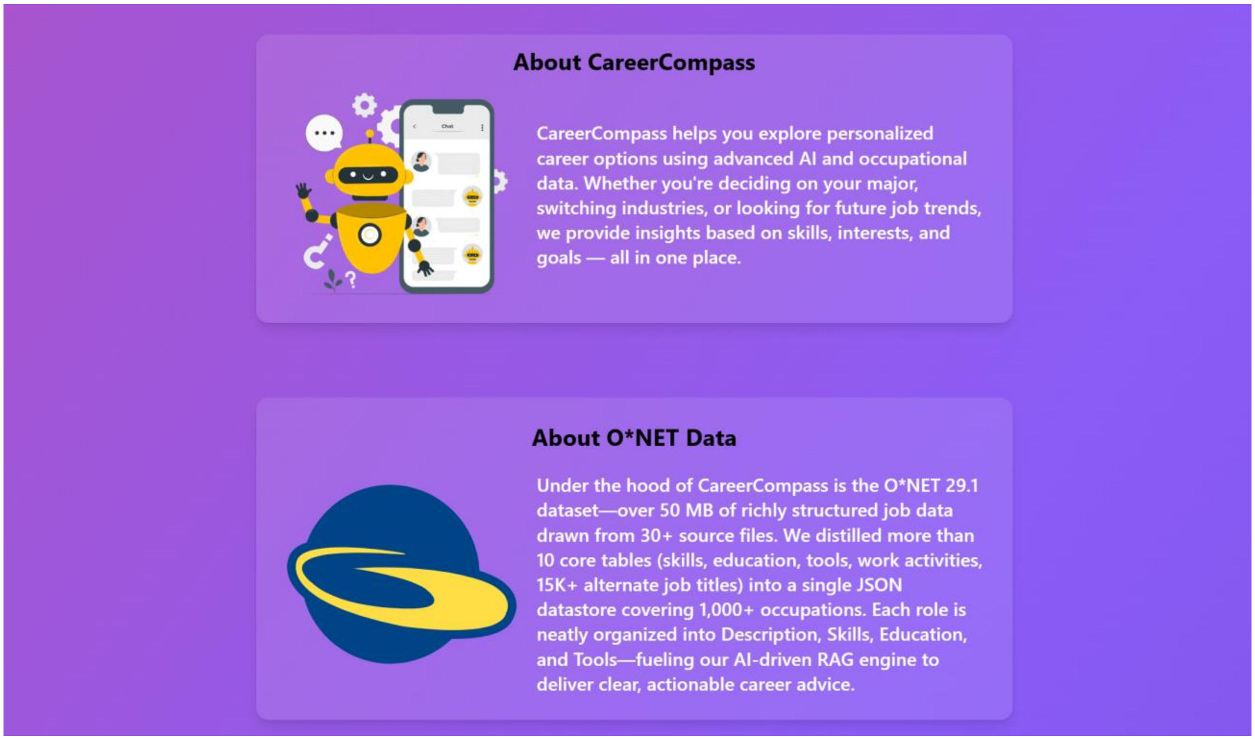
1. **Home Page**

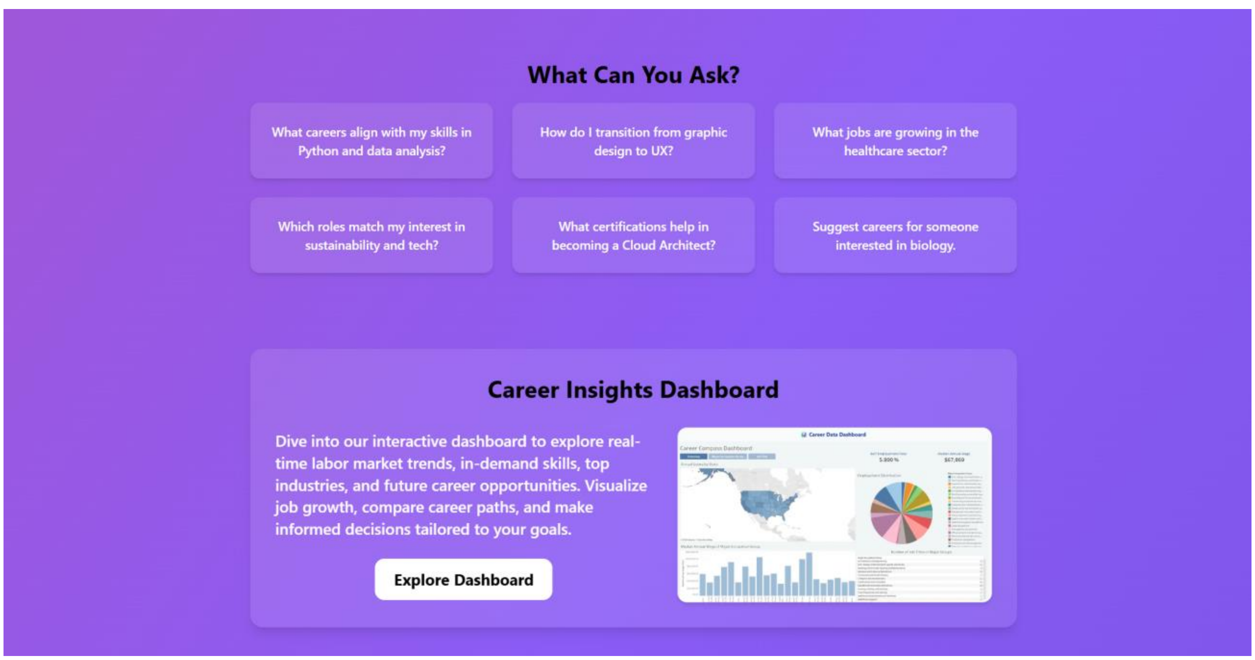
The Home Page serves as the initial entry point to CareerCompass, presenting a visually engaging layout that immediately communicates the platform's purpose. With a strong brand identity and a clear "Get Started" call-to-action, users are invited to explore further. The design is fully responsive, ensuring smooth scrolling and optimal viewing experiences on any device.

2. **Landing Page**

The Landing Page provides a brief introduction to the platform's capabilities, highlighting its AI-powered guidance and integration with reliable labor data such as O*NET. Here, users can preview and use the chatbot interface and receive an overview of the key features without needing to log in. This approach lowers entry barriers and encourages immediate interaction with the tool.

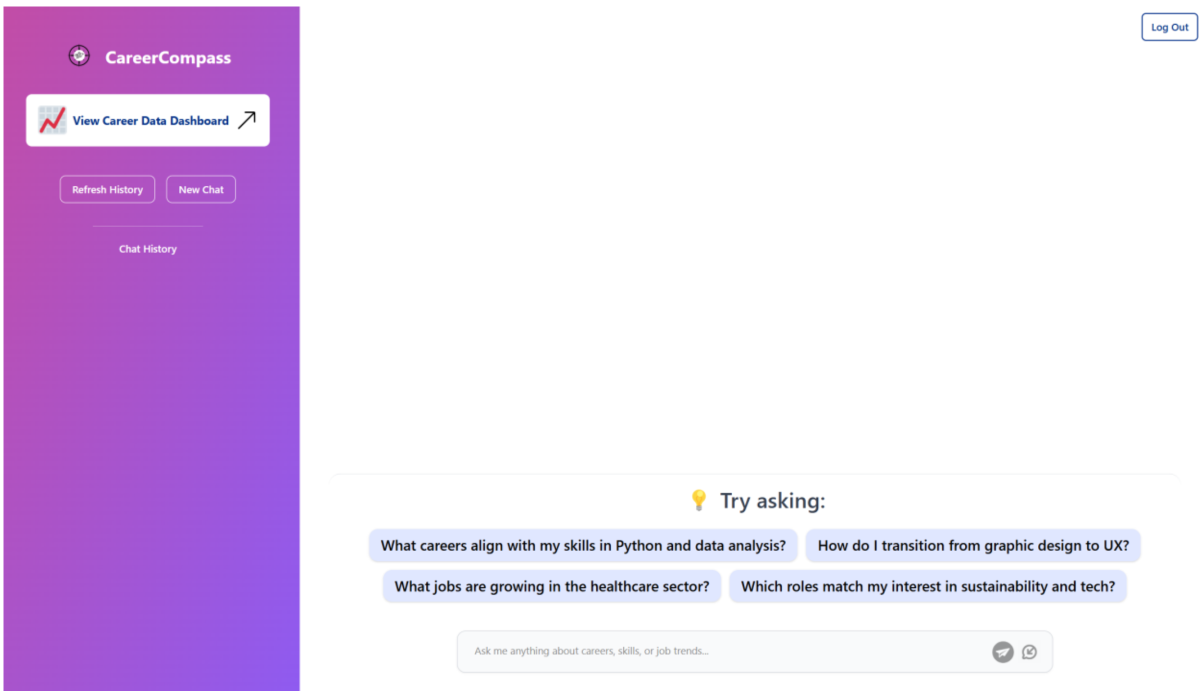



About CareerCompass
CareerCompass helps you explore personalized career options using advanced AI and occupational data. Whether you're deciding on your major, switching industries, or looking for future job trends, we provide insights based on skills, interests, and goals — all in one place.
About O*NET Data
Under the hood of CareerCompass is the O*NET 29.1 dataset—over 50 MB of richly structured job data drawn from 30+ source files. We distilled more than 10 core tables (skills, education, tools, work activities, 15K+ alternate job titles) into a single JSON datastore covering 1,000+ occupations. Each role is neatly organized into Description, Skills, Education, and Tools—fueling our AI-driven RAG engine to deliver clear, actionable career advice.


What Can You Ask?
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?
What certifications help in becoming a Cloud Architect?
Suggest careers for someone interested in biology.
Career Insights Dashboard
Dive into our interactive dashboard to explore real-time labor market trends, in-demand skills, top industries, and future career opportunities. Visualize job growth, compare career paths, and make informed decisions tailored to your goals.
Explore Dashboard
Career Data Dashboard

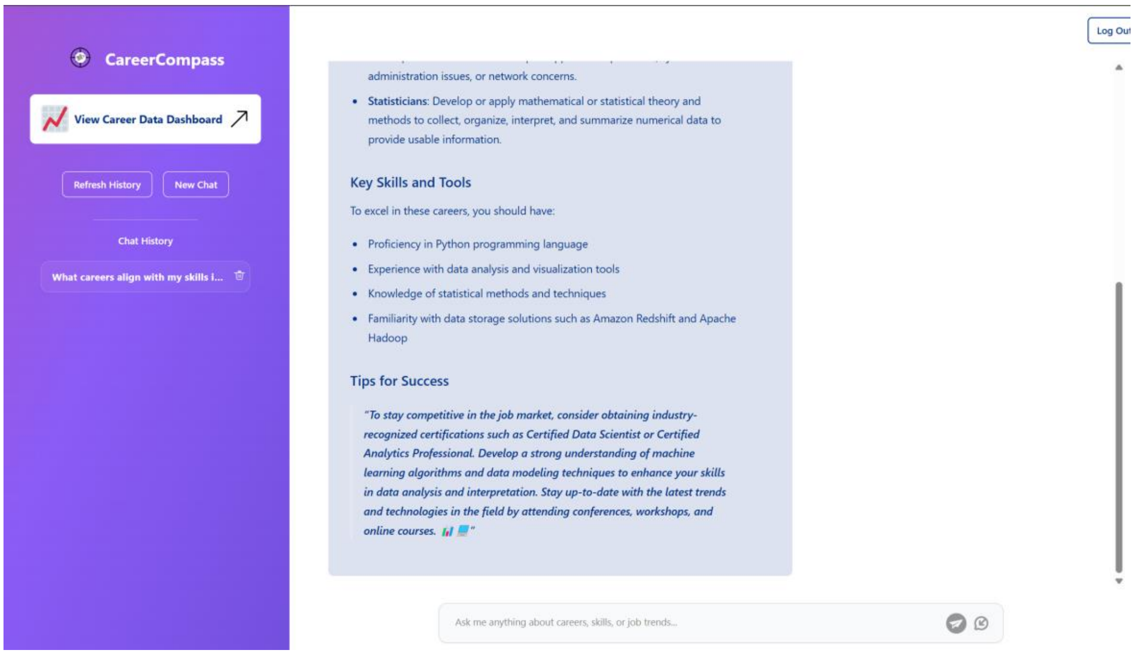
3. **Chat Interface – Starting a New Conversation**

In the Chat Interface, users engage directly with the AI to ask career-related questions, receiving structured and personalized responses in real time. There are a few example questions that the user can prompt. The interface includes a "New Chat" option that resets the session, allowing users to start fresh conversations effortlessly. It is designed for speed and clarity, ensuring that each new query is handled smoothly and responsively. Once users finish a conversation, they have the option to save it for future reference by clicking the Save Chat button.



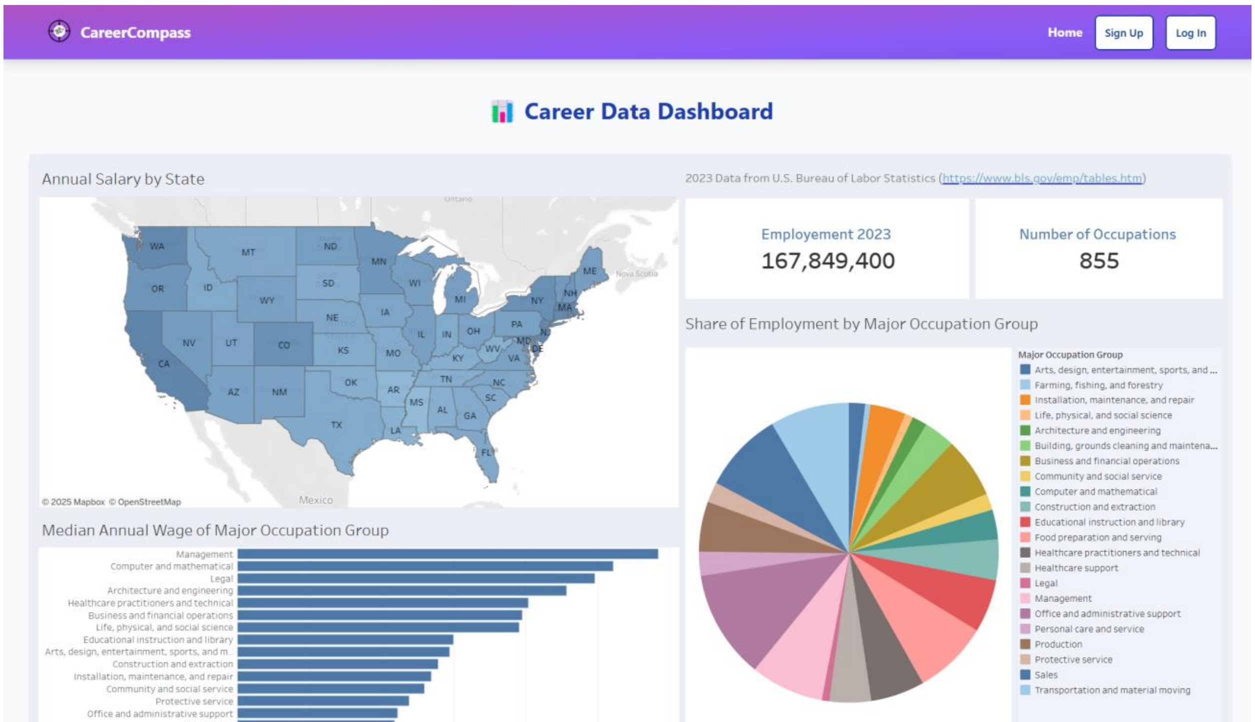
4. Save & Manage Chat History

The side navigation section has options for the user to view the data visualization dashboard, refresh the chat history or start a new conversation. The Refresh History function loads all previously saved chats with relevant timestamps and previews, making it easy to review past interactions. In addition, users can manage their saved chats with options to delete unwanted entries or retry them as needed.



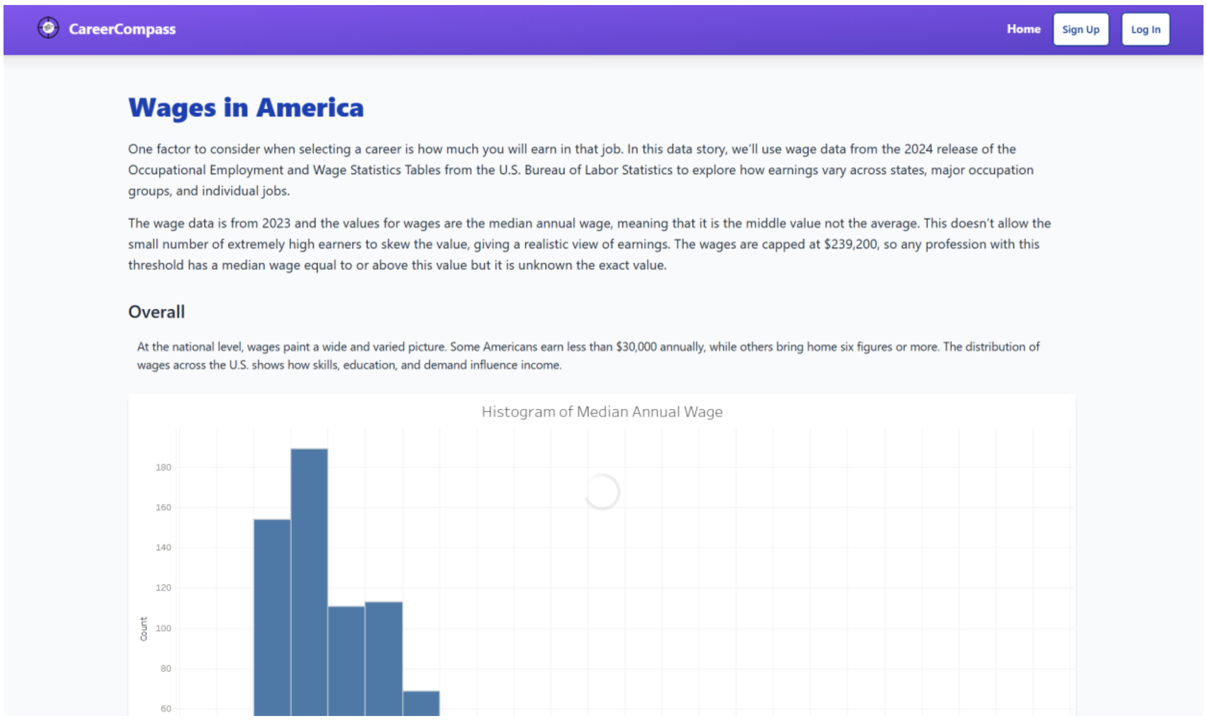
5. Career Insights Overview Dashboard

The Career Insights Overview Dashboard provides a bird's-eye view of essential job market metrics such as average wages, top roles, and employment trends. Built with Tableau and embedded using responsive containers, this dashboard helps users quickly understand the broader labor landscape. It acts as a launching point for further exploration of more detailed data stories on specific topics.



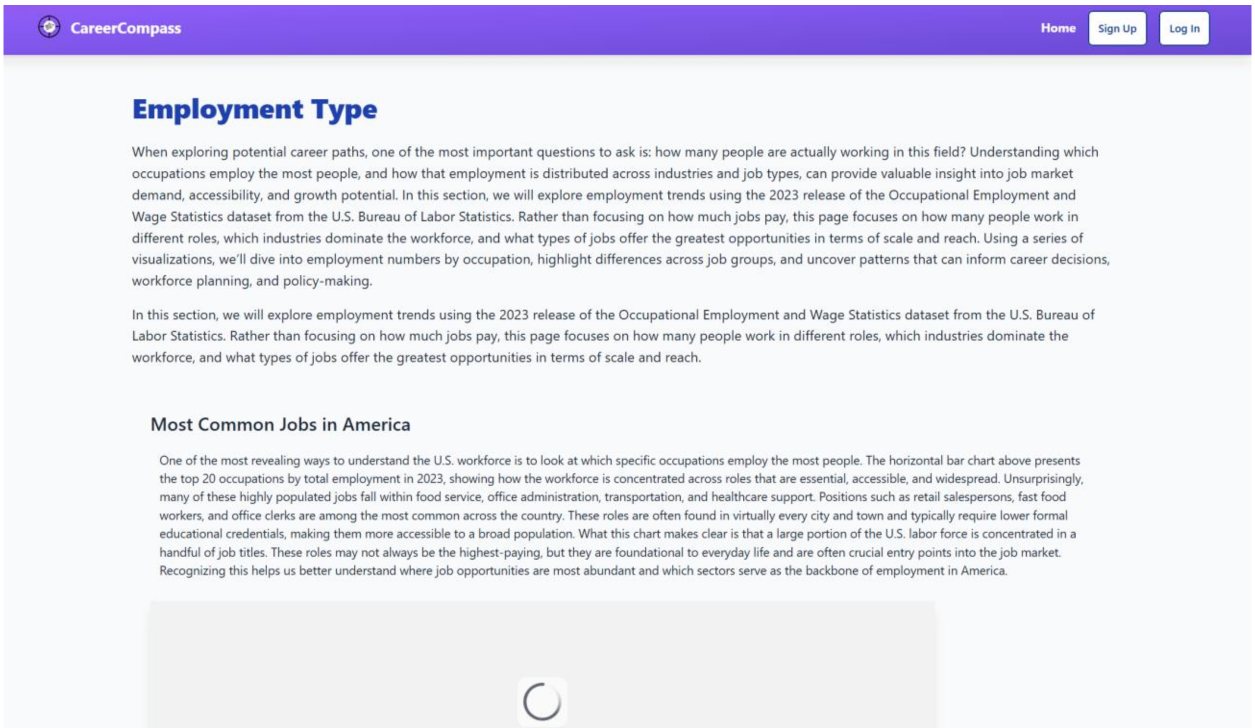
6. **Data Story: Wages in America**

   The Wages in America page offers an in-depth exploration of wage distribution across various roles and industries. It combines detailed narrative explanations with interactive Tableau visualizations, making complex compensation trends clear and accessible. This page empowers users to better align their career choices with realistic salary expectations.



7. **Data Story: Employment Trends**

   The Employment Trends page examines how job demand and employment patterns have shifted over time. Through clear, data-driven visualizations paired with concise explanatory text, users can easily identify which sectors are expanding and which are contracting. This insight aids in informed career planning and helps users anticipate future labor market changes.



Design:

The CareerCompass system was designed as a career guidance chatbot powered by Retrieval-Augmented Generation (RAG), using structured occupational data from the O*NET 29.1 database and embedding-based search. Our focus was to build a smart, explainable, and personalized system that goes far beyond generic chatbot templates by emphasizing structured knowledge retrieval, memory, and dynamic interaction. We had divided our project into three key areas namely the backend, which consists of the main retrieval and generation logic along with data processing, which was built using Next.js and Tailwind CSS to create a responsive, user-friendly interface that supports real-time chat, dashboard views, and seamless integration with backend APIs. Lastly the interactive visualizations present employment and wage data through dashboard and narrative-driven visualizations.

1. Backend:

1.1. Data Processing and Structuring

We began by extracting and transforming over 30 structured files from the O*NET 29.1 database, totaling ~50MB. This included data on skills, work activities, education levels, tools, alternate titles, and job zones, covering 1,000+ occupations.
Steps followed:

- Cleaned and standardized fields (trimmed whitespace, normalized codes).
- Merged records using O*NET-SOC (Standard Occupation Classification) as the primary key.
- Prioritized data using O*NET-provided importance scores (e.g., skills ranked by relevance).
- Added fallback placeholders like "Skills are unknown" for missing fields, to prevent hallucinations in LLM responses.
- Reformatted each job into structured Markdown-style sections: Description, Skills, Tools, Education, and Work Activities.

This restructuring allowed the LLM to generate highly contextual and explainable answers by referencing consistent, domain-specific data chunks.

1.2. Embedding and Vector Store

To enable efficient semantic search, we used OpenAI's text-embedding-ada-002 model, a state-of-the-art embedding model known for its balance between performance and cost. This model generates 1536-dimensional embeddings that capture rich semantic meaning, making it highly suitable for long-form career data from O*NET.
We embedded structured text chunks derived from job roles and stored them in ChromaDB, our chosen vector database, optimized for cosine similarity search.

Key design decisions and parameters:

- **Embedding model**: text-embedding-ada-002
  - Pros: High accuracy, strong semantic generalization
  - Cons: Requires OpenAI API key and incurs cost per token
- **Chunking strategy**:

- Chunk size: ~1000 tokens
- Overlap: ~200 tokens to preserve context continuity across segments

- **Top-k retrieval**: k = 3 based on empirical tuning for relevance and response quality
- **Similarity metric**: Cosine similarity

$$sim(u, v) = \frac{u \cdot v}{\| u \| \cdot \| v \|}$$

This setup allowed us to retrieve semantically relevant occupation chunks that were later passed to the LLM in the prompt context, enabling grounded, contextual responses. Using text-embedding-ada-002 ensured a higher-quality retrieval baseline than open-source models, which was crucial for career guidance, where small nuances in job roles can change recommendations significantly.

1.3. Retrieval-Augmented Generation (RAG) Pipeline

The core functionality of CareerCompass is built around a Retrieval-Augmented Generation (RAG) system. This architecture enables the chatbot to provide responses that are not only natural sounding but also grounded in reliable, structured job data.

Traditional language models often rely entirely on their internal knowledge, which can lead to generic or outdated answers. In contrast, our RAG approach ensures that every response is informed by up-to-date occupational information, improving accuracy and trust.

The RAG pipeline follows these key steps:

- **Step 1: User Query Embedding**
  When a user asks a question such as "What does a biomedical engineer do?", the query is converted into a numerical vector using OpenAI's text-embedding-ada-002 model. This vector has 1,536 dimensions and captures the semantic meaning of the question.
- **Step 2: Semantic Retrieval from ChromaDB**
  The system then searches our vector store, which contains thousands of embedded job role descriptions. We use ChromaDB for this step. It finds the top three most relevant documents based on cosine similarity to the user's query.

- **Step 3: Context Summarization and Structuring**
  The retrieved job roles are summarized into a consistent format. Each summary includes:
  - Job title and a short description
  - Key required skills
  - Education background
  - Common tasks, tools, and work activities
    These summaries are formatted using Markdown so that they display cleanly in the front-end interface.
- **Step 4: Prompt Construction for the LLM**
  The formatted context, along with the original user question, is combined into a detailed prompt. This prompt is then sent to LLaMA 3.3 (70B), hosted through Groq's API. Our system message ensures:
  - Responses stay focused on career guidance
  - No salary information is provided
  - Off-topic queries are rejected clearly and respectfully
- **Step 5: Final Answer Generation**
  The language model uses both the prompt and retrieved context to generate a final response. The output is informative, structured, and optimized for readability.

This approach allows CareerCompass to offer specific, helpful guidance that users can trust. By separating the retrieval of facts from the language generation process, we ensure that the advice is both grounded and personalized.

1.4. Memory and Conversation Management

To create a smooth and natural experience, CareerCompass includes a custom memory system that helps maintain the flow of conversation across multiple user inputs. Rather than treating each message in isolation, the system keeps track of what has already been asked and answered, allowing it to provide more coherent and context-aware replies.

Each user session is associated with a unique identifier, such as a username. When a user begins a conversation, the system initializes a memory buffer that includes the chatbot's system instructions and the user's interaction history. This memory is used to build prompts that reflect the user's prior questions and responses, which helps the language model avoid repeating information or asking for the same input multiple times.

“Langchain” Integration for Conversational Memory

Career Compass leverages components from the **Langchain** framework to structure and manage chat interactions more effectively. Langchain offers a clean abstraction for message formatting, memory handling, and model invocation, which enhances both maintainability and extensibility.

Key elements of Langchain integration include:

- **Message Management**:
  - We use SystemMessage, HumanMessage, and AIMessage from Langchain’s schema to format conversation turns cleanly and consistently.
  - This structure ensures clear role separation and alignment with LLM expectations.
- **Chat Model Invocation**:
  - The chat model is initialized using Langchain’s ChatOpenAI wrapper, which is configured to interface with Groq’s LLaMA 3.3 (70B) model.
  - Parameters such as temperature, max_tokens, and API endpoints are specified to ensure deterministic, grounded responses.
- **Modular and Future-Proof Design**:
  - Langchain simplifies swapping models or integrating advanced memory strategies (e.g., buffer memory or summarization memory) with minimal refactoring.
  - Its abstraction layer helps future-proof the system if more complex session-based reasoning is added later.

Key elements of the memory management system:

- **Session Tracking**
  User conversations are stored in a Python dictionary during runtime. Each session contains both the full technical prompt history and a simplified visible version of the chat, which is optimized for display in the front end.
- **Saving and Resuming Chats**
  When a user ends a conversation, the system saves the chat history to a persistent store (in this case, DynamoDB). Users can later resume past conversations by loading this history, which is then reconstructed into the correct format for the language model.

- **State Reset and Cleanup**
  An exit endpoint allows users to clear the current memory after a chat ends. This ensures that a new session starts fresh while still preserving the ability to retrieve older conversations if needed.

This memory system plays a critical role in delivering an intelligent and user-friendly experience. It gives the chatbot continuity, makes it feel responsive to the user's journey, and supports advanced features like chat resumption and history management.

1.5. Guardrails and Safety Filters

CareerCompass is designed not just to be intelligent, but also responsible. Since it provides career advice using a large language model, it was essential to build in strict controls to ensure the chatbot stays on topic, avoids sensitive areas, and maintains a professional tone throughout the conversation.

At the center of this effort is a carefully written system prompt, which guides the chatbot's behavior during every interaction. This prompt is included at the start of each session and reinforces the chatbot's role, limitations, and tone.

The system prompt enforces the **following rules:**

- The chatbot should focus only on career-related guidance and strictly avoid unrelated topics.
- It must not provide salary estimates, even if the user requests them.
- It should respond in a helpful, supportive, and professional tone, using bullet points or sections to break down complex answers.
- If a user asks a question outside the scope of career exploration, the chatbot must politely refuse to answer.

These instructions are reinforced with every new chat session and are preserved throughout the conversation history.

In addition to the prompt, further safeguards are built into the system logic:

- **Topic Filtering:** The language model is consistently reminded to ignore prompts about salary, personal advice, or unrelated topics such as politics or entertainment.
- **Content Formatting**: All responses are expected to follow a clean structure using Markdown with headings, bullet points, and consistent sectioning. This prevents the

model from producing long, unstructured or informal paragraphs that may confuse users.

- **Fallback Responses**: If no relevant job data is found for a user query, the system returns a clear fallback message rather than generating a potentially incorrect response.

These safety measures ensure that CareerCompass remains focused, consistent, and aligned with its intended purpose as a career exploration tool.

1.6. Uniqueness of Design and Key Innovations

CareerCompass was built with the goal of going beyond a typical chatbot. The backend integrates structured career data, semantic search, dynamic prompting, and session memory to provide accurate and personalized guidance. Each part of the system was designed to make the experience more useful, explainable, and trustworthy.

Several aspects of the backend make it distinct from other solutions:

- **Structured Data Grounding**
  All responses are based on real occupational data from the O*NET database. This means the answers are not purely generated by the model's training but are grounded in verified content.
- **Markdown-Formatted Context with Semantic Highlights**
  Retrieved job descriptions are formatted into clearly structured sections such as job title, skills, tools, and education. This helps the language model focus on relevant parts and ensures the response is easy to read and scan.
- **User-Aware Memory and Resumability**
  The system supports saving and resuming chat sessions. Each user's interaction history is stored and recalled later, allowing conversations to feel continuous over time.
- **Dynamic Prompting Based on Retrieved Data**
  The language model's response is shaped not only by the user query but also by relevant job content pulled in real time from the vector store. This improves the relevance and specificity of each answer.
- **Strict Topic Control with Custom Safety Layer**
  The chatbot includes strong guardrails. It avoids salary predictions, does not engage in off-topic discussions, and always responds within a defined professional scope. This was achieved through system-level prompt engineering and fallback logic.

- **Use of Open-Source Language Model**
  Instead of relying on closed-source commercial models, CareerCompass uses Meta's LLaMA 3.3 70B, an open-source large language model, hosted through Groq's high-speed inference API. This choice not only reduces dependency on proprietary systems but also supports cost-effective scalability and transparency. It demonstrates that powerful, real-time conversational tools can be built using open-source foundations when paired with thoughtful system design.
- **LangChain-Powered Orchestration**
  LangChain acts as the backbone for combining retrieval, prompt engineering, memory, and safety layers. Its modular design enabled rapid development and easy customization of complex workflows, enhancing maintainability and flexibility.

This combination of structured data, real-time retrieval, persistent memory, prompt logic, and open-source AI, woven together with LangChain, makes Career Compass a uniquely reliable and explainable tool for career exploration.

2. Frontend Part:

2.1. Framework & Architecture

The CareerCompass frontend was built using Next.js, a powerful React-based framework, paired with Tailwind CSS for efficient, utility-first styling. This tech stack enabled rapid development, mobile responsiveness, and server-side rendering where needed. We followed a modular architecture, organizing pages and components for scalability and reusability.

Each page of the platform was designed to fulfill a key user journey stage: exploration, interaction, reflection. We also integrated the frontend with **FastAPI backend endpoints**, enabling real-time data exchange for user queries, chat memory, and dynamic recommendations.

To deploy the CareerCompass platform, we used an AWS EC2 instance running Ubuntu. After cloning the Git repository and checking out the appropriate branch, a custom deployment script (post-checkout-setup.sh) was used to configure the environment. The script installs dependencies and starts the Next.js frontend in development mode. The frontend app runs persistently in the background using tmux, which allows the server process to continue running after the SSH session ends. The backend (FastAPI) was also deployed on the same EC2 instance and exposed over port 8000, providing API access for chat processing and chat memory. Chat history is stored and retrieved via AWS DynamoDB, enabling resumable and persistent conversations. Logs are written to a dedicated directory for debugging. This deployment strategy ensured low-latency interaction, secure API communication, and real-time storage, all hosted on scalable and cost-effective AWS infrastructure.

2.2. Homepage and Landing Page

We designed a hero-style homepage with a visually engaging gradient background, bold typography, and a prominent CTA ("Get Started"). Tailwind's responsive utility classes enabled instant responsiveness across screen sizes.

The landing page functions as an onboarding surface for new users.

Key features:

- Chatbot preview panel with real interface snippets
- "Try the Chatbot" CTA for frictionless access
- Highlight of AI + O*NET data integration
- Smooth scroll behavior and animated sections

Unique design decision: Users can test the chatbot without signing up, improving accessibility and reducing entry barriers.

2.3. Chat Interface & History Management

One of the core front-end features is the chat interface, built with interactive components and integrated directly with the backend's memory system.

Design highlights:

- **Real-time message handling** with automatic scroll anchoring for a smooth chat experience
- **New Chat** button that resets the current session and starts a fresh conversation
- **Refresh History** button that loads and displays previously saved chats, allowing users to view and continue past conversations
- **Save Chat** button that appears when a conversation ends, enabling users to manually store important interactions

- **Delete controls** for each saved chat, giving users control over their history and privacy

We used useEffect and useState hooks in React to sync UI state with API responses, dynamically updating the conversation window and chat memory.

2.4. Career Insights Data Visualization Dashboard Integration

The Career Data Dashboard is built in Tableau and embedded seamlessly into our frontend using responsive <iframe> containers. Tailwind's layout system helped maintain consistent visual hierarchy.

We included:

- A dedicated **Overview Dashboard** page that explores high-level trends such as **average wages**, **top job roles**, and **overall employment growth** across industries.
- Two separate **data stories**:
  - **Wages in America**, which breaks down wage distribution across roles and regions
  - **Employment Trends**, which visualizes job growth, decline, and job type distributions over time

Tailwind's layout system ensured that visualizations remained responsive and accessible. We also used visual cards and headings to guide users to each section and improve discoverability.

2.5. Design Innovations & Uniqueness

Key design decisions that differentiate our frontend from a template-based site:

- Minimalist and modern visual identity, with vibrant gradients and iconography
- Custom-built chat interface with memory and control features
- AI + visualization fusion, rarely found in traditional job platforms
- No-login chatbot preview improves accessibility and showcases value immediately
- Responsiveness, tested across devices and resolutions

3. Data Visualization Part:

CareerCompass leverages Tableau Public to present employment and wage data through dashboard and narrative-driven visualization experiences.

3.1. Data Preparation and Integration

The core dataset from O*NET lacked key quantitative measures such as employment counts and wage statistics. To fill these gaps, we sourced supplementary data from the Bureau of Labor Statistics (BLS). This required:

- Cleaning and formatting data to ensure consistency across sources.
- Creating a major occupation group based on Standard Occupational Classification (SOC) codes.
- Merging datasets using Standard Occupational Classification (SOC) codes in Tableau.

3.2. Design Choices and Visualization Formats

The visualization design was centered around two key components:

1. Overview Dashboard
   A single-page interactive dashboard was developed to provide users with a quick, high-level view of employment and wage data across various job categories and states.
2. Interactive Data Stories
   Based on feedback we felt that the dashboard alone did not provide sufficient context or guidance for data interpretation. To address this, we created two interactive data stories:
   - **Salaries** – focused on wage trends across regions, occupation groups and occupations.
   - **Employment** – highlighted job availability, workforce size, and projected growth.

Each story was structured as a webpage, with each visualization accompanying descriptions and insights to guide the user through understanding.

3.3. User Experience and Interactivity

To enhance usability and engagement, we integrated several interactive elements using Tableau's built-in functionality:

- Filters and drop-downs to allow users to explore specific occupations or industries.
- Tooltips that provided concise context or additional statistics when hovering over data points.

3.4. Distinctive Features and Customization

What sets this project apart from standard Tableau dashboards or templates is the deliberate combination of narrative storytelling and interactive analytics. Key points of differentiation include:

- A dual-format design that accommodates both exploratory and guided user experiences.
- A visual and narrative style tailored to non-expert users, including simplified terminology and clearly stated takeaways.
- Fully customized layout and content structure to create a purposeful user journey.

This project successfully translates complex occupational data into an accessible, engaging, and informative experience. Through data preparation, interactive design, and narrative-driven storytelling, the final product serves not only as a visualization tool but also as an educational resource—especially for younger audiences exploring career opportunities and labor trends to be empowered by the data instead of overwhelmed.

Retrospection:

The backend development of Career Compass was central to transforming the project from a basic chatbot into an intelligent, explainable, and user-centric career guidance system. Throughout the implementation of data pipelines, vector search, memory handling, and safety controls, several aspects stood out as key successes and challenges.

The backend development of Career Compass laid the foundation for a responsive, intelligent, and explainable career guidance system. One of the major successes was the implementation of a modular Retrieval-Augmented Generation (RAG) pipeline, which combined OpenAI embeddings with ChromaDB to deliver highly relevant and grounded responses. Our structured data processing—transforming over 30 O*NET files into Markdown-formatted job profiles—enabled context-aware answers and reduced hallucinations. Integrating LangChain for memory management and prompt orchestration streamlined the user experience by supporting session continuity, resumable chat history, and clean role separation. The decision to use Meta's open-source LLaMA 3.3 model via Groq further enhanced speed, scalability, and transparency. Together, these design choices ensured that CareerCompass provided not just accurate guidance, but a coherent and trustworthy interaction flow.

However, the backend development also presented a few key challenges. Managing session memory in a stateless FastAPI environment required careful coordination between in-memory buffers and DynamoDB persistence to ensure consistency across user interactions. Early issues with chat history linkage and prompt construction were resolved by assigning unique user identifiers and refining chunking and retrieval logic.

During the front-end development of Career Compass, several things went particularly well. Our choice of Next.js and Tailwind CSS enabled us to build a clean, responsive, and modular interface that scaled well as the project grew. Features like real-time chatbot interaction, dynamic chat history with delete functionality, and an intuitive layout for the Career Insights Dashboard helped us deliver a smooth and modern user experience. The visual design—featuring gradients, accessible typography, and iconography—established a strong identity that resonated with users. API integration was also seamless, thanks to well-managed state handling and consistent response formatting.

However, the process came with its fair share of challenges. Early in the sprint, we faced responsiveness issues on smaller screens where layouts would break or overlap. We addressed this using Tailwind's responsive utilities, refactoring several components to adapt fluidly across breakpoints. Another issue was the display of bot responses, where raw Markdown formatting like \n and ** would appear in the chat. To solve this, we integrated react-markdown to parse and render responses cleanly with proper structure and emphasis. Additionally, managing chat history proved tricky—initially, histories were not rendering correctly or linked to the wrong user sessions. We fixed this by implementing unique session IDs and timestamp-based previews, which ensured accurate and personalized history display.

Configuring EC2 access and handling background processes via tmux required some troubleshooting but enabled lightweight, persistent hosting.

The visualization team created supplemental content to the chatbot to add a visual experience. The initial data preparation and cleaning went smoothly, setting a strong foundation for the visual work in Tableau. The team effectively tackled a learning curve around designing user-friendly dashboards by iterating and organizing visualizations across multiple pages. In the final sprint, we adapted based on feedback, reused strong elements like the overview tab, and improved their storytelling by focusing on employment and wage data. Collaboration with the Front-End team helped integrate the visualizations into the site successfully.

Early in the process, a key oversight was the assumption that the O*NET database would contain all necessary numeric data, when in fact much of it resided on the Bureau of Labor Statistics website. This required a pivot to source and process data from an additional location. Later, during dashboard creation, we faced challenges in choosing appropriate visualizations and layout designs, leading to a time-consuming iteration process. Additionally, after initial dashboard feedback, we had to change direction and create new data stories, indicating some initial misalignment with user needs or expectations. These course corrections introduced additional work but ultimately improved the final product.

Recommendations for future:

On the backend, we recommend integrating additional career-related datasets beyond O*NET—such as Coursera, LinkedIn Learning, or certification databases—to broaden job exploration. This would enable features like resume suggestions tailored to selected roles, course and certification recommendations, and skill-gap analysis with career-preparation tips, all grounded in real-time data. From a front-end perspective, incorporating a user preferences dashboard, dark mode support, and interactive widgets would significantly boost personalization. Additional UX features like typing indicators, feedback prompts, and tooltip-guided onboarding can make the AI feel more conversational and intuitive. Together, these additions would make Career Compass not only more intelligent and data-rich, but also more user-centered and impactful.