**Maroun Semaan Faculty of Engineering and Architecture**

**American University of Beirut**

****

**AI-Powered Career Advisor:**

**Documentation Report**

**Presented By: Zakaria Labban**

**Hadi Elham**

**Mahmoud Chaer**

**Ali Alaaeddine**

***2. Table of Contents***

1. [*Title Page*](#1-title-page)
2. [*Table of Contents*](#2-table-of-contents)
3. [*Executive Summary*](#3-executive-summary)
4. [*Introduction*](#4-introduction)
5. [*Project Objectives*](#5-project-objectives)
6. [*Methodology*](#6-methodology)
7. [*Development Timeline*](#7-development-timeline)
8. [*Technology Stack*](#8-technology-stack)
9. [*Design and Architecture*](#9-design-and-architecture)
10. [*Implementation Details*](#10-implementation-details)
11. [*Challenges and Solutions*](#11-challenges-and-solutions)
12. [*Features and Functionality*](#12-features-and-functionality)
13. [*Testing and Validation*](#13-testing-and-validation)
14. [*Deployment*](#14-deployment)
15. [*Future Enhancements*](#15-future-enhancements)
16. [*Conclusion*](#16-conclusion)
17. [*Appendices*](#17-appendices)
18. [*Acknowledgments*](#18-acknowledgments)

***3. Executive Summary***

The AI-Powered Career Advisor is an innovative web-based platform designed to assist students and early-career professionals in navigating the complex landscape of career planning within the technology sector. Leveraging advanced artificial intelligence models, including OpenAI's GPT-4 and embedding techniques like Word2Vec, the advisor provides personalized job recommendations, skill gap analyses, and career path visualizations. Integrated with various APIs such as Adzuna, Coursera, and GitHub, the platform offers real-time job listings, relevant courses, and project opportunities to enhance users' professional development. Additionally, features like CV generation and an interactive chatbot ensure a comprehensive and user-friendly experience. Developed over an intensive eight-week period, the project showcases the effective application of AI in delivering tailored career guidance, addressing the limitations of traditional advising methods.

***4. Introduction***

**Background and Motivation**

Navigating career paths in the rapidly evolving technology sector can be daunting for students and early-career professionals. The abundance of options, coupled with the dynamic nature of tech roles, often leads to uncertainty and indecision. Traditional career advising methods may fall short in providing the personalized, data-driven insights needed to make informed decisions. Recognizing this gap, our team embarked on developing the AI-Powered Career Advisor—a tool designed to harness the power of artificial intelligence to deliver tailored career guidance, helping users align their skills and interests with suitable job opportunities.

**Problem Statement**

Individuals seeking career guidance often struggle with choosing the right path due to a lack of personalized support and the overwhelming number of options available in the tech industry. Existing career advisory tools may not adequately account for individual skills, interests, and the ever-changing job market, resulting in suboptimal career choices and reduced professional satisfaction.

**Importance of an AI-Powered Career Advisor**

An AI-powered Career Advisor offers a scalable and personalized solution to career planning challenges. By analyzing user profiles, industry trends, and comprehensive occupational data, the advisor provides actionable insights and recommendations that are both relevant and timely. This empowers users to make informed decisions, enhances their chances of success, and fosters greater satisfaction in their professional journeys.

***5. Project Objectives***

**Primary Objectives:**

1. **Personalized Job Recommendations:** Utilize AI models to analyze user profiles and provide tailored job suggestions that align with their skills, interests, and career aspirations.
2. **Skill Gap Analysis:** Assess users' current skill sets, identify areas for improvement, and recommend targeted courses and projects to bridge these gaps.
3. **Career Path Visualization:** Offer interactive visualizations that map out potential career trajectories, helping users understand the steps required to advance in their chosen fields.
4. **CV Generation:** Enable users to create professional, well-structured CVs based on their input, facilitating easy customization and export for job applications.
5. **Interactive Support:** Implement a chatbot to provide real-time assistance, answer user queries, and offer personalized career advice.

**Secondary Objectives:**

1. **Integration with External APIs:** Incorporate job listings, courses, and project opportunities through APIs like Adzuna, Coursera, and GitHub to enrich the advisor's recommendations.
2. **User Feedback Mechanism:** Establish a continuous feedback loop to refine and enhance the advisor's functionalities based on user interactions and satisfaction.
3. **Scalability and Performance:** Ensure the platform is optimized for performance and can scale to accommodate a growing user base without compromising responsiveness.

***6. Methodology***

Our project followed a structured methodology, adapting as challenges arose to ensure the successful development of the AI-Powered Career Advisor.

1. **Initial Research and Planning**
   * **Idea Conception:** Recognized the need for personalized career guidance among students in the tech field.
   * **Literature Review:** Investigated existing career advisory tools and AI applications in career planning to identify gaps and opportunities.
2. **Data Collection**
   * **Dataset Gathering:** Collected datasets from various online repositories, including GitHub, Kaggle, and the UCI Machine Learning Repository. These datasets encompassed information on career paths, required skills, industry trends, and user profiles.
   * **Data Sources:** Explored additional sources such as the O\*NET Database to ensure comprehensive coverage of the tech industry's diverse career options.
3. **Initial Model Development**
   * **Model Selection:** Planned to develop a neural network or leverage a Large Language Model (LLM) for natural language processing tasks.
   * **Resource Assessment:** Evaluated the computational and financial resources required for model training and deployment.
4. **Resource Constraints Identification**
   * **Challenges Faced:** Realized that, as students, the available resources (computational power, access to high-performance GPUs, and financial constraints) were insufficient to train large-scale models effectively.
   * **Impact:** Determined that building a neural network or an LLM from scratch was not feasible within the project's scope and resource limitations.
5. **Seeking Expert Guidance**
   * **Consultation:** Sought guidance from Professor Ammar Mohanna to navigate the resource constraints and optimize the project approach.
   * **Professor’s Advice:** Advised to utilize an already built LLM, such as OpenAI's GPT-4, to leverage existing powerful models without the need for extensive computational resources.
6. **Pivoting Strategy**
   * **Alternative Approaches:** Shifted focus towards leveraging pre-trained models and utilizing more resource-efficient algorithms to achieve the project's objectives without extensive computational demands.
   * **Tool Selection:** Decided to use cloud-based services and APIs that provide access to powerful AI models, mitigating the need for in-house resource investment.
7. **Exploration and Base Model Creation**
   * **Exploring Existing Implementations:** Investigated how existing AI-powered career advisors are built, focusing on integrating LLMs for enhanced functionality.
   * **Base Model Development:** Created an initial base model that involved simple question-answering capabilities, allowing users to interact with the advisor through predefined queries.
8. **Feature Enhancement: CV Upload and Analysis**
   * **CV Upload Functionality:** Incorporated a feature that allows users to upload their resumes (CVs) to the platform.
   * **Resume Analysis:** Utilized GPT-4 to analyze the uploaded resumes, extracting key information such as skills, experiences, and qualifications.
   * **Job Recommendations:** Based on the analysis, GPT-4 provides tailored job recommendations that align with the user's profile, skills, and interests.
9. **Integrating Job Listings API**
   * **API Exploration:** Researched various job listing APIs to source real-time job opportunities for users.
   * **API Selection:** Determined that popular APIs like LinkedIn, Indeed, and Google Jobs were not feasible due to accessibility and cost constraints.
   * **Implementation of Adzuna Search API:** Opted for the Adzuna Search API as the most viable option. Integrated it into the project to fetch relevant job listings based on user profiles.
   * **Handling Heavy Outputs:** Encountered challenges with the volume of data returned by the Adzuna API. Implemented strategies such as result filtering, pagination, and caching to manage and optimize the heavy outputs, ensuring efficient performance and a seamless user experience.
10. **Cost Constraints and Model Alternatives Exploration**
    * **Issue:** High processing costs associated with OpenAI's GPT-4 posed a significant financial challenge.
    * **Solution:** Searched for free and accessible models such as LLama-2-7B, BERT, GPT-2, and others to reduce operational costs.
    * **Outcome:** Discovered that these alternative models required GPU resources and considerable time for model training, which were not feasible given the project's resource limitations.
    * **Decision:** Ultimately decided to rely back on OpenAI's GPT-4 despite the high costs, as it remained the most practical option for achieving the project's objectives within the available constraints.
11. **Integrating Course and Project APIs**
    * **API Exploration:** Searched for APIs providing access to courses and projects to enrich the Career Advisor's recommendations.
    * **API Selection:** Identified Coursera's API and GitHub's API as viable options, offering free access to courses, projects, and their descriptions.
    * **Implementation:** Integrated Coursera's and GitHub's APIs into the platform to provide users with relevant courses and project opportunities aligned with their career recommendations.
    * **Data Management:** Implemented data handling strategies to efficiently process and present course and project information without overwhelming the user interface.
12. **Developing "Build Your CV" Feature**
    * **Identifying User Needs:** Recognized that many first and second-year students might not have an existing CV, making the CV upload feature less accessible to them.
    * **Feature Conceptualization:** Decided to create a "Build Your CV" feature that allows users to generate a professional CV in minutes.
    * **Implementation:**
      + **User Input Collection:** Designed an intuitive interface where users can input their personal information, education, skills, projects, and other relevant details.
      + **CV Generation:** Utilized GPT-4 to format the collected information into a well-structured, professional CV.
      + **Customization Options:** Provided users with customization options to tailor their CVs according to different job applications.
      + **Download and Export:** Enabled users to download their generated CVs in various formats (e.g., PDF, DOCX) for easy use.
13. **Developing the User Interface with Streamlit**
    * **Identifying UI Importance:** Recognized that an intuitive and attractive user interface is crucial for user engagement and retention.
    * **Tool Selection:** Sought an easily implementable UI framework that aligns with the team's focus on AI and machine learning rather than frontend development.
    * **Selection of Streamlit:** Chose Streamlit for its simplicity, rapid development capabilities, and suitability for data-driven applications.
    * **Streamlit Overview:** Streamlit is an open-source Python library that allows developers to create interactive and visually appealing web applications quickly without extensive frontend expertise. It integrates seamlessly with Python code, enabling the team to focus on functionality rather than design intricacies.
    * **Implementation:**
      + **UI Design:** Utilized Streamlit's widgets and layout options to design a user-friendly interface that facilitates easy navigation and interaction with the Career Advisor's features.
      + **Integration:** Integrated Streamlit with the backend components, ensuring smooth communication between the user interface and the AI-powered functionalities.
      + **Customization and Enhancements:** Leveraged Streamlit's capabilities to add customization options, interactive elements, and responsive design elements to enhance the overall user experience.
    * **Benefits Realized:** Streamlit significantly reduced the time and effort required to develop the frontend, allowing the team to concentrate on refining the AI models and backend processes. Additionally, Streamlit's real-time interactivity and simplicity facilitated rapid prototyping and iterative improvements based on user feedback.
14. **Developing Career Visualization Feature**
    * **Identifying User Needs:** Recognized that users, especially first and second-year students, may lack a clear understanding of the sequential steps required to advance in their careers.
    * **Feature Conceptualization:** Decided to create a **Career Visualization** feature that allows users to visualize their predicted career paths, including recommended jobs, courses, and projects.
    * **Design Specifications:**
      + **4-Step Career Journey:** Designed a 4-step journey where each step includes:
        - **Job Recommendation:** A sequentially recommended job role.
        - **Courses:** Four relevant courses to acquire the necessary skills for the job.
        - **Projects:** Four projects to work on that demonstrate competence in the required areas.
    * **Implementation:**
      + **Data Integration:** Utilized the integrated APIs (Adzuna, Coursera, GitHub) to fetch relevant data for jobs, courses, and projects.
      + **Visualization Tools:** Leveraged Streamlit's visualization capabilities to create interactive and intuitive graphical representations of the career paths.
      + **User Interaction:** Enabled users to interact with the visualization, allowing them to explore different career paths and understand the progression.
      + **Personalization:** Used GPT-4 to tailor the career paths based on individual user profiles, ensuring that the recommendations align with their skills, interests, and goals.
    * **Testing and Refinement:** Conducted user testing sessions to gather feedback on the usability and effectiveness of the Career Visualization feature. Iteratively refined the feature to enhance clarity, interactivity, and user engagement.
    * **Outcome:** Successfully implemented a dynamic and user-friendly Career Visualization tool that provides clear guidance on career progression, making it easier for users to plan their educational and professional development.
15. **Developing Skill Gap Analysis Feature**
    * **Identifying User Needs:** Recognized the importance of understanding users' skill levels to provide targeted career recommendations.
    * **Feature Conceptualization:** Decided to implement a **Skill Gap Analysis** feature that assesses users' current skills and identifies areas for improvement.
    * **Design Specifications:**
      + **Skill Assessment:** Developed a questionnaire or assessment tool to evaluate users' proficiency in various skills relevant to their desired career paths.
      + **Data Integration:** Incorporated data on required skills for different job roles sourced from the Adzuna API and other relevant datasets.
      + **Analysis Mechanism:** Utilized GPT-4 to analyze assessment results and identify skill gaps.
      + **Recommendations:** Generated personalized recommendations on which skills to develop, including suggested courses and projects to bridge the gaps.
    * **Implementation:**
      + **User Input Collection:** Designed an intuitive interface for users to complete skill assessments.
      + **Data Processing:** Processed assessment data to evaluate proficiency levels.
      + **Integration with Recommendations:** Linked skill gap results with the Career Visualization and job recommendation features to provide comprehensive guidance.
      + **Visualization:** Used Streamlit's visualization tools to present skill gap analysis results in an easily understandable format.
    * **Testing and Refinement:** Conducted user testing to validate the accuracy and usefulness of the skill gap analysis. Refined the assessment tools and recommendation algorithms based on feedback to ensure reliability and effectiveness.
    * **Outcome:** Implemented a robust Skill Gap Analysis feature that empowers users to identify and address their skill deficiencies, thereby enhancing their preparedness for recommended career paths.
16. **Implementing Feedback Mechanism**
    * **Identifying User Needs:** Understood that continuous user feedback is essential for refining career path recommendations and overall platform functionality.
    * **Feature Conceptualization:** Decided to implement a feedback mechanism that allows users to provide input during the career path generation process.
    * **Design Specifications:**
      + **Feedback Collection:** Designed prompts and feedback forms that appear during and after the generation of career paths.
      + **Data Integration:** Collected feedback data to understand user satisfaction and areas needing improvement.
      + **Analysis and Iteration:** Used GPT-4 to analyze feedback and adjust recommendation algorithms accordingly.
    * **Implementation:**
      + **User Interface:** Integrated feedback widgets within the Streamlit interface to capture user opinions seamlessly.
      + **Backend Processing:** Developed backend processes to store and analyze feedback data.
      + **Algorithm Refinement:** Iteratively updated the career recommendation algorithms based on aggregated feedback to enhance accuracy and user satisfaction.
    * **Testing and Refinement:** Conducted usability testing to ensure the feedback mechanism was intuitive and effective. Adjusted feedback prompts based on user interactions to maximize engagement.
    * **Outcome:** Established a continuous improvement loop where user feedback directly informs and enhances the quality of career path recommendations, leading to a more personalized and effective user experience.
17. **Implementing Chatbot Functionality**
    * **Identifying User Needs:** Recognized that users may have specific questions or require personalized support beyond predefined features.
    * **Feature Conceptualization:** Decided to implement a chatbot with the role of a career advisor to respond to all user questions and suggest actions to advance their careers.
    * **Design Specifications:**
      + **Chatbot Integration:** Integrated a conversational AI chatbot within the Streamlit interface to handle real-time user interactions.
      + **Role Definition:** Configured the chatbot to act as a knowledgeable career advisor, capable of answering queries, providing recommendations, and guiding users through the platform's features.
      + **Natural Language Understanding:** Leveraged GPT-4's capabilities to understand and respond to a wide range of user inputs effectively.
    * **Implementation:**
      + **Interface Design:** Designed a chat interface using Streamlit's components, ensuring it was easily accessible and user-friendly.
      + **Backend Integration:** Connected the chatbot to GPT-4 via OpenAI's API to handle conversational logic and response generation.
      + **Customization:** Developed predefined responses and workflows to handle common queries related to career paths, skill development, CV building, and more.
      + **Personalization:** Utilized user profile data to provide tailored responses and recommendations during interactions.
    * **Testing and Refinement:** Conducted extensive testing to ensure the chatbot accurately understood and responded to user queries. Gathered user feedback to refine conversational flows and improve response accuracy.
    * **Outcome:** Successfully implemented a responsive and intelligent chatbot that enhances user support, providing on-demand assistance and personalized career guidance, thereby increasing user engagement and satisfaction.
18. **Attempting Retrieval-Augmented Generation (RAG) System**
    * **Identifying User Needs:** Aimed to enhance the accuracy and relevance of career recommendations by incorporating external knowledge sources.
    * **Feature Conceptualization:** Decided to implement a Retrieval-Augmented Generation (RAG) system by feeding the model with books on career guidance to provide more informed and contextually accurate responses.
    * **Design Specifications:**
      + **Data Integration:** Integrated a corpus of career guidance books into the RAG framework to provide the model with additional context and information.
      + **Retrieval Mechanism:** Configured the system to retrieve relevant excerpts from the books based on user queries and inputs.
      + **Generation Process:** Utilized the retrieved information to generate more accurate and contextually relevant career recommendations and advice.
    * **Implementation:**
      + **Data Ingestion:** Processed and indexed the career guidance books to facilitate efficient retrieval.
      + **System Configuration:** Set up the RAG pipeline to combine retrieval and generation processes seamlessly.
      + **Integration with GPT-4:** Connected the RAG system with GPT-4 to leverage its language generation capabilities alongside the retrieved knowledge.
    * **Challenges Faced:**
      + **Quality of Outputs:** The RAG system did not produce sufficiently accurate or relevant recommendations, leading to suboptimal user experiences.
      + **Complexity:** Managing the integration of external data sources added complexity to the system without the desired improvements in output quality.
    * **Outcome:** The RAG system did not meet performance expectations, prompting the team to explore alternative approaches for enhancing recommendation accuracy.

***7. Development Timeline***

The AI-Powered Career Advisor was developed over an intensive eight-week period, with each week dedicated to specific tasks and milestones to ensure timely and efficient project completion.

**Week 1: Ideation and Research**

* **Conceptualization:** Defined the project's scope and objectives based on personal experiences and identified the need for an AI-driven career advisor.
* **Literature Review:** Conducted research on existing career advisory tools and AI applications to understand current solutions and identify gaps.
* **Tool Evaluation:** Investigated various GUI frameworks (Flask, Gradio, Django) and determined that Streamlit was the most suitable for our needs.

**Week 2: Data Collection and Preparation**

* **Dataset Gathering:** Collected relevant datasets from GitHub, Kaggle, UCI Machine Learning Repository.
* **Data Cleaning:** Preprocessed the collected data to ensure quality and consistency, removing duplicates and irrelevant information.

**Week 3: Initial Model Development and Integration**

* **Model Selection:** Decided to utilize OpenAI's GPT-4 for natural language processing tasks.
* **Base Model Creation:** Developed a simple question-answering model using GPT-4 to handle basic user interactions.
* **API Integration:** Began integrating the Adzuna Search API to fetch real-time job listings based on user profiles.

**Week 4: Feature Development - CV Upload and Analysis**

* **CV Upload Implementation:** Enabled users to upload their resumes in PDF format.
* **Resume Parsing:** Utilized pdfplumber and PyMuPDF to extract text data from uploaded resumes.
* **Resume Analysis:** Integrated GPT-4 to analyze extracted resume data and generate tailored job recommendations.

**Week 5: Developing "Build Your CV" and Career Visualization Features**

* **CV Generation:** Implemented the "Build Your CV" feature using python-docx and ReportLab to create customizable CVs in DOCX and PDF formats.
* **Career Visualization:** Developed the Career Visualization feature using NetworkX and PyVis to map out career paths, courses, and projects.
* **API Integration:** Continued integrating Coursera's and GitHub's APIs to enrich course and project recommendations.

**Week 6: GPT-4 and Embedding Model Implementation**

* **Embedding Model:** Implemented the Word2Vec embedding model to better understand relationships between skills and job occupations.
* **System Integration:** Linked the embedding model with the feedback mechanism to refine job title predictions based on user interactions.

**Week 7: Implementing Feedback Mechanism and Chatbot Functionality**

* **Feedback Mechanism:** Developed feedback prompts and forms within Streamlit to capture user input during career path generation.
* **Chatbot Development:** Implemented an interactive chatbot using GPT-4 to provide real-time assistance and personalized career advice.
* **User Testing:** Conducted initial testing of the feedback mechanism and chatbot to gather user feedback and identify areas for improvement.

**Week 8: Testing, Optimization, and Deployment Preparation**

* **Comprehensive Testing:** Performed thorough testing of all features to identify and fix bugs, ensuring reliability and stability.
* **Performance Optimization:** Optimized the application's performance for faster response times and efficient resource utilization.
* **Deployment Infrastructure:** Set up the necessary infrastructure for deployment, ensuring scalability and robustness.
* **Final Adjustments:** Made final refinements based on testing and user feedback.
* **Deployment:** Deployed the AI-Powered Career Advisor on a scalable cloud platform, making it accessible to end-users

***8. Technology Stack***

The AI-Powered Career Advisor leverages a combination of powerful tools, libraries, and frameworks to deliver its comprehensive functionalities. Below is a detailed overview of each tool used in the project:

1. **Streamlit**
   * **Description:** Streamlit is an open-source Python library designed to create interactive and visually appealing web applications with minimal effort. It allows developers to build data-driven apps quickly without requiring extensive frontend development knowledge.
   * **Usage in Project:** Streamlit was chosen as the primary framework for developing the user interface of the Career Advisor. Prior to selecting Streamlit, the team evaluated several other GUI frameworks, including Flask, Gradio, and Django. However, these alternatives proved to be either too complex to implement or not well-suited for rapid development needs focused on AI and machine learning features. Streamlit's simplicity and seamless integration with Python made it ideal for implementing features such as CV building, career visualization, skill gap analysis, and the chatbot interface. Its real-time interactivity enabled rapid prototyping and iterative enhancements based on user feedback.
2. **OpenAI (openai==0.27.8)**
   * **Description:** The OpenAI library provides access to OpenAI's suite of language models, including GPT-4. Version 0.27.8 refers to the specific release of the OpenAI Python client library used in this project.
   * **Usage in Project:** OpenAI's GPT-4 was integrated to handle natural language processing tasks such as resume analysis, career path generation, skill gap analysis, and chatbot interactions. By leveraging GPT-4, the Career Advisor can generate personalized recommendations, understand and respond to user queries intelligently, and process textual data effectively.
3. **pdfplumber**
   * **Description:** pdfplumber is a Python library that enables the extraction of text, tables, and metadata from PDF documents. It provides a simple API for parsing and analyzing PDF content.
   * **Usage in Project:** pdfplumber was utilized to extract and process text data from user-uploaded resumes (CVs) in PDF format. This extraction is crucial for analyzing users' skills, experiences, and qualifications, which in turn informs the personalized job recommendations provided by the Career Advisor.
4. **python-docx**
   * **Description:** python-docx is a Python library for creating, updating, and reading Microsoft Word (.docx) files. It allows for the manipulation of document elements such as paragraphs, tables, and styles programmatically.
   * **Usage in Project:** python-docx was employed to generate and format user CVs in Word document format. This functionality enables users to create professional and well-structured CVs based on their input, which can be customized and exported for various job applications.
5. **ReportLab**
   * **Description:** ReportLab is a robust Python library for generating PDFs programmatically. It offers extensive features for creating complex layouts, including text, images, tables, and graphics.
   * **Usage in Project:** ReportLab was used to generate downloadable PDF versions of user-generated CVs. This ensures that the CVs are consistently formatted and professionally presented, making them suitable for submission to potential employers.
6. **NetworkX**
   * **Description:** NetworkX is a Python library for the creation, manipulation, and study of complex networks and graph structures. It provides tools for graph analysis, visualization, and implementing various graph algorithms.
   * **Usage in Project:** NetworkX was employed to model and analyze the relationships between different career paths, skills, courses, and projects. This modeling is essential for developing the Career Visualization feature, which maps out the user's career journey and highlights the connections between various roles and necessary competencies.
7. **PyVis**
   * **Description:** PyVis is a Python library for creating interactive network visualizations in web browsers. It builds on top of NetworkX to provide dynamic and aesthetically pleasing graph representations.
   * **Usage in Project:** PyVis was used to render interactive visualizations of the users' career paths, courses, and projects. This enhanced the Career Visualization feature by allowing users to explore their career journeys through an engaging and intuitive graphical interface, facilitating a better understanding of their professional progression.
8. **PyMuPDF (pymupdf)**
   * **Description:** PyMuPDF is a Python binding for the MuPDF PDF rendering library. It allows for high-performance PDF processing, including text extraction, page manipulation, and rendering.
   * **Usage in Project:** PyMuPDF was utilized alongside pdfplumber to ensure robust and accurate extraction of content from user-uploaded PDF resumes. It provided additional capabilities for handling complex PDF structures and improved the reliability of text extraction processes.
9. **Flask**
   * **Description:** Flask is a lightweight Python web framework that provides the essentials for building web applications.
   * **Usage Consideration:** Initially considered for developing the Career Advisor's user interface. However, Flask requires extensive frontend development efforts to create interactive and user-friendly interfaces, which was beyond the team's primary focus on AI and machine learning functionalities.
10. **Gradio**
    * **Description:** Gradio is an open-source Python library for building machine learning and data science demos and web applications quickly.
    * **Usage Consideration:** Evaluated for its ease of integration with machine learning models. Despite its simplicity, Gradio lacked the flexibility and customization required for implementing complex features like career visualization and skill gap analysis.
11. **Django**
    * **Description:** Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.
    * **Usage Consideration:** Although Django offers robust features for building scalable web applications, its complexity and steeper learning curve made it less suitable for the team's immediate needs focused on developing AI-driven functionalities with limited frontend development resources.

***9. Design and Architecture***

**System Architecture Overview**

The AI-Powered Career Advisor is structured into several interconnected components, each responsible for specific functionalities. The architecture ensures seamless data flow, efficient processing, and an intuitive user experience.

1. **Frontend Interface:**
   * **Streamlit Framework:** Serves as the primary interface through which users interact with the platform. It manages user inputs, displays outputs, and facilitates navigation between different features such as CV building, career visualization, and skill gap analysis.
   * **User Interaction Elements:** Includes widgets for data input (e.g., forms for CV creation), interactive visualizations for career paths, and a chat interface for the chatbot.
2. **Backend Services:**
   * **AI Models:**
     + **GPT-4:** Handles natural language processing tasks, including resume analysis, generating job recommendations, and powering the chatbot.
     + **Word2Vec Embedding Model:** Analyzes the relationships between skills and job occupations to enhance recommendation accuracy.
   * **API Integrations:**
     + **Adzuna API:** Provides real-time job listings based on user profiles.
     + **Coursera API:** Supplies relevant courses to help users bridge skill gaps.
     + **GitHub API:** Offers project opportunities for users to build practical experience.
   * **Data Processing Modules:**
     + **Resume Parsing:** Utilizes pdfplumber and PyMuPDF to extract information from uploaded resumes.
     + **CV Generation:** Employs python-docx and ReportLab to create downloadable CVs in DOCX and PDF formats.
     + **Feedback Processing:** Collects and analyzes user feedback to refine recommendation algorithms.
3. **Data Storage:**
   * **User Data:** Stores user profiles, uploaded resumes, generated CVs, and interaction history securely.
   * **Feedback Data:** Maintains records of user feedback to inform continuous improvement processes.
4. **Integration Layer:**
   * **API Connectors:** Facilitates communication between the frontend interface and various backend services, including AI models and external APIs.
   * **Data Flow Management:** Ensures efficient data transfer and processing across different components, maintaining system responsiveness and reliability.
5. **Deployment Infrastructure:**
   * **Cloud Hosting:** Utilizes a scalable cloud platform (e.g. Azure, or Heroku) to host the application, ensuring high availability and performance.
   * **Security Measures:** Implements robust security protocols to protect user data and maintain privacy standards.

***10. Implementation Details***

The implementation of the AI-Powered Career Advisor involved several key steps, from data collection and preprocessing to model integration and feature development. Below is a detailed breakdown of each phase:

1. **Data Collection and Preprocessing:**
   * **Dataset Acquisition:** Gathered data from GitHub, Kaggle, UCI Machine Learning Repository, and the O\*NET Database. These datasets included information on job roles, required skills, industry trends, and user profiles.
   * **Data Cleaning:** Removed duplicates, handled missing values, and standardized data formats to ensure consistency and quality across all datasets.
2. **Model Development and Integration:**
   * **Base Model Creation:** Developed an initial question-answering model using GPT-4 to handle basic user interactions.
3. **Feature Development:**
   * **CV Upload and Analysis:**
     + **Implementation:** Enabled users to upload resumes in PDF format. Utilized pdfplumber and PyMuPDF to extract textual data from resumes.
     + **Analysis:** Integrated GPT-4 to analyze extracted data, identifying key skills, experiences, and qualifications to generate tailored job recommendations.
   * **Build Your CV:**
     + **Implementation:** Developed an interface using Streamlit for users to input their personal information, education, skills, and projects.
     + **Generation:** Employed python-docx and ReportLab to create customizable CVs in DOCX and PDF formats based on user input.
   * **Career Visualization:**
     + **Implementation:** Used NetworkX and PyVis to map out potential career paths, incorporating job recommendations, relevant courses, and project opportunities.
     + **Visualization:** Created interactive graphs that allow users to explore and understand their career progression visually.
   * **Skill Gap Analysis:**
     + **Implementation:** Designed a questionnaire within Streamlit to assess users' current skill levels.
     + **Analysis:** Utilized GPT-4 to process assessment results, identifying skill gaps and recommending targeted courses and projects.
   * **Feedback Mechanism:**
     + **Implementation:** Integrated feedback forms within the Streamlit interface to capture user satisfaction and suggestions during career path generation.
     + **Processing:** Analyzed feedback data using GPT-4 to refine recommendation algorithms and improve overall platform functionality.
   * **Chatbot Functionality:**
     + **Implementation:** Developed a chatbot interface using Streamlit, connected to GPT-4 via OpenAI's API.
     + **Customization:** Configured the chatbot to act as a career advisor, capable of answering user queries, providing recommendations, and guiding users through platform features.
4. **Embedding Model Implementation:**
   * **Word2Vec Integration:** Implemented a Word2Vec embedding model to better understand and predict relationships between skills and job occupations.
   * **Training:** Trained the embedding model on a curated dataset of skills and job titles to capture semantic similarities and associations.
   * **Integration:** Linked the embedding model with the feedback mechanism to refine job title predictions based on user interactions and feedback.
5. **System Integration:**
   * **API Connections:** Established connections between Streamlit, GPT-4, Adzuna, Coursera, and GitHub APIs to facilitate data flow and feature functionality.
   * **Data Flow Management:** Ensured efficient data processing and communication between frontend and backend components, maintaining system responsiveness.
6. **Testing and Optimization:**
   * **Functional Testing:** Verified that all features functioned as intended, identifying and resolving bugs and issues.
   * **Performance Optimization:** Optimized code and processes to enhance application speed and reduce latency.
   * **User Testing:** Conducted user testing sessions to gather feedback on usability and effectiveness, implementing iterative improvements based on insights gained.
7. **Deployment:**
   * **Infrastructure Setup:** Configured cloud hosting services to deploy the application, ensuring scalability and reliability.
   * **Security Implementation:** Applied security measures to protect user data and maintain privacy standards.
   * **Final Deployment:** Launched the AI-Powered Career Advisor, making it accessible to end-users.

***11. Challenges and Solutions***

Developing the AI-Powered Career Advisor presented several challenges, each requiring strategic solutions to ensure project success.

1. **Resource Constraints:**
   * **Challenge:** Limited computational resources and budget made it infeasible to train large-scale models from scratch.
   * **Solution:** Leveraged pre-trained models like OpenAI's GPT-4, utilizing cloud-based services and APIs to mitigate the need for extensive in-house resources.
2. **Data Integration Complexity:**
   * **Challenge:** Integrating multiple datasets from various sources (e.g., O\*NET, Adzuna, Coursera, GitHub) posed challenges in terms of consistency and compatibility.
   * **Solution:** Implemented robust data preprocessing pipelines to clean, standardize, and structure data uniformly, ensuring seamless integration across different components.
3. **RAG System Ineffectiveness:**
   * **Challenge:** The initial implementation of a Retrieval-Augmented Generation (RAG) system using career guidance books did not yield accurate or relevant recommendations.
   * **Solution:** Abandoned the RAG approach and instead focused on fine-tuning GPT-4 with the O\*NET Database and implementing a Word2Vec embedding model to enhance recommendation accuracy.
4. **Fine-Tuning GPT-4:**
   * **Challenge:** Fine-tuning GPT-4 with the O\*NET Database required meticulous data formatting and handling to ensure effective training.
   * **Solution:** Structured the O\*NET data into conversational Q&A pairs, enabling GPT-4 to learn and generate contextually accurate responses based on occupational data.
5. **Handling Heavy API Outputs:**
   * **Challenge:** The Adzuna API returned large volumes of data, which could overwhelm the system and degrade performance.
   * **Solution:** Implemented data management strategies such as result filtering, pagination, and caching to efficiently handle and optimize heavy API outputs, ensuring smooth performance and a seamless user experience.
6. **User Engagement and Feedback:**
   * **Challenge:** Ensuring high levels of user engagement and effectively capturing meaningful feedback for continuous improvement.
   * **Solution:** Developed intuitive feedback mechanisms and interactive features like the chatbot and career visualization tools to engage users actively. Analyzed feedback using GPT-4 to inform iterative enhancements.
7. **Balancing Functionality and Usability:**
   * **Challenge:** Integrating complex AI functionalities while maintaining an intuitive and user-friendly interface.
   * **Solution:** Chose Streamlit for its simplicity and rapid development capabilities, allowing the team to focus on functionality without compromising on user experience. Conducted user testing to refine interface elements based on real-world usage.

***12. Features and Functionality***

The AI-Powered Career Advisor encompasses a range of features designed to provide comprehensive and personalized career guidance. Below is an overview of the core functionalities:

1. **CV Upload and Analysis:**
   * **Functionality:** Allows users to upload their resumes in PDF format.
   * **Process:** Extracts textual data from uploaded resumes using pdfplumber and PyMuPDF, then analyzes the content with GPT-4 to identify key skills, experiences, and qualifications.
   * **Outcome:** Generates tailored job recommendations aligned with the user's profile.
2. **Build Your CV:**
   * **Functionality:** Enables users to create professional CVs from scratch.
   * **Process:** Users input personal information, education, skills, and projects through an intuitive interface. The system formats the input into a structured CV using python-docx and ReportLab.
   * **Customization:** Offers options to customize layout and content to suit different job applications.
   * **Export:** Allows users to download their CVs in DOCX and PDF formats.
3. **Career Visualization:**
   * **Functionality:** Provides an interactive visualization of potential career paths.
   * **Process:** Utilizes NetworkX and PyVis to map out a 4-step career journey, including sequential job recommendations, relevant courses from Coursera, and project opportunities from GitHub.
   * **Interactivity:** Users can explore different career trajectories, understanding the progression and necessary steps to advance.
4. **Skill Gap Analysis:**
   * **Functionality:** Assesses users' current skill levels and identifies areas for improvement.
   * **Process:** Users complete a questionnaire evaluating their proficiency in various skills. GPT-4 analyzes the results to pinpoint skill gaps.
   * **Recommendations:** Provides personalized suggestions for courses and projects to bridge identified gaps, sourced from Coursera and GitHub APIs.
5. **Job Recommendations:**
   * **Functionality:** Offers tailored job suggestions based on user profiles.
   * **Process:** Analyzes extracted resume data and user inputs using GPT-4 and the Word2Vec embedding model to predict suitable job titles.
   * **Data Source:** Fetches real-time job listings from the Adzuna API.
6. **Interactive Chatbot:**
   * **Functionality:** Acts as a virtual career advisor, providing real-time assistance and personalized guidance.
   * **Process:** Users can ask specific questions or seek support on various career-related topics. The chatbot, powered by GPT-4, responds intelligently and contextually.
   * **Capabilities:** Answers queries, offers recommendations, and guides users through platform features.
7. **Feedback Mechanism:**
   * **Functionality:** Captures user feedback to inform continuous improvement.
   * **Process:** Users provide feedback during and after career path generation through integrated forms.
   * **Analysis:** GPT-4 analyzes feedback to refine recommendation algorithms and enhance overall platform functionality.
8. **API Integrations:**
   * **Adzuna API:** Supplies real-time job listings tailored to user profiles.
   * **Coursera API:** Provides relevant courses to help users develop necessary skills.
   * **GitHub API:** Offers project opportunities for users to build practical experience and demonstrate competencies.

***13. Testing and Validation***

Ensuring the reliability, accuracy, and user-friendliness of the AI-Powered Career Advisor was paramount. The testing and validation process encompassed multiple stages:

1. **Unit Testing:**
   * **Purpose:** Verified the functionality of individual components and modules.
   * **Process:** Tested each feature (e.g., CV upload, resume analysis, CV generation) independently to ensure they operated as intended.
2. **Integration Testing:**
   * **Purpose:** Ensured seamless interaction between integrated components (e.g., Streamlit frontend with GPT-4 backend).
   * **Process:** Conducted end-to-end tests to validate data flow and functionality across different modules and APIs.
3. **Performance Testing:**
   * **Purpose:** Assessed the application's responsiveness and scalability under various loads.
   * **Process:** Simulated multiple concurrent users to evaluate system performance, identifying and addressing bottlenecks.
4. **User Acceptance Testing (UAT):**
   * **Purpose:** Gathered feedback from actual users to evaluate the platform's usability and effectiveness.
   * **Process:** Organized testing sessions with a group of target users, who interacted with the platform and provided insights on their experience.
5. **Feedback Analysis:**
   * **Purpose:** Analyzed user feedback to identify areas for improvement.
   * **Process:** Utilized GPT-4 to process and interpret feedback, informing iterative enhancements to features and functionalities.
6. **Accuracy Validation:**
   * **Purpose:** Ensured the precision of AI-generated recommendations and analyses.
   * **Process:** Compared AI-generated job recommendations and skill gap analyses against industry standards and expert opinions to verify accuracy.
7. **Security Testing:**
   * **Purpose:** Verified the application's security measures to protect user data.
   * **Process:** Conducted vulnerability assessments and penetration testing to identify and mitigate potential security threats.

**Testing Outcomes:**

* **Functionality:** All core features operated as intended, with CV upload and analysis, career visualization, and the chatbot providing accurate and relevant outputs.
* **Performance:** The application maintained high responsiveness and stability under simulated user loads, with optimized data handling ensuring smooth performance.
* **Usability:** User feedback highlighted the platform's intuitive interface and comprehensive feature set, with suggestions for minor enhancements to further improve user experience.
* **Accuracy:** AI-generated recommendations closely aligned with user profiles and industry standards, demonstrating high accuracy and reliability.

**Iterative Improvements:**

* **Interface Refinements:** Enhanced the user interface based on UAT feedback, improving navigation and accessibility of features.
* **Recommendation Algorithms:** Fine-tuned recommendation algorithms using feedback data to increase the relevance and personalization of job suggestions.
* **Chatbot Enhancements:** Improved the chatbot's conversational flows and response accuracy based on user interactions and feedback.

***14. Deployment***

The deployment of the AI-Powered Career Advisor involved meticulous planning and execution to ensure a smooth transition from development to a live environment.

1. **Deployment Environment Selection:**
   * **Platform:** Chose a scalable cloud platform (e.g. Azure, or Heroku) to host the application, ensuring high availability and performance.
   * **Reasons for Choice:** The selected platform offered robust support for Python-based applications, seamless integration with APIs, and the ability to scale resources based on user demand.
2. **Infrastructure Setup:**
   * **Server Configuration:** Configured cloud servers to host the Streamlit application, ensuring optimal performance and reliability.
   * **Database Integration:** Set up secure databases to store user data, feedback, and interaction history, implementing encryption and access controls to protect sensitive information.
3. **Security Implementation:**
   * **Data Protection:** Implemented SSL/TLS protocols to secure data transmission between users and the application.
   * **Authentication:** Established secure user authentication mechanisms to prevent unauthorized access.
   * **Compliance:** Ensured compliance with data privacy regulations (e.g., GDPR) by incorporating necessary consent forms and data handling procedures.
4. **Continuous Integration and Deployment (CI/CD):**
   * **Automation:** Set up CI/CD pipelines to automate testing, building, and deployment processes, facilitating rapid and reliable updates.
   * **Version Control:** Utilized Git for version control, maintaining a clean and organized codebase.
5. **Final Testing:**
   * **Pre-Deployment Testing:** Conducted comprehensive testing in the production environment to verify that all features functioned correctly post-deployment.
   * **Load Testing:** Performed additional performance testing to ensure the application could handle real-world user loads without degradation.
6. **Deployment Execution:**
   * **Launch:** Deployed the AI-Powered Career Advisor to the cloud platform, making it accessible to end-users.
   * **DNS Configuration:** Configured domain settings to map a user-friendly URL to the deployed application.
7. **Post-Deployment Monitoring:**
   * **Performance Monitoring:** Implemented monitoring tools to track application performance, uptime, and user interactions in real-time.
   * **Error Logging:** Set up logging systems to capture and alert on any errors or issues that arise, enabling swift resolution.
   * **User Support:** Established support channels to assist users with any deployment-related queries or issues.
8. **Maintenance and Updates:**
   * **Regular Updates:** Scheduled regular updates to introduce new features, improve existing functionalities, and address any security vulnerabilities.
   * **User Feedback Integration:** Continuously incorporated user feedback into the development cycle to enhance the platform's effectiveness and user satisfaction.

**Deployment Outcome:** The AI-Powered Career Advisor was successfully deployed on a scalable cloud platform, providing a stable and responsive environment for users. Post-deployment monitoring confirmed the application's reliability and performance, with initial user feedback indicating high satisfaction with the platform's functionalities and usability.

***15. Future Enhancements***

To ensure the AI-Powered Career Advisor remains relevant and continues to meet users' evolving needs, the following future enhancements are proposed:

1. **Advanced Personalization:**
   * **Dynamic Learning:** Implement machine learning algorithms that adapt to individual user behaviors and preferences over time, offering increasingly personalized recommendations.
   * **Career Path Customization:** Allow users to customize their career paths further by selecting preferred industries, company sizes, and work environments.
2. **Enhanced Chatbot Capabilities:**
   * **Multi-Language Support:** Expand the chatbot's language capabilities to cater to a global user base.
   * **Voice Interaction:** Integrate voice recognition and synthesis technologies to enable voice-based interactions with the chatbot.
   * **Contextual Memory:** Enhance the chatbot's ability to retain context across multiple interactions, providing more coherent and contextually aware responses.
3. **Integration with More APIs:**
   * **Additional Job Boards:** Incorporate APIs from other job boards like Glassdoor and Monster to provide a wider range of job listings.
   * **Educational Platforms:** Integrate with platforms like Udemy and edX to offer a broader selection of courses.
   * **Professional Networks:** Connect with LinkedIn's API to allow users to import their professional profiles directly into the platform.
4. **Mobile Application Development:**
   * **Mobile Accessibility:** Develop dedicated mobile applications for iOS and Android to provide users with on-the-go access to career guidance features.
   * **Push Notifications:** Implement push notifications to alert users about new job listings, course enrollments, and upcoming deadlines.
5. **Enhanced Data Analytics:**
   * **User Analytics Dashboard:** Create dashboards for users to track their progress, skill development, and career milestones.
   * **Platform Analytics:** Develop administrative analytics tools to monitor platform usage, feature popularity, and user engagement metrics.
6. **Gamification Elements:**
   * **Achievement Badges:** Introduce badges and rewards for users who complete courses, projects, or achieve career milestones.
   * **Progress Tracking:** Implement progress bars and milestones to motivate users and visualize their career development journey.
7. **Community Building:**
   * **User Forums:** Establish forums or discussion boards where users can share experiences, ask questions, and provide peer support.
   * **Mentorship Programs:** Facilitate mentorship connections between users and industry professionals to provide personalized guidance and networking opportunities.
8. **Enhanced Security Features:**
   * **Two-Factor Authentication (2FA):** Implement 2FA to add an extra layer of security to user accounts.
   * **Data Anonymization:** Employ data anonymization techniques to protect user privacy while enabling data-driven insights.
9. **AI Model Upgrades:**
   * **Latest AI Models:** Continuously integrate the latest advancements in AI and machine learning to enhance the platform's capabilities.
   * **Custom AI Models:** Develop proprietary AI models tailored specifically to the platform's unique requirements for even greater accuracy and personalization.
10. **Accessibility Improvements:**
    * **Inclusive Design:** Ensure the platform adheres to accessibility standards (e.g., WCAG) to accommodate users with disabilities.
    * **Language Options:** Expand language options to support a diverse, global user base.

***16. Conclusion***

The AI-Powered Career Advisor successfully addresses the critical need for personalized, data-driven career guidance within the technology sector. Developed over an intensive eight-week period, the platform integrates advanced AI models, comprehensive data sources, and user-centric features to empower students and early-career professionals in making informed career decisions. Through features like CV upload and analysis, career visualization, skill gap assessment, and an interactive chatbot, the advisor offers a holistic approach to career planning.

Despite encountering challenges such as resource constraints and initial limitations with the RAG system, the team effectively pivoted strategies, leveraging pre-trained models and fine-tuning GPT-4 with the O\*NET Database to enhance recommendation accuracy. Rigorous testing and user feedback integration ensured the platform's reliability, usability, and effectiveness.

Looking forward, the AI-Powered Career Advisor is poised for significant growth and enhancement, with planned features aimed at further personalization, expanded integrations, and broader accessibility. The project's success underscores the potential of AI in transforming career advisory services, providing users with the tools and insights needed to navigate their professional journeys confidently.

***18. Acknowledgments***

We extend our deepest gratitude to the following individuals and organizations for their invaluable support and contributions to the AI-Powered Career Advisor project:

* **Professor Ammar Mohanna:** For his expert guidance and mentorship throughout the project, helping us navigate technical challenges and refine our approach.
* **OpenAI:** For providing access to GPT-4, enabling the integration of advanced natural language processing capabilities into our platform.
* **Adzuna, Coursera, and GitHub:** For offering their APIs, which were instrumental in enriching our platform with real-time job listings, relevant courses, and project opportunities.
* **O\*NET Database:** For supplying comprehensive occupational data that significantly enhanced the accuracy and relevance of our job recommendations.
* **Our Peers and Test Users:** For their participation in testing sessions and providing constructive feedback, which was crucial in refining our application's features and user experience.
* **Family and Friends:** For their unwavering support and encouragement throughout the development process.
* **Open Source Community:** For contributing to the libraries and tools that made the development of this project possible.

***Next Steps***

The documentation for the AI-Powered Career Advisor has been comprehensively completed, encompassing all critical sections from ideation to deployment, along with detailed explanations of features, methodologies, and future enhancements. This documentation serves as a foundational reference for understanding the project's scope, implementation, and potential growth avenues.

For any further assistance or additions, feel free to reach out.

**app/career\_advisor.py**

This Python module, career\_advisor.py, is the main application script for the AI-Powered Career Advisor. It integrates functionalities such as parsing user resumes, extracting data, generating career recommendations, and interacting with external APIs to suggest jobs, courses, and projects. Below is a detailed explanation of its components:

**Imports**

* **Utility Functions**:
  + extract\_text: Extracts text from uploaded files (PDF or DOCX).
  + Functions like extract\_personal\_info, extract\_skills, get\_country\_code\_and\_location, and determine\_user\_level handle data extraction and user profiling.
* **Data Processing**:
  + Functions from data/profile\_data handle the extraction of detailed resume components (education, experience, etc.) and generate recommendations for jobs, courses, and projects.
* **External APIs**:
  + search\_courses\_on\_coursera, search\_jobs\_on\_adzuna, and search\_github\_projects enable interaction with Coursera, Adzuna, and GitHub APIs for fetching relevant resources.
* **Streamlit**:
  + Streamlit (st) provides the interface for displaying content and interacting with users.
* **Miscellaneous**:
  + Libraries such as textwrap, requests, and json manage API calls, handle text formatting, and process data.

**Global Variables**

1. **API Keys and Credentials**:
   * openai.api\_key: Stores the OpenAI API key.
   * ADZUNA\_APP\_ID and ADZUNA\_APP\_KEY: Used for Adzuna job search API.
   * coursera\_auth\_url, coursera\_payload, coursera\_token: Handles authentication and token retrieval for the Coursera API.

**Core Functions**

**display\_recommendations(resume\_text, location, country\_code)**

Displays career recommendations across three tabs: Jobs, Courses, and Projects.

* **Jobs**: Fetches job listings using the Adzuna API and displays relevant details like company, location, and salary.
* **Courses**: Displays recommended courses based on the user’s career path.
* **Projects**: Suggests GitHub projects relevant to the user’s career interests.

**regenerate\_recommendations(resume\_text, current\_feedback)**

Uses user feedback to regenerate career path recommendations.

* Updates session state variables for job, course, and project paths.
* Fetches updated course details from Coursera.

**generate\_initial\_recommendations(resume\_text, current\_feedback)**

Generates initial recommendations based on the resume text and feedback.

* Similar functionality to regenerate\_recommendations but for first-time generation.

**display\_extracted\_data()**

Displays extracted data from the user’s resume, including:

* User level
* Location
* Personal information
* Skills, education, experience, languages, awards, and projects

**career\_advisor()**

Main entry point for the Career Advisor application.

1. **Instructions**:
   * Guides the user on how to use the tool (uploading CV, analyzing data, browsing recommendations).
2. **Session State Initialization**:
   * Ensures all necessary session state variables are initialized (e.g., resume\_text, job\_path, course\_details).
3. **Feedback Form**:
   * Allows users to provide feedback on the recommendations.
   * Regenerates recommendations based on the feedback.
4. **Data Display**:
   * Displays extracted resume data and career recommendations (if available).

**Session State Variables**

* **Resume Analysis**:
  + resume\_text, user\_level, location, country\_code, personal\_info
* **Recommendations**:
  + job\_path, course\_path, project\_path, course\_details
* **Extracted Data**:
  + skills, education, experience, languages, awards, projects
* **Feedback**:
  + feedback

**External APIs**

1. **Adzuna API**:
   * Fetches job listings based on job title and location.
2. **Coursera API**:
   * Retrieves course suggestions using OAuth2 authentication.
3. **GitHub API**:
   * Searches for relevant projects based on job titles.

**User Workflow**

1. Upload a CV.
2. Extracted data is displayed for user review.
3. Initial career recommendations (jobs, courses, projects) are generated.
4. User provides feedback to refine recommendations.
5. Updated recommendations are displayed interactively.

**Dependencies**

* **Python Libraries**:
  + requests, streamlit, openai
* **Custom Modules**:
  + utils.file\_utils, data.profile\_data, utils.resume\_utils, data.api\_data

**Security Notes**

* **Sensitive Information**:
  + Ensure API keys and credentials (e.g., openai.api\_key, ADZUNA\_APP\_KEY) are stored securely and not hardcoded in production.
* **Error Handling**:
  + API calls and data processing should include robust error handling to prevent crashes.

**Career Path Visualization:**

The Career Path Visualization code provides an interactive and personalized way to visualize career progression. The tool maps job titles, recommended courses, and related projects onto a graph, showing connections between different career stages. The goal is to guide users in their career development by offering insights into potential paths and opportunities.

**Code Files Explanation**

This file serves as the primary interface for users, utilizing Streamlit to display the interactive graph and relevant information.

**Key Functions**

1. **career\_path\_visualization()**:
   * **Purpose**: Generates and displays the career path graph based on session data.
   * **Inputs**:
     + job\_path: A list of job titles.
     + course\_path: A list of recommended courses or skills.
     + project\_path: A list of related projects.
   * **Outputs**:
     + An interactive HTML-based career graph displayed in Streamlit.
     + A personalized summary of the user’s current and next career steps.
   * **Components**:
     + **Graph Rendering**: Uses generate\_career\_path\_graph() from utils.graph\_utils.
     + **Temporary File Handling**: Saves the graph to a temporary HTML file for embedding in Streamlit.
     + **Streamlit Visualization**: Displays the graph and career summary.
2. **generate\_career\_path\_graph()**:
   * **Purpose**: Creates a network graph connecting jobs, courses, and projects.
   * **Inputs**:
     + Three lists (jobs, courses, projects).
   * **Outputs**:
     + An interactive HTML-based network graph.

**Variables Explained**

1. **API Keys and Credentials**:
   * **openai.api\_key**: Used for OpenAI API calls.
   * **ADZUNA\_APP\_ID and ADZUNA\_APP\_KEY**: Adzuna credentials for fetching job data.
   * **GITHUB\_TOKEN**: Token for accessing GitHub projects.
   * **coursera\_auth\_url and coursera\_payload**: For authenticating with Coursera to fetch recommended courses.
2. **Session State Variables**:
   * **job\_path**: Stores the list of job roles.
   * **course\_path**: Stores the list of courses.
   * **project\_path**: Stores the list of projects.

**Library Usage**

1. **Streamlit**:
   * **Purpose**: Provides the user interface and interactive visualization platform.
   * **Key Components**:
     + st.header(), st.markdown(), st.subheader(): Used to display headings and descriptive text.
     + st.components.v1.html(): Embeds the HTML graph within the Streamlit app.
2. **Requests**:
   * **Purpose**: Makes API calls to fetch Coursera and other data.
   * **Usage**:
     + Fetch Coursera OAuth tokens.
3. **Tempfile**:
   * **Purpose**: Creates temporary files to store the generated graph for embedding in Streamlit.
4. **Custom Utility Module**:
   * **utils.graph\_utils**: Contains the function to generate the career path graph.

**Step-by-Step Workflow**

1. **Data Input**:
   * Users must ensure job\_path, course\_path, and project\_path are stored in the session state.
2. **Graph Generation**:
   * The generate\_career\_path\_graph() function is called to create the graph using the provided data.
3. **Temporary File Creation**:
   * The generated graph is saved as an HTML file in a temporary directory.
4. **Visualization in Streamlit**:
   * The HTML graph is embedded and displayed in the Streamlit interface.
5. **Career Summary**:
   * A personalized summary of the user’s current and next career steps is displayed alongside the graph.
6. **Descriptive Text**:
   * Guidance on how to interpret the graph is provided for better usability.

**Usage Guide**

1. **Prerequisites**:
   * Ensure API keys and credentials (OpenAI, Adzuna, GitHub, Coursera) are correctly set.
   * Data for jobs, courses, and projects must be available.
2. **Running the App**:
   * Execute streamlit run career\_path\_visualization.py to start the application.
3. **Interacting with the App**:
   * View the interactive graph.
   * Read personalized career summaries and recommendations.

**Future Enhancements**

* **Dynamic Data Fetching**: Integrate live job data from Adzuna or other sources.
* **Enhanced Visualization**: Add more styling options and advanced interactivity for the graph.
* **Broader Scope**: Include more APIs (e.g., LinkedIn) for data enrichment.

**Chatbot code:**

This application provides a conversational interface similar to ChatGPT, leveraging OpenAI's GPT-4 API. Built with the Streamlit framework, it offers interactive and personalized advice based on user profiles.

**Key Features**

1. **Interactive Chat Interface**: Allows seamless conversations between the user and the chatbot.
2. **Profile Integration**: Utilizes user-provided details (skills, education, experience, etc.) to generate personalized responses.
3. **Session Persistence**: Maintains chat history and profile data throughout the session.
4. **Debugging Context View**: Displays the exact input sent to GPT for transparency and debugging.

**Code Explanation**

**1. Libraries Used**

* **OpenAI**: Facilitates interaction with the GPT-4 API for generating chatbot responses.
* **Streamlit**: Provides a user-friendly web interface for interaction.
* **JSON**: Formats and displays complex data structures like dictionaries and lists.

**2. API Configuration**

The OpenAI API key is required for authenticating requests to the GPT-4 model. It should be kept secure and ideally stored in an environment variable for better security.

**3. Session Management**

Streamlit's session\_state is used to store chat history and profile data. This ensures persistence of conversation flow and user context across interactions during a session.

**4. Profile Context Extraction**

User-provided profile details are formatted into a structured context string. This serves as the system message for the chatbot, enabling it to offer tailored advice. The profile data includes user level, location, skills, education, experience, and suggested career paths, among other details.

**5. Chat Workflow**

* User input is captured through a text box in the interface.
* Messages from the user and responses from the chatbot are managed through the chat history stored in the session state.
* The OpenAI GPT-4 model generates responses by combining a system context with the user's conversation history.

**6. Chat History Display**

The chat interface includes a visually styled history panel that distinguishes user and assistant messages. Auto-scrolling functionality ensures that new messages are always visible.

**7. Debugging**

For transparency, the context sent to the GPT model is displayed. This aids in debugging by allowing developers to verify the input passed to the model.

**8. Optional Features**

* A "Clear Chat" button resets the conversation, enabling a fresh start.
* The application is designed to allow future expansion, such as enabling profile updates and integrating additional features.

**Deployment**

The application can be deployed using Streamlit. Running the script initializes a web interface where users can interact with the chatbot.

**Suggestions for Improvement**

1. **API Key Security**: Replace the hardcoded API key with a more secure method, such as using environment variables.
2. **Profile Editing**: Add functionality to allow users to update their profiles dynamically during the session.
3. **Dynamic Model Selection**: Provide options for users to choose between different OpenAI models.
4. **Response Formatting**: Enhance chatbot responses by adding support for rich text formatting, such as markdown.

**CV Builder Application**

**Overview**

This Streamlit application allows users to create a professional CV by filling in a series of forms about their personal information, education, skills, languages, professional experience, awards, and projects. Once the user completes the form, the app generates a profile summary and compiles the information into a downloadable PDF CV. The application leverages OpenAI and other external services to enhance user experience and data integration.

**Dependencies**

* **Streamlit**: A web framework for creating interactive applications.
* **Requests**: A library for making HTTP requests.
* **OpenAI**: Used for generating profile summaries.
* **app.profile\_app**: Contains functions for generating profile summaries and updating profiles.
* **utils.cv\_utils**: Contains the logic to create a CV PDF.

The application also uses external APIs such as OpenAI for generating summaries, Adzuna for job data, GitHub for token authentication, and Coursera for course data integration.

**Structure of the Code**

The code is structured into several key sections:

1. **API and Authentication Keys**
   * OpenAI API key: Used for generating profile summaries based on the user's input.
   * Adzuna API credentials: Not actively used in the code but set up for potential future integration related to job data.
   * GitHub Token: Not actively used in the code but set up for potential future integration related to GitHub authentication.
   * Coursera API: For retrieving Coursera courses (not actively used in the code but included for future use).
2. **Streamlit Form and UI Design**
   * The code uses st.markdown to customize the appearance of the application, including styling for headers, subheaders, and buttons.
   * The form is divided into three tabs:
     + **Basic Info**: Collects the user's personal details, such as name, job title, email, and contact details.
     + **Skills & Languages**: Allows users to input languages spoken and the skill levels they possess in various areas.
     + **Experience**: Collects professional experience, including job titles, company names, and years worked.
     + **Awards & Projects**: Enables users to input information about any awards they have received and projects they have worked on.
3. **Session State Management**
   * The app uses Streamlit's session state to manage dynamic form fields (languages, skills, education, experience, awards, and projects). It keeps track of the number of entries for each section and allows users to add more sections dynamically.
4. **CV Generation Logic**
   * After the user fills in the form and clicks the "Generate CV" button, the application:
     + Collects all the entered data.
     + Calls generate\_profile\_summary from app.profile\_app to generate a personalized summary based on the entered information.
     + Creates a PDF CV using the create\_cv\_pdf function from utils.cv\_utils.
     + Updates the profile using the update\_profile\_from\_cv function.
     + Displays a success message and provides a download button for the user to download their generated CV in PDF format.

**Key Functions**

1. **build\_cv()**
   * This is the main function that handles the entire CV generation process.
   * It contains the Streamlit UI code, the form fields for user input, and logic for adding dynamic fields (like languages, skills, experience).
   * Once the user completes the form and clicks the "Generate CV" button, it processes the input and triggers the PDF generation process.
2. **generate\_profile\_summary(data)** (imported from app.profile\_app)
   * This function takes the collected data (name, job title, skills, experience, etc.) and uses OpenAI's API to generate a professional summary for the user’s profile.
3. **create\_cv\_pdf(data, profile\_summary)** (imported from utils.cv\_utils)
   * This function uses the collected data and the profile summary to generate a PDF representation of the user's CV.
4. **update\_profile\_from\_cv(data)** (imported from app.profile\_app)
   * This function is used to update the user’s profile, potentially storing the data or updating the user’s profile in a database.

**Flow of Data**

1. The user enters their data into the form fields in the Streamlit UI.
2. When the "Generate CV" button is clicked, the entered data is passed to the backend functions.
3. The data is then processed:
   * A profile summary is generated using OpenAI’s API.
   * A PDF CV is created with the collected information and the generated summary.
   * The profile is updated using the collected data.
4. The generated CV PDF is made available for download via a Streamlit download button.

**UI Components**

* **Text Inputs**: Users can input various pieces of personal information such as name, job title, email, etc.
* **Select Boxes**: Used for selecting skill levels and language proficiency.
* **Text Areas**: Used for inputting details such as project descriptions and job responsibilities.
* **Dynamic Fields**: The user can dynamically add more languages, skills, experience, education, awards, and projects using a "➕ Add Another" button.

**API Credentials and Tokens**

* **OpenAI API Key**:
  + Used to generate profile summaries. Ensure that the API key is properly secured and not hardcoded in production environments.
* **Adzuna API ID and Key**:
  + Could be used for future job data integration.
* **GitHub Token**:
  + Reserved for future use in GitHub integrations.
* **Coursera API Token**:
  + Authenticates and retrieves data related to Coursera courses. This can be used to integrate courses into the CV in the future.

**Future Improvements**

* **Job Data Integration**: You can integrate job data from the Adzuna API to suggest relevant positions based on the user’s skills and experience.
* **Coursera Course Integration**: Link relevant Coursera courses based on the user's education and skills to improve the CV.
* **Enhanced Profile Summary**: Utilize OpenAI more to create advanced personalized summaries and add more profile enhancements.

**Ideal Job Application:**

1. **Dependencies**:
   * OpenAI: Used to generate text embeddings from OpenAI's API.
   * langchain and pgvector: Abstract the use of a vector database (PostgreSQL with PGVector) for storing and querying embeddings.
   * streamlit: Provides the user interface to interact with the app.
2. **Vector Store Initialization**:
   * The connection to a PostgreSQL database is established using PGVector to store text embeddings, allowing for efficient similarity searches.
3. **Embedding Generation**:
   * The generate\_embedding function generates embeddings for input text using the OpenAI API. These embeddings represent the semantic content of the text.
4. **Combining Skills**:
   * The combine\_skills\_from\_json function combines soft and technical skills into a single string that is used to query the database for matching documents.
5. **Similarity Search**:
   * The process\_combined\_skills function generates an embedding for the combined skills string, performs a similarity search in the vector database, and retrieves the top 10 matches. It then filters these results to show the top 3 distinct categories.
6. **Display Results**:
   * The find\_ideal\_job function integrates everything, displaying the user interface and showing the closest matching job categories and their content based on the user's skill set.

**Suggestions for Improvement:**

1. **API Key Security**:
   * It's important not to hardcode sensitive information like API keys directly in the code. Using environment variables (e.g., OPENAI\_API\_KEY) to store such keys is a more secure practice.
2. **Error Handling**:
   * Adding error handling, especially for API calls and database queries, can prevent the app from breaking due to external issues like network failures or invalid responses.
3. **UI Enhancements**:
   * Depending on the number of results, you might want to consider displaying them in a more organized format, such as cards or with expandable sections for better user experience.
4. **Category Filtering**:
   * You're filtering the results to display only the top 3 distinct categories, but it might be useful to allow the user to refine their search further or view all matches sorted by similarity.
5. **Input Validation**:
   * Before performing the similarity search, it would be helpful to check if the skills data exists in the session state and prompt the user to provide the necessary input if it's missing.

**Profile app:**

The edit\_profile function is used to allow users to edit their profile information within a Streamlit app. This function facilitates the dynamic addition and removal of sections such as personal information, skills, languages, education, professional experience, awards, and projects. The user's profile is stored in the session state and can be edited, updated, and saved.

**Functionality:**

* **Personal Information**: Users can update their name, job title, email, phone, location, date of birth, nationality, and LinkedIn URL.
* **Skills**: Users can add, edit, or remove skills. Skills are categorized into technical and soft skills.
* **Languages**: Users can update languages and their proficiency levels (Basic, Conversational, Fluent, etc.).
* **Education**: Users can add, edit, or remove their educational background, including degree, institution, years attended, and additional details.
* **Professional Experience**: Users can add or edit job titles, companies, years of employment, and job details.
* **Awards**: Users can list awards and honors they have received.
* **Projects**: Users can add or edit project titles and descriptions.

**Key Components:**

1. **Session State Management**:
   * The profile data is stored in st.session\_state.profile, which is a dictionary containing fields like name, job\_title, skills, etc.
   * It dynamically tracks the number of items (e.g., skills, languages, etc.) being edited through session state counters (e.g., edit\_skills\_count).
2. **Dynamic Editing**:
   * For each section (Skills, Languages, Education, etc.), the app generates input fields based on the current data.
   * Users can modify existing entries or add new ones.
   * Removal of items is also supported by providing a "Remove" button for each item.
3. **Adding New Entries**:
   * Each section (e.g., Skills, Languages) includes a button ("Add Skill", "Add Language", etc.) that allows users to add new entries to the respective section. This updates the session state and UI dynamically.
4. **Removing Entries**:
   * A "Remove" button next to each item in sections like Skills, Languages, Education, Experience, etc., allows users to delete specific entries from the profile.
   * When an entry is removed, the session state is updated to reflect the change, and the corresponding input fields are removed from the UI.
5. **UI Layout**:
   * The app uses columns (st.columns) to create a flexible layout for inputs. For example, in the Skills section, skill name input, level selection, and the "Remove" button are arranged in a three-column layout.
   * Each section is clearly labeled with a header (e.g., "Personal Information", "Skills", etc.) to guide the user.
6. **Skills Section**:
   * Skills are represented as a list of dictionaries in the profile (profile['skills']). Each skill consists of a skill (name) and level (expertise level).
   * Skills are edited dynamically, and changes are reflected in both the session state and the displayed profile.
7. **Languages Section**:
   * Languages are tracked with their names and proficiency levels. The level options are predefined as "Basic", "Conversational", "Proficient", "Fluent", and "Native/Bilingual".
   * Similar to the Skills section, users can add, edit, or remove languages from their profile.
8. **Education Section**:
   * This section captures the user's education history. Each entry includes Degree, Institution, Years, and additional Details.
   * Users can modify existing education entries or add new ones. The "Remove" button allows deletion of specific entries.
9. **Professional Experience Section**:
   * Users can add or edit job titles, companies, years of experience, and detailed job descriptions.
   * This section supports dynamic addition/removal of job experience entries.
10. **Awards Section**:
    * This section allows users to add or remove awards they have received.
11. **Projects Section**:
    * Users can add or edit project details, including project title and description.

**Additional Details:**

* **Skill Categorization**: The function is designed to accommodate categorization of skills (e.g., "soft skills" and "technical skills"), although categorization logic is hinted but not fully implemented in the provided code snippet.
* **Profile Save and Approve**: After editing, users can save or approve the profile changes. The profile is stored in the session state, and the user’s edits will persist for the current session.

**Notes:**

* **Session Persistence**: Changes made by the user are stored in st.session\_state and are only available for the duration of the session. To persist data across sessions, additional functionality such as saving to a database or file storage would be needed.
* **Error Handling**: The current function does not explicitly handle errors (e.g., invalid data or empty inputs), but these could be added through additional checks and user prompts.

**remove\_skill\_from\_categories(profile, skill\_key\_prefix)**

Removes a skill from both the soft skills and technical skills categories based on the skill name.

**Parameters:**

* **profile (dict)**: The user's profile that includes soft skills and technical skills.
* **skill\_key\_prefix (str)**: The prefix of the skill key in the session state used to identify which skill to remove.

**Description:**

* Extracts the skill name from the session state using the provided skill\_key\_prefix.
* Removes the skill from both the soft skills and technical skills lists within the profile.

**update\_profile\_from\_career\_advisor(user\_level, location, country\_code, personal\_info, skills, education, experience, languages, awards, projects)**

Updates the session state profile with data extracted from the career advisor.

**Parameters:**

* **user\_level (str)**: The user's career level.
* **location (str)**: The user's location.
* **country\_code (str)**: The user's country code.
* **personal\_info (dict)**: A dictionary containing personal information like name, email, phone, etc.
* **skills (list)**: A list of skills extracted from the career advisor.
* **education (list)**: A list of educational qualifications.
* **experience (list)**: A list of professional experiences.
* **languages (list)**: A list of languages spoken.
* **awards (list)**: A list of awards received.
* **projects (list)**: A list of projects completed.

**Description:**

* The function updates the session state profile with various attributes such as user level, location, and personal information.
* It ensures no duplicates exist in each section (skills, education, experience, etc.) by comparing the relevant fields (e.g., Degree, Title).
* The skills are categorized into soft\_skills and technical\_skills.

**update\_profile\_from\_cv(data)**

Updates the session state profile with data extracted from a CV.

**Parameters:**

* **data (dict)**: A dictionary containing extracted data from the CV, including skills, languages, education, experience, awards, and projects.

**Description:**

* The function merges the extracted data into the session state profile, ensuring no duplicate entries in skills, education, experience, languages, awards, and projects.
* It categorizes skills into soft\_skills and technical\_skills.

**generate\_profile\_summary(data)**

Generates a professional summary for the user's CV based on their profile data.

**Parameters:**

* **data (dict)**: A dictionary containing the user's profile data including name, job title, skills, experience, education, and languages.

**Description:**

* The function checks if the necessary fields for generating a summary are present.
* It constructs a prompt for an OpenAI model to generate a concise, professional profile summary.
* The summary is generated using the OpenAI ChatCompletion.create API, with a limit of 150 words.
* If any required fields are missing, it returns a message indicating which fields are missing.

**my\_profile()**

Displays the user’s profile and allows the user to edit it.

**Description:**

* The function checks if the profile is empty and displays a message to prompt the user to populate it.
* If in **edit mode**, the function allows the user to edit the profile.
* The profile displays sections including:
  + **Personal Information**: Name, job title, email, phone, location, date of birth, nationality, and LinkedIn.
  + **Technical Skills**: A list of the user’s technical skills.
  + **Soft Skills**: A list of the user’s soft skills.
  + **Languages**: A list of the user’s language skills.

**Notes:**

* **Session State**: The user profile is stored in the st.session\_state.profile dictionary.
* **Skills Categorization**: Skills are categorized into soft\_skills and technical\_skills, based on the skills attribute.
* **Data Uniqueness**: Duplicate entries are removed by using identifiers like skill names, degree and institution combinations, and project titles.
* **Edit Mode**: If st.session\_state.edit\_mode is True, the user can edit their profile directly. Otherwise, the profile is displayed with an option to switch to edit mode.

**Skill gap analysis:**

The skill\_gap\_analysis\_app.py is a part of the Skill Gap Analysis feature for a career advisor application. It allows users to perform an analysis of their technical skills, identify gaps based on their career level, and visualize the results through interactive charts.

**Features**

1. **Skill Gap Analysis**: Based on the user's profile, the app performs an analysis of their technical skills and compares the proficiency of each skill with a threshold determined by the user's career level.
2. **Radar Chart Visualization**: Displays a radar chart showing the proficiency of each technical skill in the user's profile.
3. **Bar Chart Visualization**: Displays a horizontal bar chart visualizing the user's proficiency in each skill.
4. **Overall Skill Completion**: Provides an overall percentage representing how close the user is to completing the required skill set for their career level.
5. **Profile Verification**: Before starting the analysis, the app verifies if the user's profile is available and approved.

**Functionality**

1. **skill\_analysis()**:
   * The main entry point for performing the skill gap analysis. It verifies whether the user's profile is available and approved before proceeding.
   * If the profile is valid, it calls the perform\_skill\_gap\_analysis() function to display the analysis.
2. **perform\_skill\_gap\_analysis(profile)**:
   * **User Profile**: The function receives the user's profile as input, which contains details such as the user's technical skills and career level.
   * **User Level & Thresholds**: Based on the user's career level (e.g., student, entry-level professional, etc.), the app defines a maximum skill completion threshold that serves as the target for proficiency.
   * **Skills Data**: The user's technical skills and proficiency levels are extracted from the profile.
   * **Skill Level Weights**: Each skill proficiency level (e.g., Beginner, Amateur, Competent) is associated with a relative weight that represents the percentage of maximum proficiency.
   * **Radar and Bar Chart**: The function generates two visualizations:
     + A radar chart displays the user's proficiency in each skill, where the chart is filled according to the user's skill levels.
     + A bar chart provides a horizontal view of the proficiency levels for each skill.
   * **Skill Completion Calculation**: The app calculates the average proficiency of all technical skills and determines the overall skill completion relative to the user's career level.
   * **Display and Feedback**: The app shows the radar and bar charts side by side and provides an overall progress bar indicating how close the user is to completing their skill set.
3. **Session State**:
   * The function stores the calculated total\_completion and max\_percentage in the session state, which can be used for further analysis or report generation.
4. **Progress Feedback**:
   * Displays a progress bar that visually represents the user's overall skill completion, along with a textual summary of the results.

**Key Components**

* **Logging**: Logs user levels, completion percentages, and other relevant information for debugging and analysis purposes.
* **OpenAI API**: The app uses OpenAI's API key (though not explicitly used in the provided code snippet).
* **Adzuna API, GitHub Token, and Coursera Credentials**: These credentials are initialized but not actively used within the code, potentially for future expansions like retrieving data from these services.

**Dependencies**

* **Streamlit**: Used for creating the interactive web interface.
* **Plotly**: Used for generating radar and bar charts.
* **Requests**: For making API calls to external services such as Coursera.
* **OpenAI**: For AI-related functionalities (not actively used in this version).
* **Logging**: For tracking and debugging the application.

**Error Handling**

* The app checks if the profile is available and approved before proceeding with the skill gap analysis. If the profile is incomplete or unapproved, the app informs the user and halts the analysis.
* Warnings are logged when the user’s level is unrecognized or if data is missing.

**Future Enhancements**

* Integration with more APIs for fetching additional skill data or job market trends.
* Personalized recommendations for skill improvement based on gaps identified.
* Generation of downloadable reports summarizing the analysis results.

**Api data:**

1. **API Key Management:**

* To enhance security, avoid embedding API keys directly in the code. Store them in environment variables or use a secrets manager to protect sensitive information.

1. **Error Handling:**

* While error handling is in place, you might want to add more detailed logging or retry logic for temporary failures in external API requests.

1. **Response Parsing:**

* Before parsing the API responses, ensure that the response is valid and that the status code indicates success. This will help avoid potential errors if the API doesn't return the expected format.

1. **Description Truncation:**

* You're truncating job descriptions to 300 characters, but it could be more user-friendly to format them with ellipses and make sure they fit neatly in the UI. Consider adding some visual consistency for these long descriptions.

1. **Function Refactoring:**

* Some of the API request logic is repeated across different functions. You could consider creating a helper function for making API calls to reduce redundancy and improve maintainability.

1. **Text Wrapping:**

* The textwrap module is imported but not used in your code. It could be beneficial to leverage this module to format text descriptions neatly when they exceed a certain width, ensuring consistency.

1. **Code Structure and Readability:**

* The code is fairly organized, but as your project grows, you may want to break it down into smaller, more manageable modules. This will improve maintainability and scalability, especially for larger codebases.

**Profile Data:**

This code implements a system that extracts various details from resumes, including education, professional experience, and languages. It leverages OpenAI's GPT-4 model to process the resume text and return structured data in JSON format. The code integrates multiple APIs, such as OpenAI, GitHub, Adzuna, and Coursera, and provides a user interface using Streamlit.

**Functional Components**

**1. OpenAI Integration**

The system uses OpenAI's GPT-4 model to analyze the resume text and extract specific information. The model is provided with a set of instructions and examples to guide the extraction process for each category (education, experience, languages). Each request to the OpenAI API includes the resume content and a predefined system message to instruct the model on how to extract the desired data.

**2. Extract Education Details**

* **Purpose**: This function processes a resume and extracts education-related details, such as degree, institution, years, and additional details (if available).
* **Output**: The extracted education details are returned in a JSON format containing an array of education entries. Each entry includes:
  + Degree
  + Institution
  + Years
  + Details (can be a string or list)

**3. Extract Professional Experience**

* **Purpose**: This function processes a resume to extract professional experience, including job title, company, years, and job details.
* **Output**: The extracted experience details are returned in JSON format. Each entry includes:
  + Title
  + Company
  + Years
  + Details

**4. Language Extraction and Detection**

* **Purpose**: This function identifies the languages mentioned in a resume and their proficiency levels. If no languages are explicitly listed, it tries to infer the language of the resume based on the content.
* **Output**: The extracted languages are returned in JSON format. Each language entry includes:
  + Language
  + Level (e.g., Native, Fluent, Intermediate)

**5. Streamlit Interface**

* **Purpose**: The system integrates with Streamlit to provide a simple user interface for displaying the extracted data. The extracted information, including education, experience, and languages, is shown in text areas for review.
* **Features**:
  + Displays the raw responses from the OpenAI API for debugging purposes.
  + Shows extracted information in a user-friendly format, with clear labels and structured output.
  + Handles errors and displays appropriate error messages for missing or invalid data.

**6. External API Integration**

* **Adzuna**: Used for job-related information and employment statistics (though not directly utilized in the functions shown, it is initialized for potential future extensions).
* **GitHub**: A GitHub token is initialized, which could be used for integrations or accessing public repositories.
* **Coursera**: The system uses Coursera API credentials to potentially fetch course data (although the usage within the code is not active in the current implementation).

**Error Handling**

The code is designed to handle various types of errors gracefully:

* **JSON Parsing Errors**: If the response from OpenAI does not contain valid JSON, an error message is displayed.
* **Invalid or Missing Data**: If required fields such as degree, institution, or title are missing, the system will notify the user.
* **API Errors**: If the OpenAI API request fails (e.g., due to network issues or rate limits), appropriate error messages are displayed.

**Security Considerations**

* **API Keys**: Sensitive API keys (for OpenAI, Adzuna, GitHub, and Coursera) are included in the code, which should be kept secure. It is recommended to store them in environment variables or configuration files instead of directly in the code to avoid security risks.

**Limitations**

* The system relies on the clarity and consistency of the resume format. If the resume content is not structured well or contains too many variations in formatting, the extraction process might not be accurate.
* The system may not always accurately detect all languages, especially if they are not explicitly listed in the resume text.
* While the Coursera and Adzuna APIs are initialized, their functionality is not fully integrated into the resume extraction process.

**Future Enhancements**

* **Enhanced API Integrations**: Integrating the Coursera and Adzuna APIs could provide additional information, such as relevant courses or job recommendations based on the extracted resume data.
* **User Input Handling**: Enhancing the Streamlit interface to allow users to upload resumes directly and manage the extracted data more interactively.
* **Data Validation**: Implementing more robust data validation and cleaning methods to handle edge cases and variations in resume formats.

**extract\_awards(resume\_text)**

* **Purpose:** Extracts awards or recognitions from a given resume text.
* **Parameters:**
  + resume\_text (str): The resume text from which awards need to be extracted.
* **Returns:**
  + awards\_list (list): A list of awards found in the resume, each represented as a string.
* **Detailed Explanation:**
  + The function sends a request to the OpenAI API to extract awards or recognitions in a JSON format.
  + It parses the JSON response and extracts the awards, which are returned as a list.
  + In case of errors, the function provides feedback via Streamlit components.
  + If no awards are found, an empty array is returned.

**2. extract\_projects(resume\_text)**

* **Purpose:** Extracts project details (Title and Description) from the provided resume text.
* **Parameters:**
  + resume\_text (str): The resume text from which project information should be extracted.
* **Returns:**
  + projects\_list (list): A list of dictionaries, each containing Title and Description of a project.
* **Detailed Explanation:**
  + This function sends the resume text to the OpenAI API, requesting the extraction of project information in a strict JSON format with a projects array.
  + Each project entry includes a Title and Description key.
  + If multiple descriptions are found in list format, they are combined into a single string.
  + The extracted projects are returned as a list, with error handling in case of invalid or missing JSON.

**3. generate\_job\_path(resume\_text, feedback)**

* **Purpose:** Generates a four-step career path based on the user’s resume and feedback.
* **Parameters:**
  + resume\_text (str): The user’s resume text.
  + feedback (str): Additional user feedback to guide the career path.
* **Returns:**
  + steps (list): A list of four career job titles, in order of progression.
* **Detailed Explanation:**
  + This function sends the resume and feedback to the OpenAI API, requesting a career path with clear job titles likely to appear on Adzuna.
  + The result is a list of job titles, with the system ensuring that each step reflects a progression in career levels.
  + The function returns only the job titles, stripping any unnecessary formatting or text.

**4. generate\_course\_path(resume\_text, feedback)**

* **Purpose:** Generates a four-step learning and development path with course recommendations based on the user’s career path, resume, and feedback.
* **Parameters:**
  + resume\_text (str): The user’s resume text.
  + feedback (str): Additional user feedback to guide the course path.
* **Returns:**
  + steps (list): A list of four course titles, each aligned with the corresponding career step.
* **Detailed Explanation:**
  + This function first generates a career path using generate\_job\_path(), then sends the resume, feedback, and career path to the OpenAI API to generate corresponding course titles.
  + These courses are typically found on platforms like Coursera and represent progressive learning aligned with the career steps.

**5. generate\_project\_path(resume\_text, feedback)**

* **Purpose:** Generates a four-step open-source project path based on the user’s career progression and feedback.
* **Parameters:**
  + resume\_text (str): The user’s resume text.
  + feedback (str): Additional user feedback to guide the project path.
* **Returns:**
  + steps (list): A list of four project titles, representing progressively complex open-source projects.
* **Detailed Explanation:**
  + This function first generates a career path using generate\_job\_path(), and then uses the resume and feedback to generate a sequence of GitHub project titles for each career step.
  + The generated projects help the user to build skills progressively, with each project corresponding to a higher career level.

**6. generate\_course\_descriptions(course\_details)**

* **Purpose:** Generates descriptions for the provided course details, especially for those without a description.
* **Parameters:**
  + course\_details (list): A list of dictionaries, where each dictionary contains course details (title, url, and possibly a description).
* **Returns:**
  + descriptions (list): A list of descriptions, either generated by the function (if the course has no description) or taken from the provided details.
* **Detailed Explanation:**
  + The function iterates through each course in the provided list, checking if a description exists.
  + If no description is available or the description is deemed insufficient, the function uses the OpenAI API to generate a concise description.
  + The generated descriptions are appended to the list and returned.

**General Notes:**

* **Error Handling:** In each function, error handling is implemented using try-except blocks to catch JSON parsing issues and other potential exceptions. If errors occur, appropriate error messages are shown through Streamlit.
* **API Usage:** The functions rely on OpenAI's API to process the provided input (resume text, feedback, etc.) and generate the required output (awards, projects, career path, etc.).
* **Streamlit Integration:** Several of the functions make use of Streamlit components (st.text\_area, st.info, st.error) to display debug information or user feedback. This integration helps with debugging and provides clear responses during the execution.

**CV Utils:**

This script generates a professional CV in PDF format using the reportlab library. It takes personal and professional information (such as name, job title, contact information, education, skills, etc.) and formats them neatly into a structured CV. The generated CV includes sections for personal details, profile summary, languages, skills, education, professional experience, awards, and projects. The layout is designed with appropriate fonts, styles, and margins.

**Key Functions:**

**1. draw\_wrapped\_text(c, text, x, y, max\_width, leading=12)**

**Purpose**:  
This function is responsible for drawing wrapped text on the canvas. It uses the Paragraph class from the reportlab library to handle the text wrapping and rendering on the PDF.

**Parameters**:

* c: The canvas object on which text is drawn.
* text: The text to be displayed.
* x: The x-coordinate for the starting point of the text.
* y: The y-coordinate for the starting point of the text.
* max\_width: The maximum width the text can occupy.
* leading (optional): Line height for the text. Defaults to 12.

**Returns**:  
The adjusted y-coordinate after the text is drawn, to be used for the next element on the page.

**2. create\_cv\_pdf(data, profile\_summary)**

**Purpose**:  
This function generates a CV in PDF format based on the data provided. It formats the information into various sections (e.g., personal details, education, skills, etc.) and saves it to a PDF file. The output is a structured document ready for printing or sharing.

**Parameters**:

* data: A dictionary containing the user's data (name, job title, contact info, etc.). The structure of the data is expected to be as follows:
  + name: The name of the individual.
  + job\_title: The job title of the individual.
  + email, phone, location, dob, nationality: Contact information fields.
  + linkedin: LinkedIn profile URL (optional).
  + languages: A list of languages spoken, each containing Language and Level.
  + skills: A list of skills, each containing skill and level.
  + education: A list of education details, each containing Degree, Institution, Years, and optional Details.
  + experience: A list of work experiences, each containing Title, Company, Years, and optional Details.
  + awards: A list of awards.
  + projects: A list of projects, each containing Title and Description.
* profile\_summary: A short profile description or summary of the individual's professional background.

**Returns**:  
A buffer containing the generated PDF file.

**Workflow:**

1. **Setting Up the Canvas**:  
   The script starts by setting up the canvas object using the reportlab canvas.Canvas class. It defines the page size as A4 and sets the margins for the page.
2. **Header Section**:
   * The name of the individual is displayed at the top of the page in a large, bold font.
   * The job title is displayed underneath the name in a smaller font.
   * A horizontal line is drawn beneath the header to separate it from the rest of the content.
3. **Contact Information**:
   * A string with the contact information (email, phone, location, etc.) is displayed below the header.
   * If the LinkedIn profile is available, it is appended to the contact information.
4. **Profile Summary**:
   * The "Profile Summary" section is created with a section title in bold and centered on the page.
   * The profile summary text is wrapped and displayed in the available space.
5. **Additional Sections**:
   * **Languages**: The "Languages" section lists the languages spoken and their proficiency levels.
   * **Skills**: The "Skills" section lists the skills and the proficiency levels.
   * **Education**: The "Education" section lists degrees, institutions, and years attended, with optional details about the courses.
   * **Professional Experience**: The "Professional Experience" section lists job titles, companies, and years of employment, along with details of job responsibilities.
   * **Awards**: The "Awards" section lists any professional awards or recognitions.
   * **Projects**: The "Projects" section lists titles of important projects, followed by descriptions.
6. **Footer with Page Number**:  
   A footer is added to the PDF with the page number, centered at the bottom of the page.
7. **Saving the PDF**:  
   The canvas.save() method is used to finalize the PDF document, and it is saved to a BytesIO buffer. The buffer is returned to be used or saved externally.

**Libraries Used:**

* **reportlab**:  
  The reportlab library is used for creating PDFs. It provides functionalities to draw text, lines, and shapes, and to format text in a structured way. Specifically, the canvas object is used to draw text, while the Paragraph class is used to handle text wrapping and advanced formatting.
* **requests**:  
  The requests library is used to interact with external APIs (like Coursera) to fetch tokens for authentication.

**Customization:**

You can customize the generated CV by modifying:

* The fonts used (change Times-Roman or Times-Bold to other available fonts).
* The colors used (adjust the header\_color, section\_color, and text\_color values).
* The margins and layout of the page.

**File Utils:**

**File: file\_utils.py**

This module provides utilities for extracting text from various document types, such as PDF and DOCX files. It defines functions to handle text extraction from these files using appropriate libraries.

**Imports**

* fitz (PyMuPDF): A library for extracting text from PDF files.
* docx.Document: A class from the python-docx package to read DOCX files and extract text from them.

**Functions**

**extract\_text(file, file\_type)**

**Description:**  
This function determines the file type (PDF or DOCX) and calls the appropriate extraction function based on the file type.

**Parameters:**

* file: The file object to extract text from. Should be in a binary format.
* file\_type: A string that specifies the type of file. Possible values are:
  + "pdf": Indicates the file is a PDF.
  + "docx": Indicates the file is a DOCX document.

**Returns:**

* A string containing the extracted text from the specified file.

**Raises:**

* ValueError: If an unsupported file type is provided.

**extract\_text\_from\_pdf(file)**

**Description:**  
Extracts text from a PDF file using the PyMuPDF (fitz) library.

**Parameters:**

* file: The file object (in binary format) for the PDF.

**Returns:**

* A string containing all the text extracted from the PDF.

**Details:**

* The function opens the PDF file and iterates through each page.
* Text is extracted from each page using the get\_text() method and concatenated into a single string, with a newline character added between pages.

**extract\_text\_from\_docx(file)**

**Description:**  
Extracts text from a DOCX file using the python-docx library.

**Parameters:**

* file: The file object for the DOCX document.

**Returns:**

* A string containing the text extracted from the DOCX file.

**Details:**

* The function reads the DOCX file using python-docx and extracts the text from each paragraph.
* The text of all paragraphs is concatenated into a single string with a newline character separating each paragraph.

**Graph Utils:**

This script generates a career path graph using the PyVis library, which maps job roles, courses, and projects. The graph helps visualize the progression of a career path, including relevant courses and projects at each job stage. The script also handles API integrations with Adzuna, GitHub, and Coursera.

**Imports**

* requests: Used to make HTTP requests, particularly for fetching data from APIs like Coursera.
* os: Provides functions to interact with the operating system (though it's not used in the provided code).
* openai: Used to interact with OpenAI's API (though it's not used in the provided code).
* pyvis.network: Used to create and manipulate interactive network graphs.

**Constants and Credentials**

* **OpenAI API Key**: openai.api\_key for authenticating requests to OpenAI's services (not used in this script).
* **Adzuna API**: Contains ADZUNA\_APP\_ID and ADZUNA\_APP\_KEY for integrating with Adzuna's job search API (not used in this script).
* **GitHub Token**: GITHUB\_TOKEN for GitHub API access (not used in this script).
* **Coursera Credentials**: Includes coursera\_auth\_url, coursera\_payload, and coursera\_token for fetching Coursera's access token using OAuth2. The token is used to authenticate requests to Coursera's API to access courses and resources.

**Function: generate\_career\_path\_graph(job\_path, course\_path, project\_path)**

**Description:**  
This function generates a career path graph that visually represents the progression of a career, including jobs, courses, and projects.

**Parameters:**

* job\_path: A list of job titles representing the career path.
* course\_path: A list of courses associated with each job in the career path.
* project\_path: A list of projects linked to each job in the career path.

**Returns:**

* A Network object representing the career path graph.

**Steps Involved:**

1. **Create Network Object**:  
   A Network object from the PyVis library is created to visualize the graph. The network is directed, with a height of 800px and width of 100%.
2. **Add Nodes for Jobs, Courses, and Projects**:  
   For each stage in the career path (from the provided job\_path, course\_path, and project\_path), three nodes are added:
   * **Job Node**: Represents a job in the career path.
   * **Course Node**: Represents a course related to the job.
   * **Project Node**: Represents a project associated with the job.

These nodes are colored as follows:

* + Job nodes: lightcoral
  + Course nodes: lightblue
  + Project nodes: lightgreen

1. **Add Edges Connecting Jobs, Courses, and Projects**:  
   Edges are added between the job nodes and the respective course and project nodes, with:
   * Blue edges between jobs and courses.
   * Green edges between jobs and projects.
2. **Add Edges Connecting Jobs in Order**:  
   Edges are also added between the job nodes to represent career progression, using gray edges with a width of 2.
3. **Configure Physics of the Network**:  
   The force\_atlas\_2based method is used to apply a dynamic layout with specified gravity, central gravity, spring length, and damping values to make the network interactive and visually appealing.

**Usage Example**

python

Copy code

# Example paths for jobs, courses, and projects

job\_path = ["Junior Developer", "Software Engineer", "Senior Software Engineer", "Tech Lead"]

course\_path = ["Python Basics", "Advanced Java", "System Design", "Leadership in Tech"]

project\_path = ["Web App Development", "API Integration", "Cloud Migration", "Team Management"]

# Generate the career path graph

career\_graph = generate\_career\_path\_graph(job\_path, course\_path, project\_path)

# Show the generated graph

career\_graph.show("career\_path\_graph.html")

In the example, a list of jobs, courses, and projects are provided. The generate\_career\_path\_graph function is called with these lists, and the result is displayed using the show() method, which renders the graph to an HTML file.

**Resume Utils:**

1. **OpenAI Integration**: You're using OpenAI’s GPT-4 model to process the resume text and extract various details:
   * **User Level**: Based on the resume, you classify the user's career level (e.g., undergraduate student, entry-level professional, etc.).
   * **Location and Country Code**: You extract the location (city) and determine the corresponding country code.
   * **Personal Information**: Extracting details like name, email, phone, date of birth, etc., in JSON format.
   * **Skills Extraction**: Extracting both technical and soft skills from the resume and categorizing them with proficiency levels.
2. **External API Integration**:
   * **Adzuna API**: Used for job search or data related to job listings (credentials are provided).
   * **GitHub API**: Used for GitHub integrations (access token provided).
   * **Coursera API**: Fetches Coursera credentials (client ID and secret are used for OAuth2 token).
3. **Logging**: You’ve implemented logging for tracking errors and events in the app.log file.
4. **Streamlit Interface**: You're using Streamlit to display the extracted skills and other resume details in the web interface.

**Suggestions for Improvement:**

1. **Sensitive Information**: Avoid storing API keys, tokens, and credentials in the code directly (as seen with OpenAI, Adzuna, GitHub, and Coursera credentials). Use environment variables or encrypted secrets management systems to handle them securely.
2. **Error Handling**: In functions like determine\_user\_level and extract\_skills, it would be good to provide a more robust mechanism for retries or error recovery, especially in case of network or API failures.
3. **Skill Categorization**: It appears you are categorizing skills into **soft** and **technical** categories. Make sure that your categorization logic is robust and handles ambiguous cases well. You might want to create a predefined set of keywords or categories to assist the model.
4. **Streamlit UI**: The code uses Streamlit’s text\_area to display skills. Ensure the UI/UX is user-friendly by adding headings, tooltips, or more structure to the input and output areas.
5. **Token Management**: You may want to handle API tokens more efficiently. For instance, the openai.api\_key is hardcoded, which is not ideal for deployment or sharing. Use environment variables like os.environ.get("OPENAI\_API\_KEY") to keep things secure.

**Next Steps:**

* **Testing**: Test each function with sample resume data to ensure that the extraction and categorization work as expected.
* **Security**: Refactor the code to use environment variables or a more secure method to handle API keys and tokens.
* **Streamlit UI Enhancements**: Improve the interface to display extracted data more clearly and interactively.

**Configure.py:**

This script integrates with multiple external APIs, including OpenAI, Adzuna, GitHub, and Coursera. The primary functionality is to securely load API credentials from environment variables and use them to interact with these APIs. It performs token generation for Coursera's API and sets up the environment for API usage. The credentials for all APIs are stored securely in environment variables.

**Libraries Used**

* **os**: The os library is used to access environment variables, allowing the script to securely retrieve API keys.
* **requests**: This library is used for sending HTTP requests to external APIs.
* **openai**: The OpenAI API is used to interact with GPT models or other OpenAI services.

**Code Explanation**

1. **Import Libraries**:
   * os: This library is used to access environment variables for API keys and other configurations.
   * requests: This is used to send HTTP requests for authentication and API interactions.
   * openai: This library allows integration with OpenAI's API services for tasks like text generation, embeddings, and more.
2. **Loading API Credentials**: The credentials for all APIs (OpenAI, Adzuna, GitHub, and Coursera) are loaded from environment variables using os.getenv(). This ensures that sensitive data like API keys are kept out of the source code.
   * openai.api\_key: The OpenAI API key is loaded from the environment variable OPENAI\_API\_KEY.
   * ADZUNA\_APP\_ID and ADZUNA\_APP\_KEY: Credentials for interacting with the Adzuna API are loaded from ADZUNA\_APP\_ID and ADZUNA\_APP\_KEY.
   * GITHUB\_TOKEN: The GitHub personal access token is loaded from GITHUB\_TOKEN.
   * coursera\_client\_id and coursera\_client\_secret: These values are used to authenticate with Coursera's API.
3. **Coursera Token Generation**: To interact with Coursera’s API, an OAuth2 token is needed. This is obtained by sending a POST request to Coursera's OAuth2 token generation endpoint. The request includes the client\_id, client\_secret, and grant\_type.
   * **Authentication Request**: A POST request is sent to Coursera’s token generation URL (https://api.coursera.com/oauth2/client\_credentials/token) with the client\_id, client\_secret, and the grant\_type set to client\_credentials.
   * **Token Extraction**: If the response status code is 200 (success), the token is extracted from the response JSON and stored in the variable coursera\_token. If the request fails, the token is set to an empty string.
4. **Example of Direct API Credential Setup (Unsafe Practice)**: In the second section of the code, hardcoded API keys are used directly in the script. This practice is **not recommended** because it exposes sensitive information (such as OpenAI API key, Adzuna credentials, GitHub token, and Coursera credentials) in the source code. For security purposes, always load these keys from environment variables as shown in the first section.
   * **Hardcoded OpenAI API Key**: The OpenAI API key is directly assigned in the script (unsafe practice).
   * **Hardcoded Adzuna API Keys**: Credentials for Adzuna API are also directly included in the script.
   * **Hardcoded GitHub Token**: The GitHub personal access token is hardcoded in the script.
   * **Hardcoded Coursera Credentials**: Both the client\_id and client\_secret for Coursera’s API are embedded in the code.

**Security Considerations**

* **Environment Variables**: It is recommended to store sensitive information like API keys in environment variables to prevent them from being exposed in the code. This helps secure sensitive data, especially in public or shared repositories.
* **Avoid Hardcoding Keys**: Hardcoding keys and tokens in the code is a security risk. Always use secure methods to handle API credentials, such as loading them from .env files or using secure secrets management services like Azure Key Vault, or similar tools.
* **Production Environment**: Ensure that sensitive keys are securely managed in production environments using secure infrastructure and practices.

**Main:**

The **AI-Powered Career Advisor** is a web-based application that offers various features aimed at assisting users with their career development. These features include personalized career advice, CV building, skill gap analysis, career path visualization, and chatbot interactions. The application integrates multiple external services, such as OpenAI, Adzuna, GitHub, and Coursera, to provide comprehensive career guidance and recommendations.

**Key Features**

1. **Career Advisor**: Provides tailored career advice based on the user's skills, goals, and interests.
2. **Build Your CV**: A tool that helps users create and customize their CV by extracting data from existing documents and offering formatting options.
3. **My Profile**: Displays and allows the user to edit their profile, including their career history, education, and skills.
4. **Skill Gap Analysis**: Analyzes the user's current skills and compares them against the skills required for their desired career, providing recommendations to bridge any gaps.
5. **Career Path Visualization**: Visualizes potential career paths using network graphs, showing various job roles and the required skills to progress through them.
6. **Chat with Career Advisor**: A chatbot that uses OpenAI's language model to answer career-related questions and provide advice in a conversational format.
7. **Find your Ideal Job**: Matches the user's profile with relevant job opportunities based on their preferences and qualifications.

**Application Architecture**

The application is built using **Streamlit**, a Python framework for building interactive web applications. The page is structured with a sidebar that allows users to navigate between different features, and the main content area dynamically updates based on the selected feature.

* **Session State Management**: The application uses **Streamlit session state** to store and persist data across different tabs. This includes the user's profile, career path, and related information, ensuring that data is retained while navigating between features.
* **External API Integration**:
  + **OpenAI**: Used for providing AI-powered career advice and interacting with the chatbot.
  + **Adzuna**: Retrieves job listings and recommendations based on the user's profile.
  + **GitHub**: Analyzes the user's projects and repositories to suggest relevant career development paths.
  + **Coursera**: Suggests online courses and certifications to help users acquire the skills necessary for career advancement.

**Security Considerations**

* **API Credentials**: All API keys and sensitive information are securely stored in environment variables. This prevents the exposure of credentials in the source code, which could lead to unauthorized access.
* **Session Management**: The application ensures proper handling of session states, preventing data leakage between users and maintaining privacy and security.

**Technology Stack**

* **Streamlit**: Used to create the interactive frontend.
* **OpenAI**: For natural language processing tasks and career advice generation.
* **Requests**: For making HTTP requests to external APIs like Adzuna, GitHub, and Coursera.
* **NetworkX** and **PyVis**: Used to visualize career paths and job role networks.
* **Python Libraries**: pdfplumber, docx, reportlab, and json are used for document processing, PDF generation, and handling user data.

**How to Use**

1. **Set Up**: Install the required Python libraries and ensure API credentials are configured in environment variables.
2. **Run the Application**: Launch the application using Streamlit, which automatically handles routing to different features based on user selection.
3. **Interacting with Features**: From the sidebar, users can select a feature to interact with, such as career advice, CV building, or job search.

**Requirements:**

**Libraries and Their Purpose:**

1. **streamlit**:
   * **Purpose**: Streamlit is used to build interactive web applications directly from Python scripts. In your project, it's used for creating the frontend interface, enabling the navigation between different features like Career Advisor, CV Building, Skill Gap Analysis, etc.
2. **openai==0.27.8**:
   * **Purpose**: This library is used to interact with OpenAI’s API, which powers the chatbot and career advisor features of your application. It allows the app to generate AI-powered responses for career advice and other tasks based on natural language processing.
3. **pdfplumber**:
   * **Purpose**: Pdfplumber helps in extracting text and tables from PDF documents. It can be used for processing CVs and other PDFs to extract and manipulate the content.
4. **python-docx**:
   * **Purpose**: This library is used for reading, writing, and modifying Word documents (.docx). It can be used for processing resumes or any documents in the .docx format.
5. **reportlab**:
   * **Purpose**: ReportLab is used for generating PDF files. In your application, it could be used to generate and customize CVs or career reports in PDF format, utilizing features like text wrapping and custom page sizes.
6. **network**:
   * **Purpose**: Likely a reference to the networkx library, which is used for creating, analyzing, and visualizing complex networks and graphs. This could be used for the Career Path Visualization feature to model job roles and career paths.
7. **pyvis**:
   * **Purpose**: PyVis is a Python library for visualizing networks in the web browser. It builds on networkx and can be used to create interactive visualizations of career paths or skill networks, helping users explore their career options.
8. **plotly**:
   * **Purpose**: Plotly is a popular library for creating interactive plots and charts. In your case, it might be used to visualize data, such as skill gaps, career progression, or project timelines, in an interactive format.
9. **pymupdf**:
   * **Purpose**: PyMuPDF (also known as Fitz) is used for reading and writing PDFs and other document types. It offers advanced document manipulation, such as extracting text and images, and might be used for document analysis or manipulation in your app.

**Note: The following will be implemented in the future**

**Fine-Tuning GPT-4 with the O\*NET Database**

* + **Identifying User Needs:** Seeking to further enhance the accuracy and relevance of job title predictions and career path recommendations by leveraging comprehensive occupational data.
  + **Understanding the O\*NET Database:**
    - **Description:** The O\*NET (Occupational Information Network) Database is a comprehensive, authoritative source of occupational information in the United States. It contains detailed data on various job roles, including required skills, competencies, knowledge areas, job tasks, and other relevant attributes.
    - **Contents:** The database includes information on over 900 occupations, providing insights into the skills required, educational prerequisites, work activities, and other factors essential for each job role.
  + **Process of Fine-Tuning GPT-4:**
    - **Data Conversion:** Converted the O\*NET Database files into a chat-friendly format suitable for fine-tuning GPT-4. This involved structuring the data into question-and-answer pairs, where each entry corresponds to specific occupational details and related queries.
    - **Formatting:** Organized the data to mimic conversational interactions, enabling GPT-4 to better understand and generate contextually accurate responses based on the occupational information.
    - **Fine-Tuning Steps:**
      * **Data Preparation:** Extracted relevant fields from the O\*NET Database, such as job titles, required skills, tasks, and knowledge areas. Structured this information into a dialogue format where prompts represent user inquiries about specific occupations, and responses provide detailed answers based on the database.
      * **Training Configuration:** Configured the fine-tuning parameters for GPT-4, ensuring that the model focuses on accurately mapping user queries to the relevant occupational data.
      * **Training Execution:** Employed OpenAI's fine-tuning API to train GPT-4 on the prepared O\*NET-based dataset. This process involved iterative training sessions to refine the model's ability to generate precise and informative responses.
      * **Validation and Testing:** Conducted extensive testing to evaluate the model's performance in accurately predicting job titles and providing relevant career recommendations based on user inputs. Adjusted training parameters and dataset structures as necessary to enhance accuracy.
  + **Integration with Existing Systems:**
    - **Embedding Model:** Utilized the fine-tuned GPT-4 model in conjunction with the previously implemented Word2Vec embedding model. The embedding model processes user-provided skills and matches them with relevant job titles, while the fine-tuned GPT-4 enhances the recommendation accuracy by leveraging detailed occupational data from the O\*NET Database.
    - **Feedback Mechanism:** Fed the output from the fine-tuned GPT-4 model into the existing feedback mechanism. This integration allows the system to continuously learn from user interactions and improve the relevance of job title predictions and career path recommendations.
  + **Benefits Realized:**
    - **Enhanced Accuracy:** The fine-tuned GPT-4 model demonstrated improved accuracy in predicting job titles and generating relevant career paths by leveraging the detailed occupational data from the O\*NET Database.
    - **Informed Recommendations:** Users received more informed and contextually appropriate career recommendations, aligning closely with their skills and desired job roles.
    - **Personalization:** The integration of the O\*NET Database enabled the Career Advisor to offer highly personalized guidance, taking into account the specific requirements and nuances of various occupations.
  + **Outcome:** Successfully implemented the fine-tuned GPT-4 model with the O\*NET Database, resulting in significantly improved job title predictions and more accurate career path generation, thereby addressing the shortcomings observed with the RAG system.
  + **(future work) Fine-Tuning GPT-4:** Enhanced GPT-4's capabilities by fine-tuning it with the O\*NET Database. This involved:
    - **Data Preparation:** Extracted relevant occupational data and structured it into conversational formats.
    - **Training Process:** Used OpenAI's fine-tuning API to train GPT-4 on the prepared dataset, improving its ability to generate accurate and contextually relevant responses.
    - **Validation:** Tested the fine-tuned model to ensure it provided precise job title predictions and career path recommendations.