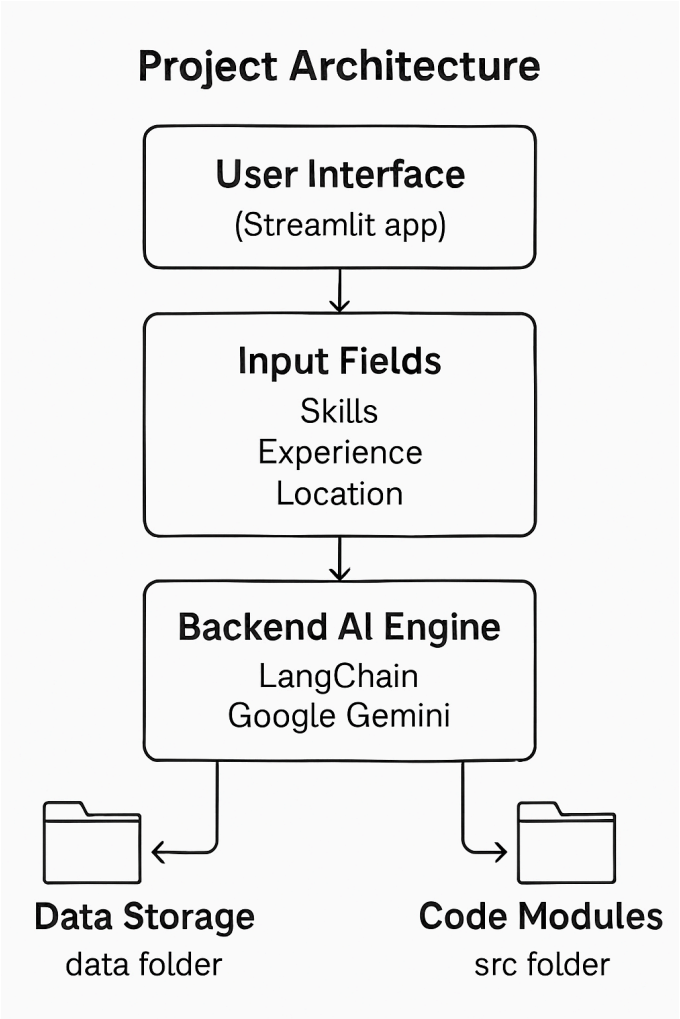
# Job Recommendation GenAI — Project Report

## Problem Explanation

In today’s competitive job market, finding relevant job opportunities tailored to individual skills, experience, and location is challenging. Traditional job portals offer generic listings that may not match a candidate’s unique profile effectively. This project addresses this problem by developing an AI-powered job recommendation app that leverages user inputs (skills, experience level, location) to provide personalized job suggestions, thereby increasing the chances of a successful match.



## Use of RAG with LangChain + Gemini

The project incorporates advanced AI retrieval and generation techniques:

- RAG (Retrieval-Augmented Generation): Combines retrieval of relevant information from external datasets with generative AI to produce more accurate and context-aware job recommendations.
- LangChain: Facilitates building AI applications by providing flexible tools for chaining together multiple components such as retrievers, language models, and APIs.
- Gemini (Google's AI): Utilized as the underlying generative AI model integrated via LangChain to understand user queries, analyze job data, and generate customized recommendations.

Together, these technologies enable the app to effectively search across job listings and intelligently generate personalized suggestions in real-time.

## Folder/Code Walkthrough

- app.py: The main Streamlit application file containing the user interface, input handling, and triggering backend logic.
- src/: Contains supporting Python modules and helper scripts that implement the core functionality such as data retrieval, processing, and integration with LangChain and Gemini.
- data/: Holds datasets or configuration files relevant to job listings or application parameters.
- .gitignore: Specifies files and folders to exclude from version control (e.g., virtual environments, cache).
- README.md: Documentation explaining the project overview, features, setup instructions, and usage.

The code structure follows modular programming principles for easy maintenance and extension. The app.py file orchestrates the interaction between front-end inputs and backend AI processing via the modules in src/.

## Challenges Faced and How You Solved Them

- Module Installation and Dependency Issues:
  Encountered errors like "No module named 'langchain_google_genai'". Resolved by identifying and installing the missing libraries and managing dependencies with careful pip installs.
- Git and GitHub Authentication Problems:
  Faced push errors due to branch mismatches and authentication, especially with two-factor authentication. Solved by configuring Git user identity, using Personal Access Tokens for secure authentication, and learning Git commands for pull/rebase/push operations.
- Empty Folder Handling in Git:
  Git does not track empty folders, affecting the project's folder structure representation on GitHub. Resolved by adding placeholder `.gitkeep` files to ensure folder visibility.
- Setting Up Streamlit and Running the App:
  Guided through Command Prompt navigation, virtual environment activation, package installation, and Streamlit server startup to smoothly run the app locally.

## Summary of What You Learned

- Gained practical experience in building AI-powered applications using modern tools and frameworks like Streamlit, LangChain, and Google's Gemini.
- Understood the importance of AI retrieval-augmented generation (RAG) in enhancing personalized user experiences.
- Developed skills in managing Python environments, handling dependencies, and troubleshooting software errors.
- Improved command-line and Git/GitHub proficiency, especially around repository initialization, branch management, authentication, and collaboration workflows.
- Learned best practices around project organization, documentation including README creation, and preparing professional deliverables for software projects.