CS673 Software Engineering
Team 5 - CVCoach
Software Design Document

Your project Logo here if any

Team Member	Role(s)	Signature	Date
Linchen Xu	Team Leader, Design and Implementation Leader	*Linchen Xu*	09/14/2024
Lin Ma	Configuration Leader	*Lin Ma*	09/14/2024
Zihan Zhou	Security Leader	*Zihan Zhou*	09/14/2024
Zhen Cao	QA Leader	*Zhen Cao*	09/14/2024
Haochen Sun	Requirement Leader	*Haochen Sun*	09/19/2024

Revision history

Version	**Author**	**Date**	**Change**
1.0.0	**Linchen Xu adamma@bu.edu caoz229@bu.edu zhzhjycs@bu.edu**	**10/17/2024**	**Create initial document**

• Introduction

This document will include our software architecture both frontend and backend, class diagram, database design, security design and our key algorithms for resume evaluation. Our software focuses on using AI to automate and improve the process of resume evaluation and interview preparation. By incorporating Retrieval-Augmented Generation (RAG), we ensure that our application can provide more accurate, context-relevant reviews.

• Software Architecture

Our software system follows a client-server architecture, where the frontend and backend are decoupled into separate repositories. The frontend handles user interaction and data presentation, while the backend provides data processing and serves as the interface with the database and external services.

1. High-Level Architecture

The system is divided into two main components:

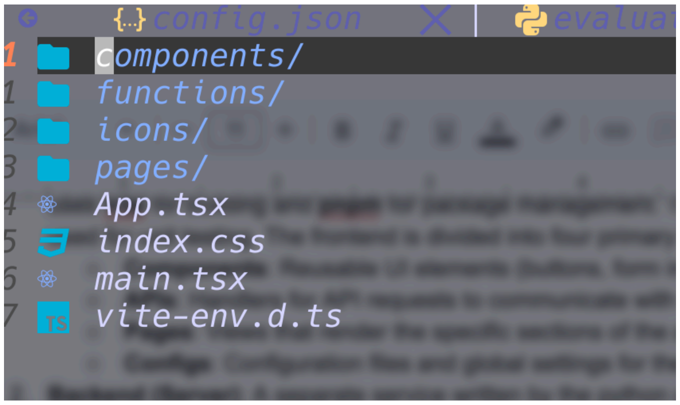
1. **Frontend (Client)**: Developed using React and TypeScript, this part of the system is responsible for user interaction, rendering the user interface, and managing the state. It uses **Vite** for building and **pnpm** for package management. The **Vitest** framework is used for unit testing. The frontend is divided into four primary folders:
    - **Components**: Reusable UI elements (buttons, form inputs, etc.).

  - **APIs**: Handlers for API requests to communicate with the backend.
  - **Pages**: Views that render the specific sections of the application.
  - **Configs**: Configuration files and global settings for the frontend.
2. **Backend (Server)**: A separate service written by the python and using flask framework to hold a server.

2. Frontend

Frontend Component Decomposition

The **frontend** system is decomposed into several key components, folders, and sections:



```
components/
functions/
icons/
pages/
App.tsx
index.css
main.tsx
vite-env.d.ts
```

a. Components Folder

- **Purpose**: Holds reusable UI components to create a consistent look and feel across the application.
- **Examples of Components**:
  - `TextArea.tsx:`Text input for user queries.
- **Dependencies**:
  - MUI Component Lib

b. Functions Folder

- **Purpose**: Manages all API requests to the backend services and utilizations.
- **Examples of API Handlers**:
  - `api.ts`: Sends and retrieves chat messages from the server.
  - `api.test.ts`: Handles resume upload and resume analysis requests.
- **Dependencies**:
  - `HTTP libraries`: Axios for making requests to the backend.
  - `Vitest`: An UT lib in frontend.
  - `Other libs`: UUID to generate a random uuid.

c. Pages Folder

- **Purpose**: Contains views that represent individual screens or pages in the application.
- **Examples of Pages**:
    - `ChatBox.tsx`: Interacts with users and accepts user's input.
    - `Content.tsx`: Displays the Q&A history.
    - `Header.tsx`: Displays title and desc.
- **Dependencies**:
    - MUI Component Lib
    - `APIs`: Pages call APIs to display dynamic data (e.g., message history or resume analysis results).

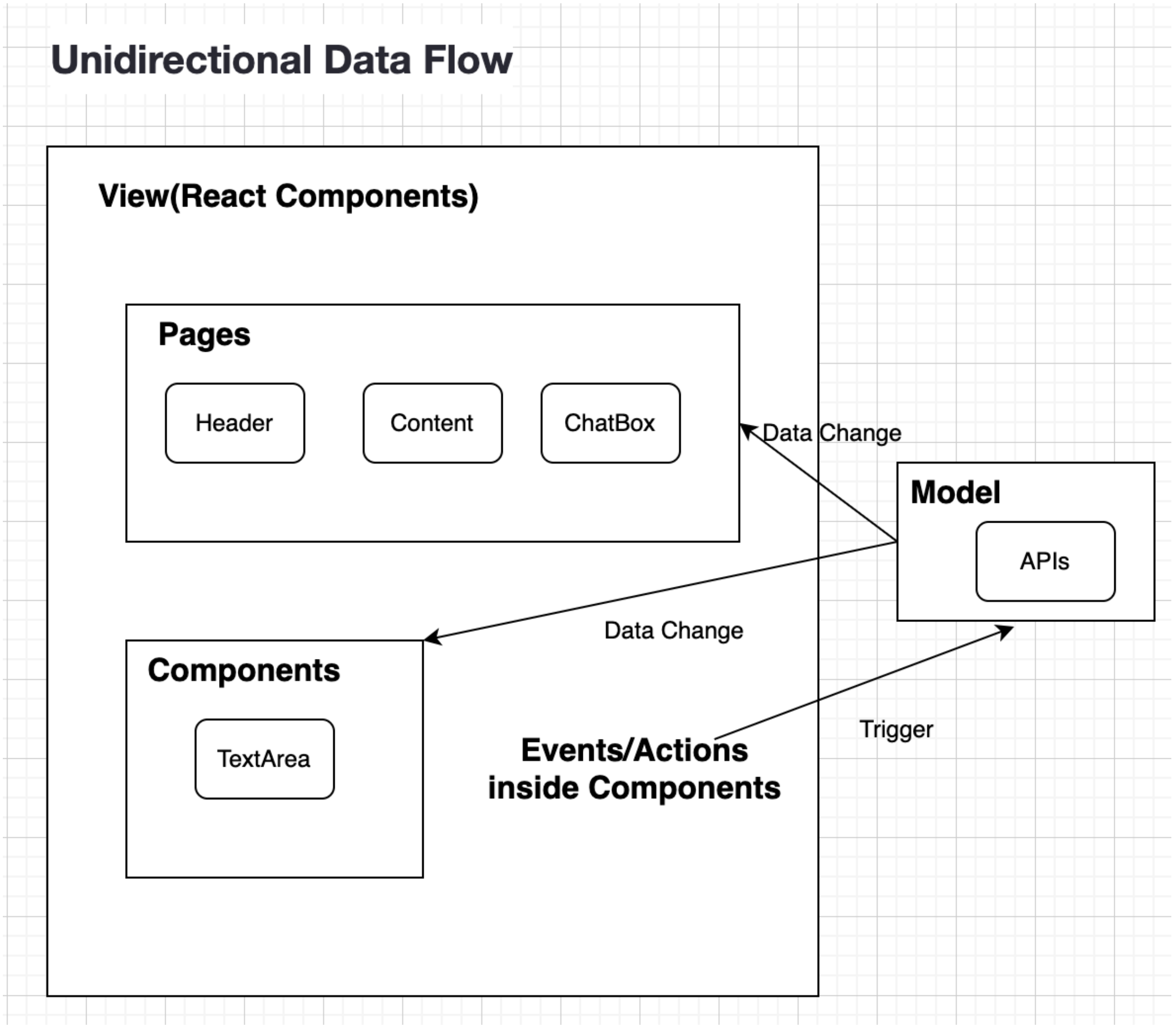
d. Assets Folder

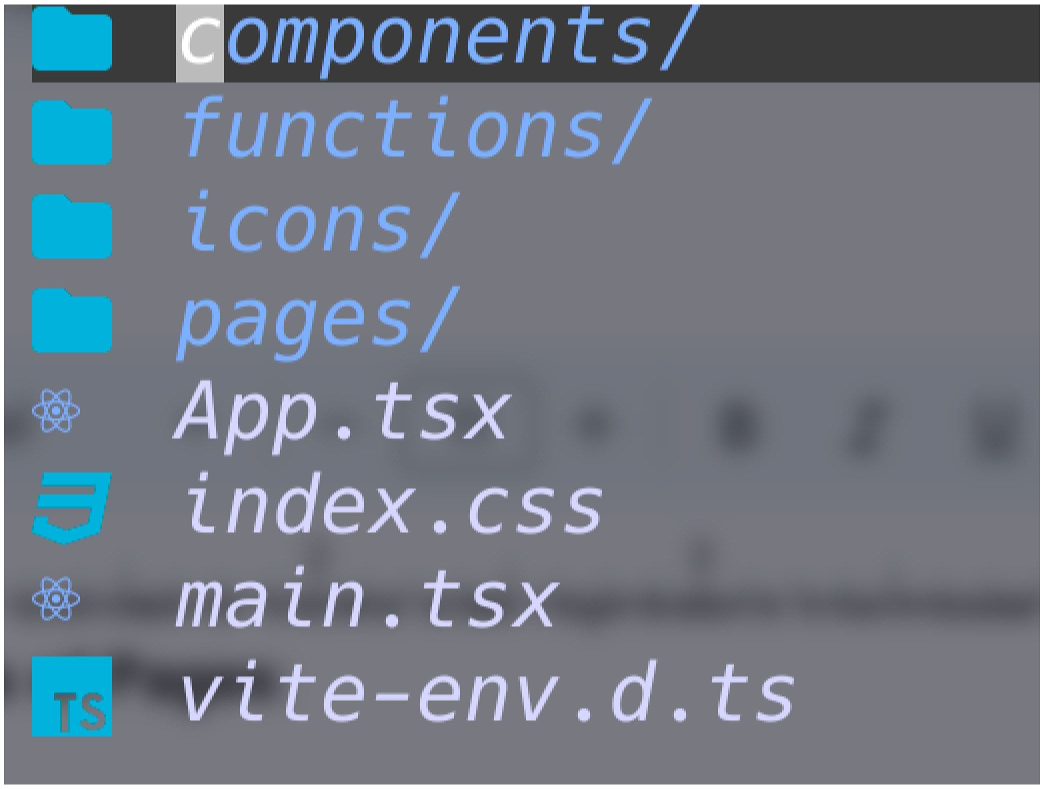
- **Purpose**: Stores static resources like icons and images.

Frontend Frameworks and Tools Used

- **React**: Main frontend framework for building the user interface.
- **TypeScript**: Provides static typing to enhance code quality and development efficiency.
- **Vite**: Modern frontend build tool that improves development speed and production performance.
- **pnpm**: Package management tool used to efficiently install and manage dependencies.
- **Vitest**: Unit testing framework used to ensure that components, API calls, and logic are working as expected.

Frontend Diagram



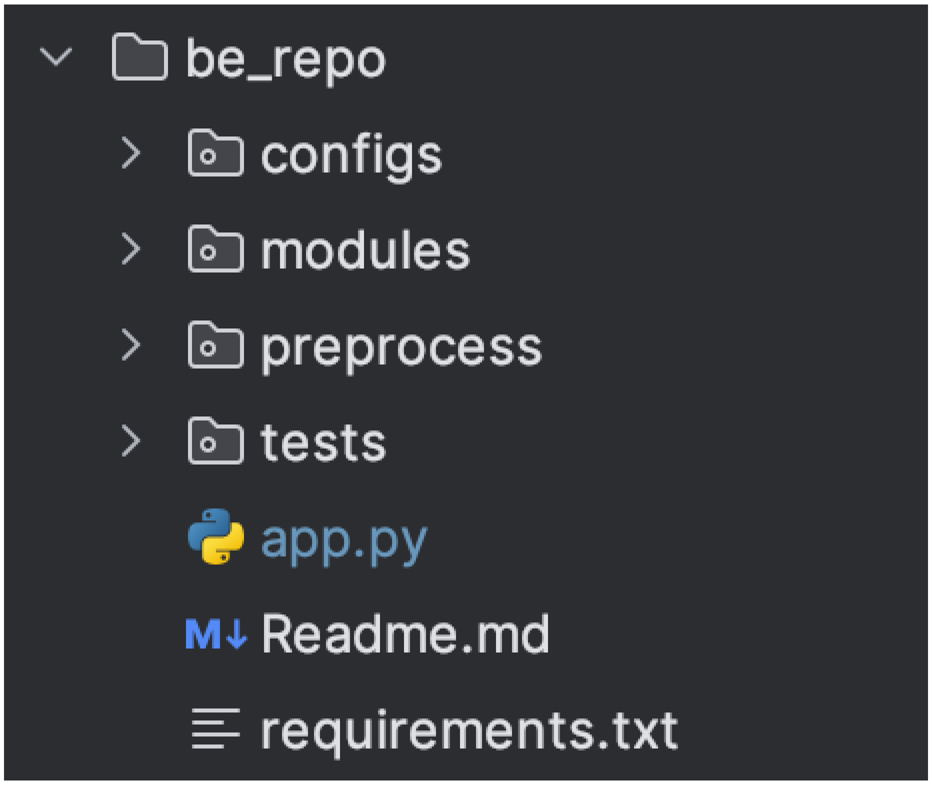


```
components/
functions/
icons/
pages/
App.tsx
index.css
main.tsx
vite-env.d.ts
```

3. **Backend**

Backend Component Decomposition

The **backend** system is decomposed into several key components, folders, and sections:



```
be_repo
  configs
  modules
  preprocess
  tests
  app.py
  Readme.md
  requirements.txt
```

a. Modules Folder

- **Purpose**: Contain the modules responsible for handling various components of the application logic.
- **Examples of Components**:
  - `upload.py`: Upload user's resume to our MongoDB.
  - `evaluator.py`: Retrieve the corresponding resume from MongoDB using the user's ID and return evaluation generated by the ChatGPT.

b. Preprocess Folder

- **Purpose**: Responsible for preprocessing the dataset to prepare it for further analysis, evaluation, and matching.
- **Examples of Preprocessing**:
  - `gpt_evaluation.py`: Score each resume based on predefined criteria.
  - `extract_job_keywords`: Extract keywords to be used later for matching resumes with job postings.

c. Tests Folder

- **Purpose**: Contain all the unit tests using Pytest.
- **Examples of Pages**:

- `test_upload.py`: unit test for upload.py.

d. Configs Folder

- **Purpose**: Contain configuration files and settings such as our API keys.

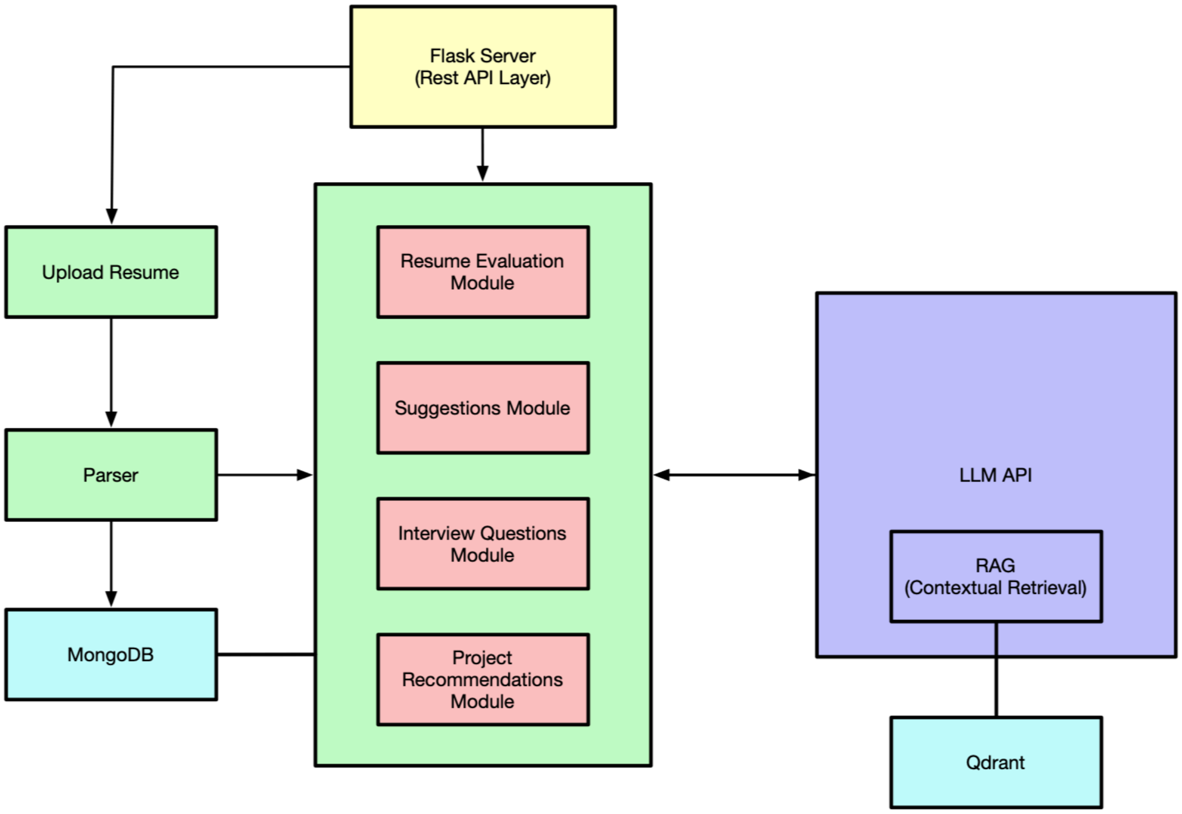
e. app.py

- **Purpose**: Serve as the entry point for application and contains various API endpoints for handling requests and providing responses.

Backend Frameworks and Tools Used

- **Flask**: Python web framework used to build our backend API of the application. It handles routing, request processing and interaction between the client and server.
- **Pytest**: A testing framework used to run unit tests for our python application.

Backend Diagram



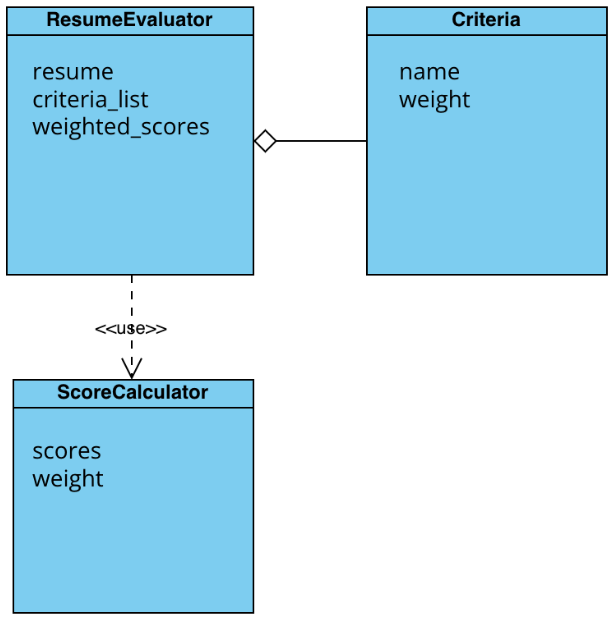
- Basic functionality
  - Resume evaluation

- Suggestions for improvement
- Interview question generation
- Project recommendations

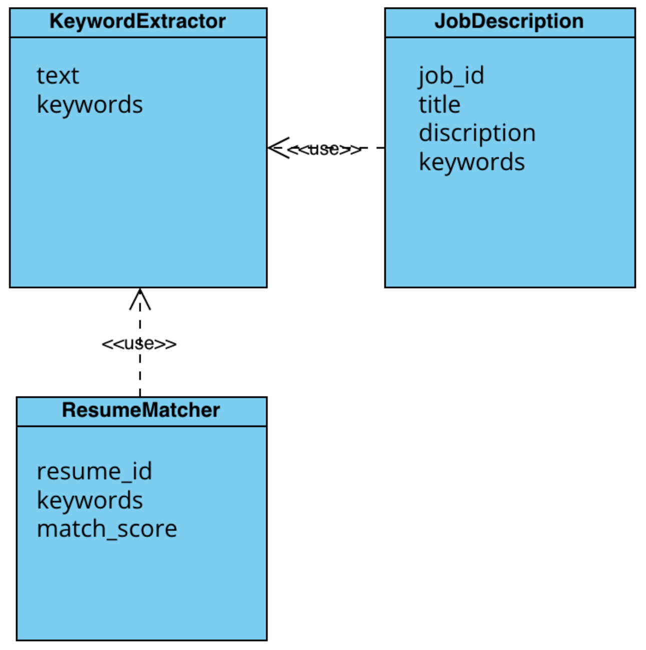
Class Diagram

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

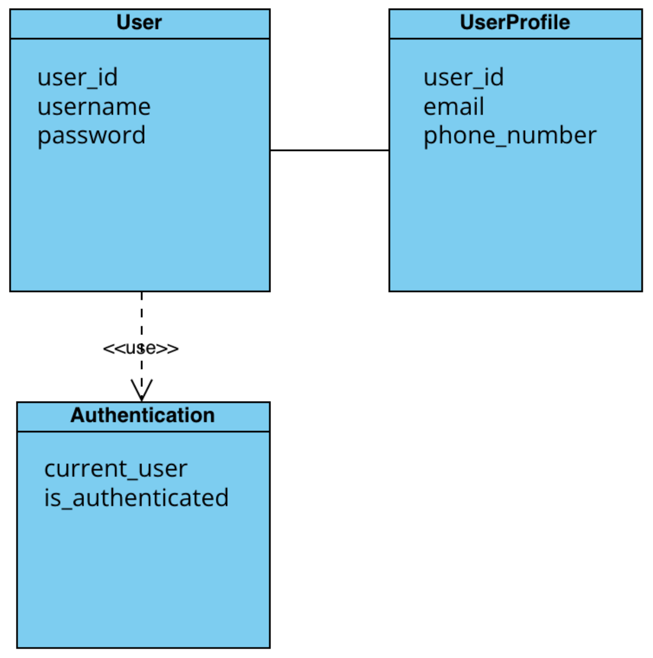
1. Resume Evaluation Component
   - Classes:
     - ResumeEvaluator: load resume and compute the final score.
     - Criteria: criterias used to evaluate a resume.
     - ScoreCalculator: aggregate scores and calculate a weighted score.
   - Relationships:



2. Job Keyword Extraction Component
   - Classes:
     - KeywordExtractor: extract keywords from job descriptions
     - JobDescription: details of a job posting
     - ResumeMatcher: match resumes to job descriptions based on the keywords
   - Relationships:

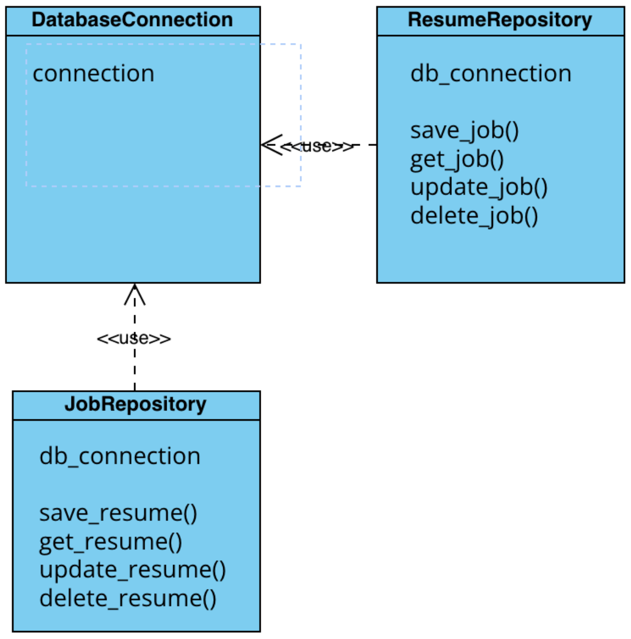


3. User Management Component
   - Classes:
     - User: resume’s owner, system user
     - UserProfile: user information
     - Authentication: user login information
   - Relationships:



4. Database Interaction Component

- Classes:
    - DatabaseConnection: manage database connections
    - ResumeRepository: CRUD for resumes
    - JobRepository: CRUD for job descriptions
- Relationships:



- UI Design

    We don't have a particular UI design, but we are following some existed and popular guidelines as follows:

    Icon: https://iconoir.com/
    Basic Components: MUI
    Style Guideline: tailwindcss basic styles

- Database Design (if applicable)

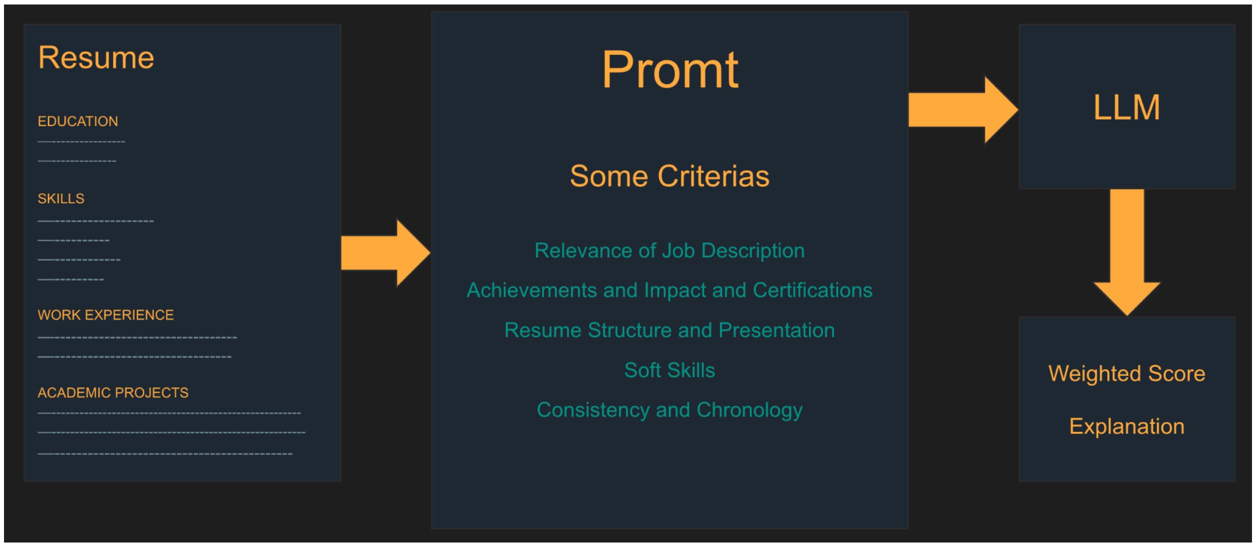
    MongoDB is used for storing resumes uploaded by users. We also plan to put our OpenAI Key and other sensitive data into our MongoDB database.

    Qdrant online vector database. Using a vector database for easy OpenAI embedding and retrieval, With a Embedding size of 1536 for usual data embedding.

- Security Design
    1. Frontend

    a. Avoid the XSS attack within the input.
    b. Check out dependencies' versions and make sure they're on a safe version.
2. Backend
    a. Integration of Cloudflare to keep our DNS safe from like DDoS attacks
    b. Dependencies checking (in CI)
    c. Input filter, avoid the XSS attack.
    d. Isolate the privacy data from the request and even tracking.
    e. Isolate the sensitive user data from RAG retrieval.

Business Logic and/or Key Algorithms


Resume
EDUCATION
SKILLS
WORK EXPERIENCE
ACADEMIC PROJECTS
Promt
Some Criterias
Relevance of Job Description
Achievements and Impact and Certifications
Resume Structure and Presentation
Soft Skills
Consistency and Chronology
LLM
Weighted Score
Explanation

Resume score calculation:

We have come up with some **criteria** that are relevant to resume evaluation. Based on their priority, we use this weighted score below to generate a prompt along with the resume. Then the **prompt** will be delivered to the LLM for resume analysis to get a weighted score and explanation for each score.

Score = relevance of job description * xx% + achievements * xx% + education and certification * xx% + resume structure and presentation * xx% + soft skills * xx% + consistency and chronology * xx%. This score indicates how well the candidate's resume fits the given job description.

For the criterion' relevance of job description', we use keywords matching to enhance the evaluation of LLM.

Resume and Job description matching:

We will use keywords extracted from job descriptions to match against keywords in the user's resume. This will help improve the relevance and alignment between the user's skills and job requirements.

We plan to use **TF-IDF** and **cosine similarity** to do this matching.

- ## Design Patterns

  MVC Pattern

  - **Model**: Manage the data and our business logic. We have various modules for fetching resumes from MongoDB, calculating resume scores and so on.
  - **View**: The frontend interface that the user interacts with.
  - **Controller**: In app.py, we handle the communication between the model and the view, processing API calls.

- ## Any Additional Topics you would like to include.

  - Frontend:
    - Linter: ESLint recommended rules
    - Package manager: Pnpm
    - Building tools: Vite
  - API documentation:
    - Finish by iteration 1:
      - Resume Upload
      - Resume Analysis
    - Finish by iteration 2:
      - Login
      - Ask a question
      - Interview question recommendation

- ## References

- ## Glossary