**AI-Powered Resume Builder and Job Matcher**

**Project Description**

This web application helps users create professional resumes and matches them with suitable job opportunities based on their skills, experience, and preferences. The platform uses Machine Learning to analyze job descriptions and user profiles, ensuring personalized and relevant job recommendations.

**Core Features**

1. **Resume Builder**
   * **What it does**: Allows users to create customized resumes using pre-designed templates.
   * **Implementation Details**:
     + Users input their personal details, skills, education, and experience.
     + Templates styled with HTML and CSS for a polished look.
     + PDF export functionality using libraries like “pdfkit or xhtml2pdf.
     + Option to include AI-suggested keywords to make resumes ATS-friendly (Applicant Tracking System).
2. **Job Matching System**
   * **What it does**: Recommends jobs based on user skills, experience, and preferences.
   * **Implementation Details**:
     + Use ML models like NLP-based algorithms (e.g., TF-IDF or BERT) to analyze job descriptions and user profiles.
     + SQL database to store job listings and user profiles.
     + A similarity scoring system for matching user skills with job requirements.
     + Filters for location, job type, and salary range.
3. **Skill Gap Analysis**
   * **What it does**: Highlights skills missing in the user’s profile compared to job requirements and suggests courses to fill these gaps.
   * **Implementation Details**:
     + NLP model to identify key skills from job descriptions.
     + Integration with course platforms like Udemy or Coursera via APIs to recommend relevant courses.
4. **Dashboard for Job Applications**
   * **What it does**: Allows users to track job applications and view the status.
   * **Implementation Details**:
     + SQL tables to manage job applications and their statuses.
     + JS-based dashboard for visualizing progress (e.g., applied, under review, rejected).
5. **Admin Panel for Employers**
   * **What it does**: Enables employers to post job listings and view potential candidates.
   * **Implementation Details**:
     + Employers can upload job descriptions.
     + A Flask backend to manage job postings.
     + SQL database to store employer data and job postings.

**Tech Stack**

1. **Frontend**:
   * **HTML/CSS**: For designing user interfaces.
   * **JavaScript**: For dynamic features like real-time suggestions and progress tracking.
2. **Backend**:
   * **Flask**: To handle user requests and server-side processing.
   * **REST APIs**: For communication between frontend and backend.
3. **Database**:
   * **SQL**: To store user profiles, resumes, job listings, and application data.
4. **Machine Learning**:
   * **NLP Models**:
     + For extracting and comparing skills, job descriptions, and profiles.
     + Libraries like spaCy or transformers (BERT).
   * **Scikit-learn**: For skill similarity scoring.
5. **PDF Export**:
   * Libraries like xhtml2pdf or reportlab to generate downloadable resumes.

**Workflow**

1. **User Registration/Login**
   * Users sign up or log in using credentials.
   * Session management using Flask.
2. **Resume Creation**
   * Users fill out a form with their details, which is stored in a SQL database.
   * Select a template, and the system formats the data.
   * Option to download the resume as a PDF.
3. **Job Matching**
   * The system uses ML to recommend jobs based on user skills and preferences.
   * Users can filter results or search for specific keywords.
4. **Application Tracking**
   * Users can apply for jobs through the platform.
   * The status of each application is updated dynamically.
5. **Admin (Employer) Role**
   * Employers post job listings and review matched candidates.

**Additional Features to Enhance the App**

1. **Interactive UI**:
   * Drag-and-drop resume sections.
   * Real-time keyword suggestions for resumes.
2. **Analytics Dashboard**:
   * Visual insights into job market trends (e.g., in-demand skills, popular roles).
3. **Email Notifications**:
   * Notify users about job matches or application updates.
4. **AI Cover Letter Generator**:
   * Generate tailored cover letters for job applications.
5. **Integration with LinkedIn**:
   * Import LinkedIn profiles to prefill user data.

This project not only showcases your skills in full-stack development and machine learning but also addresses a real-world problem, making it an excellent addition to your portfolio! Let me know if you need help starting or designing specific components.

**Project Folder Structure :**

/resume-job-matcher

│

├── /static

│ ├── /css

│ │ └── styles.css # Main CSS file for the app

│ ├── /js

│ │ └── app.js # JavaScript for dynamic functionality

│ └── /images # Store static images and icons

│

├── /templates

│ ├── base.html # Base layout for all pages

│ ├── job\_match.html # Displays job matches and allows applying to them

│ ├── home.html # Homepage

│ ├── register.html # User registration form

│ ├── login.html # User login page

│ ├── dashboard.html # User dashboard with resume creation

│ ├── job\_match.html # Job recommendations page

│ ├── admin.html # Admin panel for employers

│ └── error.html # Error page

│

├── /models

│ └── ml\_model.pkl # Trained machine learning model for job matching

│

├── /data

│ └── job\_listings.db # SQL database for job postings

│

├── app.py # Main Flask application file

├── database.py # Database models and connection

├── requirements.txt # Python dependencies

├── README.md # Documentation

└── .gitignore # Git ignore file

**Workflow by Feature**

**1. User Registration/Login**

**Files Involved**:

* app.py
* register.html
* login.html
* SQL database (users table)

**Workflow**:

1. User navigates to the registration/login page.
2. Form data (username, email, password) is submitted to the backend (/register or /login routes in Flask).
3. Passwords are hashed using a library like bcrypt before storing in the database.
4. Flask sessions manage user authentication status.

**2. Resume Creation**

**Files Involved**:

* app.py
* dashboard.html
* styles.css
* PDF libraries (xhtml2pdf)

**Workflow**:

1. User fills in details (e.g., name, skills, experience) in the dashboard form.
2. Data is stored in a SQL table (resumes).
3. User selects a resume template, and data is dynamically rendered into HTML.
4. The Flask backend generates a downloadable PDF using libraries like xhtml2pdf.

**3. Job Matching System**

**Files Involved**:

* app.py
* ml\_model.pkl
* job\_match.html
* SQL database (jobs and users tables)

**Workflow**:

1. User data (skills, experience) is fetched from the database.
2. Job postings from the SQL database are analyzed using the pre-trained ML model (ml\_model.pkl).
3. The model computes similarity scores between user profiles and job descriptions.
4. Matched jobs are displayed on the job\_match.html page, allowing users to filter by location, salary, etc.

**4. Skill Gap Analysis**

**Files Involved**:

* app.py
* ML model (ml\_model.pkl)
* job\_match.html
* External API integration (e.g., Coursera)

**Workflow**:

1. The ML model compares user skills with required skills in job descriptions.
2. Missing skills are identified and displayed to the user.
3. Flask fetches relevant courses using an external API and displays them.

**5. Dashboard for Job Applications**

**Files Involved**:

* app.py
* SQL database (applications table)
* dashboard.html
* app.js

**Workflow**:

1. Users can view jobs they applied to on the dashboard.
2. Flask queries the applications table to get the current status of each application.
3. JavaScript dynamically updates the application status (e.g., Applied, Under Review).

**6. Admin Panel for Employers**

**Files Involved**:

* app.py
* admin.html
* SQL database (jobs table)

**Workflow**:

1. Employers log in to the admin panel.
2. They can post job descriptions, which are stored in the SQL database.
3. Flask renders a list of matched candidates for each job posting using the ML model.

**Python Files Overview**

**1. app.py**

This is the main Flask application file. It:

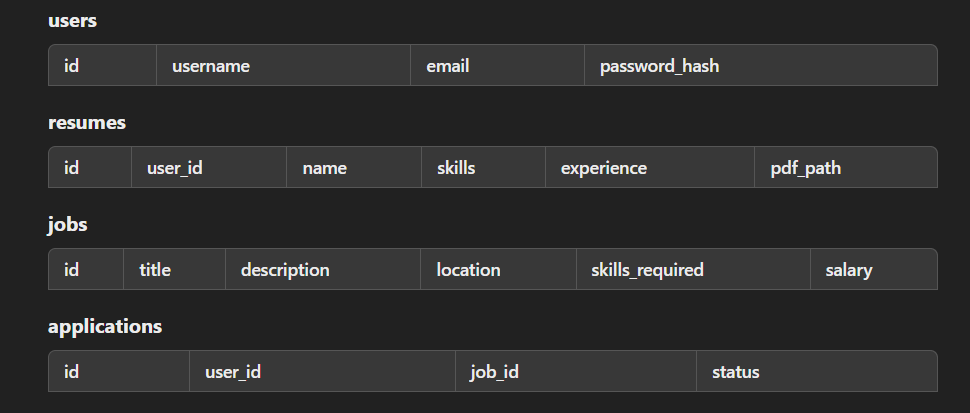
* Manages routes (e.g., /register, /login, /dashboard).
* Connects the frontend (HTML templates) with the backend logic.
* Calls the ML model for recommendations.
* Handles form submissions and data storage.

**2. database.py**

Handles database operations:

* Creates tables (users, jobs, resumes, applications).
* Queries the database for data retrieval and updates.

**SQL Database Schema**

****