BANGALORE UNIVERSITY

Jnana Bharathi Campus, Bangalore-560056

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report On

"AI RESUME BUILDER"

Submitted by

KIRAN J (P03NK23S126039)

Under the guidance of

DR. M HANUMANTHAPPA

Senior Professor and Coordinator

Department of Computer Science and Applications

Towards

MAJOR PROJECT

Prescribed by the BANGALORE UNIVERSITY

MASTER OF COMPUTER APPLICATIONS

For the academic year **2024-2025**

BANGALORE UNIVERSITY

Jnana Bharathi Campus, Bangalore-560056

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report On

"AI RESUME BUILDER"

CERTIFICATE

This is to certify that **KIRAN J (P03NK23S126039)** has satisfactorily completed the project titled **"Ai Resume Builder"** towards Mini project Lab prescribed by the Bangalore University for the 4th Semester Master of Computer Application course in academic year 2024-2025.

DR. M HANUMANTHAPPA

Guide, Senior Professor and coordinator

Examiners:

Name of Candidate: KIRAN J
Register no: P03NK23S126039
Examination Center: **Bangalore University**
Date of Examination: 21-07-2025

DECLARATION

I hereby declare that the project report, titled **"AI RESUME BUILDER"**, is prepared in partial fulfilment of the requirements for the award of the degree of Master of Computer Applications of Bangalore University for the academic year 2024-2025. Further, the matter embodied in dissertation has not been submitted previously by anybody for the award of any Degree or Diploma to any other University.

KIRAN J (P03NK23S126039)

DATE:

PLACE: BANGALORE

ACKNOWLEDGEMENT

The satisfaction that I feel at the successful completion of the lab work title is **“AI RESUME BUILDER”** would be incomplete if I do not mention the names of the people whose valuable guidance and management has made this project work a success.

It is a great pleasure to express my gratitude and respect to all those who Inspired and helped me in completing this project.

It is pleasure to thank my internal guide, HOD The Coordinator of the computer science & application, Bangalore University, **Dr. M HANUMANTHAPPA,** who has given me ideas and guidance to make the report a highly useful & knowledge base experience and for his valuable assistance and cooperation.

It is also pleasure to express my gratitude to all Teaching and Non-teaching staff members of department of Computer Science and Application for their encouragement and providing valuable requirement. With a deep sense of indebtedness, I convey my heartiest thanks to my parents who have taken effort and given me such an opportunity. Their love and blessing has given me strength to acquire knowledge and gain experience in my life. As well as my friends who have helped for this accomplished task.

ABSTARCT

The "AI Resume Builder" represents a comprehensive, intelligent ecosystem designed to revolutionize the recruitment lifecycle by bridging the operational gap between talent acquisition and career management through the advanced capabilities of Generative AI. Functioning as a sophisticated dual-sided platform, it empowers job seekers with a high-utility toolkit that transcends basic document formatting; this includes an AI-powered ingestion engine that parses complex PDF or DOCX files into structured data to eliminate repetitive data entry, a generative text enhancer that leverages Natural Language Processing to rewrite professional summaries and experience descriptions for maximum impact, and a predictive scoring system that offers real-time, quantitative feedback to align resumes with industry standards before they are rendered into polished, multi-template PDF exports. Simultaneously, the system transforms the hiring workflow for employers by acting as a next-generation Applicant Tracking System (ATS) that moves beyond archaic keyword matching, utilizing deep semantic analysis to rank candidates based on the nuanced relevance of their skills to specific job descriptions, while also generating executive summaries and custom-tailored interview questions to accelerate decision-making. This complex functionality is orchestrated within a robust technical environment built on the Django framework's Model-View-Template (MVT) architecture, employing a scalable, asynchronous infrastructure using Celery and Redis to offload resource-intensive AI computations—driven by the Google Gemini API—thereby ensuring a seamless, high-performance user experience that maintains responsiveness even during heavy data processing loads.

Table of Contents

Chapter 1: Introduction

1.1 Project Description

The "AI Resume Builder" is a sophisticated, database-driven web application designed to bridge the gap between job seekers and employers using the power of Generative Artificial Intelligence (AI). The platform is built on the Django framework and provides two distinct interfaces for its primary users: **Job Seekers** and **Employers**.

For Job Seekers: The system acts as an intelligent co-pilot for resume creation and job hunting.

- **Resume Management:** Users can create, store, and manage multiple versions of their resumes.
- **AI-Powered Parsing:** Instead of starting from scratch, a user can upload their existing resume in PDF or DOCX format. The system asynchronously extracts the text and uses the Google Gemini AI to parse it into structured JSON data, which then pre-fills the resume-building form.
- **AI Text Enhancement:** Users can leverage AI to rewrite and enhance key sections of their resume, such as the "Professional Summary" or "Experience Description," to be more impactful and professional.
- **AI Scoring & Feedback:** The system provides a real-time "Resume Score" (0-100) and actionable feedback points generated by AI, helping users understand their resume's strengths and weaknesses.
- **PDF Generation:** Users can download their resume in four distinct professional templates: Classic, Modern, Professional, and Creative.
- **AI Job Matching:** The platform intelligently compares the user's resume against active job listings and displays a "Match Score," ranking jobs by relevance.
- **AI Interview Prep:** After applying for a job and being invited to an interview, the user can access an "AI Interview Prep" tool that generates practice questions and sample STAR-method answers based on *both* their resume and the specific job description.

For Employers: The system functions as a smart and efficient Applicant Tracking System (ATS).

- **Job Posting:** Employers can create, manage, and post job listings, specifying details, requirements, and salary information.
- **AI Job Description Generator:** To assist in posting, employers can use an AI tool to generate a professional job description and requirements list based on a title and keywords.

- **Applicant Ranking:** When viewing applicants for a job, the system automatically displays all candidates ranked by their AI-calculated "Match Score," allowing recruiters to instantly focus on the most qualified individuals.
- **AI Applicant Summary:** Employers can get a concise, AI-generated summary for any applicant, highlighting their fit for the role.
- **Interview Scheduling:** The platform includes a workflow for proposing interview slots and confirming schedules with candidates.

This project is not just a simple CRUD application; it is a full-fledged ecosystem that uses an asynchronous task queue (Celery with Redis) to handle computationally expensive AI tasks, ensuring the user interface remains fast and responsive.

1.2 Benefits and Objectives

Benefits:

- **For Job Seekers:**
  - **Saves Time:** AI parsing and enhancement drastically reduce the time needed to create a high-quality resume.
  - **Improves Quality:** AI feedback and scoring help users craft resumes that are more likely to succeed.
  - **Reduces "Job Search Fatigue":** AI matching filters out irrelevant jobs and highlights the best fits.
  - **Increases Confidence:** The AI Interview Prep tool empowers candidates to walk into interviews well-prepared.
- **For Employers:**
  - **Saves Time:** AI ranking and summarization eliminate the need to manually read hundreds of irrelevant resumes.
  - **Reduces Bias:** Initial screening is based on objective skill and experience matching rather than unconscious bias.
  - **Improves Quality of Hire:** By focusing on the best-matched candidates, employers can spend more time engaging with top talent.
  - **Streamlines Workflow:** Integrates posting, tracking, and scheduling in one platform.

Objectives:

- To design and develop a secure, scalable, and robust two-sided web platform for job recruitment.
- To integrate the Google Gemini Generative AI API to perform complex NLP tasks: text parsing, summarization, enhancement, and scoring.
- To implement an asynchronous task queue using Celery and Redis to manage long-running AI processes without blocking the user interface.
- To develop a sophisticated job matching algorithm that provides a quantitative score for resume-to-job compatibility.
- To create a complete application lifecycle management system, from posting a job to scheduling an interview.
- To provide a clean, modern, and responsive user interface using Tailwind CSS and Alpine.js.
- To secure the application using Django's built-in security features and django-allauth for robust user authentication, including social login.

1.3 Overview

This document provides a complete technical overview of the "AI Resume Builder" project.

- **Chapter 2 (Literature Review)** discusses the current landscape of resume builders and job portals, identifying the gap this project aims to fill. It also details the technology stack and system requirements.
- **Chapter 3 (System Analysis)** defines the problems with existing systems, the proposed solution, and outlines the functional and non-functional requirements.
- **Chapter 4 (System Design)** presents the high-level system architecture, including Data Flow Diagrams (DFDs), Activity Diagrams, an ER Diagram, and the MVT (Model-View-Template) structure.
- **Chapter 5 (Methodology)** describes the Agile software development process used to build the project.
- **Chapter 6 (Code Snippets)** provides extensive samples of the most critical code from the project's core modules.
- **Chapter 7 (Snapshots)** illustrates the final application through a series of screenshots of its key features.

- **Chapter 8 (Conclusion)** summarizes the project's achievements and discusses potential enhancements for future work.

Chapter 2: Literature Review

2.1 Literature Survey

The field of Human Resources Technology (HR-Tech) has seen significant evolution. The initial phase consisted of simple digital job boards (e.g., Monster, CareerBuilder). The second phase, dominated by platforms like **LinkedIn** and **Indeed**, introduced social networking and powerful search algorithms, but still relied heavily on keyword matching. An applicant's success often depended on their ability to "keyword-stuff" their resume to pass initial automated screens.

Simultaneously, a separate industry of online resume builders (e.g., **Zety, Resume.io, Canva**) emerged. These tools excel at creating visually appealing templates but offer limited *content* intelligence. They may provide simple grammar checks or phrase suggestions, but they do not deeply analyze the user's content in the context of a specific job.

The "Existing System" can therefore be characterized as a fragmented ecosystem. A job seeker might use:

- **MS Word** to write their resume.
- **Grammarly** to check for errors.
- **Zety** to format it into a template.
- **LinkedIn** to search for jobs.
- **A spreadsheet** to track their applications.

This process is manual, inefficient, and disconnected. Recruiters face a parallel problem: they are inundated with hundreds of visually-diverse resumes, many of which are poorly written or irrelevant, forcing them to spend seconds on each, relying on quick pattern matching.

The introduction of Generative AI (like GPT-4 and Google's Gemini) marks the beginning of a third phase. Current AI-powered tools are still nascent. Some tools offer "AI resume writers" that generate generic, often-plagiarized summaries. Other ATS systems use AI for basic keyword extraction.

A clear gap exists for a single, integrated platform where Generative AI is not just an add-on but the core engine. A system that can intelligently *parse* unstructured documents, *critique* and *enhance* content, *quantitatively score* a resume's quality, *dynamically match* that resume against job descriptions with nuanced understanding, and *prepare* the candidate for the final interview, all within one seamless workflow.

2.2 Existing and Proposed System

Existing System

The existing "system" is the fragmented and manual process described above.

- **Resume Creation:** Relies on user's writing skill and knowledge of (often arbitrary) ATS keywords. Templates are "dumb" and do not adapt to content.
- **Job Search:** Based on simple keyword and location filters. Results are voluminous and often irrelevant.
- **Applicant Screening:** A manual, time-consuming process for recruiters, leading to high "time-to-hire" and potential for bias.
- **Feedback Loop:** Non-existent. Job seekers apply and often hear nothing back, with no understanding of why.

Proposed System

The "AI Resume Builder" proposes an integrated, intelligent ecosystem to solve these problems.

- **Centralized Platform:** A single web application for both seekers and employers.
- **AI as a Co-pilot:** Generative AI is woven into every step:
  - **Parsing:** Solves the "blank page" problem by ingesting an existing resume, saving hours of data entry.
  - **Enhancement:** Bridges the writing-skill gap, allowing users to sound more professional.
  - **Scoring:** Provides an immediate, objective feedback loop that was previously missing.
  - **Matching:** Moves beyond keywords to semantic understanding. The AI can understand that "managed a team of 5" is relevant for a "Senior" role, even if the resume doesn't use the word "senior."
  - **Summarization:** Saves recruiters time by providing a "TL;DR" for each candidate's fit.
  - **Preparation:** Uses the exact job and resume data to create highly relevant interview questions, closing the loop from application to interview.
- **Asynchronous Architecture:** By using Celery and Redis, the system offloads heavy AI tasks, keeping the user experience fast and fluid. The user can upload their resume and be redirected to a validation page while the AI parses in the background.

2.3 Tools and Technologies Used

- **Backend:**
  - **Python 3.11+:** Core programming language.
  - **Django:** High-level Python web framework for rapid development (MVT).
  - **Gunicorn:** WSGI HTTP server for production deployment.
- **Frontend:**
  - **HTML5:** Standard markup language for web pages.
  - **Tailwind CSS:** A utility-first CSS framework for rapid UI development.
  - **Alpine.js:** A minimal JavaScript framework for declarative, reactive UIs.
- **Database:**
  - **PostgreSQL:** Primary relational database for production (via dj-database-url).
  - **SQLite3:** Default database for local development.
- **AI:**
  - **Google Gemini:** Generative AI model used for all intelligent features.
- **Async Tasks:**
  - **Celery:** Asynchronous task queue for background processing.
  - **Redis:** In-memory data store, used as the Celery broker and result backend.
- **Authentication:**
  - **django-allauth:** Handles user registration, login, password reset, and social auth (Google & GitHub OAuth 2.0).
- **PDF Tools:**
  - **WeasyPrint:** Generates high-quality PDF documents from HTML/CSS templates.
  - **PyMuPDF (fitz):** High-performance PDF text and image extraction.
  - **python-docx:** Library for reading and parsing DOCX files.
- **Deployment:**
  - **Docker:** Containerization for consistent production environments.

  - **Railway.app:** Cloud platform for deploying the application, database, and worker.

- **Utilities:**
  - **python-dotenv:** Manages environment variables from a .env file.
  - **crispy-forms:** Manages Django form rendering with Tailwind CSS.
  - **icalendar:** Generates .ics calendar files for interviews.

2.4 Hardware and Software Requirements

Hardware Requirements (Development)

- **Processor:** Intel Core i3 (or equivalent) or higher.
- **RAM:** 8 GB or more (recommended for running multiple services).
- **Hard Disk:** 20 GB of free space.
- **Network:** Internet connection (for API calls and package installation).

Software Requirements (Development)

- **Operating System:** Windows 10/11, macOS, or a modern Linux distribution.
- **IDE:** VS Code, PyCharm, or any modern text editor.
- **Core:** Python 3.11+, Git.
- **Services:** PostgreSQL (v12+), Redis (v6+).
- **Browser:** Google Chrome, Mozilla Firefox, or Microsoft Edge.

Software Requirements (User)

- **Operating System:** Any (Windows, macOS, Linux, Android, iOS).
- **Browser:** A modern, up-to-date web browser (e.g., Chrome, Firefox, Safari, Edge).
- **Network:** A stable internet connection.

Chapter 3: System Analysis

3.1 Problems in Existing System

The current, fragmented system of job hunting and recruiting is plagued by inefficiencies and misaligned incentives for both parties.

- **For Job Seekers:**
  - **High Friction:** Creating a resume is a difficult, time-consuming task. Many users are not professional writers and struggle to articulate their value.
  - **The "Black Box" ATS:** Applicants "keyword-stuff" their resumes to pass automated screening tools (ATS), often at the expense of readability.
  - **Lack of Feedback:** Most applications receive no response, leaving the user with no idea how to improve.
  - **Irrelevant Results:** Traditional job searches are noisy, returning thousands of poor-fit jobs, leading to application fatigue.
  - **Interview Anxiety:** Candidates are often unprepared for specific, competency-based questions.
- **For Employers:**
  - **Application Overload:** Popular job postings can receive hundreds of applications, many from unqualified candidates.
  - **High "Time-to-Hire":** Recruiters spend the majority of their time on low-value tasks like manual screening rather than on high-value tasks like interviewing.
  - **Poor-Fit Candidates:** Keyword-based ATS systems can be gamed, leading to candidates who look good on paper but lack the true required skills.
  - **Recruiter Bias:** Unconscious human bias can filter out excellent candidates during the initial 6-second manual scan.

3.2 Proposed Solution

The "AI Resume Builder" is a holistic platform designed to solve these specific problems by embedding intelligence at each step of the process.

- **Solving Seeker Problems:**
  - **High Friction:** AI Resume Parsing (from PDF/DOCX) and AI Text Enhancement directly solve this. The user goes from an old file or a rough draft to a professional resume in minutes.
  - **The "Black Box" ATS:** AI Resume Scoring provides the transparency the black box lacks. It tells the user *why* their resume is strong or weak.
  - **Lack of Feedback:** The AI Resume Scoring feature *is* the feedback loop, providing instant, objective advice.
  - **Irrelevant Results:** AI Job Matching inverts the search. Instead of the user hunting for keywords, the system *ranks* jobs based on a deep, semantic understanding of their resume, bringing the best-fit jobs to the top.
  - **Interview Anxiety:** The AI Interview Prep tool provides a safe, relevant way to practice, building confidence.
- **Solving Employer Problems:**
  - **Application Overload:** AI Job Matching provides the first-line defense. Recruiters immediately see applicants ranked by their match score.
  - **High "Time-to-Hire":** AI Applicant Summary allows a recruiter to understand a candidate's fit in 10 seconds, not 10 minutes, drastically reducing screening time.
  - **Poor-Fit Candidates:** The AI matching is semantic, not just keyword-based. It understands context, making it harder to "game" and resulting in a more accurate list of qualified candidates.
  - **Recruiter Bias:** By providing an objective score and summary, the system empowers recruiters to focus on skills and experience first.

3.3 Functional Requirements

Functional requirements define the specific behaviors and functions of the system.

- **User Management**
  - System shall allow users to register as either a 'Job Seeker' or 'Employer'.
  - System shall allow users to log in using their username/email and password.
  - System shall support social authentication (Google, GitHub) via django-allauth.

- System shall provide a secure password reset mechanism via email.
- Job Seeker users shall have a profile (JobSeekerProfile).
- Employer users shall have a profile (EmployerProfile).
- Employers must complete an "onboarding" form (company details) after first login.

- **Resume Management**
  - A Job Seeker shall be able to create and edit a resume, section by section.
  - Resume sections shall include: Experience, Education, Skills, Projects, Certifications, Achievements, Languages, and Hobbies.
  - A Job Seeker shall be able to upload an existing resume (PDF or DOCX).
  - The system shall parse the uploaded resume text and use AI to extract structured data.
  - The user shall be presented with a validation form pre-filled with this extracted data.
  - A Job Seeker shall be able to trigger an AI text enhancement for their professional summary and experience descriptions.
  - The system shall provide an AI-generated score (0-100) and actionable feedback for the user's resume.
  - The system shall regenerate the AI score asynchronously whenever the resume is updated.
  - A Job Seeker shall be able to download their resume as a PDF in four different templates (Classic, Modern, Professional, Creative).
  - PDF generation shall be an asynchronous task to prevent UI blocking.
- **Job & Application Management (FR-J)**
  - An Employer shall be able to create, read, update, and delete job postings.
  - The system shall provide an "AI Job Description Generator" to assist employers.
  - A Job Seeker shall be able to browse and search all active job postings.
  - The system shall display an "AI Match Score" to the Job Seeker for each job, based on their primary resume.
  - A Job Seeker shall be able to apply for a job.

  - ➢ The system shall prevent a user from applying if their resume is incomplete (missing summary, education, or skills).
  - ➢ An Employer shall be able to view all applicants for their job, ranked by match score.
  - ➢ An Employer shall be able to trigger an "AI Applicant Summary" for any applicant.
  - ➢ An Employer shall be able to update an applicant's status (e.g., 'Submitted', 'Under Review', 'Interview', 'Rejected').

- **Interview Management**
  - ➢ An Employer shall be able to schedule an interview by proposing time slots.
  - ➢ A Job Seeker shall be able to view proposed slots and confirm one.
  - ➢ A Job Seeker shall be able to access an "AI Interview Prep" tool for any job they have an 'Interview' status for.
  - ➢ The system shall generate practice questions and answers based on the user's resume and the job description.
- **Admin & Feedback**
  - ➢ A staff user shall be able to access a custom admin dashboard.
  - ➢ An admin shall be able to view system statistics (user counts, bug reports).
  - ➢ An admin shall be able to view and resolve bug reports.
  - ➢ Any user shall be able to submit a bug report with an optional screenshot.
  - ➢ Any user shall be able to submit general feedback.

3.4 Non-functional Requirements

Non-functional requirements define the quality attributes and constraints of the system.

- **Performance**
  - ➢ All AI-driven tasks (parsing, scoring, PDF gen) shall be executed asynchronously using Celery to ensure the UI remains responsive.
  - ➢ Job list page load time shall be under 3 seconds, even with AI matching (achieved via caching JobMatchScore).
  - ➢ AI text enhancement shall return a response in under 5 seconds.

- **Scalability**
  - ➢ The system shall be containerized using Docker for portable and scalable deployment.
  - ➢ The system shall support a production-grade database (PostgreSQL).
  - ➢ The Celery-Redis architecture shall allow for scaling workers horizontally to handle increased load.
- **Usability**
  - ➢ The user interface shall be modern, clean, and intuitive (using Tailwind CSS).
  - ➢ The application shall be fully responsive and functional on mobile, tablet, and desktop browsers.
  - ➢ The resume builder shall provide real-time feedback (AI score) and clear validation.
- **Security**
  - ➢ All user passwords shall be securely hashed using Django's default password hasher.
  - ➢ The system shall be protected against common web vulnerabilities (CSRF, XSS, SQL Injection) via Django's built-in protections.
  - ➢ All user-submitted content (resumes, job descriptions) shall be sanitized to prevent prompt injection attacks.
  - ➢ User authentication shall be robust, supporting email/password and OAuth 2.0 (Google, GitHub).
  - ➢ Users shall only be able to access and modify their own data (e.g., a user cannot edit another user's resume).
- **Reliability**
  - ➢ The system shall use a production-grade WSGI server (Gunicorn) for stable deployment.
  - ➢ Asynchronous tasks shall include retry logic to handle transient API failures.
  - ➢ PDF generation shall be robust, supporting four distinct templates without failure.

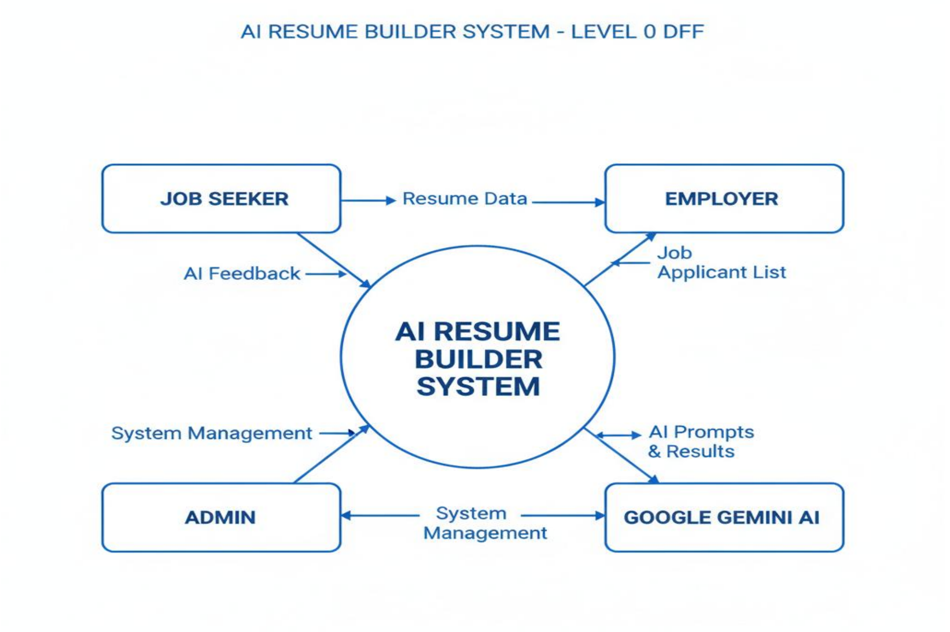
- **Maintainability**
    - ➢ The project shall be structured into modular Django apps (users, resumes, jobs, pages).
    - ➢ Code shall follow PEP 8 standards and include comments for complex logic.
    - ➢ Environment configuration shall be managed via a .env file, not hardcoded.

Chapter 4: System Design

4.1 Data Flow Diagram (DFD)

Data Flow Diagrams (DFDs) illustrate how data moves through the system.

Level 0: Context Diagram

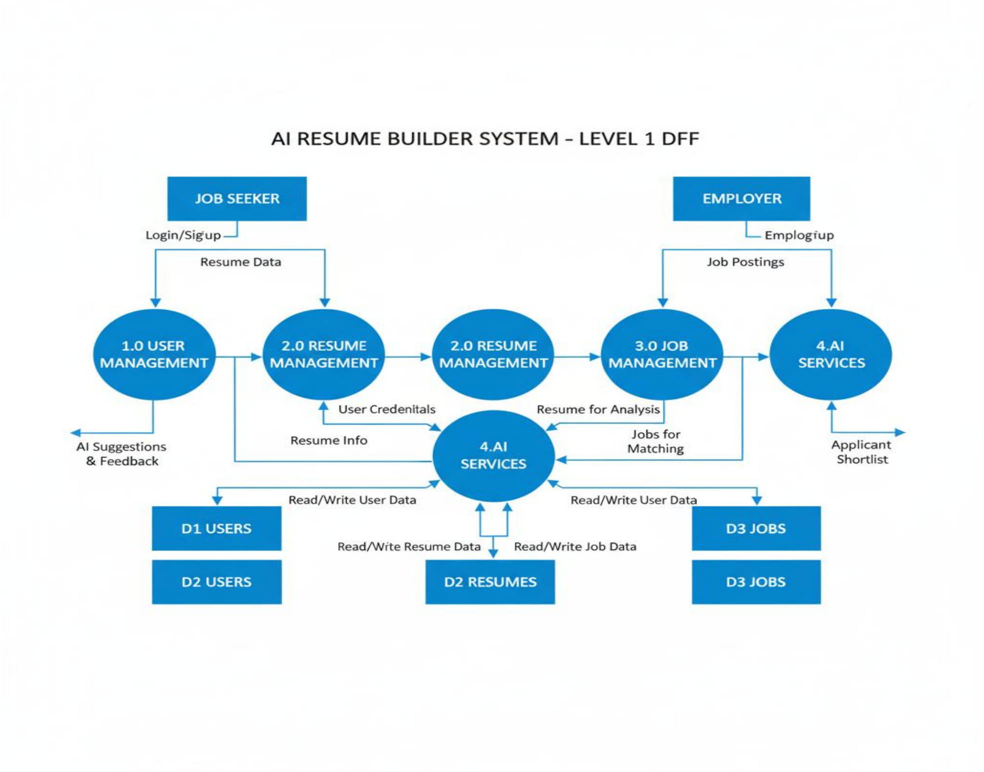


The Context Diagram shows the entire system as a single process interacting with external entities.

- **External Entities:**
  - **Job Seeker:** Provides personal data, resume content, and job applications. Receives resume drafts, AI scores, PDF resumes, and job listings.
  - **Employer:** Provides company data and job postings. Receives applicant lists, AI summaries, and interview confirmations.
  - **Admin:** Manages the system, views bug reports, and monitors users.
  - **Google Gemini AI:** Receives text prompts (resume text, job descriptions). Provides structured JSON, enhanced text, scores, and summaries.
  - **Email Service (Resend):** Receives email data (e.g., password reset). Sends emails to users.

- **Process:**
    - **0. AI Resume Builder System:** The entire web application.

Level 1: DFD



The Level 1 DFD breaks the system into its main sub-processes.

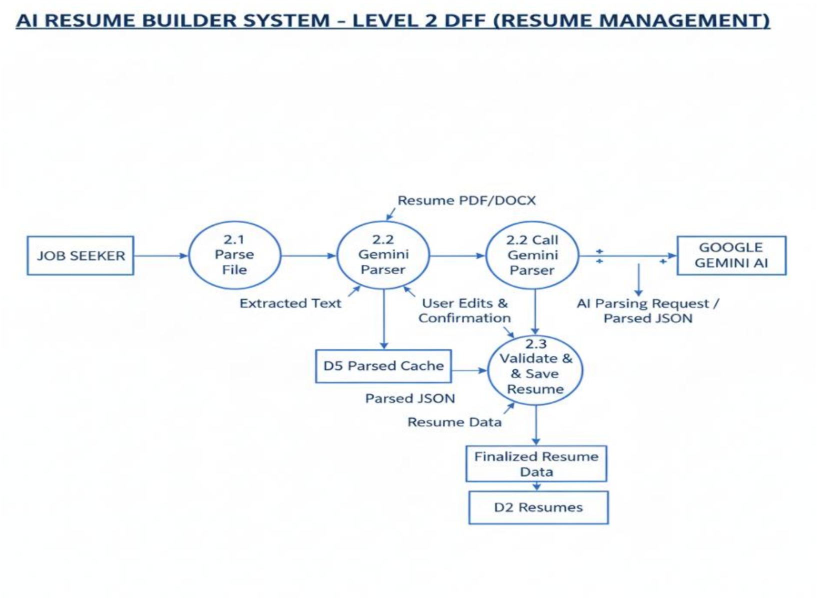
- **Processes:**
    - **Manage Users:** Handles registration, login, social auth, and profile updates.
    - **Manage Resumes:** Handles resume creation, editing, uploading, parsing, scoring, enhancement, and PDF generation.
    - **Manage Jobs:** Handles job posting, listing, applying, matching, applicant viewing, and interview scheduling.
    - **Manage Admin:** Handles bug reporting, feedback, and the admin dashboard.

- **Data Stores:**
    1. **D1 - Users:** Stores CustomUser, JobSeekerProfile, EmployerProfile.
    2. **D2 - Resumes:** Stores Resume, Experience, Education, Skill, etc.

3. **D3 - Jobs:** Stores JobPosting, Application, Interview, JobMatchScore.
4. **D4 - Feedback:** Stores BugReport, Feedback.

- **AI Service:** This is a conceptual process representing the AI logic in parser.py and matcher.py, which communicates with the external Gemini API.
-

Level 2: DFD (Example: Process 2.0 Resume Management)

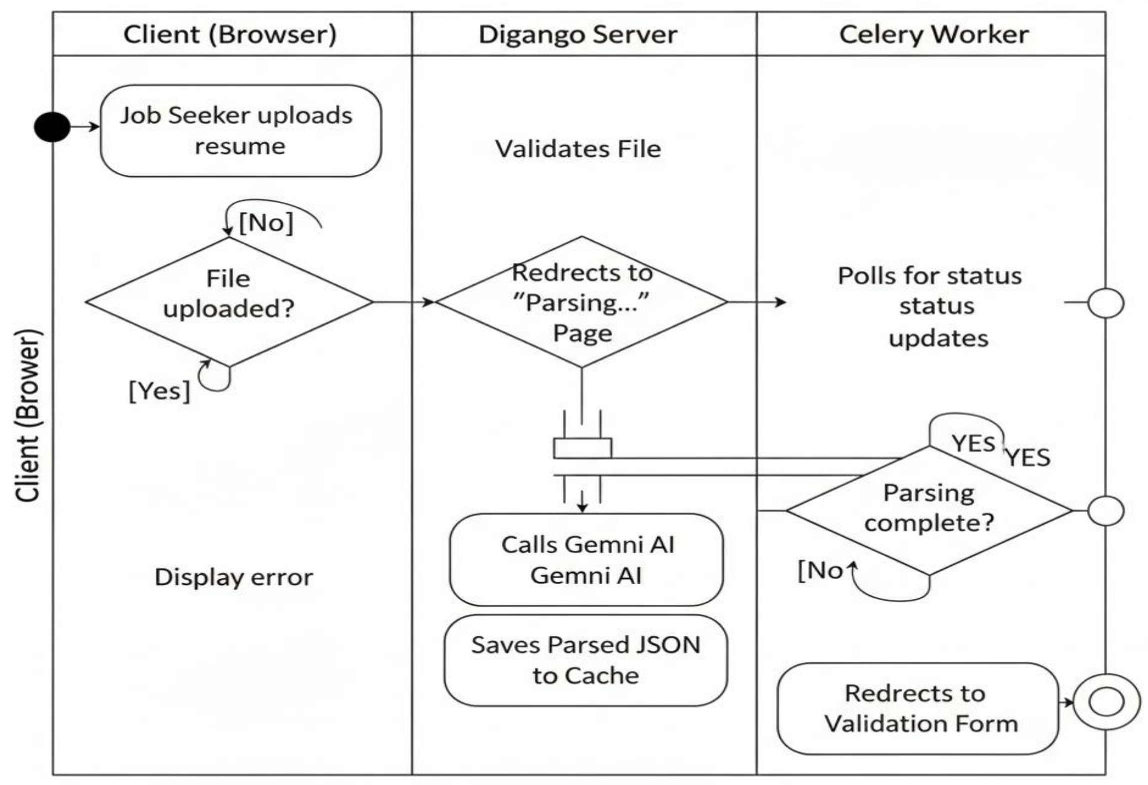


This diagram details a single process from Level 1.

- **This DFD shows the asynchronous parsing flow:**
  - The Job Seeker Uploads File to process 2.1 Parse Resume File.
  - 2.1 (a Celery task) extracts text and calls 2.2 Call Gemini Parser.
  - 2.2 gets Parsed JSON from the Gemini API.
  - The Parsed JSON is saved to the D5 ParsedResumeCache data store.
  - The Job Seeker is redirected to 2.3 Validate & Save Resume.
  - This process reads from the D5 cache, displays the data in a form, and on submission, saves the final, validated data to the main D2 Resumes data store.

4.2 Activity Diagram

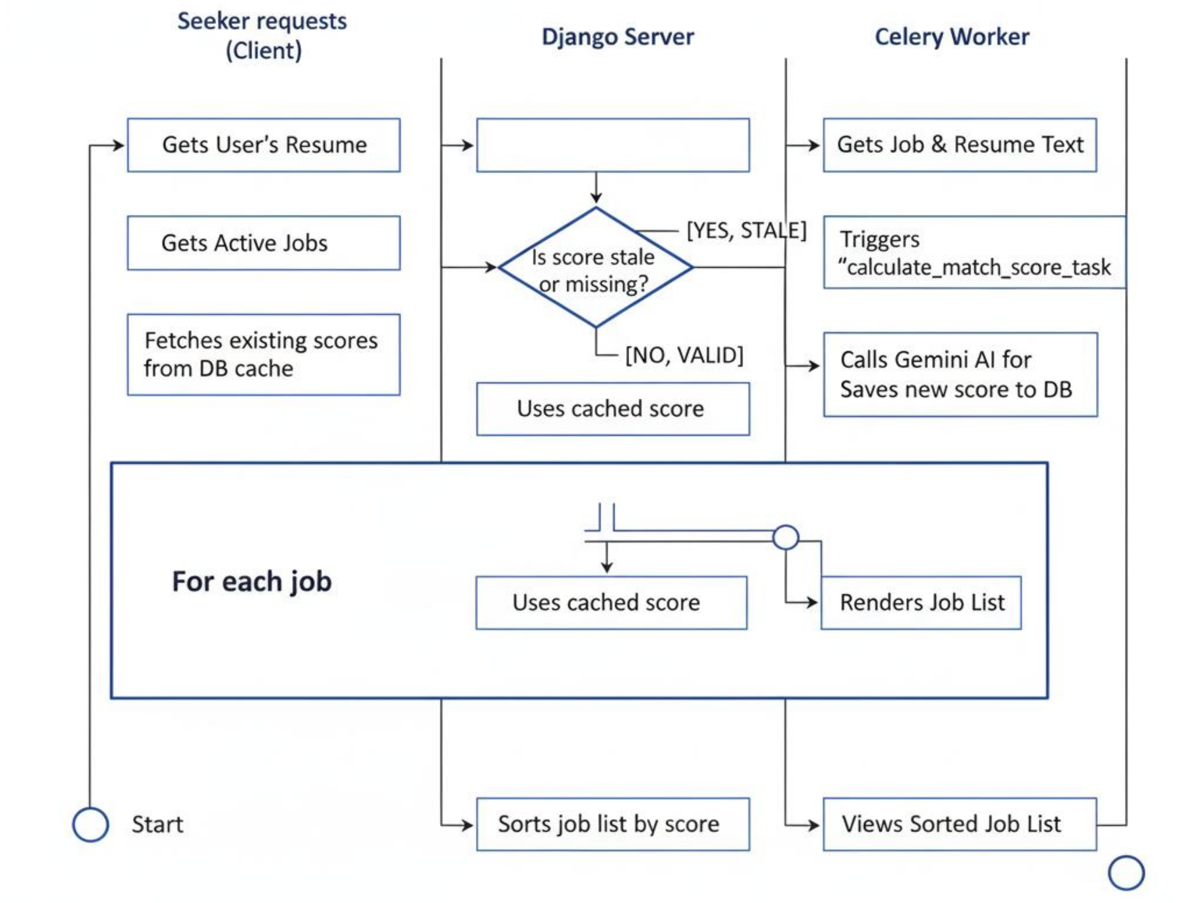
Activity Diagram: AI Resume Parsing



1. **Actor:** Job Seeker
2. **Swimlanes:** Client (Browser), Django Server, Celery Worker, Google Gemini API.
3. **Flow:**
    - **Client:** User visits Resume Dashboard and submits the upload form.
    - **Server:** upload_resume_view receives the file. It validates the file type/size.
    - **Server:** The file content is read and base64-encoded.
    - **Server:** Triggers the parse_resume_task.delay() Celery task, passing the user ID and file content.
    - **Server:** Redirects the user to the parsing_progress.html page.

- **Celery Worker:** Picks up the task.
- **Celery Worker:** Decodes the file and uses PyMuPDF/docx to extract text.
- **Celery Worker:** Calls parse_text_with_gemini, which sends the text to the **Google Gemini API**.
- **Google Gemini API:** Returns structured JSON.
- **Celery Worker:** Saves the JSON to the ParsedResumeCache model, linked to the user.
- **Client:** The parsing_progress.html page polls the check_parsing_status_view.
- **Server:** check_parsing_status_view checks if a ParsedResumeCache entry exists for the user.
- **Server:** If not found, returns {'status': 'PENDING'}. Client polls again.
- **Server:** If found, returns {'status': 'SUCCESS'}.
- **Client:** Redirects to validate_resume_data_view.
- **Server:** validate_resume_data_view loads the JSON from the cache, populates the formsets, and renders the validation page.
- **Client:** User reviews, corrects, and submits the validation form.
- **Server:** The POST request to validate_resume_data_view validates all formsets and saves the data to the final Resume, Experience, etc. models.
- **Server:** Deletes the ParsedResumeCache entry and redirects to the resume_builder_view.

Activity Diagram: AI Job Matching

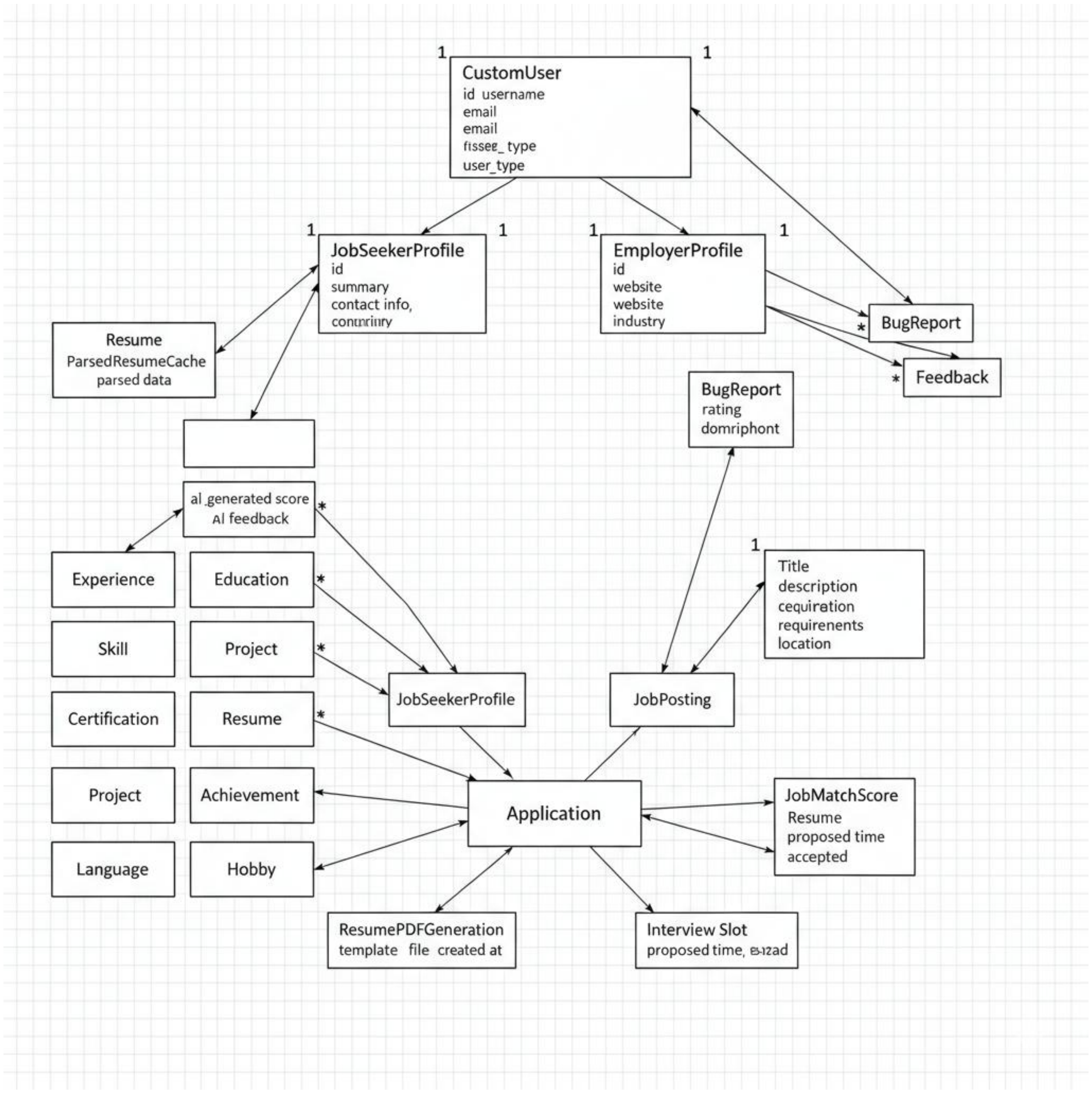


1. **Actor**: Job Seeker
2. **Swimlanes:** Client (Browser), Django Server, Celery Worker.
3. **Flow:**
   - **Client**: User visits the job_list_view.
   - **Server**: Receives the request.
   - **Server:** Fetches the user's most recently updated Resume.
   - **Server**: Fetches all active JobPosting objects.

- **Server:** Fetches all *existing* JobMatchScore entries for this user's resume in a single query.
- **Server**: Begins a loop: For each JobPosting...
- **Server:** Checks if a JobMatchScore exists in the fetched cache.
- **Server:** Checks if the score is stale (e.g., resume.updated_at > score.last_calculated or job.updated_at > score.last_calculated).
- **Server (Decision):**
    - **[If Stale or Missing]:** Triggers the calculate_and_save_match_score_task.delay() Celery task for this resume/job pair. The score displayed for now will be 0 or the old score.
    - **[If Not Stale]:** Uses the valid score from the cache.
- **Server (Fork):** (In parallel)
    - **Celery Worker:** Picks up the task.
    - **Celery Worker:** Fetches resume text and job text.
    - **Celery Worker:** Calls calculate_match_score (which calls the Google Gemini API).
    - **Celery Worker:** Saves the new score to the JobMatchScore table.
- **Server (Join):** After iterating all jobs...
- **Server:** Sorts the job list (with their current scores) from highest to lowest.
- **Server:** Renders the job_list.html template, passing the sorted list.
- **Client:** Sees the job list, ranked by match score. When the page is refreshed later, the newly calculated scores will appear.

4.3 Entity-Relationship (ER) Diagram

An Entity-Relationship (ER) Diagram is a graphical representation of the database schema. It shows the entities (which map to Django Models/database tables) and the relationships between them. This is crucial for understanding the data structure of the application, which is managed by the Django ORM.

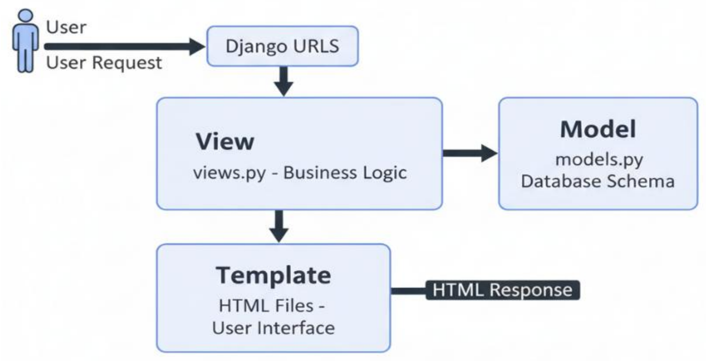


Key Entities and Relationships:

- **CustomUser:** The base user model. It has a one-to-one relationship with *either* JobSeekerProfile or EmployerProfile, determined by the user_type field.
- **JobSeekerProfile:** Stores all seeker-specific data. It has a one-to-many relationship with Resume (one seeker can have multiple resumes).
- **Resume:** This is the central entity for a job seeker. It has one-to-many relationships with Experience, Education, Skill, Project, Certification, Achievement, Language, and Hobby (one resume contains many of each).

- **EmployerProfile:** Stores all employer-specific data. It has a one-to-many relationship with JobPosting (one employer can post multiple jobs).
- **JobPosting:** The job listing entity created by an employer.
- **Application:** This is a crucial junction table. It has a many-to-one relationship with JobSeekerProfile and a many-to-one relationship with JobPosting, linking a specific seeker to a specific job. It also has a foreign key to the Resume used for the application.
- **Interview:** Has a one-to-one relationship with Application, as an application can only have one interview record.
- **JobMatchScore:** A cache table with foreign keys (many-to-one) to Resume and JobPosting, forming a unique pair to store a calculated score.
- **ParsedResumeCache & ResumePDFGeneration:** These are utility models that relate to JobSeekerProfile or Resume to track the state of asynchronous tasks.

4.4 MVT Architecture



This project is built using Django, which follows a Model-View-Template (MVT) architecture.

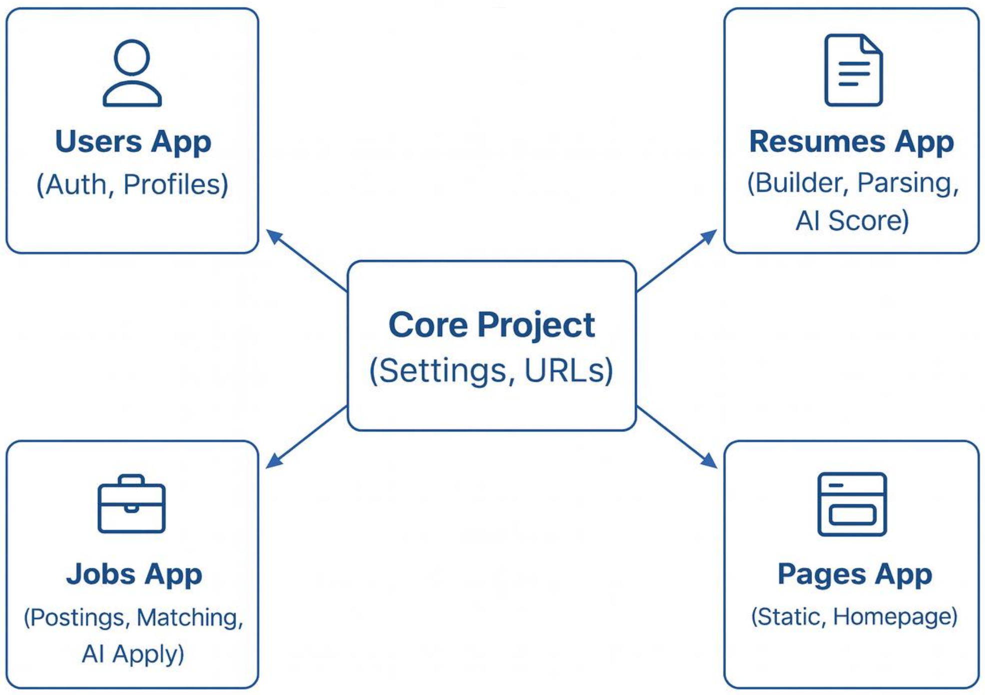
- **Model:** Represents the data structure and logic. All models are defined in the models.py file of their respective app.

- users/models.py: Defines CustomUser (extending AbstractUser), JobSeekerProfile, and EmployerProfile. This is the foundation of the system's data.
- resumes/models.py: Defines Resume as the central model, with related models for each section: Experience, Education, Skill, Project, Certification, Achievement, Language, Hobby. It also defines ParsedResumeCache and ResumePDFGeneration for managing async tasks.
- jobs/models.py: Defines JobPosting, Application (linking JobPosting and JobSeekerProfile), Interview, InterviewSlot, and JobMatchScore (the cache for AI scores).

- **View:** The business logic layer. It receives HTTP requests, interacts with the models (and AI services), and renders a template. All views are in the views.py file of their respective app.
  - users/views.py: Handles register_view, login_view, logout_view, employer_onboarding_view.
  - resumes/views.py: Contains the most complex logic. resume_builder_view handles all AJAX (add, edit, delete) requests for resume sections. upload_resume_view and validate_resume_data_view manage the AI parsing flow. download_resume_pdf and preview_resume_view handle PDF generation. enhance_description_api is the endpoint for AI text enhancement.
  - jobs/views.py: Handles job_list_view (which triggers matching), apply_for_job_view (which checks resume completeness), view_applicants_view (which shows ranked applicants), and the various AI API endpoints (generate_job_description_api, generate_applicant_summary_api, generate_interview_prep_api).
- **Template:** The presentation layer, written in HTML with Tailwind CSS and Alpine.js. All templates are in the templates/ directory of their app.
  - templates/base.html: The main site template, including the navbar, footer, and toast/modal components.
  - resumes/resume_builder.html: A complex, single-page application (SPA) style page managed by Alpine.js, which dynamically loads and submits forms.
  - resumes/validate_resume.html: The dynamic formset page for validating parsed resume data.
  - jobs/job_list.html: Displays job postings, conditionally showing AI match scores.

- resumes/resume_pdf_*.html: The four distinct templates (classic, modern, professional, creative) used by WeasyPrint to generate PDFs.

4.5 Core Modules



The project is logically divided into four main Django applications:

1. **users (App):**
   - **Purpose:** Manages user authentication, registration, and profiles.
   - **Key Components:**
     - models.py: CustomUser, JobSeekerProfile, EmployerProfile.
     - views.py: register_view, login_view, edit_profile_view.
     - forms.py: CustomUserCreationForm, ProfileUpdateForm, EmployerOnboardingForm.
     - adapters.py: Custom django-allauth adapters to redirect users based on their type (job_seeker vs. employer).

2. **resumes (App):**
   - **Purpose:** Core of the Job Seeker experience. Manages all aspects of resume creation, AI analysis, and PDF generation.
   - **Key Components:**
     - models.py: Resume and all related section models (Experience, Education, etc.), plus ParsedResumeCache and ResumePDFGeneration.
     - views.py: resume_builder_view (AJAX handler), upload_resume_view (parsing flow), download_resume_pdf (PDF flow), enhance_description_api (AI enhancement).
     - parser.py: Contains the core AI logic for parse_text_with_gemini, enhance_text_with_gemini, and score_and_critique_resume.
     - tasks.py: Celery tasks parse_resume_task, update_resume_score_task, generate_resume_pdf_task.
     - templates/resumes/: Contains the resume_builder.html UI and all four resume_pdf_*.html templates.
3. **jobs (App):**
   - **Purpose:** Core of the Employer experience and the bridge between seekers and employers.
   - **Key Components:**
     - models.py: JobPosting, Application, Interview, JobMatchScore.
     - views.py: job_list_view (shows matches), apply_for_job_view (handles applications), view_applicants_view (shows ranked list), generate_applicant_summary_api, generate_interview_prep_api.
     - matcher.py: Contains the AI logic for calculate_match_score, generate_job_description, generate_applicant_summary, generate_interview_prep.
     - tasks.py: Celery task calculate_and_save_match_score_task to (re)calculate match scores in the background.

4. **pages (App):**
   - **Purpose:** Handles the main homepage, static content, and site-wide utilities like bug reporting.
   - **Key Components:**
     - views.py: home_view (routes users to the correct dashboard), report_bug_view, submit_feedback_view.
     - models.py: BugReport, Feedback.
     - templates/pages/: Contains the different homepages (home_public.html, home_seeker.html, home_employer.html).

Chapter 5: Methodology

5.1 Software Development Model

This project was developed using an **Agile development methodology**, specifically inspired by **Scrum** and **Kanban**. Given the nature of the project—a solo developer integrating complex, experimental AI features—a rigid waterfall model was unsuitable.

Why Agile?

- **Flexibility:** The capabilities of the Generative AI (Gemini) were not fully known at the outset. An agile approach allowed for iterative development, where features were built, tested with the AI, and refined based on the quality of the AI's output.
- **Iterative Development:** The project was built in feature-based sprints:
    - **Sprint 1: Core Foundation:** User models, auth, basic resume models, and templates.
    - **Sprint 2: Core Resume AI:** AI parsing, validation flow, and AI text enhancement.
    - **Sprint 3: PDF & Scoring:** PDF generation with WeasyPrint and async AI resume scoring.
    - **Sprint 4: Employer & Jobs:** Job posting, job list, and application models.
    - **Sprint 5: Core Matching AI:** JobMatchScore model, calculate_and_save_match_score_task, and display on the job list.
    - **Sprint 6: Full Lifecycle AI:** Interview models, AI Interview Prep, and AI Applicant Summary.
    - **Sprint 7: Admin & Polish:** Admin dashboard, bug reporting, and UI refinement.
- **Rapid Feedback:** This iterative process allowed for continuous testing of the AI prompts. For example, the parse_text_with_gemini prompt was refined multiple times to improve its JSON output accuracy.

5.2 Project Phases

The project followed the standard Software Development Life Cycle (SDLC) phases, adapted for an agile model.

1. **Phase 1: Requirement Analysis & Planning**

- **Activity:** Identified the core problem (fragmented job-hunting process), defined the two main user roles (Seeker, Employer), and researched the feasibility of using Generative AI to solve the key pain points.
- **Deliverables:** Functional & Non-functional requirements (see Chapter 3), Technology Stack selection (Django, Celery, Gemini).

2. **Phase 2: System Design**
   - **Activity:** Designed the database schema (Models), application architecture (MVT, Core Modules), and user flow (DFDs, Activity Diagrams).
   - **Deliverables:** Database schema, Architecture diagrams (see Chapter 4).
3. **Phase 3: Implementation (Coding)**
   - **Activity:** Wrote the Python (Django, Celery) and frontend (HTML/Tailwind/Alpine.js) code. This was the longest phase, broken into the sprints described in 5.1. A key part of this phase was "prompt engineering"—crafting and testing the prompts in parser.py and matcher.py to get reliable, structured, and secure responses from the Gemini API.
   - **Deliverables:** The complete, functional source code (see Chapter 6).
4. **Phase 4: Testing**
   - **Activity:**
     - **Unit Testing:** Wrote unit tests (using django.test.TestCase) for core functionality, such as form validation (resumes/tests.py) and view permissions (jobs/tests_views.py).
     - **Integration Testing:** Manually tested the complete user flows, such as:
       - Upload -> Parse -> Validate -> Save -> Download PDF.
       - Update Resume -> Score Regenerates -> Job Match Score Updates.
     - **AI Testing:** Used test_api.py to confirm API key and connectivity. Manually tested AI features with edge cases (e.g., empty text, malicious prompts) to ensure robustness.
   - **Deliverables:** A set of passing unit tests and a stable application.

5. **Phase 5: Deployment**
   - **Activity:** Containerized the application using Dockerfile. Set up production services (PostgreSQL, Redis) on the Railway.app cloud platform. Configured environment variables (e.g., DJANGO_SECRET_KEY, GOOGLE_AI_API_KEY, DATABASE_URL). Deployed the Django app (Gunicorn) and the Celery worker as separate services.
   - **Deliverables:** A publicly accessible, fully functional web application.
6. **Phase 6: Maintenance**
   - **Activity:** This is an ongoing phase. It involves monitoring application logs for errors, responding to user feedback, and fixing bugs. The BugReport model and admin dashboard are key tools for this phase.
   - **Deliverables:** A stable and reliable service for users.

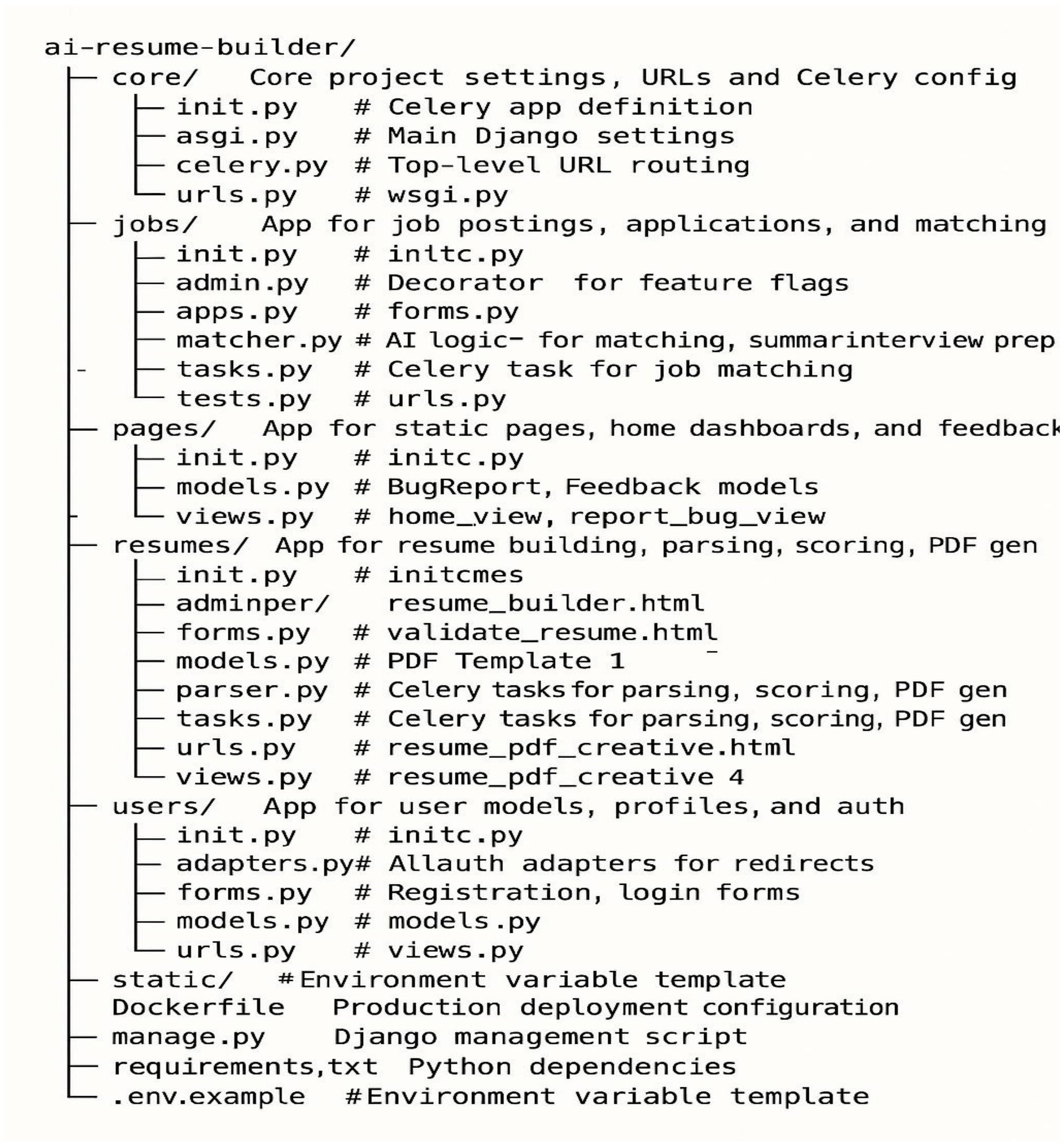
Chapter 6: Code Snippets

This chapter contains key code snippets from the project's core modules, demonstrating the implementation of its main features.

(Reference: https://github.com/kiruuuuuuu/ai-resume-builder)

6.1 Project Structure

The project is organized into a core project folder and four distinct Django apps: users, resumes, jobs, and pages.



```
ai-resume-builder/
├─ core/   Core project settings, URLs and Celery config
│   ├─ init.py    # Celery app definition
│   ├─ asgi.py    # Main Django settings
│   ├─ celery.py  # Top-level URL routing
│   └─ urls.py    # wsgi.py
├─ jobs/    App for job postings, applications, and matching
│   ├─ init.py    # initc.py
│   ├─ admin.py   # Decorator  for feature flags
│   ├─ apps.py    # forms.py
│   ├─ matcher.py # AI logic- for matching, summarinterview prep
-   ├─ tasks.py   # Celery task for job matching
│   └─ tests.py   # urls.py
├─ pages/   App for static pages, home dashboards, and feedback
│   ├─ init.py    # initc.py
│   ├─ models.py  # BugReport, Feedback models
-   └─ views.py   # home_view, report_bug_view
├─ resumes/  App for resume building, parsing, scoring, PDF gen
│   ├─ init.py    # initcmes
│   ├─ adminper/    resume_builder.html
│   ├─ forms.py   # validate_resume.html
│   ├─ models.py  # PDF Template 1
│   ├─ parser.py  # Celery tasks for parsing, scoring, PDF gen
│   ├─ tasks.py   # Celery tasks for parsing, scoring, PDF gen
│   ├─ urls.py    # resume_pdf_creative.html
│   └─ views.py   # resume_pdf_creative 4
├─ users/   App for user models, profiles, and auth
│   ├─ init.py    # initc.py
│   ├─ adapters.py# Allauth adapters for redirects
│   ├─ forms.py   # Registration, login forms
│   ├─ models.py  # models.py
│   └─ urls.py    # views.py
├─ static/   #Environment variable template
│  Dockerfile   Production deployment configuration
├─ manage.py    Django management script
├─ requirements,txt  Python dependencies
└─ .env.example  #Environment variable template
```

6.2 Key Model Definitions (models.py)

users/models.py

Defines the two types of users in the system.

```
from django.db import models
from django.contrib.auth.models import AbstractUser


class CustomUser(AbstractUser):
    """
    Custom user model to differentiate between Job Seekers and Employers.
    """
    USER_TYPE_CHOICES = (
        ('job_seeker', 'Job Seeker'),
        ('employer', 'Employer'),
    )
    user_type = models.CharField(max_length=10, choices=USER_TYPE_CHOICES, default='job_seeker')


class JobSeekerProfile(models.Model):
    """
    Profile for users who are seeking jobs.
    Linked one-to-one with the CustomUser model.
    """
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE, primary_key=True)
    profile_photo = models.ImageField(upload_to='profile_photos/', null=True, blank=True)
    full_name = models.CharField(max_length=255, null=True, blank=True)
    professional_summary = models.TextField(null=True, blank=True)
    phone_number = models.CharField(max_length=20, null=True, blank=True)
    address = models.CharField(max_length=255, null=True, blank=True)
    date_of_birth = models.DateField(null=True, blank=True)
    portfolio_url = models.URLField(null=True, blank=True)
    linkedin_url = models.URLField(null=True, blank=True)

    def __str__(self):
        return self.user.username

class EmployerProfile(models.Model):
    """
    Profile for users who are employers/recruiters.
    Linked one-to-one with the CustomUser model.
    """
    INDUSTRY_CHOICES = [
        ('Technology', 'Technology'),
        ('Finance', 'Finance'),
        # ... other choices
        ('Other', 'Other'),
    ]

    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE,
primary_key=True)
    company_name = models.CharField(max_length=255, null=True, blank=True)
    company_website = models.URLField(null=True, blank=True)
    company_description = models.TextField(null=True, blank=True)
    company_logo = models.ImageField(upload_to='company_logos/', null=True, blank=True)
    company_bio = models.TextField(null=True, blank=True, help_text="Extended company biography for public profile")
    location = models.CharField(max_length=255, null=True, blank=True)
    industry = models.CharField(max_length=50, choices=INDUSTRY_CHOICES, null=True, blank=True)

    def __str__(self):
        return self.company_name or self.user.username
```

resumes/models.py

Defines the Resume and its child sections, as well as models for managing async tasks.

```
from django.db import models
from users.models import JobSeekerProfile

class ParsedResumeCache(models.Model):
    profile = models.OneToOneField(JobSeekerProfile, on_delete=models.CASCADE, primary_key=True)
    parsed_data = models.JSONField()
    created_at = models.DateTimeField(auto_now_add=True)

class Resume(models.Model):
    profile = models.ForeignKey(JobSeekerProfile, on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    score = models.IntegerField(null=True, blank=True)
    feedback = models.JSONField(null=True, blank=True)

    def __str__(self):
        return self.title

class Experience(models.Model):
    resume = models.ForeignKey(Resume, on_delete=models.CASCADE)
    job_title = models.CharField(max_length=255)
    company = models.CharField(max_length=255)
    start_date = models.DateField(null=True, blank=True)
    end_date = models.DateField(null=True, blank=True)
    description = models.TextField(null=True, blank=True)
    # ... other models: Education, Skill, Project, etc. ...

class ResumePDFGeneration(models.Model):
    """Tracks async PDF generation tasks for resumes."""
    STATUS_CHOICES = [
        ('pending', 'Pending'),
        ('processing', 'Processing'),
        ('completed', 'Completed'),
        ('failed', 'Failed'),
    ]

    resume = models.ForeignKey(Resume, on_delete=models.CASCADE,
related_name='pdf_generations')
    template_name = models.CharField(max_length=50)
    accent_color = models.CharField(max_length=50, default='blue')
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending')
    task_id = models.CharField(max_length=255, null=True, blank=True, help_text="Celery task
ID")
    pdf_file = models.FileField(upload_to='generated_pdfs/', null=True, blank=True)
    pdf_content = models.BinaryField(null=True, blank=True, help_text="PDF content stored in database for Railway compatibility")
    error_message = models.TextField(null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    completed_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        ordering = ['-created_at']
```

jobs/models.py

Defines the JobPosting and the Application that links Seekers and Employers.

```
from django.db import models
from users.models import CustomUser, JobSeekerProfile, EmployerProfile
from django.utils import timezone
from resumes.models import Resume


class JobPosting(models.Model):
    employer = models.ForeignKey(EmployerProfile, on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    description = models.TextField()
    requirements = models.TextField()
    location = models.CharField(max_length=255)
    salary_min = models.IntegerField(null=True, blank=True, help_text="Minimum salary in USD")
    salary_max = models.IntegerField(null=True, blank=True, help_text="Maximum salary in USD")
    vacancies = models.PositiveIntegerField(default=1, help_text="Number of available positions for this role.")
    application_deadline = models.DateTimeField(null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title

    @property
    def is_active(self):
        if self.application_deadline:
            return timezone.now() < self.application_deadline
        return True

class Application(models.Model):
    TEMPLATE_CHOICES = [
        ('classic', 'Classic'),
        ('modern', 'Modern'),
        ('professional', 'Professional'),
    ]
    STATUS_CHOICES = [
        ('Submitted', 'Submitted'),
        ('Under Review', 'Under Review'),
        ('Shortlisted', 'Shortlisted'),
        ('Interview', 'Interview'),
        ('Offered', 'Offered'),
        ('Rejected', 'Rejected'),
    ]
    job_posting = models.ForeignKey(JobPosting, on_delete=models.CASCADE)
    applicant = models.ForeignKey(JobSeekerProfile, on_delete=models.CASCADE)
    resume = models.ForeignKey(Resume, on_delete=models.SET_NULL, null=True, blank=True, help_text="The specific resume version used for this application")
    applied_at = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=50, choices=STATUS_CHOICES, default='Submitted')
    resume_template = models.CharField(max_length=20, choices=TEMPLATE_CHOICES, default='classic')


class Interview(models.Model):
    INTERVIEW_STATUS_CHOICES = [
        ('Proposed', 'Proposed'),
        ('Scheduled', 'Scheduled'),
        ('Completed', 'Completed'),
        ('Cancelled', 'Cancelled'),
    ]
    application = models.OneToOneField(Application, on_delete=models.CASCADE)
    confirmed_slot = models.DateTimeField(null=True, blank=True)
    interview_details = models.TextField(null=True, blank=True) # e.g., Meet link, address
    status = models.CharField(max_length=20, choices=INTERVIEW_STATUS_CHOICES, default='Proposed')


class JobMatchScore(models.Model):
    """
    Stores the calculated match score between a resume and job posting
    to act as a cache and improve performance.
    """
    resume = models.ForeignKey(Resume, on_delete=models.CASCADE)
    job_posting = models.ForeignKey(JobPosting, on_delete=models.CASCADE)
    score = models.IntegerField()
    last_calculated = models.DateTimeField(auto_now=True)

    class Meta:
        unique_together = ('resume', 'job_posting')
```

6.3 Core AI Logic (parser.py, matcher.py)

resumes/parser.py (Resume AI)

This file contains the AI logic for parsing, enhancing, and scoring a resume.

```
import os
import google.generativeai as genai
import fitz  # PyMuPDF
import docx
import logging
import re
import json
# ... other imports

def _get_gemini_model(model_name: str = 'models/gemini-2.5-flash'):
    """Initializes and returns a Gemini model instance."""
    try:
        from django.conf import settings
        api_key = settings.GOOGLE_AI_API_KEY
        genai.configure(api_key=api_key)
        return genai.GenerativeModel(model_name)
    except Exception as e:
        logger.error(f"Failed to initialize Gemini model: {e}")
        return None


def _call_gemini_with_retry(model, prompt, max_retries=3, base_delay=2, timeout_seconds=60):
    # ... (Implementation of exponential backoff retry logic) ...


def parse_text_with_gemini(text: str) -> Dict[str, Any]:
    """
    Sends resume text to the Gemini API and asks it to parse the content
    into a structured JSON format. Includes a few-shot example for better accuracy.
    """
    model = _get_gemini_model()
    # ... (Sanitize input text) ...
    prompt = f"""
    You are an expert resume parsing system. Analyze the following resume text and extract the information into a structured JSON object.

    The JSON object must have the following top-level keys: "personal_details", "professional_summary", "experience", "education", "skills", "projects", "certifications", "achievements", "languages", "hobbies".

    - "personal_details" should be an object with keys: "full_name", "email", ...
    - "experience", "education", etc. should be lists of objects...
    - "skills" should be a list of objects with "name" and "category" keys...

    --- EXAMPLE ---
    INPUT TEXT: "Jane Doe - Software Engineer. Contact: jane.d@email.com..."
    EXPECTED JSON OUTPUT:
    {{
      "personal_details": {{ "full_name": "Jane Doe", "email": "jane.d@email.com", ... }},
      "professional_summary": "Software Engineer",
      "experience": [ ... ],
      ...
    }}
    ---

    --- ACTUAL RESUME TEXT TO PARSE ---
    {sanitized_text}
    ---

    CRITICAL: You MUST return ONLY the raw JSON object.
    """
    # ... (Call _call_gemini_with_retry, parse JSON response) ...

def enhance_text_with_gemini(text_to_enhance: str, context: str) -> str:
    """
    Uses Gemini to rewrite and improve a piece of text from a resume.
    """
    model = _get_gemini_model()
    # ... (Sanitize input text) ...
    prompt = f"""
    You are an expert career coach. Rewrite and enhance the following text for a resume's '{context}'
section to be more professional and impactful.
```

```
    ... (Context-specific instructions) ...

    CRITICAL: You MUST return ONLY the enhanced plain text.

    ENHANCED TEXT:
    """
    # ... (Call _call_gemini_with_retry, clean and return text) ...

def score_and_critique_resume(full_resume_text: str) -> Dict[str, Any]:
    """
    Uses Gemini to provide a holistic score and feedback for a resume.
    """
    model = _get_gemini_model()
    # ... (Sanitize input text) ...
    prompt = f"""
    You are an expert and encouraging career coach. Analyze the complete resume text.
    Provide a holistic quality score from 0 to 100 and 2-3 brief, actionable, and encouraging
feedback points.

    **Scoring Guidelines:**
    - **Fresher/Entry-Level Resume:** If no 'Work Experience' section, evaluate as a fresher's
resume.
    - **Impact & Action Verbs (40%):**
    - **Clarity & Readability (30%):**
    - **Completeness & Detail (30%):**

    CRITICAL: You MUST return ONLY a single, raw JSON object with two keys: "score" (an
integer) and "feedback" (a list of strings).
```

```
    Example: {{"score": 85, "feedback": ["Your project descriptions are detailed.", "Consider adding a link to your GitHub."]}}
    """
    # ... (Call _call_gemini_with_retry, parse JSON response) ...
```

jobs/matcher.py (Job Matching AI)

This file contains the AI logic for matching resumes to jobs and assisting employers.

```
# ... (Imports including _get_gemini_model, _call_gemini_with_retry) ...


def _extract_job_details(job_text: str) -> dict:
    """Uses Gemini to parse a job description into a structured format."""
    model = _get_gemini_model()
    prompt = f"""
    Analyze the job posting text and extract key requirements into a structured JSON object.
    Keys: "required_skills" (list), "nice_to_have_skills" (list), "required_experience_years" (int).

    CRITICAL: You MUST return ONLY the raw JSON object.
    """
    # ... (Call API, parse JSON) ...


def score_resume_with_gemini(resume_text: str, job_details: dict) -> int:
    """Calculates the match score using the Gemini API with structured data."""
    model = _get_gemini_model()
    prompt = f"""
    You are a very strict technical recruiter. Analyze the RESUME against the structured JOB REQUIREMENTS and produce a realistic match score from 0-100.

    **Scoring Guidelines:**
```

```
     - **Experience (50%):**
     - **Required Skills (35%):** Penalize heavily for missing required skills.
     - **Nice-to-Have Skills (15%):**

    Return ONLY a single raw JSON object with one key: "score". Example: {{"score": 78}}
    """
    # ... (Call API, parse JSON, return score) ...

def calculate_match_score(resume_text, job_description_text):
    """
    Primary function to calculate match score.
    Step 1: AI analyzes the job description.
    Step 2: AI scores the resume against the structured job details.
    """
    job_details = _extract_job_details(job_description_text)
    if not job_details:
        job_details = {"requirements": job_description_text} # Fallback

    gemini_score = score_resume_with_gemini(resume_text, job_details)
    return gemini_score

def generate_applicant_summary(resume_text: str, job_description: str) -> str:
    """
    Uses Gemini to generate a 3-bullet summary of an applicant's fit.
    """
    model = _get_gemini_model()
    prompt = f"""
    You are an expert recruiter. Analyze the candidate's resume and the job description.
    Create a concise 3-bullet summary highlighting the candidate's fit for this role.

    CRITICAL: You MUST return ONLY a single raw JSON object with one key: "summary"
    Example: {{"summary": "- First bullet\\n- Second bullet\\n- Third bullet"}}
    """
    # ... (Call API, parse JSON, return summary string) ...

def generate_interview_prep(resume_text: str, job_description: str) -> dict:
    """
    Uses Gemini to generate interview questions and STAR-method answers.
    """
    model = _get_gemini_model()
    prompt = f"""
    You are an AI career coach. Generate 5 likely interview questions based on the candidate's resume and the job description.
    For each question, provide a sample answer using the STAR method (Situation, Task, Action, Result).

    CRITICAL: You MUST return ONLY a single raw JSON object with one key: "questions" (an array of 5 objects).
    Example: {{"questions": [{{"question": "Q1", "answer": "STAR answer 1"}}, ...]}}
    """
    # ... (Call API, parse JSON, return dict) ...
```

6.4 Asynchronous Tasks (tasks.py)

resumes/tasks.py

These tasks run in the background to handle slow processes.

```
from celery import shared_task
from .parser import extract_text_from_docx, extract_text_from_pdf, parse_text_with_gemini,
get_full_resume_text, score_and_critique_resume
from .models import Resume, ParsedResumeCache, ResumePDFGeneration
import base64
import tempfile
import os


@shared_task
def parse_resume_task(user_id, filename, file_content_b64):
    """
    Asynchronous task to parse a resume file and store the structured data.
    """
    try:
        file_content = base64.b64decode(file_content_b64)

        with tempfile.NamedTemporaryFile(delete=False, suffix=os.path.splitext(filename)[1]) as
temp_file:
            temp_file.write(file_content)
            temp_file_path = temp_file.name

        text = ""
        if filename.lower().endswith('.pdf'):
            text = extract_text_from_pdf(temp_file_path)
        elif filename.lower().endswith('.docx'):
            text = extract_text_from_docx(temp_file_path)

        if text:
            structured_data = parse_text_with_gemini(text)
            if structured_data:
                ParsedResumeCache.objects.update_or_create(
                    profile_id=user_id,
                    defaults={'parsed_data': structured_data}
                )
    # ... (Error handling and cleanup) ...

@shared_task
def update_resume_score_task(resume_id):
    """
    Asynchronous task to calculate and save the AI score and feedback for a resume.
    """
    try:
        resume = Resume.objects.get(id=resume_id)
        full_resume_text = get_full_resume_text(resume)

        if not full_resume_text.strip() or len(full_resume_text.strip()) < 100:
            resume.score = 0
            resume.feedback = json.dumps(["Resume is too empty to score."])
            resume.save()
            return

        score_data = score_and_critique_resume(full_resume_text)

        if score_data and 'score' in score_data:
            resume.score = score_data.get('score')
            resume.feedback = json.dumps(score_data.get('feedback', []))
            resume.save()
    # ... (Error handling) ...

@shared_task
def generate_resume_pdf_task(pdf_generation_id, base_url):
    """
    Asynchronous task to generate a PDF resume using WeasyPrint.
    """
    try:
        from weasyprint import HTML, CSS
        pdf_gen = ResumePDFGeneration.objects.get(id=pdf_generation_id)
        pdf_gen.status = 'processing'; pdf_gen.save()

        resume = pdf_gen.resume
        context = _get_resume_context(resume) # Helper function to get resume data
        context['accent_color'] = pdf_gen.accent_color # Pass color

        template_path = f'resumes/resume_pdf_{pdf_gen.template_name}.html'
        template = get_template(template_path)
        html = template.render(context)

        html_obj = HTML(string=html, base_url=base_url)
        pdf_bytes = html_obj.write_pdf(...)

        # Save PDF content directly to the database for Railway compatibility
        pdf_gen.pdf_content = pdf_bytes
        pdf_gen.status = 'completed'
        pdf_gen.completed_at = timezone.now()
        pdf_gen.save()
    # ... (Error handling) ...
```

jobs/tasks.py

This task calculates the match score between one resume and one job.

```
from celery import shared_task
from .models import JobPosting, JobMatchScore
from resumes.models import Resume
from resumes.parser import get_full_resume_text
from .matcher import calculate_match_score


@shared_task
def calculate_and_save_match_score_task(resume_id, job_id):
    """
    Asynchronous task to calculate and save the match score
    between a resume and a job posting.
    """
    try:
        resume = Resume.objects.get(id=resume_id)
        job = JobPosting.objects.get(id=job_id)

        resume_text = get_full_resume_text(resume)
        job_text = f"{job.title} {job.description} {job.requirements}"

        score = calculate_match_score(resume_text, job_text)
        JobMatchScore.objects.update_or_create(
            resume=resume,
            job_posting=job,
            defaults={'score': score}
        )
    except (Resume.DoesNotExist, JobPosting.DoesNotExist):
        pass # Handle error
    except Exception as e:
        pass # Handle error
```

6.5 Core Views Logic (views.py)

resumes/views.py

This view handles the AJAX-driven resume builder.

```
@login_required
def resume_builder_view(request):
    try:
        profile = request.user.jobseekerprofile
        resume, created = Resume.objects.get_or_create(
            profile=profile,
            defaults={'title': f"{profile.full_name or request.user.username}'s Resume"}
        )

        if request.method == 'POST' and request.headers.get('x-requested-with') ==
'XMLHttpRequest':
            return handle_ajax_request(request, resume)

        # ... (Prepare context for initial page load) ...
    return render(request, 'resumes/resume_builder.html', context)

def handle_ajax_request(request, resume):
  action = request.POST.get('action')
  model_name = request.POST.get('model_name')
  pk = request.POST.get('pk')

  model_map = { 'experience': (Experience, ExperienceForm), ... }
  Model, Form = model_map[model_name]
  if action == 'add':
    form = Form(request.POST)
    if form.is_valid():
      item = form.save(commit=False); item.resume = resume; item.save()
      resume.score = None; resume.save() # Invalidate score
      transaction.on_commit(lambda: update_resume_score_task.delay(resume.id))
      # ... (Trigger job match recalculations) ...
      item_html = render_to_string(...)
      return JsonResponse({'status': 'success', 'item_html': item_html})
    else:
      return JsonResponse({'status': 'error', 'errors': form.errors}, status=400)

  elif action == 'get_edit_form':
    item = get_object_or_404(Model, pk=pk, resume=resume)
    form = Form(instance=item)
    form_html = render_to_string(...)
    return JsonResponse({'status': 'success', 'form_html': form_html})
    elif action == 'update':
        item = get_object_or_404(Model, pk=pk, resume=resume)
        form = Form(request.POST, request.FILES, instance=item)
        if form.is_valid():
            item = form.save()
            resume.score = None; resume.save() # Invalidate score
            transaction.on_commit(lambda: update_resume_score_task.delay(resume.id))
            # ... (Trigger job match recalculations) ...
            item_html = render_to_string(...)
            return JsonResponse({'status': 'success', 'item_html': item_html, 'pk': pk})
        else:
            return JsonResponse({'status': 'error', 'errors': form.errors}, status=400)

    elif action == 'delete':
        item = get_object_or_404(Model, pk=pk, resume=resume)
        item.delete()
        resume.score = None; resume.save() # Invalidate score
        transaction.on_commit(lambda: update_resume_score_task.delay(resume.id))
        # ... (Trigger job match recalculations) ...
        return JsonResponse({'status': 'success'})

    return JsonResponse({'status': 'error', 'message': 'Invalid action.'}, status=400)
```

jobs/views.py

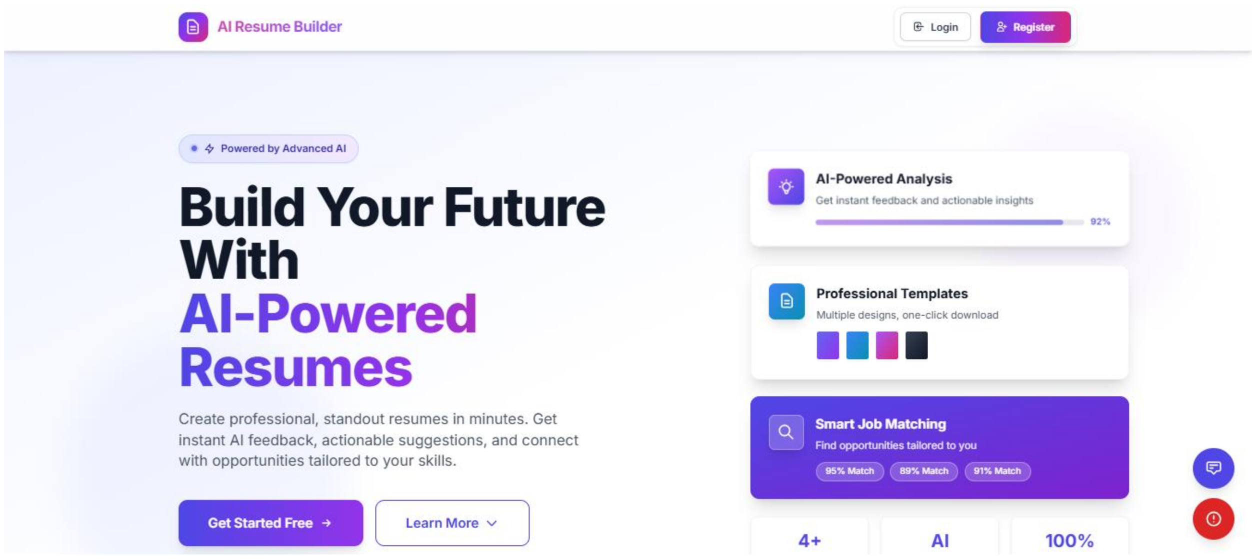
This view shows the job list with AI-powered match scores.

```
@job_feature_disabled
def job_list_view(request):
    jobs = JobPosting.objects.filter(
        Q(application_deadline__gte=timezone.now()) | Q(application_deadline__isnull=True)
    )

    # ... (Search and filter logic) ...

    has_resume = False
    if request.user.is_authenticated and request.user.user_type == 'job_seeker':
        try:
            # Get the user's most recently updated resume
            applicant_resume                                                                  =
Resume.objects.filter(profile=request.user.jobseekerprofile).latest('created_at')
            has_resume = True

            jobs_with_scores = []
            job_ids = [job.id for job in jobs]

            # Fetch all existing scores from the cache in one DB query
            existing_scores = JobMatchScore.objects.filter(
                resume=applicant_resume,
                job_posting_id__in=job_ids
            ).values('job_posting_id', 'score', 'last_calculated')

            scores_map = {item['job_posting_id']: item for item in existing_scores}

            for job in jobs:
                cached_score_data = scores_map.get(job.id)
                # Check if the cache is stale
                is_stale = (not cached_score_data or
                            applicant_resume.updated_at > cached_score_data['last_calculated'] or
                            job.updated_at > cached_score_data['last_calculated'])

                if is_stale:
                    # If stale, trigger a background task to recalculate
                    calculate_and_save_match_score_task.delay(applicant_resume.id, job.id)

                # Show the cached score for now (0 if it doesn't exist)
                score = cached_score_data['score'] if cached_score_data else 0
                jobs_with_scores.append({'job': job, 'score': score})

            return render(request, 'jobs/job_list.html', {
                'jobs_with_scores': jobs_with_scores,
                'has_resume': has_resume
            })

        except (Resume.DoesNotExist, JobSeekerProfile.DoesNotExist):
            pass # Fallback to showing jobs without scores

    return render(request, 'jobs/job_list.html', {
        'jobs': jobs,
        'has_resume': has_resume
    })
```

Chapter 7: Snapshots

This chapter presents screenshots of the final application, illustrating the key features and user flows.

7.1 Public and Authentication



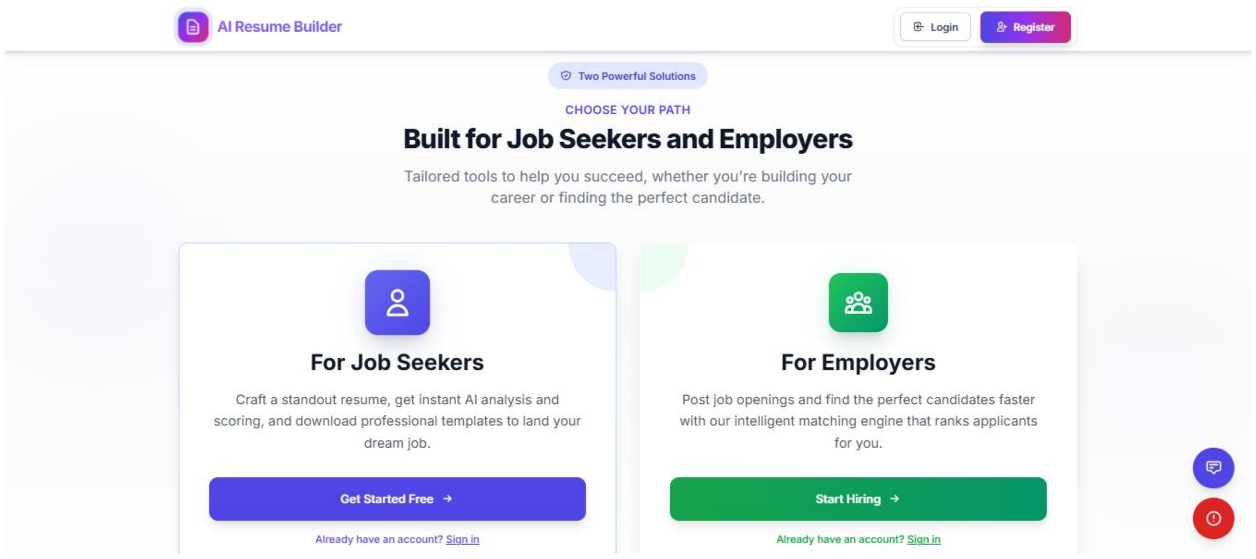
Build Your Future With AI-Powered Resumes

Create professional, standout resumes in minutes. Get instant AI feedback, actionable suggestions, and connect with opportunities tailored to your skills.

AI-Powered Analysis
Get instant feedback and actionable insights

Professional Templates
Multiple designs, one-click download

Smart Job Matching
Find opportunities tailored to you



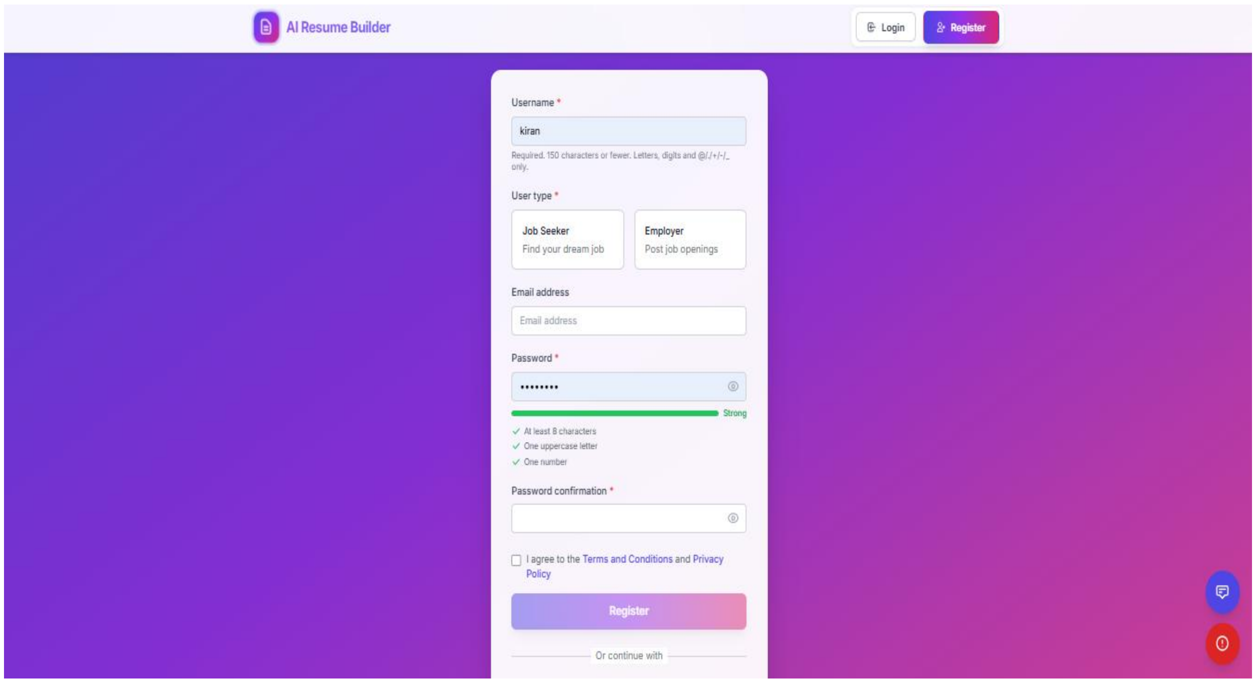
Built for Job Seekers and Employers

Tailored tools to help you succeed, whether you're building your career or finding the perfect candidate.

For Job Seekers
Craft a standout resume, get instant AI analysis and scoring, and download professional templates to land your dream job.
Already have an account? Sign in

For Employers
Post job openings and find the perfect candidates faster with our intelligent matching engine that ranks applicants for you.
Already have an account? Sign in

Figure 7.2: User Registration Page



AI Resume Builder

Login

Register

Username *

kiran

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

User type *

Job Seeker
Find your dream job

Employer
Post job openings

Email address

Email address

Password *

••••••••

Strong

✓ At least 8 characters
✓ One uppercase letter
✓ One number

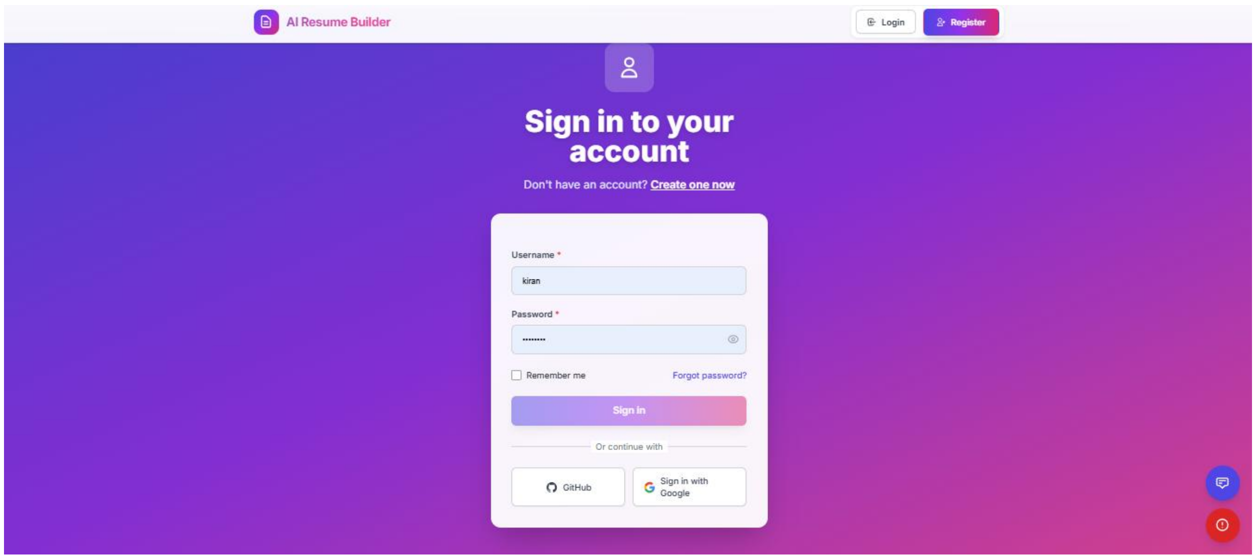
Password confirmation *

☐ I agree to the Terms and Conditions and Privacy Policy

Register

Or continue with

Figure 7.3: User Login Page



AI Resume Builder

Login

Register

Sign in to your account

Don't have an account? Create one now

Username *

kiran

Password *

••••••••

☐ Remember me

Forgot password?

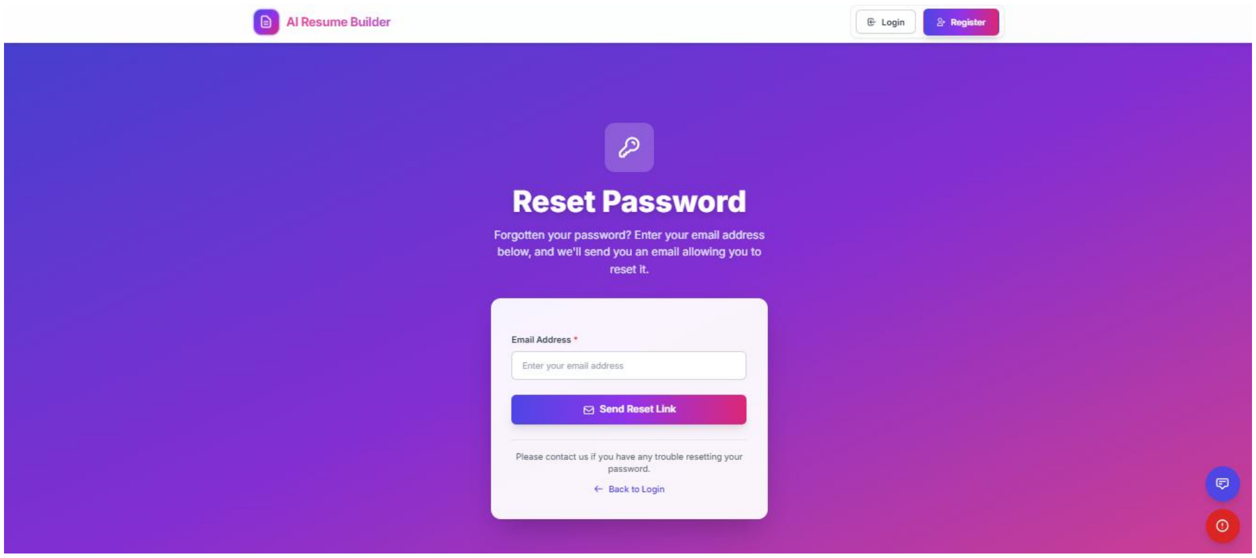
Sign in

Or continue with

GitHub

Sign in with Google

Figure 7.4: Password Reset Page


Reset Password

Forgotten your password? Enter your email address below, and we'll send you an email allowing you to reset it.

Email Address *

Enter your email address

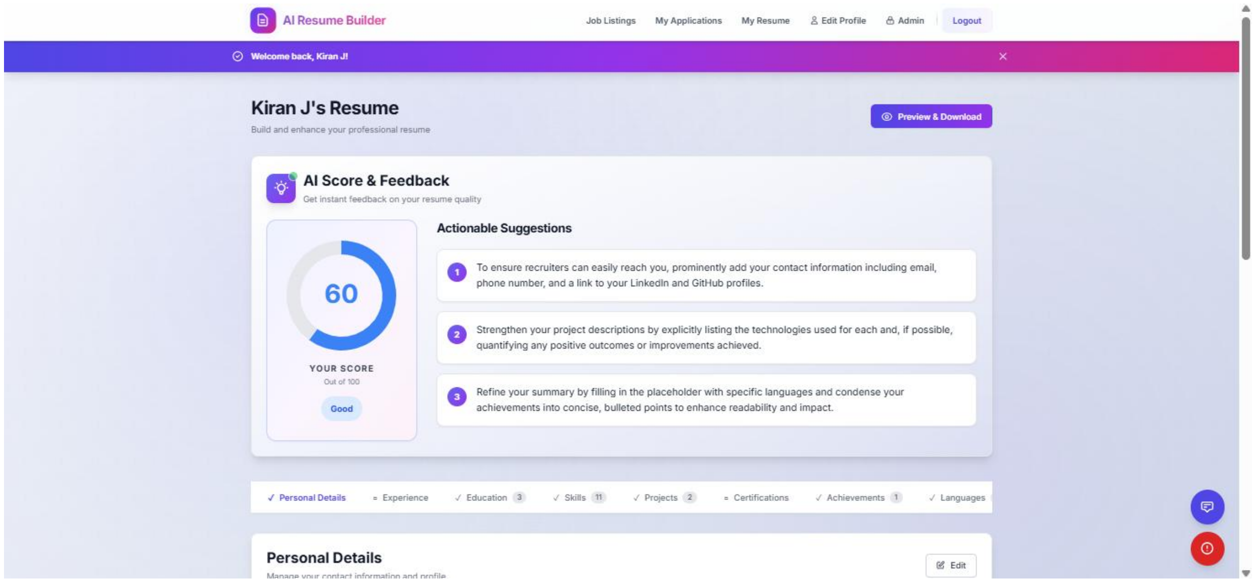
Send Reset Link

Please contact us if you have any trouble resetting your password.

Back to Login

7.2 Job Seeker Flow

Figure 7.5: Job Seeker Dashboard (Home)


Welcome back, Kiran J!

Kiran J's Resume

Build and enhance your professional resume

AI Score & Feedback

Get instant feedback on your resume quality

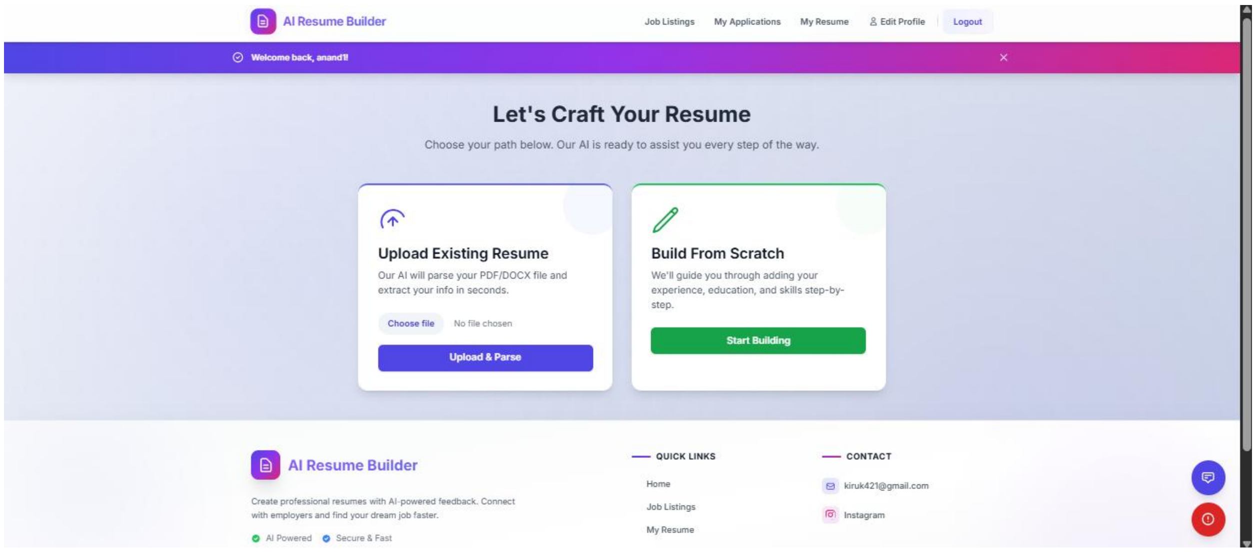
60 YOUR SCORE Out of 100 Good

Actionable Suggestions

1. To ensure recruiters can easily reach you, prominently add your contact information including email, phone number, and a link to your LinkedIn and GitHub profiles.
2. Strengthen your project descriptions by explicitly listing the technologies used for each and, if possible, quantifying any positive outcomes or improvements achieved.
3. Refine your summary by filling in the placeholder with specific languages and condense your achievements into concise, bulleted points to enhance readability and impact.

Personal Details

Figure 7.6: Resume Dashboard



AI Resume Builder   Job Listings   My Applications   My Resume   Edit Profile   Logout

Welcome back, anand!!

Let's Craft Your Resume

Choose your path below. Our AI is ready to assist you every step of the way.

Upload Existing Resume

Our AI will parse your PDF/DOCX file and extract your info in seconds.

Choose file   No file chosen

Upload & Parse

Build From Scratch

We'll guide you through adding your experience, education, and skills step-by-step.

Start Building

AI Resume Builder

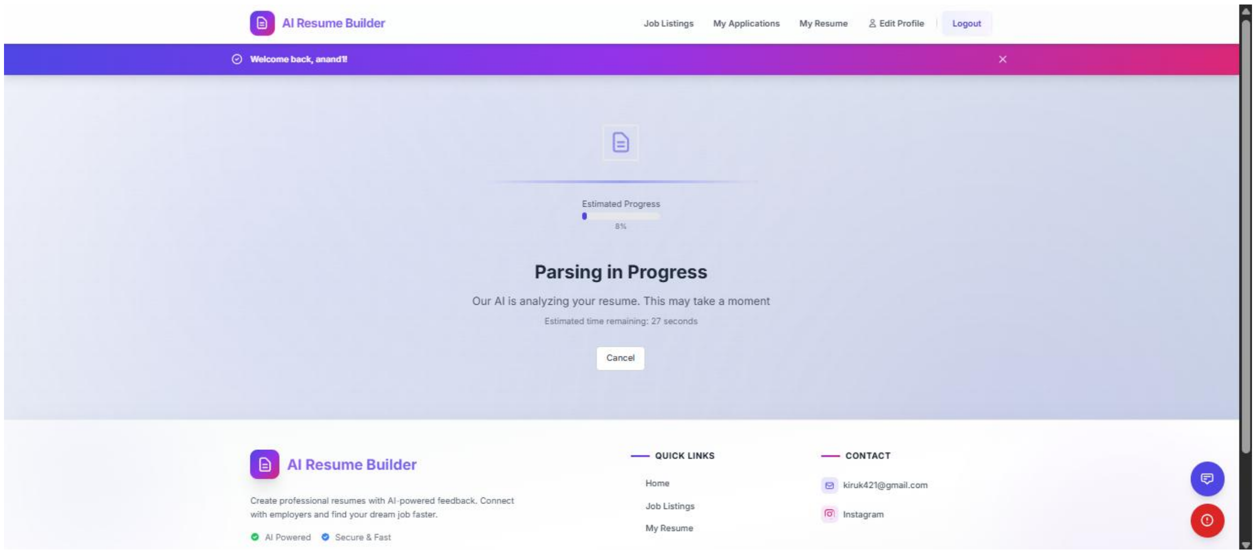
Create professional resumes with AI-powered feedback. Connect with employers and find your dream job faster.

AI Powered   Secure & Fast

QUICK LINKS: Home, Job Listings, My Resume

CONTACT: kiruk421@gmail.com, Instagram

Figure 7.7: AI Parsing in Progress



AI Resume Builder   Job Listings   My Applications   My Resume   Edit Profile   Logout

Welcome back, anand!!

Estimated Progress

8%

Parsing in Progress

Our AI is analyzing your resume. This may take a moment

Estimated time remaining: 27 seconds

Cancel

AI Resume Builder

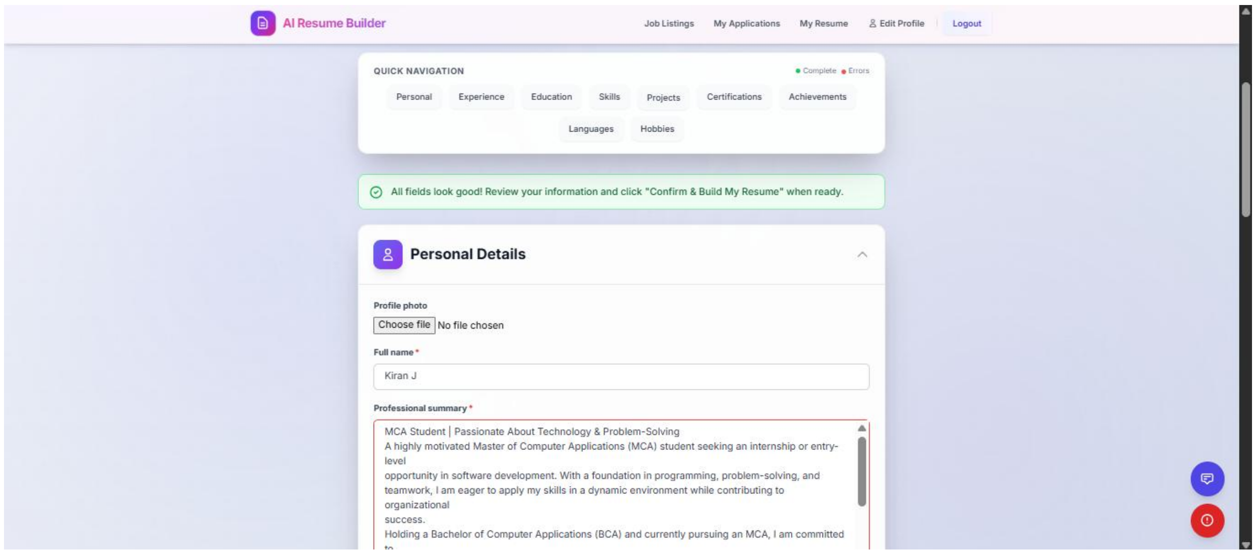
Create professional resumes with AI-powered feedback. Connect with employers and find your dream job faster.

AI Powered   Secure & Fast

QUICK LINKS: Home, Job Listings, My Resume

CONTACT: kiruk421@gmail.com, Instagram

Figure 7.8: Resume Validation Form



All fields look good! Review your information and click "Confirm & Build My Resume" when ready.

Personal Details

Profile photo

Choose file No file chosen

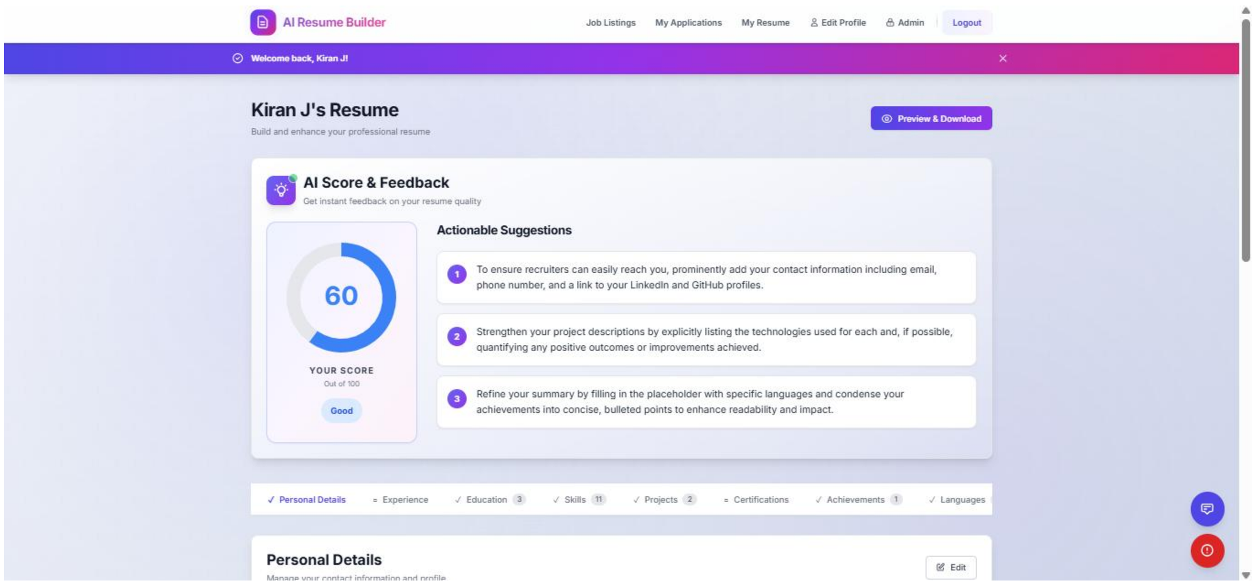
Full name *

Kiran J

Professional summary *

MCA Student | Passionate About Technology & Problem-Solving
A highly motivated Master of Computer Applications (MCA) student seeking an internship or entry-level
opportunity in software development. With a foundation in programming, problem-solving, and teamwork, I am eager to apply my skills in a dynamic environment while contributing to organizational
success.
Holding a Bachelor of Computer Applications (BCA) and currently pursuing an MCA, I am committed

Figure 7.9: Main Resume Builder UI



Welcome back, Kiran J!

Kiran J's Resume

Build and enhance your professional resume

AI Score & Feedback

Get instant feedback on your resume quality

60

YOUR SCORE

Out of 100

Good

Actionable Suggestions

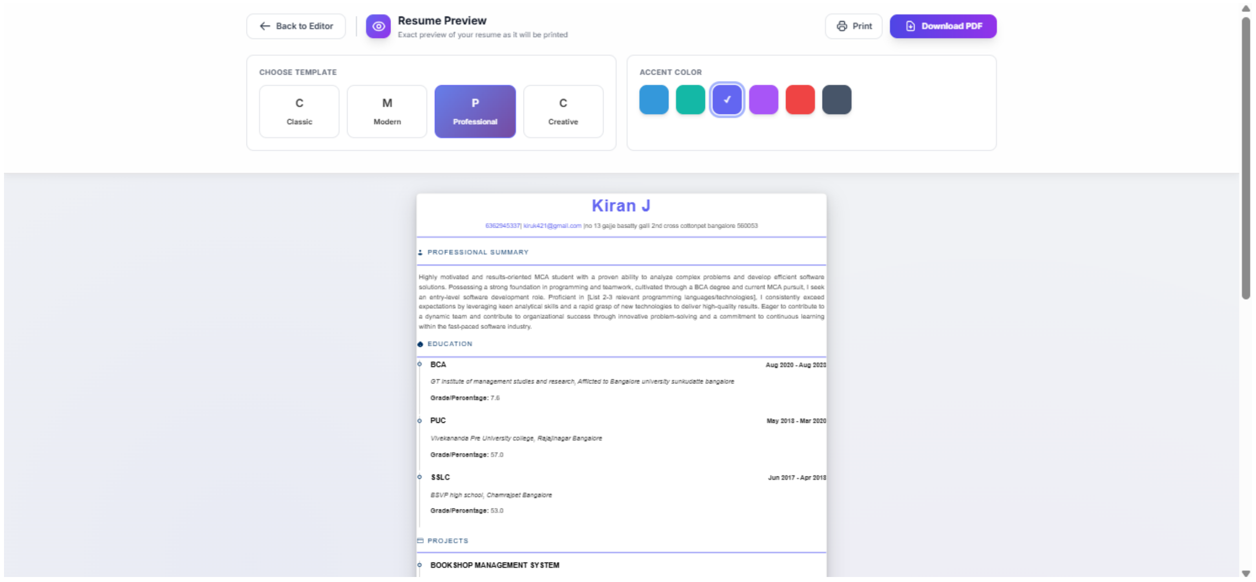
1. To ensure recruiters can easily reach you, prominently add your contact information including email, phone number, and a link to your LinkedIn and GitHub profiles.
2. Strengthen your project descriptions by explicitly listing the technologies used for each and, if possible, quantifying any positive outcomes or improvements achieved.
3. Refine your summary by filling in the placeholder with specific languages and condense your achievements into concise, bulleted points to enhance readability and impact.

✓ Personal Details · Experience · ✓ Education 3 · ✓ Skills 11 · ✓ Projects 2 · Certifications · ✓ Achievements 1 · ✓ Languages

Personal Details

Manage your contact information and profile

Figure 7.10: Resume Preview (Classic Template)



Kiran J

6362945337| kiruk421@gmail.com |no 13 gajje basatty galli 2nd cross cottonpet bangalore 560053

PROFESSIONAL SUMMARY

Highly motivated and results-oriented MCA student with a proven ability to analyze complex problems and develop efficient software solutions. Possessing a strong foundation in programming and teamwork, cultivated through a BCA degree and current MCA pursuit, I seek an entry-level software development role. Proficient in [List 2-3 relevant programming languages/technologies]. I consistently exceed expectations by leveraging keen analytical skills and a rapid grasp of new technologies to deliver high-quality results. Eager to contribute to a dynamic team and contribute to organizational success through innovative problem-solving and a commitment to continuous learning within the fast-paced software industry.

EDUCATION

BCA Aug 2020 - Aug 2023

GT institute of management studies and research, Afflicted to Bangalore university sunkudatte bangalore

Grade/Percentage: 7.6

PUC May 2018 - Mar 2020

Vivekananda Pre University college, Rajajinagar Bangalore

Grade/Percentage: 57.0

SSLC Jun 2017 - Apr 2018

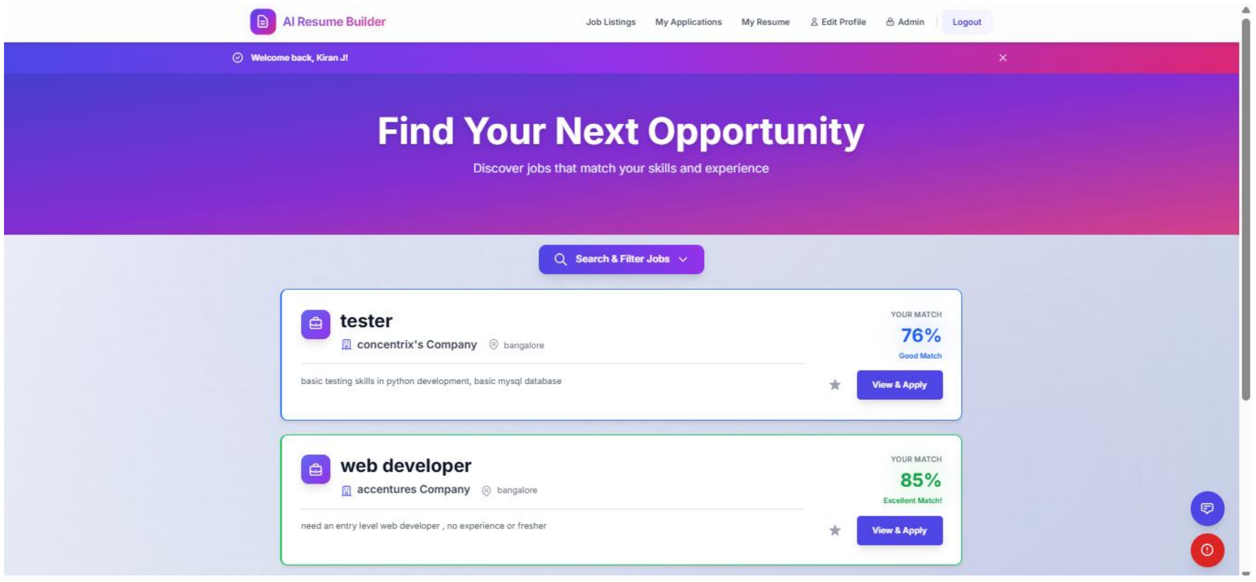
BSVP high school, Chamrajpet Bangalore

Grade/Percentage: 53.0

PROJECTS

BOOKSHOP MANAGEMENT SYSTEM

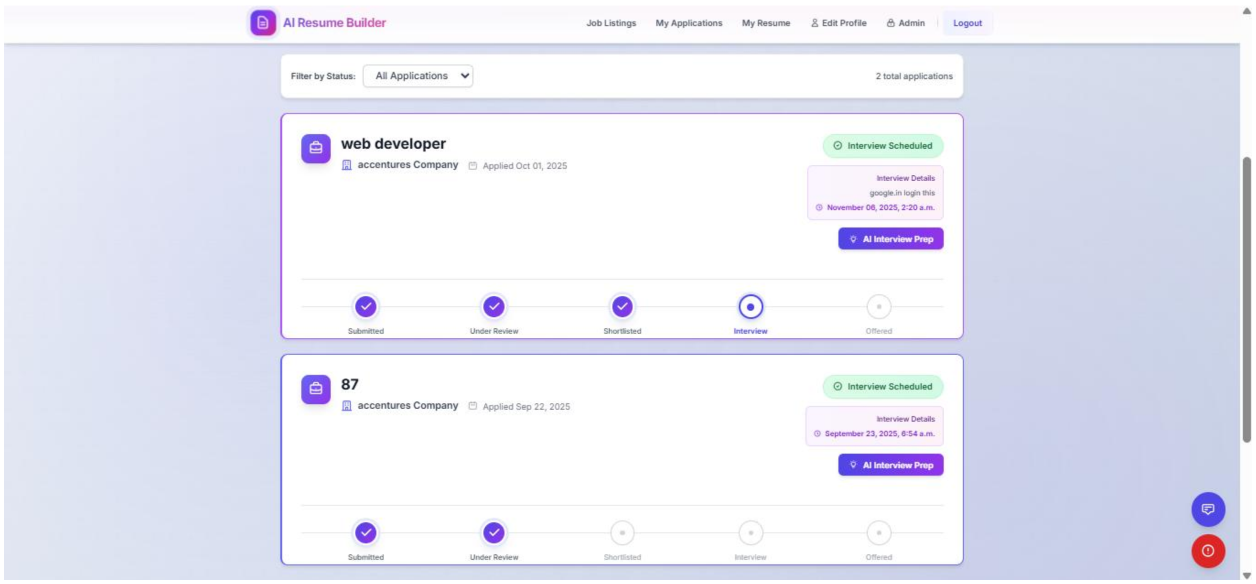
Figure 7.11: AI Job Matching List



tester
concentrix's Company bangalore
basic testing skills in python development, basic mysql database
YOUR MATCH 76% Good Match
View & Apply

web developer
accentures Company bangalore
need an entry level web developer , no experience or fresher
YOUR MATCH 85% Excellent Match!
View & Apply

Figure 7.12: Application Tracking



Filter by Status: All Applications

2 total applications

web developer

accentures Company   Applied Oct 01, 2025

Interview Scheduled

Interview Details
google.in login this
November 06, 2025, 2:20 a.m.

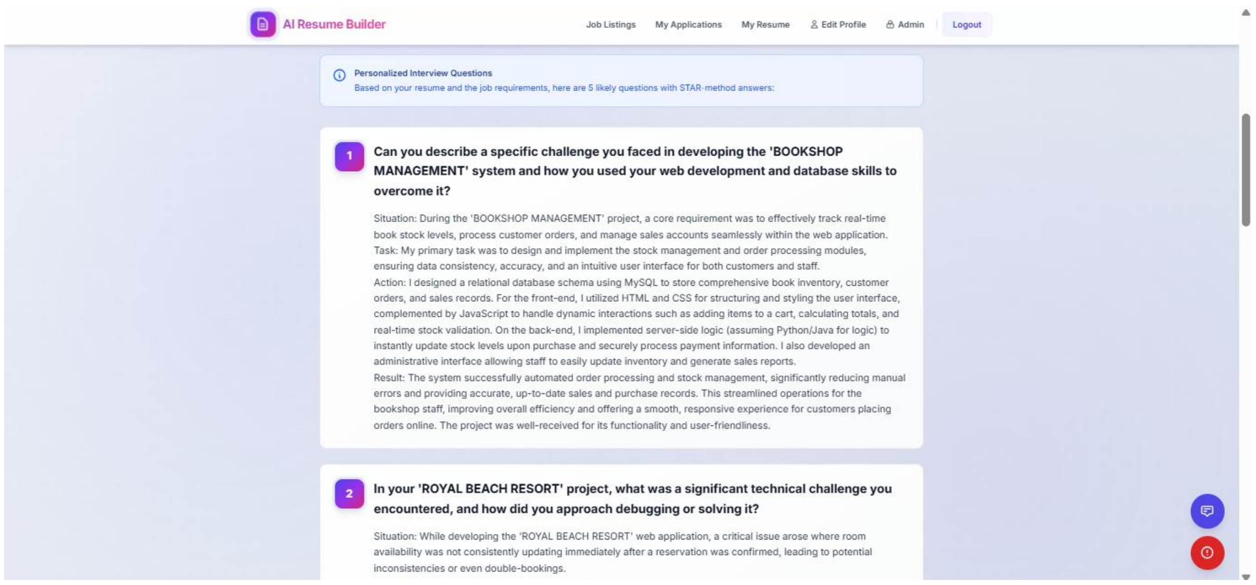
87

accentures Company   Applied Sep 22, 2025

Interview Scheduled

Interview Details
September 23, 2025, 6:54 a.m.

Figure 7.13: AI Interview Preparation Tool



Personalized Interview Questions
Based on your resume and the job requirements, here are 5 likely questions with STAR-method answers:

1 **Can you describe a specific challenge you faced in developing the 'BOOKSHOP MANAGEMENT' system and how you used your web development and database skills to overcome it?**

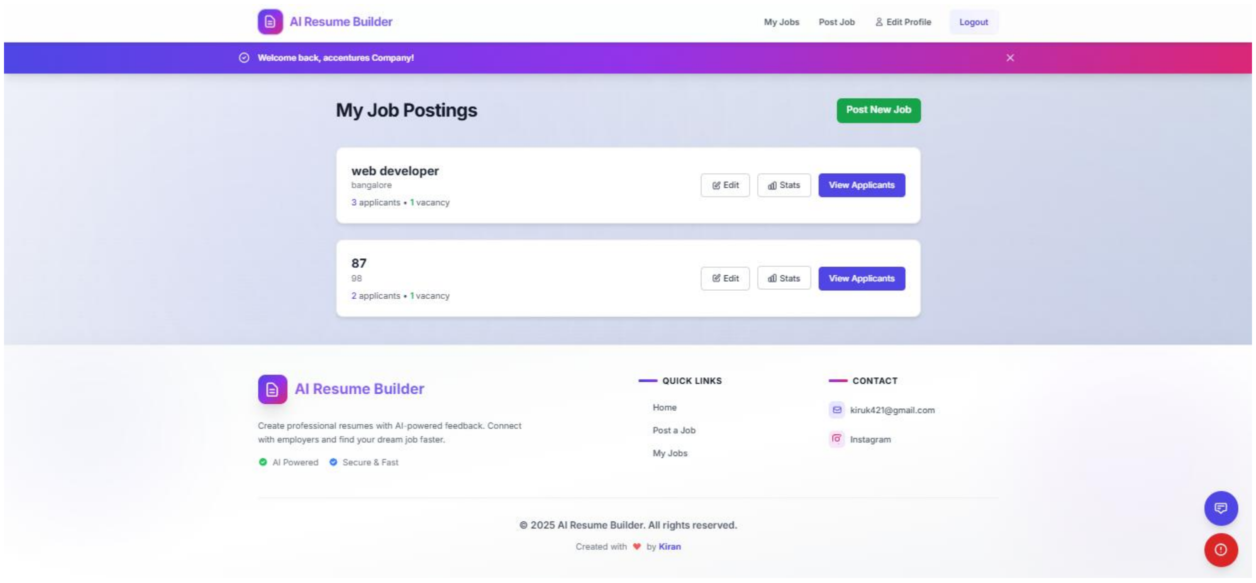
Situation: During the 'BOOKSHOP MANAGEMENT' project, a core requirement was to effectively track real-time book stock levels, process customer orders, and manage sales accounts seamlessly within the web application.
Task: My primary task was to design and implement the stock management and order processing modules, ensuring data consistency, accuracy, and an intuitive user interface for both customers and staff.
Action: I designed a relational database schema using MySQL to store comprehensive book inventory, customer orders, and sales records. For the front-end, I utilized HTML and CSS for structuring and styling the user interface, complemented by JavaScript to handle dynamic interactions such as adding items to a cart, calculating totals, and real-time stock validation. On the back-end, I implemented server-side logic (assuming Python/Java for logic) to instantly update stock levels upon purchase and securely process payment information. I also developed an administrative interface allowing staff to easily update inventory and generate sales reports.
Result: The system successfully automated order processing and stock management, significantly reducing manual errors and providing accurate, up-to-date sales and purchase records. This streamlined operations for the bookshop staff, improving overall efficiency and offering a smooth, responsive experience for customers placing orders online. The project was well-received for its functionality and user-friendliness.

2 **In your 'ROYAL BEACH RESORT' project, what was a significant technical challenge you encountered, and how did you approach debugging or solving it?**

Situation: While developing the 'ROYAL BEACH RESORT' web application, a critical issue arose where room availability was not consistently updating immediately after a reservation was confirmed, leading to potential inconsistencies or even double-bookings.

7.3 Employer Flow

Figure 7.14: Employer Dashboard (Home)



Welcome back, accentures Company!

My Job Postings

web developer
bangalore
3 applicants • 1 vacancy

87
98
2 applicants • 1 vacancy

AI Resume Builder

Create professional resumes with AI-powered feedback. Connect with employers and find your dream job faster.

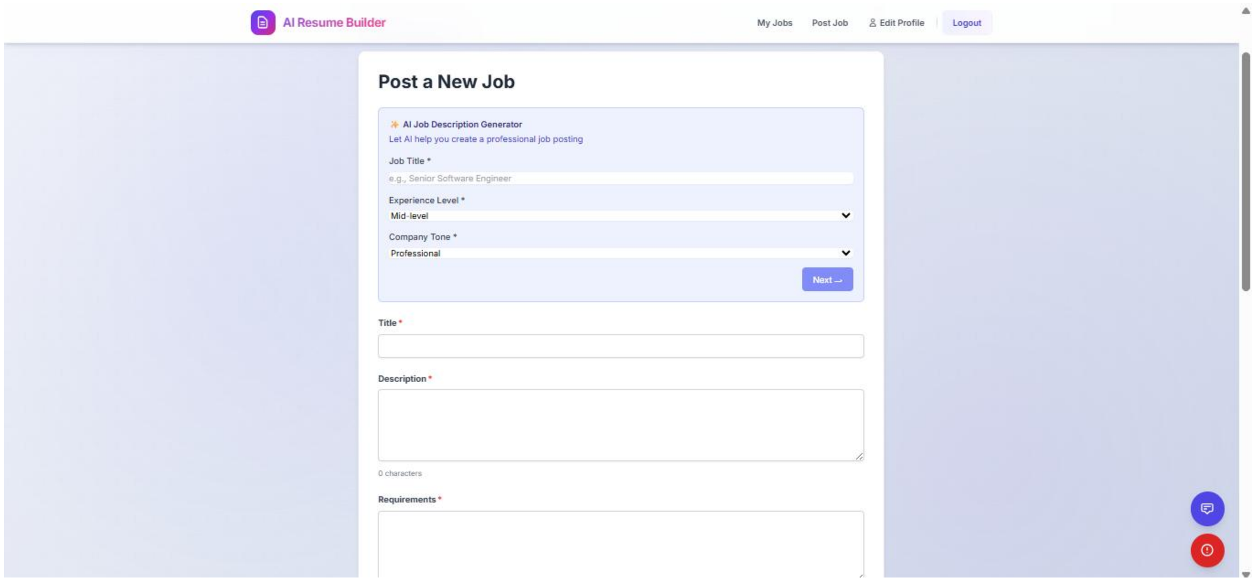
AI Powered   Secure & Fast

kiruk421@gmail.com

© 2025 AI Resume Builder. All rights reserved.

Created with ♥ by Kiran

Figure 7.15: Post Job Page with AI Generator



Post a New Job

AI Job Description Generator
Let AI help you create a professional job posting

Job Title *
e.g., Senior Software Engineer

Experience Level *
Mid-level

Company Tone *
Professional

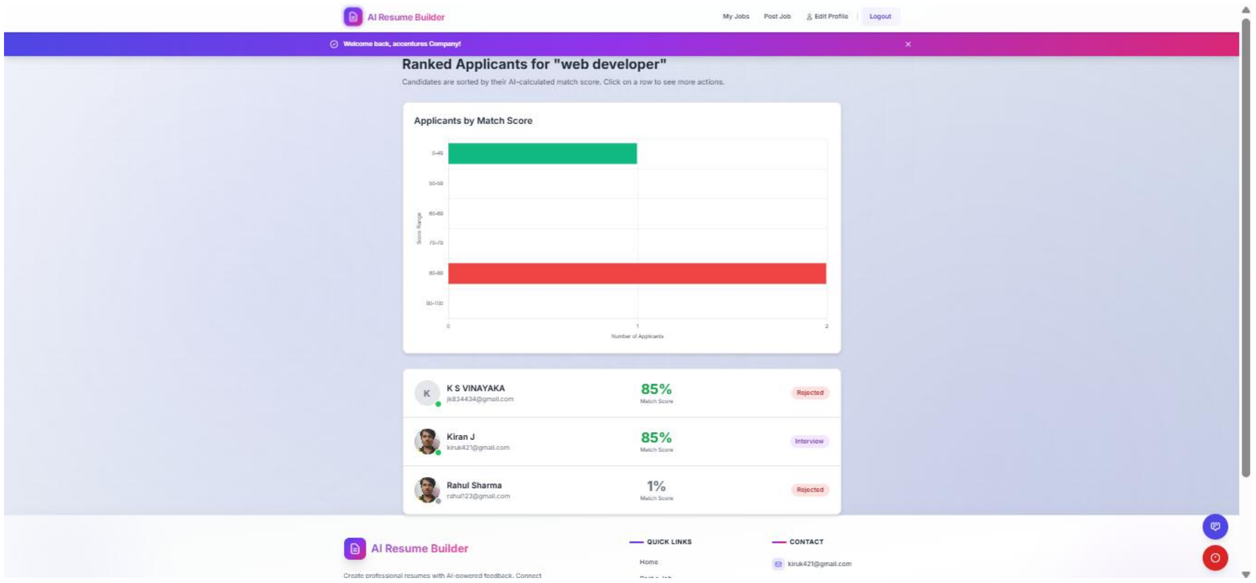
Title *

Description *

0 characters

Requirements *

Figure 7.16: Ranked Applicants List



Ranked Applicants for "web developer"

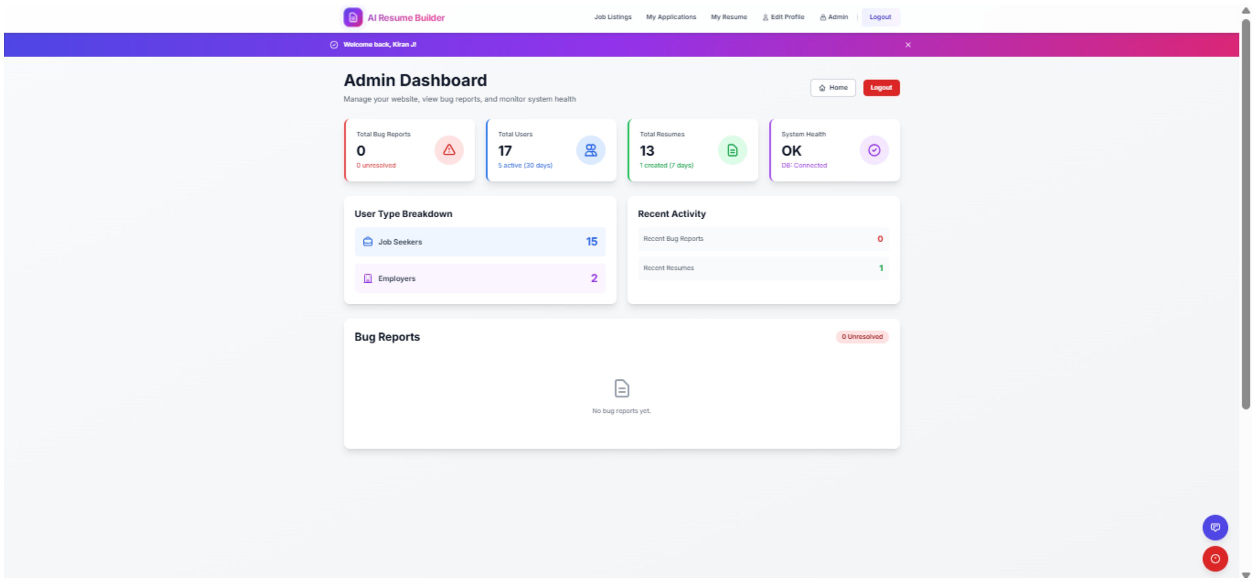
Candidates are sorted by their AI-calculated match score. Click on a row to see more actions.

K S VINAYAKA jk834434@gmail.com	85% Match Score	Rejected
Kiran J kiruk421@gmail.com	85% Match Score	Interview
Rahul Sharma rahul123@gmail.com	1% Match Score	Rejected

Create professional resumes with AI-powered feedback. Connect

7.4 Admin Flow

Figure 7.17: Main Admin Dashboard



Admin Dashboard

Manage your website, view bug reports, and monitor system health

Total Bug Reports	Total Users	Total Resumes	System Health
0	17	13	OK
0 unresolved	5 active (30 days)	1 created (7 days)	DB: Connected

User Type Breakdown

Job Seekers	15
Employers	2

Recent Activity

Recent Bug Reports	0
Recent Resumes	1

Bug Reports — 0 Unresolved

No bug reports yet.

Chapter 8: Conclusion and Future Work

8.1 Conclusion

The **AI Resume Builder** project successfully showcases the design, development, and deployment of a modern, intelligent web application tailored for the HR-Tech domain. By leveraging a robust backend stack—**Django, PostgreSQL, Celery, and Redis**—along with the integration of powerful Generative AI (**Google Gemini**), the system effectively resolves key inefficiencies faced by both job seekers and employers. The project achieves all its core objectives by offering a dual-sided platform, integrating AI across six major features (resume parsing, enhancement, scoring, matching, summarization, and interview preparation), and implementing an asynchronous architecture via Celery to maintain responsiveness during resource-intensive AI operations. Additionally, it ensures a secure, authenticated environment with social login, while delivering tangible value through time savings, improved resume quality, and objective AI-driven candidate-job matching. Overall, the deployed application stands as a scalable, maintainable, and feature-rich platform—serving as a strong proof-of-concept for the future of AI-powered recruitment.

8.2 Future Work

While the current system is fully functional, the platform is designed for extensibility. Future work can be focused on enhancing the AI capabilities and broadening the platform's feature set.

- **Enhanced AI & Matching:**
  - **Real-time Feedback:** Instead of a single score, develop an AI agent that provides interactive, line-by-line suggestions as the user types.
  - **Skill Gap Analysis:** Have the AI suggest specific skills or courses a user should pursue to better match a desired job.
  - **Employer-Side Search:** Allow employers to write a job description and have the AI search the entire database of job seekers, ranking them by match score.

- **Platform Features:**
  - **More Resume Templates:** Expand the library of WeasyPrint templates.
  - **Resume Sharing:** Allow users to generate a unique, shareable link for their resume.

   - **In-Platform Messaging:** Add a chat system for employers and candidates to communicate directly.
   - **Full ATS Integration:** Expand the employer tools to include features like managing interview feedback, sending offer letters, and candidate pipelines.

- **Technical Enhancements:**
   - **WebSocket Integration:** Use Django Channels to provide real-time updates (e.g., "Your resume has finished parsing," "You have a new job match") without needing to poll.
   - **Advanced Analytics:** Build out a more detailed analytics dashboard for employers (e.g., time-to-hire, application drop-off rates).
   - **Mobile Application:** Develop native mobile (iOS/Android) apps that connect to the existing Django backend.

Chapter 9: Bibliography

1. **Django Software Foundation.** (2025). *Django Documentation*. Retrieved from https://www.djangoproject.com/
2. **Google.** (2025). *Google AI for Developers: Gemini API*. Retrieved from https://ai.google.dev/
3. **Redis.** (2025). *Redis Documentation*. Retrieved from https://redis.io/documentation
4. **PostgreSQL Global Development Group.** (2025). *PostgreSQL Documentation*. Retrieved from https://www.postgresql.org/docs/
5. **Kozea.** (2025). *WeasyPrint: The Awesome Document Factory*. Retrieved from https://weasyprint.org/
6. **Tailwind Labs Inc.** (2025). *Tailwind CSS Documentation*. Retrieved from https://tailwindcss.com/
7. **Alpine.js.** (2025). *Alpine.js Documentation*. Retrieved from https://alpinejs.dev/
8. **Python Software Foundation.** (2025). *Python 3.11 Documentation*. Retrieved from https://docs.python.org/3.11/
9. Celery Project. (2025). Celery: Distributed Task Queue. Retrieved from https://docs.celeryq.dev/
10. **Project GitHub Repository.** (2025). *ai-resume-builder by kiran j*. Retrieved from https://github.com/kiruuuuuuu/ai-resume-builder