PROJECTREPORT

ON

JobAi: Automated Job Portal with Auto-Apply

Submitted By

PRUDHWI RAJ KRISHNA V
MGP20NMC043

To

the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of

Integrated Master of Computer Applications

Under the Guidance of

Dr. Akhil Mathew Philip

DEPARTMENT OF COMPUTER APPLICATIONS

Saintgits College of Engineering (Autonomous), Pathamuttom,

Kottayam, Kerala-686532

APRIL 2025

SAINTGITS COLLEGE OF ENGINEERING (AUTONOMOUS)

Kottukulam Hills, Pathamuttom, Kottayam, Kerala

BONAFIDE CERTIFICATE

Certified that the report entitled **"JobAi: Automated Job Portal with Auto-Apply"** submitted by **"PRUDHWI RAJ KRISHNA V, MGP20NMC043"** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Integrated Master of Computer Applications is a bonafide record of the project work carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Akhil Mathew Philip
Internal Guide

Dr. Libin M Joseph
Project Co-Ordinator

Dr. Rajesh K. S
Head of the Department

Dr. Sudha T
Principal

Viva-voce held on: .....

ACKNOWLEDGEMENT

At the outset, I thank the lord almighty for his abundant grace, strength and hope to make our endeavor a success. I express my deep-felt gratitude to Dr. Sudha T., Principal, Saintgits College of Engineering (Autonomous) for her warm support with regard to the work and to the management for providing the facilities required.

I would like to place my deep sense of gratitude to Prof. Mini Punnoose, Director-MCA, Department of Computer Applications, Saintgits College of Engineering (Autonomous) for her constant encouragement. I express my gratitude to Dr. Rajesh K.S, Head of the Department, Department of Computer Applications, Saintgits College of Engineering (Autonomous), for his support and guidance.

I am profoundly grateful to Dr. Akhil Mathew Philip, my project guide and Dr. Libin M Joseph, the project coordinator for their valuable guidance, help, suggestions and assessment.

Furthermore, I would like to thank all others especially my parents and numerous friends. This project report would not have been a success without their inspiration, valuable suggestions and moral support from them throughout its course.

PRUDHWI RAJ KRISHNA V

LLPIN-AAD-7207
GST:32AALFT2481P1ZN
www.tissertechnologies.com

TISSER TECHNOLOGIES
technology that is tailor made...

To,

Date:15/04/2025

Head of the Department

Computer Applications Saintgits

College of Engineering

Dear Sir/Madam,

This is to certify that Mr. PRUDHWI RAJ KRISHNA V, Saintgits College of Engineering, Kottayam has successfully completed an Internship in Software Development using Python Django from 20-01-2025 to 20-04-2025.During the period of internship program with us, He had been exposed to different processes and was found diligent, hardworking and inquisitive.

We wish him all success in his future endeavors.

For Tisser Technologies

ABSTRACT

The "JobAi: Automated Job Portal with Auto-apply" is an innovative platform designed to simplify and enhance the job search experience for fresh graduates and professionals. Leveraging artificial intelligence, the system provides tailored job recommendations based on user-specific parameters such as CGPA, preferred job category, and professional experience. The platform is designed for accessibility and usability, offering secure login for job seekers, recruiters. Job Seekers can upload their resumes, creating comprehensive profiles for potential employers to review. The AI algorithms ensure that relevant job opportunities are presented on user dashboards, streamlining the search process and saving time. The most attractive features that separate this project from others is that it reduces time consumption and avoids wastage of time by automatically search and apply for jobs which is perfect matching with seeker's job preferences in profile and/or Resume. This project has additional modules of auto-apply to jobs which is perfect match to the jobseeker's profile. Furthermore, the system integrates email notifications to keep users informed about job applications and interview schedules, fostering effective communication between job seekers and recruiters. This project demonstrates the potential of AI in transforming traditional job search methodologies into a seamless and personalized experience.

Technologies used: Django, Python, HTML, CSS, JavaScript, Bootstrap, MySQL

CONTENTS

Contents **Page No.**

LIST OF TABLES

Title **Page No**

LIST OF FIGURES

Title **Page No**

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION TO THE PROJECT

In today's fast-paced digital world, job seekers face the challenge of finding suitable job opportunities while competing against a large pool of applicants. The traditional job search process is often time-consuming and requires individuals to manually browse job portals, tailor resumes, and submit applications. To streamline this process, JobAi leverages artificial intelligence (AI) and automation to assist job seekers in discovering relevant opportunities and applying for jobs efficiently.

JobAi is an intelligent job search platform that helps candidates identify, evaluate, and apply for suitable positions based on their qualifications, experience, and career preferences. By incorporating resume parsing, AI-driven job matching, and auto-apply functionality, JobAi significantly reduces the time and effort required for job hunting. The system ensures that users receive personalized job recommendations and enhances their chances of securing employment.

The primary goal of JobAi is to eliminate the repetitive and tedious aspects of job searching by providing a fully automated job application process. Many job seekers struggle with optimizing their resumes, identifying the right opportunities, and applying to multiple jobs efficiently. JobAi addresses these issues by:

- Analyzing uploaded resumes to extract key details such as skills, experience, and qualifications.
- Matching job seekers with the most relevant job postings based on AI-driven algorithms.
- Automatically applying to job listings that align with a candidate’s profile.
- Generating customized cover letters tailored to each job application.
- Providing real-time job notifications and updates on application statuses.

Many job seekers spend countless hours searching for jobs, filling out application forms, and tailoring their resumes for different positions. The sheer volume of job postings makes it challenging for candidates to filter and apply only to the most relevant ones. Additionally, manual applications often lead to human errors, inconsistencies, and delays in the job search process.

The key challenges that JobAi aims to address include:

- The time-consuming nature of job searching and application submission.
- The lack of personalization in job recommendations.
- The difficulty in creating effective resumes and cover letters for different job roles.
- The missed opportunities due to delayed applications or lack of job alerts.

1.2 ORGANIZATION PROFILE

Tisser Technologies LLP is a prominent web and software development company headquartered in Kottayam, Kerala, with a branch office in Nairobi, Kenya. Since its inception in 2007, the company has transformed from a design-focused startup into a full-fledged technology provider offering a wide range of digital solutions. The journey began with modest design services, which gradually laid the foundation for a larger vision. Over the years, Tisser Technologies has built a reputation for innovation, commitment, and customer-centric service, successfully delivering over 6,000 projects to more than 4,000 clients around the globe.

Tisser Technologies offers a comprehensive suite of services that spans design, development, and digital marketing. Its design capabilities include website interfaces, logos, and graphic solutions tailored to enhance brand identity. On the development front, Tisser creates dynamic websites, custom software systems, and mobile applications built for performance and scalability. Marketing services include search engine optimization (SEO), email campaigns, Google Ads, and other digital promotions designed to enhance visibility and online engagement.

In addition to services, Tisser also develops ready-made software products across various domains. Some of its flagship products include pharmacy management systems, clinic and hospital automation software, customized ERP solutions, temple management portals and complete matrimony platforms. These products are crafted to meet the specific demands of industries such as healthcare, retail, education, government, and non-profit organizations.

Tisser's portfolio includes collaboration with well-known brands such as Fedserv, along with numerous startups and enterprises. The company has successfully executed high-level web projects for government bodies and provided online infrastructure to schools and colleges across India. In the non-profit sector, Tisser has developed web applications aimed at expanding outreach and simplifying donation processes.

1.3 OBJECTIVES OF THE PROJECT

JobAi focuses on automating job apply by matching the jobseekers' profile with company's job criteria in jobs posted in the site. It employs AI-driven algorithms to match job postings with

candidate profiles based on skills, qualifications, and experience, ensuring better job compatibility. The system also automates the job application process while limiting applications to 2-3 jobs per session and avoiding duplicate submissions. A comprehensive dashboard is provided to track applied jobs, review matched positions, and manage application statuses effectively. To enhance user engagement, a notification system alerts users about new job matches and tracks application progress. Additionally, some algorithms are leveraged to extract structured resume data, including name, email, skills, and qualifications, and display them in a Django form. Also some algorithms for job matching with the profile and display automated job recommendations based on some criteria in job matching prioritizing highest job preference and give more priority to nearest last date to apply first.

1.4 PURPOSE, SCOPE, AND APPLICABILITY OF THE PROJECT

The primary purpose of JobAi is to create a seamless and efficient job search and application process by providing an AI-driven platform where companies can post job listings and candidates can apply directly. The system minimizes manual effort, ensures relevant job matches, and prevents duplicate applications, ultimately increasing job search success rates.

JobAi is designed for both job seekers and employers. It allows companies to post job openings, while AI-based algorithms help match candidates based on their skills, qualifications, and experience. The system includes automated application submission with a limit on the number of applications per session, a comprehensive dashboard for tracking applications, and AI-driven resume parsing. Additionally, the system provides an interview self-assessment feature where users can preview recorded videos of their interviews, allowing them to evaluate their posture and presentation, making JobAi a holistic job search and application management tool.

This project is applicable to job seekers looking for a streamlined way to find and apply for jobs efficiently. It also benefits companies seeking a simplified method to post job openings and receive applications from suitable candidates. The AI-driven approach ensures better job-candidate matching, making the platform useful for recruitment agencies, HR professionals, and individual job seekers aiming to optimize their job search strategy.

CHAPTER 2
REQUIREMENTS AND ANALYSIS

2.1 EXISTING SYSTEM

A traditional job portal is an online platform that connects job seekers with employers. The process is mostly manual, where candidates browse job listings and apply by submitting their resumes and other details. These portals typically offer a wide range of functionalities for both employers and job seekers.

Key Components of a Job Portal:

- Job Listings: Employers post job openings with details such as job title, description, required qualifications, location, and salary.
- Search Functionality: Job seekers can search for jobs based on filters such as location, industry, experience level, and salary range.
- Job Application Process:
  - Manual Applications: Job seekers select a job listing and manually upload their resumes and cover letters.
  - Profile Creation: Job seekers create a profile with their personal details, work experience, education, and skills. Their resume is either uploaded or auto-generated from the profile.
  - Application Tracking: Candidates track the status of their applications (e.g., submitted, reviewed, interview, rejected).

2.1.1 LIMITATIONS OF EXISTING SYSTEM

Traditional job search platforms often come with a set of challenges that JobAi seeks to address. Many existing systems require job seekers to sift through endless job listings manually, which can be time-consuming and inefficient. Moreover, job applications typically lack smart filters, meaning candidates might apply for roles that don't align with their skills or experience. This can lead to duplicate applications, which not only wastes time but also diminishes the likelihood of receiving meaningful responses. In addition, most traditional portals don't offer a seamless way for users to track their application progress, leaving candidates in the dark about where they stand. And while some platforms provide interview preparation tools, they often miss the mark by not including a self-assessment feature, preventing candidates from reviewing and refining their performance based on video previews.

2.2 PROPOSED SYSTEM

JobAi aims to revolutionize the traditional job search experience by integrating intelligent features that address common pain points for both job seekers and employers. The system will incorporate AI-driven automation, streamlined workflows, and personalized support to ensure efficiency, relevance, and a more effective job application process. Below is an overview of the Proposed System:

2.2.1 JUSTIFICATION OF PROPOSED SYSTEM

- Traditional job search platforms often require job seekers to manually browse through numerous job listings, leading to an inefficient and time-consuming application process. Many candidates struggle with identifying the most relevant opportunities, as these platforms lack intelligent filtering mechanisms to match job seekers with roles that truly align with their skills and experience. JobAi addresses this challenge by leveraging AI-driven job matching, ensuring that candidates receive personalized job recommendations based on their profiles, thereby significantly reducing the time spent searching for jobs. This approach enhances the overall efficiency of the job search process, making it more targeted and productive.
- Another major limitation of existing job portals is the lack of an integrated system for tracking applications. Candidates often apply to multiple jobs but have no centralized way to monitor their progress, leading to confusion and missed opportunities. JobAi introduces a real-time application tracking system that keeps job seekers informed about the status of their applications at every stage. Additionally, by preventing duplicate applications, the system ensures that candidates do not waste time applying for the same job multiple times, improving the overall job search experience while helping employers manage applications more effectively.
- Furthermore, while some job portals offer interview preparation tools, they typically lack self-assessment features that enable candidates to review and refine their performance. JobAi fills this gap by integrating AI-powered mock interviews and video analysis, allowing users to record and assess their responses. This feature provides valuable feedback on speech clarity, body language, and confidence levels, helping candidates improve their interview skills before facing real employers. By incorporating this self-improvement mechanism, JobAi empowers job seekers to present themselves more professionally and increase their chances of securing job offers.
- In addition to benefiting job seekers, JobAi enhances the hiring process for employers by ensuring that only well-matched candidates apply for their job openings. With intelligent application filtering,

recruiters receive applications that are more relevant to their job requirements, reducing the time spent reviewing unqualified applicants. The system's automation also facilitates smoother communication between candidates and employers, ensuring timely responses and follow-ups. By addressing inefficiencies on both sides of the hiring process, JobAi streamlines recruitment and enhances the overall experience for job seekers and employers alike.

- Overall, the proposed JobAi system introduces a smarter, more efficient, and user-friendly job search experience. By eliminating redundant tasks, improving job-candidate matching, and offering advanced self-assessment tools, JobAi significantly enhances the job search process. This innovation not only saves time for job seekers but also increases their chances of landing the right job while helping employers connect with qualified candidates more effectively.

2.2.2 BENEFITS OF PROPOSED SYSTEM

1. Efficiency and Time-Saving

- Job seekers no longer have to spend hours scrolling through irrelevant job listings. The AI-driven recommendations ensure they only see positions that match their qualifications and preferences, making the job search process faster and more efficient.

2. Improved Matching Accuracy

- With smart filters and AI-driven recommendations, candidates are more likely to apply for jobs that align with their skill sets, leading to better-fit job placements and reducing the chances of rejection due to mismatched qualifications.

3. Enhanced Application Tracking

- Candidates gain real-time insights into the status of their applications, reducing anxiety and keeping them informed throughout the hiring process.

4. Self-Improvement Tools for Interviews

- The self-assessment and AI-driven feedback features offer candidates a unique opportunity to refine their interview skills before engaging with potential employers. This leads to better performance and increased chances of securing the job.

5. Reduced Redundancy

- The system's duplicate detection feature ensures that candidates don't waste time applying to the same job multiple times, while also reducing employer frustration over receiving repeated applications.

6. A Seamless Experience

- JobAi integrates smoothly with existing job portals, providing a unified, efficient experience for jobseekers without requiring them to abandon other platforms they are accustomed to using.

2.3 FEASIBILITY STUDY

A feasibility study is essential in assessing the practicality and viability of implementing JobAi as an AI-driven job search and application management platform. This study evaluates JobAi's feasibility across four key areas: technical feasibility, economic feasibility, operational feasibility, and legal/social feasibility. By analyzing these aspects, we can determine the project's sustainability and the potential impact it will have on job seekers and employers.

2.3.1 Technical Feasibility

The technical feasibility of JobAi assesses whether the required technology, infrastructure, and expertise are available to develop and maintain the system effectively. JobAi is built using modern web technologies, including Django for the backend, React or Vue.js for the frontend, and AI-based algorithms for job matching and interview analysis. These technologies are widely used and supported, making implementation realistic.

AI-driven job matching requires natural language processing (NLP) and machine learning models to analyze resumes and job descriptions. spaCy can be used to extract relevant skills and qualifications. These tools are well-documented, open-source, and scalable, ensuring that JobAi's AI features are both practical and efficient.

Furthermore, the platform needs cloud-based hosting for scalability and performance. Cloud solutions like AWS, Google Cloud, or Microsoft Azure offer reliable storage, database management, and computing power to support JobAi's AI-driven functionalities. Given the

advancements in cloud computing, integrating AI-based recommendation engines and real-time application tracking is technically achievable.

Overall, JobAi is technically feasible because the required technologies, programming frameworks, and cloud services are readily available, scalable, and cost-effective.

2.3.2 Operational Feasibility

· Operational feasibility assesses how well JobAi aligns with the needs of job seekers and employers, as well as the ease of implementation and user adoption. Job seekers often struggle with job search inefficiencies, application tracking, and interview preparation, while employers face challenges in screening large volumes of applicants and reducing hiring time. JobAi is designed to address these issues by offering:

· AI-powered job recommendations, ensuring candidates only apply to relevant opportunities.

· Real-time application tracking, preventing confusion about application statuses.

· Automated interview coaching, allowing job seekers to refine their skills.

· Employer-side candidate filtering, reducing the burden of manual resume screening.

· The user interface is designed to be intuitive, mobile-friendly, and easy to navigate, ensuring that users with minimal technical skills can leverage its features. Job seekers can seamlessly upload their resumes, apply to jobs, and receive feedback, while employers can manage job postings and filter applicants without extensive training.

· To ensure operational success, JobAi requires marketing and onboarding strategies such as:

· Integration with existing job portals to attract initial users.

· Partnerships with universities and career services to promote AI-driven job search tools.

· Freemium model adoption, allowing users to experience the benefits before committing to paid services.

· Since JobAi aligns well with industry demands and offers tangible improvements over traditional job search methods, it is operationally feasible and has a high potential for user adoption.

2.3.3 Economic Feasibility

- Economic feasibility examines whether JobAi is financially viable in terms of development, deployment, and maintenance costs compared to the expected benefits. Developing JobAi requires investment in AI model training, database management, web hosting, and frontend/backend development. These costs can be controlled by leveraging open-source AI tools and cloud services with flexible pricing models.

- A cost-benefit analysis suggests that JobAi can generate revenue through multiple channels:

- Freemium Model: Job seekers can use basic job search features for free, while premium users can access AI-powered job matching, resume enhancement, and interview coaching.

- Subscription Plans for Employers: Companies can pay for premium job postings, automated candidate screening, and advanced applicant tracking.

- Advertisement Revenue: Sponsored job listings and employer branding options can provide an additional revenue stream.

- API Integrations: JobAi can charge job portals for integrating its AI-powered matching system into their platforms.

- The demand for AI-driven recruitment solutions is growing, and job seekers are increasingly willing to invest in tools that enhance their chances of securing employment. By reducing time-to-hire for companies and improving the success rate of job seekers, JobAi can provide significant cost savings and efficiency gains for both parties. Given these revenue opportunities, the project is economically feasible and has the potential for long-term profitability.

2.3.4 Scheduling Feasibility:

- Project Timeline: Develop a detailed project timeline with milestones and deadlines for each phase of development, testing, deployment, and maintenance.
- Resource Allocation: Allocate human and financial resources efficiently to meet project deadlines and milestones within the specified budget constraints.
- Risk Management: Identify potential risks and challenges that may impact the project schedule, such as technical issues, regulatory hurdles, or resource constraints, and develop contingency plans to mitigate this risk.

2.4 PLANNING AND SCHEDULING

GANTT CHART

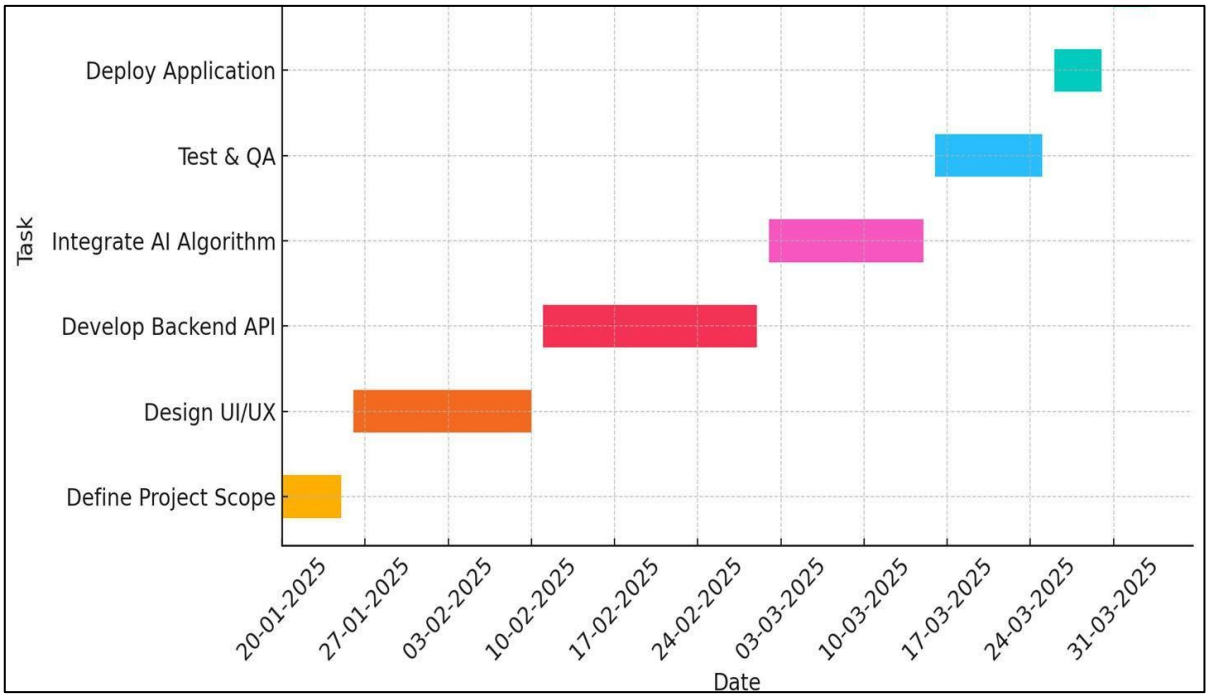

Fig. 2.4.1 Gantt Chart

CHAPTER 3
SYSTEM SPECIFICATION

3.1 SOFTWARE AND HARDWARE SPECIFICATION FOR DEVELOPMENT, IMPLEMENTATION

3.1.1 Hardware Specifications

The hardware configuration of the system on which the package was developed is as follows.

- Processor : Intel core i5 architecture
- Ram : 8GB or more
- Hard Disk : upto 1TB

3.1.2 Software Specifications

The package has been developed using the following specification.

- Operating system : All operating system
- Server : MySQL
- Web Browser : Almost on all web browsers
- Server-side scripting language : Python Django
- Client-side scripting language : Html, CSS, Bootstrap, JavaScript
- Tools : Visual Studio Code, OpenAI API

3.2 TOOLS AND PLATFORMS USED

"JobAi: Automated Job Portal with Auto-apply" is designed with broad accessibility in mind, ensuring compatibility across various platforms and tools. It operates seamlessly on all operating systems and can be hosted using MySQL for server-side database management. Users can access the platform via almost any web browser, facilitating easy adoption. Built with Python Django for server-side scripting and

HTML/JavaScript for client-side scripting, "JobAi: Automated Job Portal with Auto-apply" offers a robust and dynamic user experience. Development and management are streamlined through Visual Studio Code, ensuring efficient collaboration and code management. This comprehensive approach ensures that "JobAi: Automated Job Portal with Auto-apply" is not only accessible but also user-friendly, enabling healthcare professionals and patients alike to engage with the platform effortlessly.

3.2.1 Microsoft Visual Studio Code

Microsoft Visual Studio Code, commonly referred to as VS Code, is a versatile and powerful source code editor developed by Microsoft. Built on open- source technologies, VS Code is renowned for its extensive feature set, lightweight design, and robust performance, making it a preferred choice among developers across various programming languages and platforms. With its intuitive user interface and customizable layout, VS Code offers a seamless coding experience, empowering developers to write, debug, and deploy code with ease.

One of the standout features of Visual Studio Code is its vast ecosystem of extensions, which enhances its functionality and adaptability to different workflows and programming languages. From syntax highlighting and code completion to debugging tools and version control integration, the rich library of extensions allows developers to tailor VS Code to their specific needs and preferences. Furthermore, VS Code's seamless integration with other Microsoft products, such as Azure and GitHub, provides developers with a comprehensive development environment that streamlines the entire software development lifecycle. Whether working on personal projects or collaborating with teams, Microsoft Visual Studio Code stands as a versatile and indispensable tool for modern software development.

3.2.2 Python 3.10

Python is a versatile and powerful programming language known for its simplicity, readability, and flexibility. Developed in the late 1980s by Guido van Rossum, Python has since gained widespread popularity among developers, educators, scientists, and professionals across various domains. Its elegant syntax and easy-to-understand code structure make it an ideal choice for beginners learning to code, while its extensive libraries and frameworks cater to the needs of advanced developers tackling complex projects

Python's versatility extends to the realm of machine learning, where it serves as a primary

language for building and training sophisticated models. With libraries such as TensorFlow, PyTorch, and scikit-learn, Python empowers developers and data scientists to create powerful machine learning algorithms and models with ease. One of Python's strengths in machine learning lies in its rich ecosystem of libraries and frameworks designed specifically for this purpose. TensorFlow, developed by Google, and PyTorch, maintained by Facebook's AI Research lab, are two of the most popular deep learning frameworks that offer high-level abstractions for building neural networks. These frameworks provide efficient computation on both CPUs and GPUs, allowing developers to train complex models on massive datasets.

Additionally, scikit-learn, a widely-used machine learning library in Python, offers a comprehensive set of tools for classical machine learning algorithms such as linear regression, decision trees, and support vector machines. Its user-friendly API and extensive documentation make it an excellent choice for beginners and seasoned practitioners alike.

Python's popularity in machine learning extends beyond its libraries to its ease of use and readability. Its simple syntax and dynamic typing facilitate rapid prototyping and experimentation, enabling developers to iterate quickly and explore different model architectures and hyperparameters. Furthermore, Python's vibrant community and extensive online resources provide ample support and guidance for developers embarking on their machine learning journey.

Python's versatility and rich ecosystem make it an indispensable tool for machine learning practitioners. Whether you're building deep learning models with TensorFlow and PyTorch or exploring classical machine learning algorithms with scikit-learn, Python offers the flexibility and power needed to tackle a wide range of machine learning tasks effectively. Its simplicity, readability, and extensive support for libraries and frameworks make Python the language of choice for machine learning development.

3.2.3 Django 5.1.5

Django is a high-level web framework for building dynamic web applications using the Python programming language. It follows the Model-View- Controller (MVC) architectural pattern, emphasizing rapid development, scalability, and reusability of components. Django provides a robust set of features and tools that streamline the development process and promote best practices in web development.

At its core, Django emphasizes DRY (Don't Repeat Yourself) principles, encouraging developers to write clean, concise code by minimizing redundancy and maximizing code reuse. It comes bundled with a built-in administration interface, authentication system, and URL routing mechanism, reducing the need for boilerplate code and simplifying common development tasks.

Django's ORM (Object-Relational Mapping) system abstracts database interactions, allowing developers to define data models using Python classes without directly writing SQL queries. This abstraction layer facilitates database portability and simplifies data manipulation, validation, and querying operations.

Django's templating engine enables the creation of dynamic and reusable HTML templates, providing a clean separation between presentation and business logic. Templates support inheritance, template inheritance, and template tags, making it easy to build complex and maintainable user interfaces.

Furthermore, Django's ecosystem includes a vast array of third-party libraries and packages that extend its functionality and cater to various use cases, such as RESTful API development, user authentication, and content management.

Overall, Django offers a powerful and versatile framework for building web applications, from simple prototypes to complex, scalable projects. Its emphasis on rapid development, code simplicity, and scalability makes it a popular choice among developers for a wide range of web development tasks.

3.2.4 MySQL 8.0

MySQL is an open-source relational database management system (RDBMS) developed by Oracle Corporation. It is widely used for managing databases in various applications, from small websites to large-scale enterprise systems. MySQL stores data in tables that are linked by predefined relationships, allowing users to perform complex queries and operations on large sets of structured data.

MySQL is a robust and widely used relational database management system (RDBMS) that ensures efficient storage, retrieval, and management of data in JobAI. It is particularly effective in handling structured data and complex queries, which makes it ideal for an automated job portal like JobAI. Below, I'll outline the specific ways MySQL will be used in the JobAI project:

The database for JobAI will consist of several tables designed to store key information about users, job listings, applications, and related data. The relational nature of MySQL allows for normalization of data, reducing redundancy and ensuring data integrity.

MySQL follows the client-server model, where the MySQL server handles database operations and the client interacts with the server to query and manipulate data.

3.2.5 JavaScript

JavaScript is a powerful client-side scripting language widely used to create interactive and dynamic web applications. In the JobAI project, JavaScript is primarily utilized to manage real-time user interactions, such as handling quiz logic in the mock interview module, managing video recording using the Media Recorder API, and controlling modal windows for displaying questions or video previews. It enables seamless communication between the user interface and underlying processes without requiring page reloads, thereby enhancing responsiveness. JavaScript interacts with HTML and CSS through DOM manipulation, allowing the system to update content dynamically based on user input or system events. Additionally, its asynchronous capabilities through promises and async/await make it ideal for handling time-based actions, like countdown timers during assessments or delayed feedback. Overall, JavaScript contributes significantly to making the JobAI system user-friendly, efficient, and suitable for real-time web-based interactions.

3.2.6 HTML

HTML (Hyper Text Markup Language) is the foundational technology used to create and structure the web pages in the JobAI system. HTML plays a central role in defining the layout, structure, and content of all interfaces without relying on any server-side rendering. It organizes key sections of the application, such as the registration and login forms, job seeker dashboard, company portal, job listings, mock interview modules, and resume upload interfaces. Using semantic elements like <header>, <nav>, <section>, and <form>, the structure is made clear and accessible, ensuring a smooth and intuitive user experience across different devices and screen sizes.

HTML forms are crucial for capturing user input, including profile details, resume files, quiz responses, and job preferences. These inputs are handled entirely on the client side, with JavaScript validating and processing the data in real time. Interactive components like buttons, dropdowns, file inputs, and modals are all structured using HTML, allowing users to seamlessly navigate and interact with the system. Video recording during mock interviews, quiz modals, and application tracking features are integrated into the HTML layout to maintain consistency and ease of use.

Overall, HTML forms the backbone of the JobAI interface, providing a clear, organized, and user-friendly framework for all front-end interactions.

3.2.7 CSS

CSS (Cascading Style Sheets) is an essential technology in the JobAI system, responsible for defining the visual styling and layout of the application's user interface. Since JobAI is built entirely on the client side, CSS plays a vital role in transforming static HTML structures into an interactive, modern, and responsive web interface. It controls key visual aspects such as spacing, colors, typography, borders, hover effects, animations, and element positioning, ensuring a consistent and intuitive user experience across all modules—including job seeker dashboards, company pages, job listings, quizzes, and interview preview windows.

To accelerate development and maintain a cohesive design system, the JobAI project also integrates Bootstrap, a popular CSS framework. Bootstrap provides a rich collection of pre-defined components like buttons, modals, cards, forms, navbars, and grid systems, allowing developers to build a responsive layout quickly and efficiently. The framework's responsive utilities and mobile-first design ensure that the platform automatically adjusts to different screen sizes, offering a seamless experience on desktops, tablets, and smartphones without the need for writing custom media queries in most cases.

CSS Flexbox and Grid are used alongside Bootstrap’s grid system to create well-organized, flexible layouts for displaying resumes, job listings, ATS results, and user statistics. Custom styles are applied where necessary to override default Bootstrap components, tailoring the look and feel to match the unique branding and functionality of the JobAI platform. Visual feedback mechanisms such as form validation

highlights, button transitions, and alert boxes enhance usability and make the application more interactive.

By combining custom CSS with Bootstrap's powerful components and responsive utilities, the JobAI system achieves a balance of design flexibility, consistency, and rapid development.

3.2.8 OPENAI API

The OpenAI API is a key component in enhancing the intelligence and automation capabilities of the JobAI platform. It is used to power several advanced features, particularly within the mock interview and quiz modules. By leveraging OpenAI's natural language processing (NLP) capabilities, the system can dynamically generate interview questions tailored to different job roles and difficulty levels. These questions are not hardcoded but are created in real-time using prompts sent to the OpenAI API, making each quiz session unique and contextually relevant. This not only simulates a realistic interview experience but also ensures that users are continuously challenged with new content, improving their preparation over time.

Additionally, the OpenAI API is used to evaluate user responses during quizzes. The system can analyze text input from candidates and provide automated feedback or scoring based on relevance, accuracy, and clarity. This intelligent evaluation process adds a layer of personalization and objectivity to the assessment, something that traditional rule-based systems struggle to achieve. Since the JobAI project is built as a client-side application, the interaction with the OpenAI API is securely handled using asynchronous JavaScript methods, ensuring smooth performance and fast responses without relying on a full server-side infrastructure.

By integrating the OpenAI API, the JobAI system transforms from a static job platform into an intelligent career assistant capable of simulating interview scenarios, evaluating answers, and supporting job seekers through smart automation. This significantly elevates the overall functionality, making the system more adaptive, scalable, and aligned with real-world hiring standards.

CHAPTER 4
SYSTEM DESIGN

4.1 MODULE DESCRIPTIONS

The project is divided into 3 modules they are:

- Company Module
- Jobseeker Module
- Interview Module
- ATS Module

4.1.1 Company Module

The Company Module is designed to streamline the hiring process for employers by allowing them to register, post jobs, and automatically process applications based on ATS (Applicant Tracking System) scores. This module enhances recruitment efficiency by leveraging AI to evaluate candidate applications and send automated acceptance or rejection emails, along with automatically generated admit cards for shortlisted candidates.

Employers can register and log in to access a dedicated dashboard where they can manage job postings and track applications. Once registered, they can create job listings, specifying requirements such as qualifications, skills, and experience levels. These job postings are then made available to job seekers who can apply directly through the platform.

When an application is submitted, the system automatically processes it using ATS-based scoring. This evaluation considers factors like keyword matching, skills alignment, and resume quality. Based on the ATS score, the system determines whether the candidate meets the job requirements.

- If the candidate's score meets the threshold set by the employer, they receive an acceptance email along with an automatically generated admit card. This admit-card contains essential details such as interview date, time, venue (or online meeting link), job role, and other relevant instructions.
- If the candidate does not meet the criteria, the system sends a rejection email, ensuring professional and timely communication.

By automating resume screening, email communication, and admit card generation, the Company Module significantly reduces manual effort for recruiters. This feature ensures a faster, more efficient, and structured hiring process while providing candidates with timely updates and essential interview details.

4.1.2 Jobseeker Module:

The Jobseeker Module is designed to provide candidates with a seamless and efficient job search experience by integrating AI-driven job matching, automated applications, and interview preparation tools. This module ensures that job seekers can easily register, find relevant jobs, apply efficiently, and track their application progress, all while receiving AI-powered support to optimize their chances of success.

Candidates begin by registering and logging in, where they can set up their profile, including personal details, resume upload, skills, and job preferences. Once registered, the system extracts key details from the uploaded resume, such as name, email, qualifications, skills, experience, and other relevant information. This resume extraction feature helps in automatically matching candidates with suitable job opportunities based on their profile and preferences.

To make the application process faster and smarter, the module includes an auto-apply feature that submits applications on behalf of the candidate. The system prioritizes applications based on high-preference jobs first, ensuring that job seekers have a better chance at landing their desired roles. To prevent unnecessary spam or redundancy, the auto-apply feature is designed to limit applications to a reasonable number and avoid duplicate applications.

A key aspect of any job application is a strong cover letter. The Jobseeker Module includes AI-powered cover letter generation, where the system automatically creates customized cover letters tailored to each job application. This saves time and helps candidates make a stronger impression on potential employers by highlighting their skills and qualifications in a well-structured and professional manner.

In addition to job applications, the module also includes a Mock Interview Feature, where candidates can take AI-generated mock interviews to practice their responses. The system records the session, allowing candidates to review their video performance after completion. They can preview and download the recorded session for self-assessment, helping them identify areas of improvement in their posture, speaking clarity, and overall confidence.

For job seekers who prefer manual job search, the module allows them to search for jobs based on industry, skills, location, and other filters. It also includes a shortlisting feature, where

candidates can save jobs they are interested in and revisit them later before applying. This helps them organize and prioritize their job hunt effectively.

Once a candidate has applied for jobs, the Application Status Viewer allows them to track the real-time status of their applications, whether they are under review, shortlisted, or rejected. To keep job seekers engaged and informed, the system also sends automated follow-up emails with updates about their job applications. These emails ensure that candidates stay on top of their job search without constantly checking manually.

Overall, the Jobseeker Module is built to make the job search and application process efficient, automated, and highly personalized. By leveraging AI-driven job matching, auto-apply functionality, cover letter generation, interview preparation, and real-time tracking, this module provides job seekers with a comprehensive and streamlined job-hunting experience.

4.1.3 Interview Module:

The Mock Interview Module is designed to help candidates practice and refine their interview skills in a structured and interactive manner. This module integrates AI-generated quizzes, video recording, and self-assessment tools to provide a realistic interview experience. By leveraging the OpenAI API, it creates dynamic interview questions tailored to the job role or industry, ensuring that candidates receive relevant and varied challenges.

During the mock interview session, the OpenAI API generates a set of interview questions based on predefined criteria. Candidates respond to these questions in real time, simulating a real interview scenario. After completing the quiz, OpenAI evaluates their responses, providing insights or scores to help users identify strengths and areas for improvement. This AI-driven evaluation enhances the learning process by offering structured feedback on answers.

To further enhance self-assessment, the module records the candidate's video throughout the session. This allows candidates to review their body language, facial expressions, and speaking clarity. Once the quiz modal is closed, the recorded video is made available for preview, enabling users to reflect on their performance and make necessary adjustments. Unlike real-time posture monitoring, this system empowers candidates to analyze their own posture and engagement at their own pace.

Additionally, candidates have the option to download their recorded mock interview session for future reference. This feature helps track progress over multiple practice sessions, allowing

users to compare past performances and make continuous improvements. By eliminating the need for live interviewers and real-time AI-based posture tracking, this module offers a lightweight and user-friendly solution that focuses on self-improvement.

With its combination of AI-powered quizzes, automated evaluation, and video-based self-review, the Mock Interview Module provides an effective and engaging way for candidates to prepare for real-world interviews. This system ensures a personalized and insightful practice environment, enabling users to build confidence and refine their interview techniques over time.

4.1.4 ATS Module:

The ATS Module is designed to automate and streamline the application screening process, allowing recruiters to efficiently evaluate candidates with minimal manual effort. This system utilizes AI-driven ATS scoring to assess applications based on predefined criteria such as resume keyword matching, skills alignment, and experience level.

In the applicants' details UI, recruiters have access to a table displaying all job applicants. Each row contains relevant candidate information, along with a dedicated column showing the ATS score. This score helps recruiters quickly determine a candidate's suitability for the job.

A key feature of this module is the automated decision-making button. Recruiters can click a button to automate the screening process, which triggers the ATS system to evaluate the application. Once processed, the system replaces the button with either "Accepted" or "Rejected" based on the candidate's ATS score. This eliminates the need for manual selection and provides instant feedback on applicant suitability.

For accepted candidates, the system automatically sends an acceptance email along with an auto-generated admit card containing interview details. If a candidate is rejected, the system sends a polite rejection email, ensuring timely and professional communication.

By automating application evaluation, acceptance/rejection updates, and communication, the ATS Module significantly reduces recruiter workload while ensuring a fair, efficient, and data-driven hiring process.

4.2 SCHEMA DESIGN

Database Name: JobAi

1. Table Name: **Company**

Description : This table is used to store the company details.
Primary key : id

Fields	**Type**	**Description**
Company ID	int	Company ID
Company Logo	varchar	Company Logo
Company Name	varchar	Company Name
Email	varchar	Company's Email
Password	varchar	Password
Company Type	varchar	Type of Company
Location	varchar	Location of Company

Table 4.2.1: Company

2. Table Name: **Jobseeker_registration**

Description : This table is to store patient details.
Primary key : User ID

Fields	**Type**	**Description**
User ID	int	User ID
User Name	varchar	User Name
Email	varchar	Email
Phone	varchar	Phone Number
Password	varchar	Password

Table 4.2.2: Jobseeker Registration

3. Table Name: **Jobseeker Profile**

Description : This table is used to store the doctor details.
Primary key :Profile id

Fields	**Type**	**Description**
Profile ID	int	Profile ID unique
Resume	varchar	Resume uploaded by user
User ID	int	User Id automatic generated
DOB	varchar	Date Of Birth
Image	varchar	Profile image
Email	varchar	Email address of user
Phone No.	varchar	Phone Number
Address	varchar	Address
Highest Qualification	varchar	Highest Qualification
Job Preference(only one)	Varchar	One main Job Preference
Nationality	varchar	Indian/Outside India
Job Location	varchar	Preferred Job Location
University Name	varchar	University name

Table 4.2.3: Jobseeker Profile

4. Table Name: **Company Job List**

Description : This table is used to store the laboratory details.
Primary Key : id

Fields	**Type**	**Description**
ID	int	ID of Company Joblist
Job ID	int	ID of Job
Company ID	int	Company ID
Job Number	int	Unique Number company given for Job
Job Description	varchar	Job Descriptions
Job Position	varchar	Job Position
Eligibility	varchar	Eligibility Criteria
Skills	varchar	Skills Required
Job posted date	varchar	Date when Job Posted
Last Date of Application	varchar	Last Date till which job application accepts

Table 4.2.4: Company Job List

5. Table Name: **Job Titles**

Description : This table is used to store the job titles.
Primary Key : id

Fields	**Type**	**Description**
ID	int	ID of Table
Job Title	varchar	Position of Job

Table 4.2.5: Job Titles

6. Table Name: **Job Applications**

Description : This table is used to store job application details.
Primary key : id

Fields	**Type**	**Description**
id	Integer Field	Unique application ID
jobseeker	Foreign Key (Jobseeker Profile)	Candidate who applied
company_joblist_id	Foreign Key(Company Job Listing)	Job being applied for
applied_at	Date-Time Field	Timestamp of application
ats_score	Integer Field (nullable)	Auto-calculated ATS score
status	Text Field (default: "Applied")	Application status (Applied, Accepted, Rejected)

Table 4.2.6: Job Application

7. Table Name: **Jobseeker Notifications**

Description : This table is used to store the notification details.
Primary Key : id.

Fields	**Type**	**Description**
id	Integer	Unique identifier
jobseeker_profile	Foreign Key (jobseeker_profile)	Links notification to a jobseeker
company_job	Foreign Key (company_joblist)	Links notification to a job
message	Text Field	Notification content
is_read	Boolean Field	False if unread, True if read
created_at	Date-Time Field	Timestamp when notification was created

Table 4.2.7: Job Notification

4.3 PROCEDURAL DESIGN

JobAI is an AI-powered job portal that automates job matching, applications, and interview preparation for job seekers while streamlining the recruitment process for companies using an ATS-driven system.

Job seekers begin by registering and logging into their profiles, where their resumes are automatically extracted to populate key details such as qualifications, skills, and experience. AI-powered job matching suggests suitable opportunities, and an auto-apply feature prioritizes high-preference jobs while preventing duplicate applications. Candidates can also generate automated cover letters tailored to each job. The system provides an application status tracker and sends follow-up notifications regarding job applications.

The Mock Interview Module allows candidates to take AI-generated quizzes while recording their sessions. After the quiz, they can preview and download the video for self-assessment. This helps job seekers refine their posture, speaking clarity, and confidence for real interviews.

For companies, JobAI offers a Company Module where recruiters can register, log in, and post jobs. Applications are automatically processed using ATS scoring, evaluating resumes based on skill matching and experience. Recruiters can process applications at the click of a button, after which the system replaces the button with "Accepted" or "Rejected" based on the ATS score. Accepted candidates receive an automated email with an admit card, while rejected candidates receive a polite rejection email.

Additionally, job notifications keep candidates informed about new opportunities, interviews, and application updates. The system ensures an efficient, automated, and data-driven hiring process, benefiting both job seekers and employers.

.

4.3.1 . Flow Charts and Block Diagrams

- Flowchart

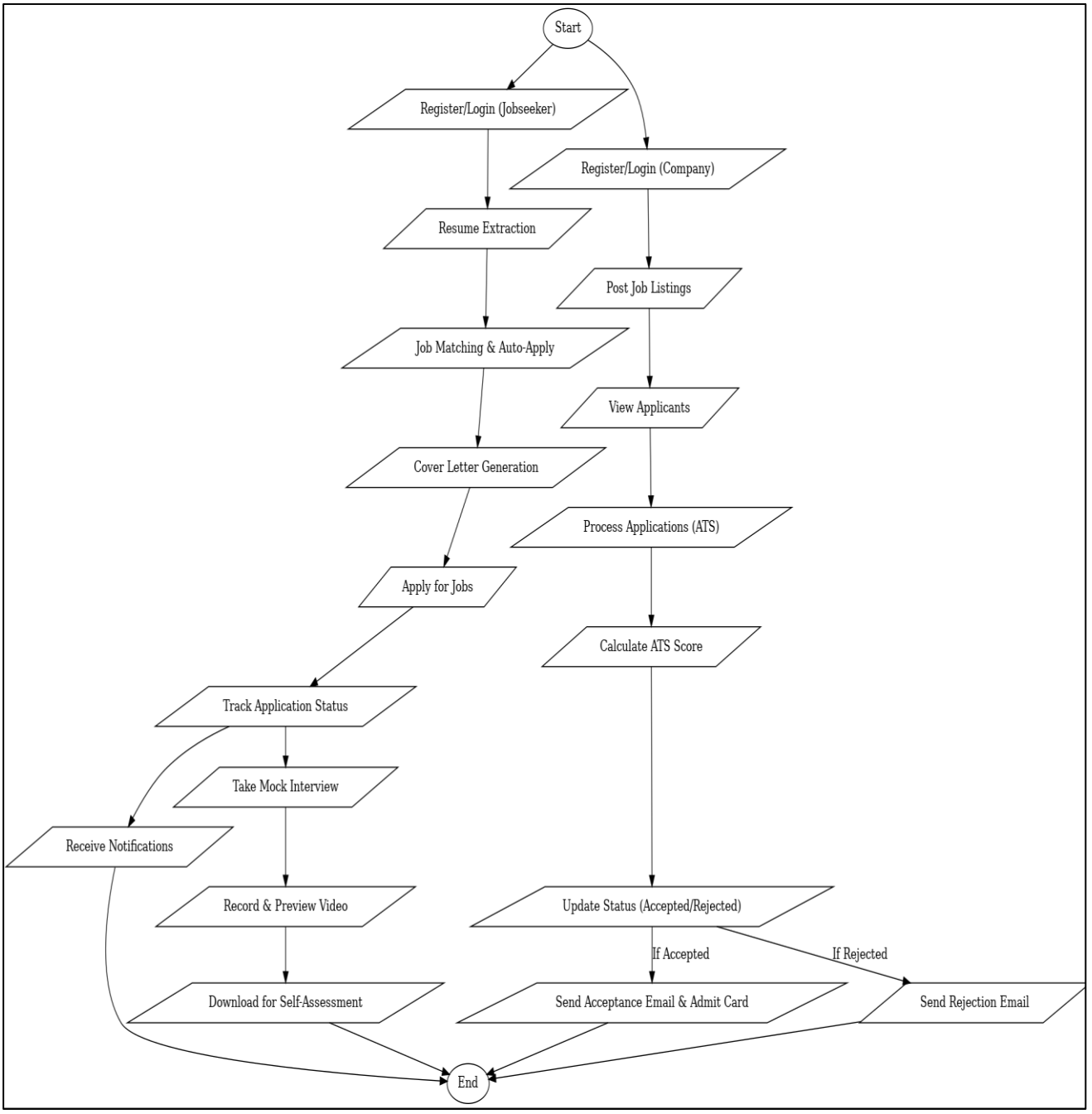

Fig 4.3.1:Flowchart

4.3.2 Use Case Diagram

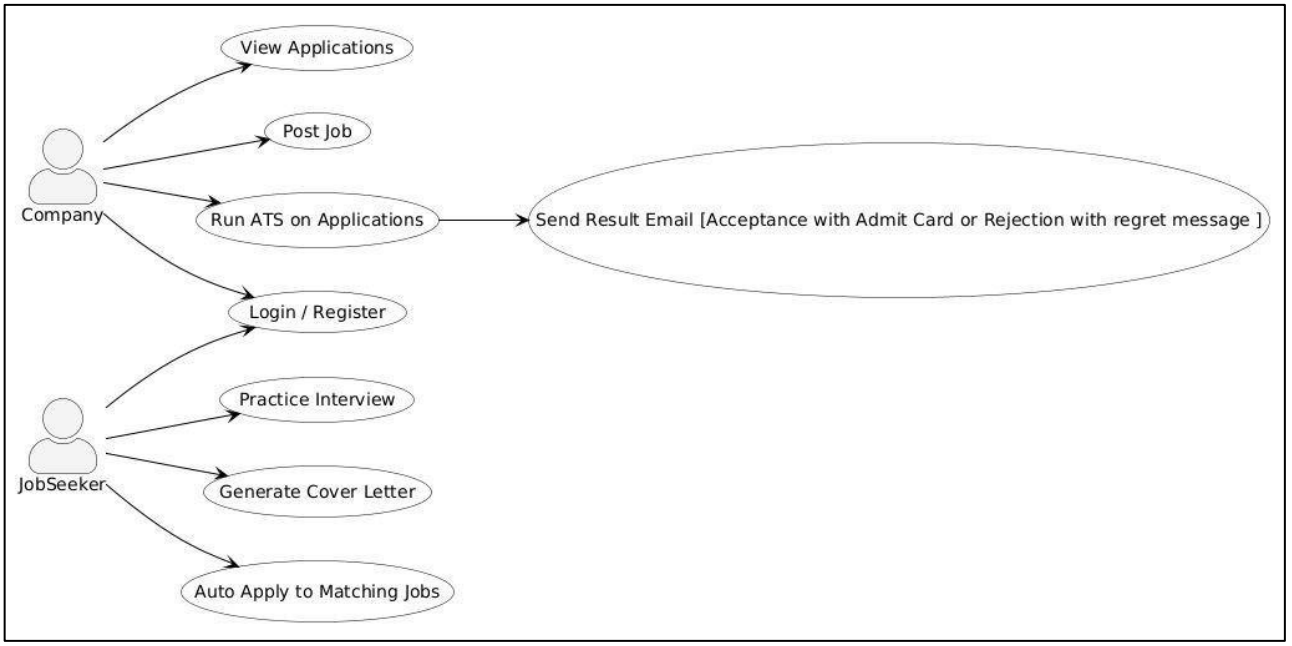

Fig. 4.3.2 Use-Case Diagram

CHAPTER 5
AGILE DOCUMENTATION

5.1 AGILE ROADMAP

The Agile roadmap for JobAi: Automated Job Portal with Auto-apply outlines a systematic approach to developing and deploying the platform in iterative phases. Beginning with data collection and model development, each phase is organized into weekly sprints, ensuring focused and incremental progress towards the project's goals. Build an AI-powered Job Portal platform that simplifies the job search and hiring process, offering seamless resume parsing, job matching, and automated application to jobs for which jobseeker's profile matches to an extent . Key phases and tasks include:

Sprint 1: Foundation & User Management

- **Duration**: Weeks 1-2
- **Key Objectives**:
  - Implement core user authentication (jobseeker & company registration/login).
  - Develop a robust resume upload and parsing module.
  - Set up basic project infrastructure (repository, CI/CD pipeline).
- **Milestones**:
  - Secure login and session management
  - Resume parser integrated with file validation
- **Outcome**: A stable platform where users can sign up, log in, and upload resumes.

Sprint 2: Job Posting & Matching Engine

- **Duration**: Weeks 3-4
- **Key Objectives**:
  - Create job posting functionality for companies.
  - Build a preliminary job matching algorithm to pair jobseekers with relevant job listings.
  - Integrate a simple user dashboard to view matches.
- **Milestones**:
  - Job posting forms and listings management
  - Initial job matching logic based on resume content and job criteria

- **Outcome**: A functional system where companies can post jobs and jobseekers receive initial job recommendations.

Sprint 3: ATS Scoring & Auto-Application

- **Duration**: Weeks 5-6
- **Key Objectives**:
  - Develop an ATS scoring system to evaluate applicant resumes automatically.
  - Implement the auto-apply feature that allows jobseekers to apply to matched jobs with one click.
  - Enhance user dashboards with ATS score displays.
- **Milestones**:
  - ATS score algorithm integrated with OpenAI services
  - Auto-application logic with duplicate prevention
- **Outcome**: Automated candidate evaluation and application, providing clear insights through ATS scores.

Sprint 4: Mock Interview & Feedback System

- **Duration**: Weeks 7-8
- **Key Objectives**:
  - Introduce an AI-powered mock interview module.
  - Enable video recording or text-based responses during mock interviews.
  - Generate feedback and improvement suggestions for jobseekers.
- **Milestones**:
  - AI-generated interview questions
  - Feedback mechanism for interview responses
- **Outcome**: An interactive module that helps jobseekers prepare for real interviews while gathering performance data.

Sprint 5: Communication & Notifications

- **Duration**: Weeks 9-10
- **Key Objectives**:
  - Implement a comprehensive notification system for both jobseekers and companies.
  - Develop email automation for sending application status updates, admit cards, and

feedback.

- Refine dashboards to reflect real-time application statuses.

- **Milestones**:
  - Automated email system for acceptance/rejection notifications
- **Outcome**: Clear, timely communication ensuring all users are informed about job application progress.

Sprint 6: Dashboard Enhancements & Analytics

- **Duration**: Weeks 11-12
- **Key Objectives**:
  - Enhance user dashboards with analytics and performance metrics.
  - Provide detailed insights for companies on applicant trends.
  - Optimize the overall user interface and experience based on feedback.
- **Milestones**:
  - Analytical reports and trend visualizations
  - Refined UI/UX design for improved accessibility and responsiveness
- **Outcome**: A polished, data-driven interface that empowers users with actionable insights.

5.2 AGILE PROJECT PLAN

- **Agile Plan:**

Project Plan	Status
Data collection and Annotation	Completed on January 20 2025
Web Platform Development	Completed on March 28 2025
Database Designing	Completed on March 16 2025
Integrating Algorithms	Completed on March 31 2025

Table 5.2.1: Agile Plan

5.3 AGILE USER STORY

As an IT professional, I want the JobAi: Automated Job Portal with Auto-apply

platform to automate job application and make it simpler to jobseekers for applying to all matching jobs to their profile without need to waste much time in search and manual apply. This enables jobseekers not to skip any job opportunities due to lack of time. .

Tasks:

Data Collection and Preprocessing:

o Gather resumes (DOC/DOCX) and associated profile metadata from jobseekers.

o Collect job listings and requirement details from companies.

o Manually review a sample of resumes and job posts to validate parsing accuracy.

o Preprocess text by cleaning, normalizing, and extracting key sections (contact, skills, education).

o Standardize file formats and data schemas to ensure consistency across the platform.

• **Model Development and Training:**

o Select and configure NLP and AI services (e.g., OpenAI GPT models) for resume parsing, job matching, and ATS scoring.

o Define ATS-scoring criteria (skills match, readability, grammar) and implement prompt templates.

o Train and fine-tune models using GPU-accelerated resources and representative datasets.

o Evaluate model performance on validation sets; iterate on prompts and hyperparameters to improve accuracy.

o Apply regularization and prompt-engineering techniques to prevent overfitting and ensure robust generalization.

• **Web Platform Development:**

o Design responsive user interfaces for jobseekers, recruiters, and administrators.

o Implement core features: user registration/login, resume upload, job posting, recommendation feeds, and auto-apply workflows.

o Integrate AI endpoints for on-the-fly resume parsing, match scoring, and mock-interview question generation.

o Build backend APIs for authentication, data processing, notifications, and file storage.

o Incorporate video recording and playback for mock interviews, plus a feedback submission interface.

• **Database Modules Setup:**

o Provision a secure relational database (e.g., MySQL) to store users, resumes, job listings, applications, and notifications.

o Define a clear schema with foreign-key relationships and uniqueness constraints (e.g., one application per jobseeker-job pair).

o Implement data validation rules and integrity checks at the model and database levels.

o Create Django ORM models and migrations to manage schema evolution.

o Ensure seamless integration between the database layer and backend services for reliable data exchange.

• **Testing and Validation:**

o Write unit tests for parsing logic, matching algorithms, ATS scoring, and API endpoints.

o Perform integration tests to verify end-to-end workflows (upload → match → apply → notify).

o Conduct system testing on staging with representative user scenarios and edge cases.

o Validate AI outputs—match scores, parsing results, interview feedback—against ground-truth samples and stakeholder review.

o Gather continuous feedback from pilot users to refine usability, fix bugs, and adjust feature behavior.

• **Deployment and Maintenance:**

o Deploy the JobAI platform using containerized services (Docker) and CI/CD pipelines (GitHub Actions).

o Configure cloud infrastructure for scalability, security, and high availability.

o Provide onboarding materials and training sessions for jobseekers and recruiters.

o Monitor performance metrics (response times, error rates, model latency) and set up alerting.

o Schedule regular maintenance windows for security patches, dependency updates, and feature roll-outs.

5.4 AGILE SPRINT BACKLOG

Date: 15/1/2025

o Project kickoff and team alignment.

o Initial requirements gathering: identify core modules (authentication, resume parsing,

matching, ATS, interviews).

o Set up version control, issue tracker, and project board.

• **Date: 20/1/2025**

o Collect sample resumes (.doc/.docx) and job listings from partner companies.

o Research existing job-portal workflows and best practices.

• **Date: 26/1/2025**

o Finalize module requirements and define user stories.

o Document data schema for users, resumes, jobs, applications, and notifications.

• **Date: 27/1/2025**

o Create wireframes and UI mockups for jobseeker and company dashboards.

o Review and iterate on UX flows for resume upload, job posting, and application tracking.

• **Date: 2/2/2025**

o Design database schema and write initial Django models.

o Prepare training/validation datasets for resume parsing and matching (cleaning, normalization).

• **Date: 10/2/2025**

o Initialize Django project and configure settings (database, static/media, security).

o Implement user authentication (registration, login, session management).

• **Date: 15/2/2025**

o Develop jobseeker profile module: resume upload UI and backend storage.

o Integrate OpenAI API for resume parsing; extract name, contact, skills, education

• **Date: 20/2/2025**

o Build company module: job posting form, listing management, and CRUD APIs.

o Notify matching jobseekers upon new job postings.

• **Date: 25/2/2025**

o Implement job matching engine: score jobs by relevance using AI prompts.

o Display personalized recommendations in the jobseeker dashboard.

• Date: 27/2/2025

o Integrate ATS scoring: calculate and store 0–100 scores for each application.

o Automate accept/reject logic based on threshold; generate admit cards.

• Date: 29/2/2025

o Develop mock interview module: generate 5 role-specific questions via OpenAI.

o Build response capture (text/video) and feedback workflow.

• Date: 1/3/2025

o Create notification center: in-app alerts and email templates for status updates.

o Implement real-time application tracking and read/unread flags.

• Date: 5/3/2025

o Refine UI/UX: responsive design tweaks, accessibility improvements.

o Conduct user walkthroughs and gather feedback for adjustments.

• Date: 10/3/2025

o Execute comprehensive testing: unit tests, integration tests, end-to-end scenarios.

o Validate AI outputs against ground-truth samples; tune prompts and thresholds.

• Date: 15/3/2025

o Set up CI/CD pipeline (GitHub Actions): automated builds, tests, and deployments.

o Deploy to staging environment; perform smoke tests.

• Date: 20/3/2025

o Conduct training sessions and prepare user documentation for jobseekers and recruiters.

o Address any critical bugs or UX issues identified in staging.

• Date: 25/3/2025

o Production deployment: launch JobAI platform for live use.

o Monitor system health, performance metrics, and error logs.

• Date: 1/4/2025

o Post-launch review: collect user feedback, track key usage metrics.

o Plan next set of enhancements and backlog reprioritization.

5.5 AGILE TEST PLAN

The Agile Test Plan for JobAi: Automated Job Portal with Auto-apply outlines a flexible and iterative approach to testing within the dynamic environment of Agile development. Emphasizing adaptability and continuous improvement, the plan focuses on testing processes that align with the incremental delivery of features and the evolving needs of stakeholders.

Automated Job Portal with Auto-apply outlines a flexible and iterative approach to testing within the dynamic environment of Agile development. Emphasizing adaptability and continuous feedback, the plan focuses on validating functionalities in line with incremental feature releases and real-time stakeholder expectations. The test strategy integrates continuous integration and automation, ensuring high software quality throughout the development lifecycle. Below are the key components:

❖ Introduction to Testing Approach:

- A well-defined Agile testing methodology is followed, aligning with the iterative nature of Agile sprints.

- Testing is not a phase but an ongoing activity, integrated into every stage of development.
- Focus is placed on collaboration between testers, developers, and product owners.

❖ Testing Strategy:

- Testing is conducted at various levels: unit, integration, system, and acceptance.
- Test coverage includes functionality, security, usability, and performance.
- Emphasis is placed on early detection of issues and continuous validation through sprint cycles.

❖ Test Environment Setup:

- Configured isolated development, staging, and production environments using Django and MySQL.
- Employed version-controlled test environments with seeded data to simulate real job applications and resume uploads.
- Incorporated third-party APIs (e.g., OpenAI, email services) in the testbed to mirror real-world interactions.

❖ Test Cases:

- Developed test cases for all core modules including resume parsing, job matching, auto-apply, ATS scoring, and mock interview.
- Included edge case validations like invalid resume formats, duplicate job applications, and threshold-based ATS decisions.
- Automated unit and integration tests using Django's testing framework and PyTest.

❖ Defect Management Procedures:

- Bugs and issues are logged in a sprint-wise backlog with priority tags.
- Quick triage is conducted in daily stand-ups; blockers are resolved before moving to the next sprint.
- Continuous regression testing ensures that new features don't break existing functionality.

❖ Commitment to Continuous Improvement:

- Regular retrospectives are conducted at the end of each sprint to refine test approaches.
- Metrics such as test coverage, bug counts, and pass/fail ratios are tracked for performance analysis.
- Feedback loops from users, demo sessions, and UAT are incorporated to enhance the platform's reliability.

❖ Agile Test Activities for JobAi-Automated Job Portal with Auto-apply:

- **Resume Parsing Validation**:
  Validate that resume uploaded in .doc/.docx formats are accurately parsed using OpenAI API. Extracted fields like name, contact info, education, and skills are compared against ground truth.
- **Job Matching Accuracy Test**:
  Test the AI's job recommendation engine by comparing user profiles with recommended listings. Validate ranking logic and ensure alignment with user preferences.
- **Auto-apply Feature Testing**:
  Test conditions under which the auto-apply module functions—priority sorting, job limits, and duplication checks. Simulate both successful and failed applications.
- **ATS Score Calculation Test**:
  Verify that ATS scoring correctly reflects job-resume alignment. Test edge conditions such as extremely low or high scores and ensure decisions (Accept/Reject) trigger appropriate notifications.
- **Mock Interview Module Testing**:
  Validate question generation using OpenAI prompts based on job roles. Check video/audio recording functionality, playback, and storage.
- **UI/UX and Responsiveness Testing**:
  Test interfaces for both jobseekers and companies across devices and browsers. Ensure intuitive navigation, consistent layout, and accessibility compliance.
- **Database & Security Testing**:
  Confirm data integrity during concurrent actions like job applications and resume uploads. Validate user authentication and role-based access control.

CHAPTER 6
IMPLEMENTATION AND TESTING

6.1 TESTING METHODS

Testing is a critical process aimed at assessing the quality, functionality, and reliability of a system. It involves systematically executing predefined test cases to identify defects or discrepancies between expected and actual outcomes. Testing ensures that the software meets specified requirements, operates as intended, and delivers value to users.

The primary goal of testing is to uncover defects and errors early in the development lifecycle, reducing the likelihood of costly issues in production. Through testing, we can validate the correctness of code, identify potential weaknesses or vulnerabilities, and ensure that the system behaves as expected under various conditions. Testing also helps improve software maintainability and scalability by identifying areas for optimization or enhancement.

Testing encompasses various levels and types, including unit testing, integration testing, system testing, and acceptance testing. Each type of testing focuses on different aspects of the system, from individual components to the system as a whole, and serves specific objectives in ensuring software quality. Overall, testing is an essential practice in software development, contributing to the delivery of reliable, high-quality software products that meet user needs and expectations.

6.1.1 Levels of Testing

The different levels of testing are:

- Unit testing
- Integration testing
- System testing
- User acceptance testing
- Regression Testing

- **Unit testing**

Unit testing for JobAi: Automated Job Portal with Auto-Apply plays a pivotal role in ensuring the reliability, functionality, and maintainability of its individual components and modules. This testing approach involves isolating and validating each unit of code, typically at the method or function level, to ensure that it behaves as expected in various scenarios.

At the heart of JobAi's unit testing strategy lies the verification of backend services, data extraction logic, job matching algorithms, and frontend interactions. For backend services, unit tests scrutinize the business logic responsible for tasks such as resume parsing, keyword extraction, ATS scoring, and job filtering. This includes testing different scenarios and edge cases to ensure accurate and consistent results.

Unit tests for database interactions focus on verifying the correctness of data retrieval, storage, and update operations. Mocking techniques are used to simulate data environments, ensuring the system handles data operations under diverse conditions.

On the frontend, unit tests validate UI components such as registration forms, resume upload interfaces, and status dashboards. Testing includes validating user inputs, error messages, and form behavior to ensure an intuitive experience for jobseekers and recruiters.

JobAi's unit testing leverages tools like `pytest` and JavaScript-based libraries for frontend tests. Continuous integration (CI) pipelines are integrated to automate these tests during every update, promoting rapid bug detection and delivery of reliable features.

❖ **Integration Testing**

Integration testing is a crucial component of JobAi's quality assurance strategy, ensuring that all system components work together seamlessly to deliver the intended functionality and user experience. This testing approach focuses on validating the interactions and interfaces between different modules, services, and APIs within the JobAi platform.

At its core, integration testing for JobAi involves verifying the communication between resume parsing, job-matching logic, auto-apply scripts, and database modules. This includes testing the API responses, data flow, and trigger actions when resumes are uploaded or applications are submitted.

Integration points such as automatic email notifications, application status updates, and ATS-based decision handling are thoroughly tested to ensure correctness across modules. For example, once a jobseeker uploads their resume, the integration testing ensures resume extraction flows into job matching, and valid applications are created automatically without user intervention.

Tools like Postman, Selenium, and Cypress are used to automate integration testing, validating form submissions, auto-apply triggers, and user session flows. CI tools automatically run these tests with every new deployment to catch integration flaws early.

❖ **System Testing**

System testing for JobAi: Automated Job Portal with Auto-Apply involves evaluating the integrated system as a whole to ensure that it meets the specified requirements and functions correctly in its intended environment. Several methods can be employed during system testing to comprehensively assess the platform's functionality, usability, performance, and reliability.

• **Functional Testing:** This method verifies the core functionalities of the JobAi platform, including jobseeker registration, resume upload, job matching, auto-application logic, and application tracking. Test cases are created for all features to ensure they behave as intended.

• **User Interface Testing:** UI testing assesses the usability, responsiveness, and accessibility of the JobAi platform. This includes testing across various devices, browsers, and screen sizes to confirm that the interface is consistent and user-friendly.

• **Performance Testing:** Performance tests evaluate JobAi's behavior under different traffic levels. It measures how the auto-apply logic performs when multiple users are uploading resumes or when mass job data is fetched. These tests ensure responsiveness even under peak usage.

• **Security Testing:** Security tests are conducted to detect vulnerabilities in user authentication, resume storage, and data integrity. The system is checked for risks such as unauthorized data access, SQL injection, and CSRF attacks.

• **Compatibility Testing:** This ensures that the JobAi platform functions uniformly across different OS platforms and browsers like Chrome, Firefox, and Safari. It confirms that no functional or UI breakdown occurs in different configurations.

❖ **User Acceptance Testing (UAT)**

User Acceptance Testing (UAT) for JobAi ensures that the platform meets end-user expectations and business requirements. In UAT, real users test the platform in a production-like environment by uploading resumes, verifying job matches, and assessing the auto-apply functionality.

Feedback from UAT leads to final improvements before deployment, focusing on intuitive workflows, application status clarity, and success messages. This phase confirms that the software delivers real-world value.

Regression Testing

Regression Testing in JobAi involves re-testing existing features after each new deployment to ensure that previously working functionality remains unaffected. This includes testing core features like resume parsing, job matching, and auto-apply logic after introducing enhancements or bug fixes.

Regression tests are automated where possible, using scripts and CI/CD pipelines to validate the system's stability.

6.2 IMPLEMENTATION APPROACHES

Setup Environment:

• Install necessary technologies and dependencies required for JobAi: Automated Job Portal with Auto-Apply development, such as Python, Django, and additional libraries for PDF/Doc parsing and resume keyword extraction.

• Set up the development environment on the developer's system by installing Python and other required tools.

Create Virtual Environment:

•Navigate to the directory containing the JobAi project code.

• Create a virtual environment using a tool like virtualenv or venv to isolate project dependencies.

Install Dependencies:

• Install all necessary Python dependencies using pip within the virtual environment.

• This includes libraries for web development, PDF parsing (PyMuPDF or pdfminer), text extraction, and database connectivity.

Run JobAi Application:

• Navigate to the directory containing the JobAi application code.

• Run the Django application by executing the manage.py script.

• Ensure that the required services, such as the database server, are running before starting the application.

Access Application:

• Once the application is running, access it through a web browser using the appropriate URL (http://127.0.0.1:8000 or deployment IP).

Deployment:

• The system can be containerized using Docker or deployed on cloud platforms (Heroku, AWS, etc.)

for production.

• Environment variables and database credentials are configured appropriately.

Testing and Validation:

• Conduct thorough testing using unit, integration, and system tests.

• Validate all modules including resume upload, job matching, auto-apply, and application tracking before production release.

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

JobAi: Automated Job Portal with Auto-apply is a transformative solution designed to streamline and optimize the job search and recruitment process through the intelligent integration of artificial intelligence, automation, and user-centered design. By offering advanced features such as AI-powered resume screening, smart job recommendations, auto-application functionality, and real-time status tracking, JobAi significantly reduces the time, effort, and complexity involved in job seeking and hiring.

The system not only enhances the experience for job seekers by matching them to the most relevant opportunities based on their skills, qualifications, and preferences, but also provides companies with a robust platform to identify the most suitable candidates quickly and efficiently through ATS-based applicant ranking and automated communication. Its modular design, including dedicated interfaces for job seekers and employers, ensures a seamless workflow across all phases of the hiring process—from application submission to interview scheduling and final selection.

Built using modern web technologies and machine learning models, and supported by a scalable backend infrastructure, JobAi is not only robust and efficient but also adaptable to future enhancements. It has the potential to expand into more intelligent features like interview analysis, skill gap detection, and career development tools, making it an all-encompassing career platform.

In conclusion, JobAi serves as a comprehensive and forward-thinking approach to tackling the inefficiencies of traditional recruitment methods. It is a step toward democratizing access to employment opportunities and empowering users to take charge of their career paths with confidence and clarity. As it continues to evolve, JobAi is poised to redefine how individuals connect with meaningful work and how organizations discover top talent in the digital age.

7.2 FUTURE SCOPE

- **Enhanced AI-Powered Job Matching with Deep Learning**

The future of JobAi lies in the integration of advanced deep learning models that can further

refine job recommendations based on a candidate's career history, skills, preferences, and behavioral patterns. By implementing Natural Language Processing (NLP) techniques and semantic analysis, the platform can achieve a more sophisticated understanding of job descriptions and resumes, ensuring an even higher accuracy in job matching. Furthermore, incorporating real-time labor market trends and economic factors can allow JobAi to proactively suggest career shifts, skill upgrades, or emerging job roles, helping job seekers stay competitive in an ever-evolving job market.

- **Advanced Interview Preparation and Real-Time Feedback**

With the increasing use of AI in recruitment, JobAi will integrate real-time mock interview simulations using conversational AI and facial expression analysis. The platform will be able to assess a candidate's confidence, communication skills, and posture during video interviews, providing instant feedback to improve their performance. Further advancements will include voice modulation analysis, sentiment detection, and AI-generated suggestions for answering behavioral interview questions more effectively. This feature will prepare job seekers to face real-world interview scenarios with confidence and improve their chances of securing their desired jobs.

- **Mobile App with Offline Features and Notifications**

To improve accessibility, JobAi will launch a fully functional mobile app with offline capabilities, allowing users to track job applications, prepare for interviews, and receive notifications even without an active internet connection. Smart notifications will provide timely reminders for application deadlines, scheduled interviews, and recruiter responses, ensuring that users never miss important job opportunities.

- **Predictive Career Path Analysis and Future Job Market Trends**

By leveraging AI and machine learning, JobAi will offer predictive career path analysis, helping users understand long-term career prospects and potential job shifts. The system will analyze employment trends, industry demand, and personal career progression to suggest the best future job opportunities for users. It will also provide insights into upcoming skills in demand, helping users proactively acquire new competencies to stay ahead in their careers.

CHAPTER 8
APPENDICES

8.1 SCREENSHOTS

Fig 8.1.1 Landing Page

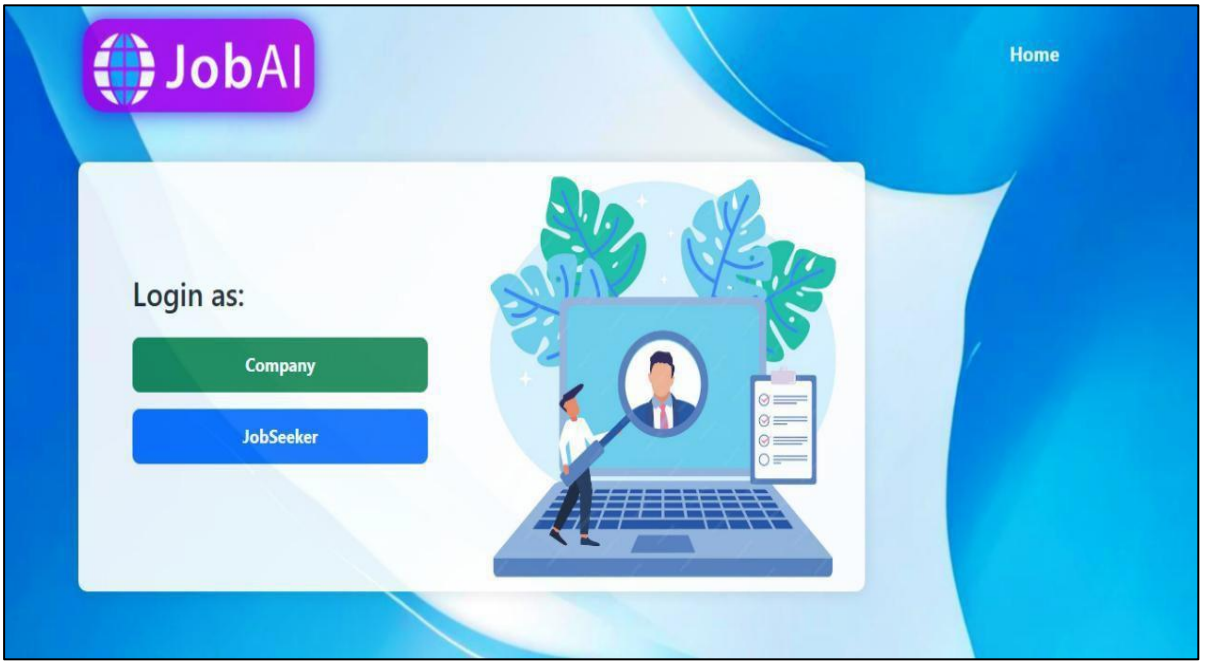

Fig 8.1.2 User Type Page

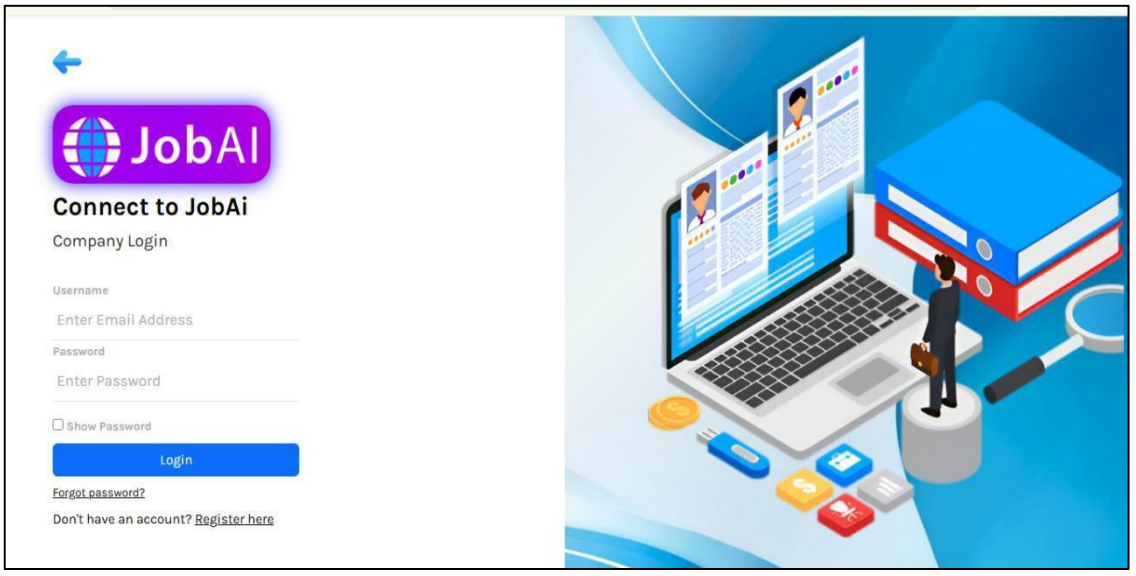

Fig 8.1.3 Company Login Page

Fig 8.1.4 Jobseeker Login Page

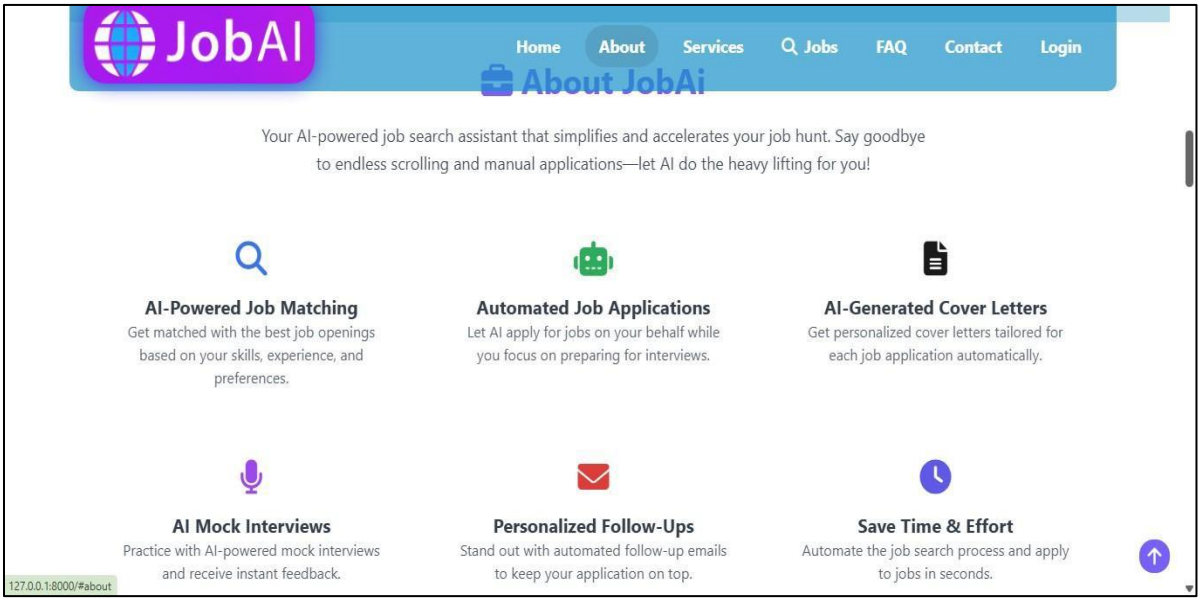

Fig 8.1.5 About Section

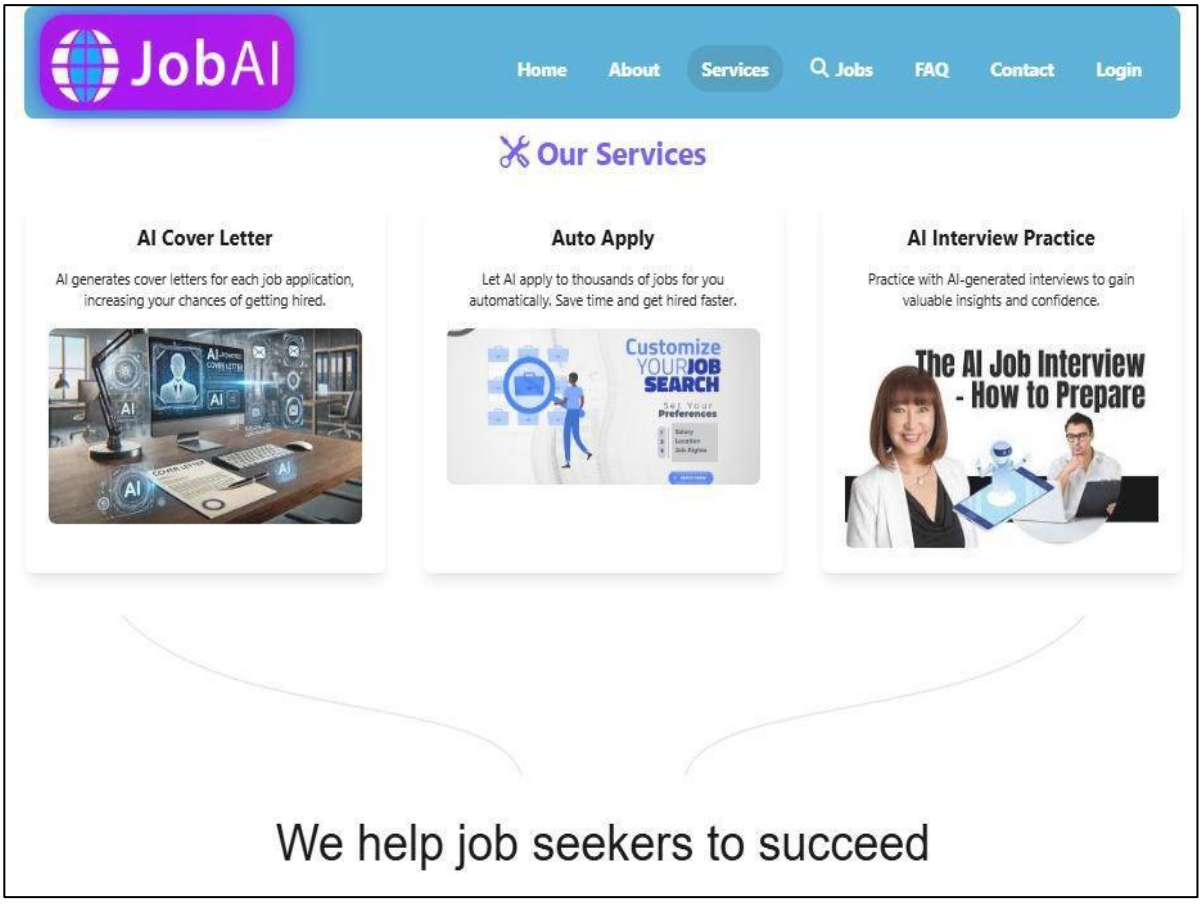

Fig 8.1.6 Services Section Page

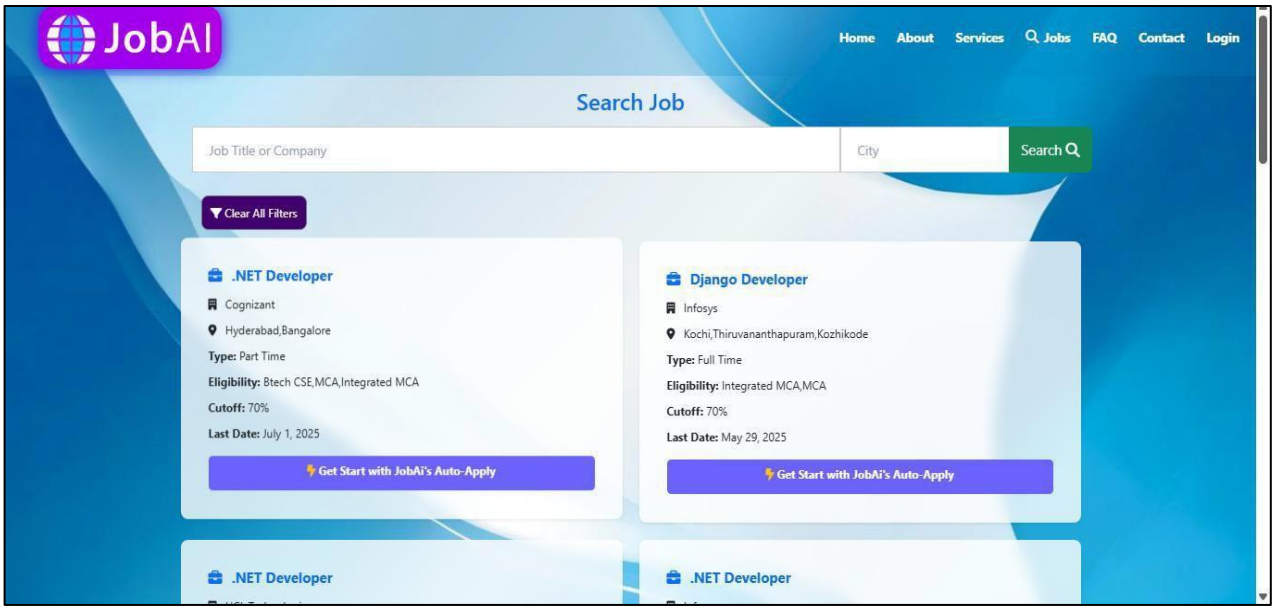

Fig 8.1.7 Job Listing Page

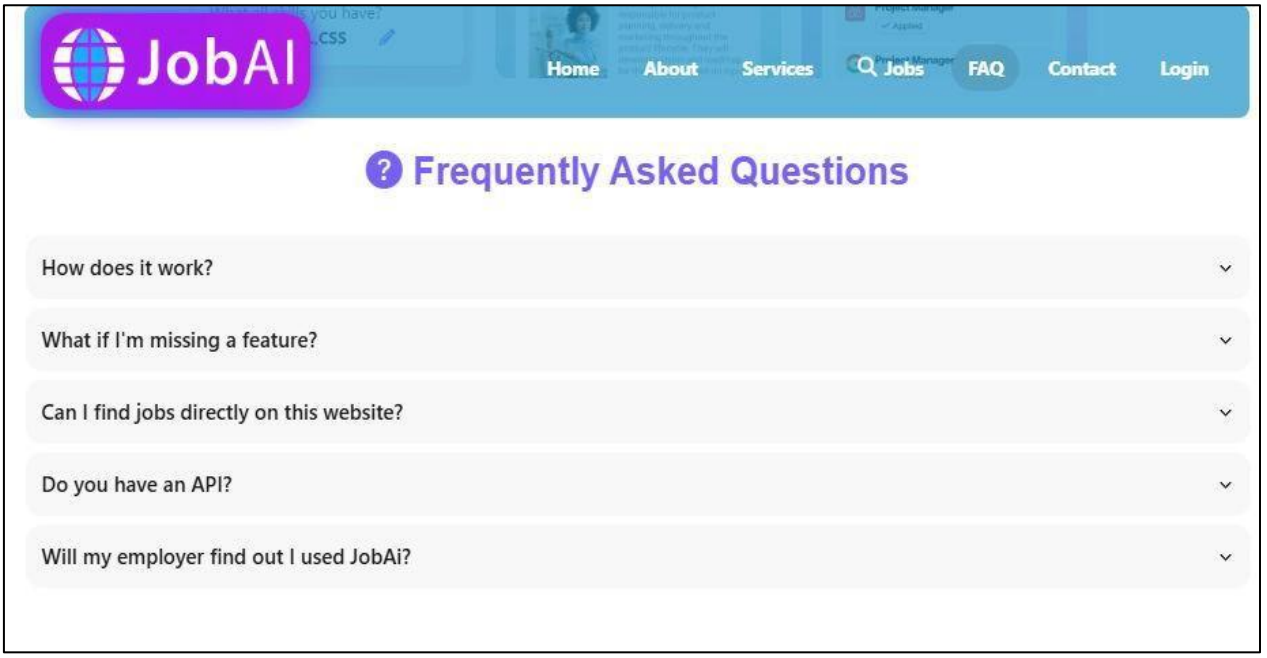

Fig 8.1.8 FAQ Section

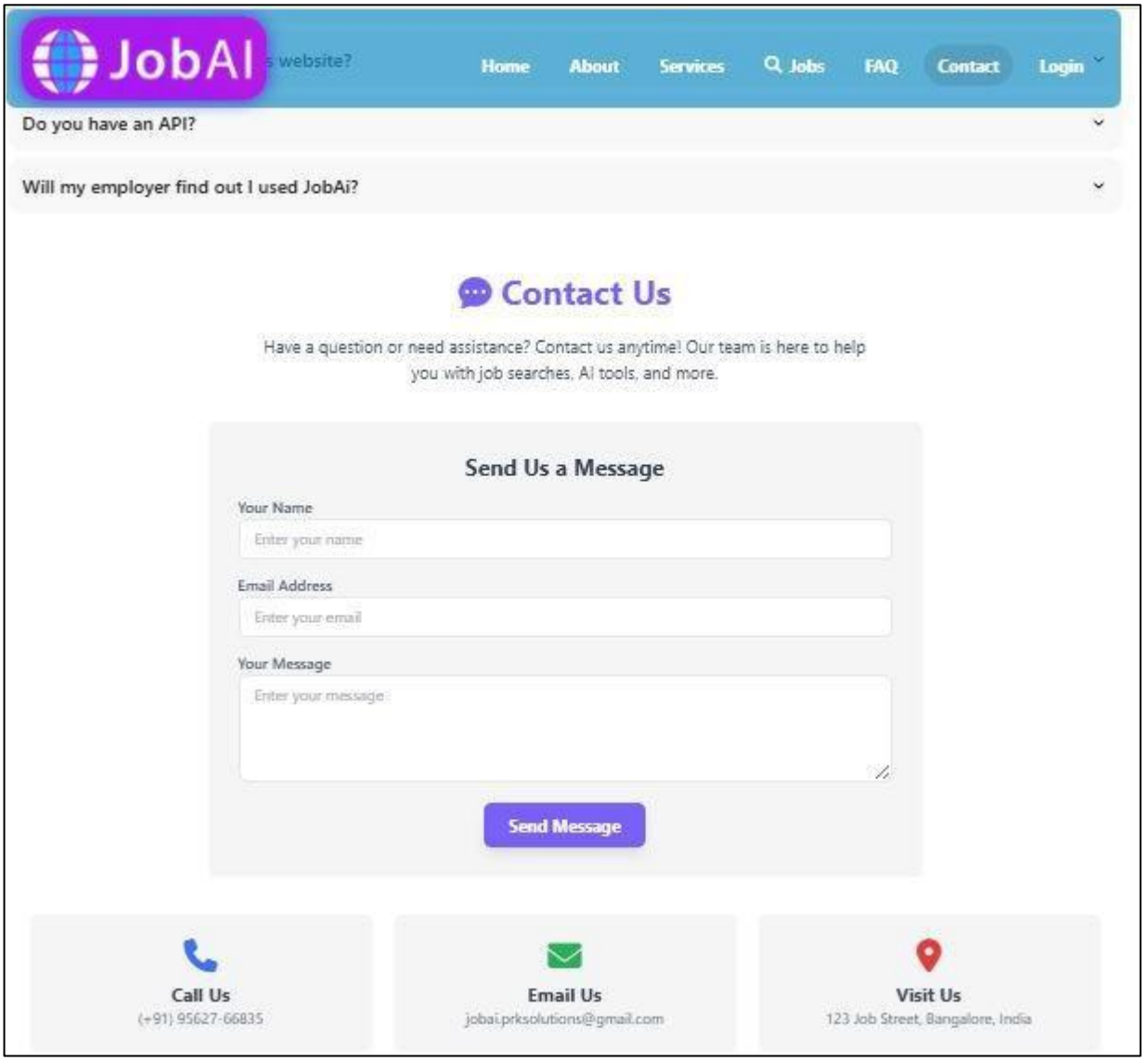

Fig 8.1.9 Contact Section

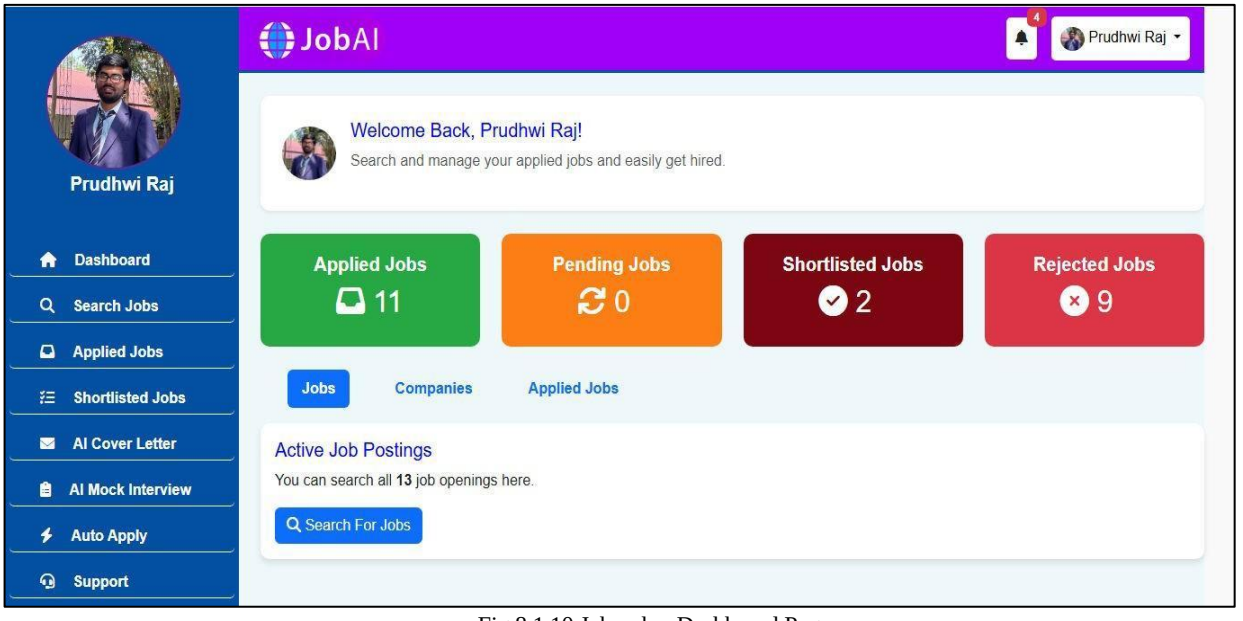

Fig 8.1.10 Jobseeker Dashboard Page

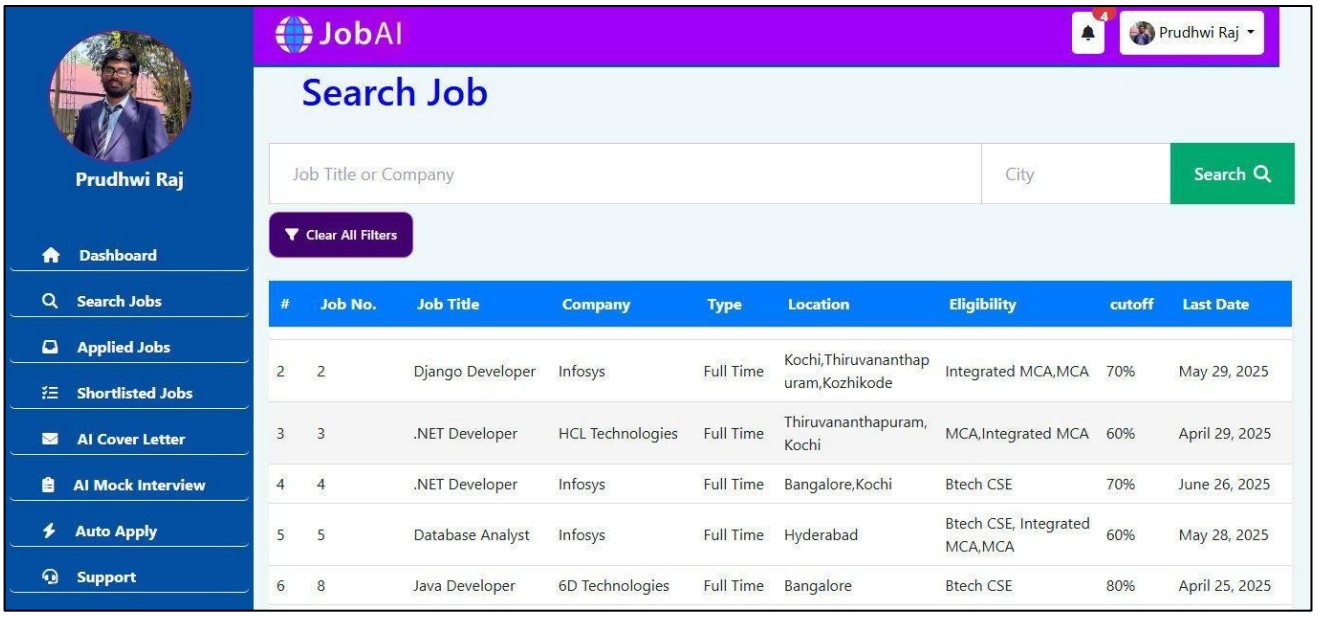

Fig 8.1.11 Jobseeker Job Search Page

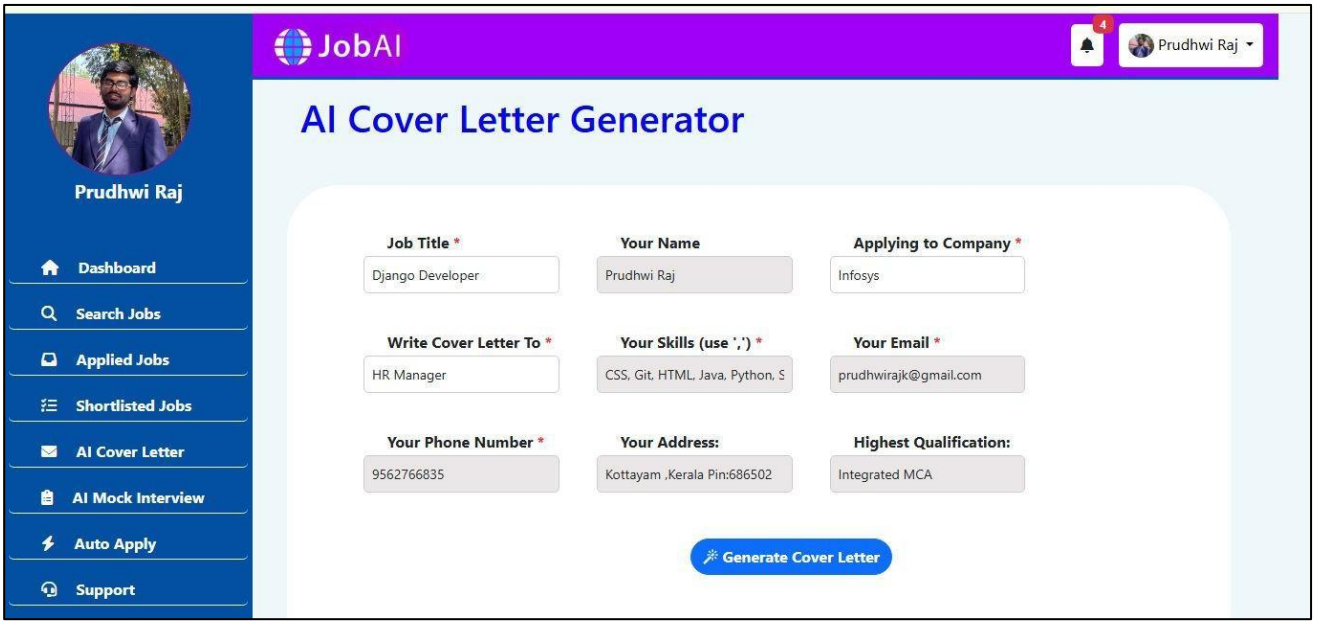

Fig 8.1.12 AI Cover Letter Generator Page

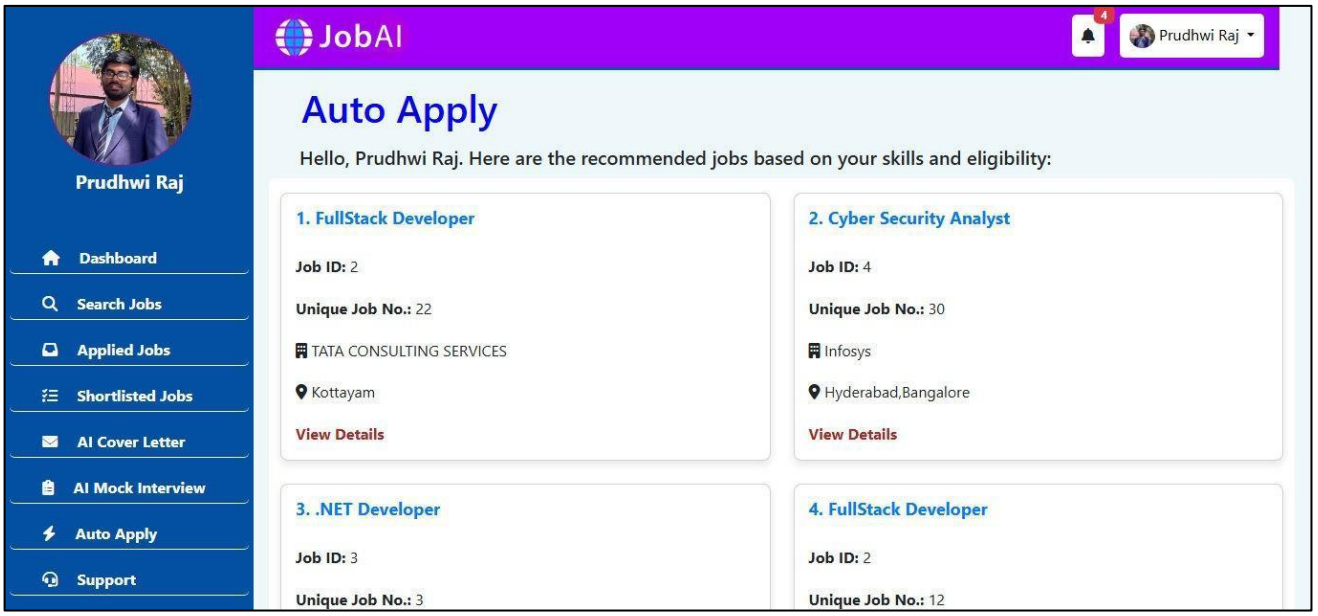

Fig 8.1.13 Auto-Apply on Recommended Jobs Page

Fig 8.1.14 Support Page

Fig 8.1.15 Jobseeker Settings Page

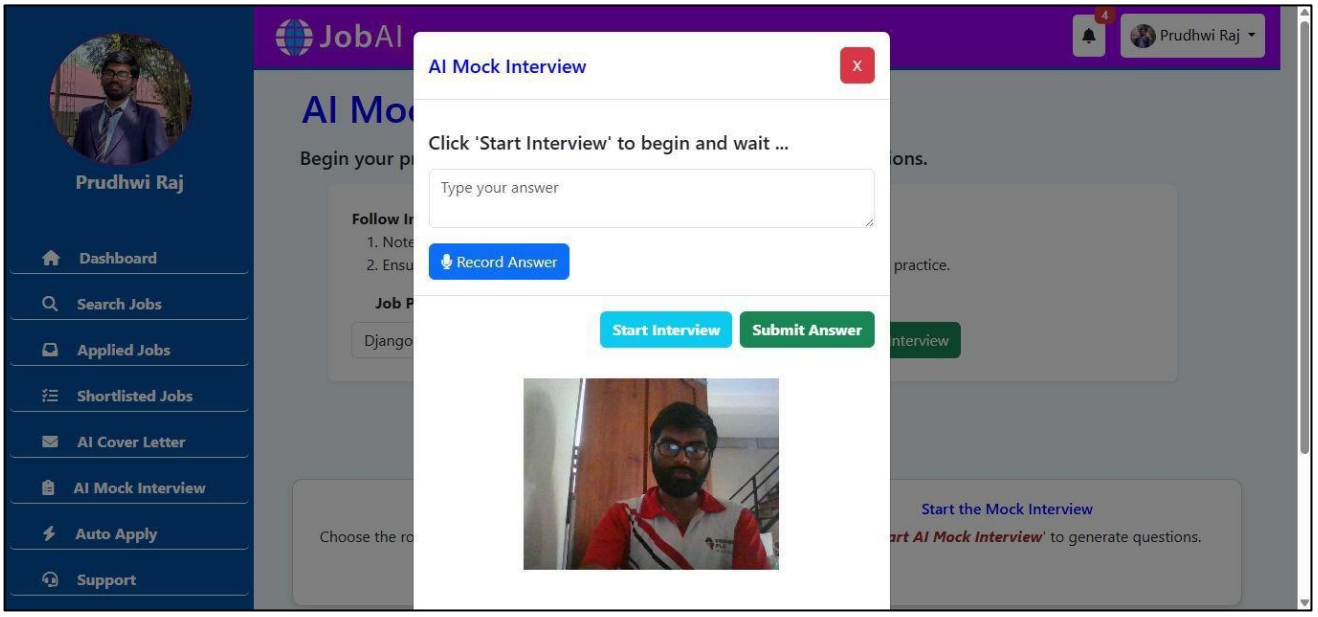

Fig 8.1.16 Mock Interview Page

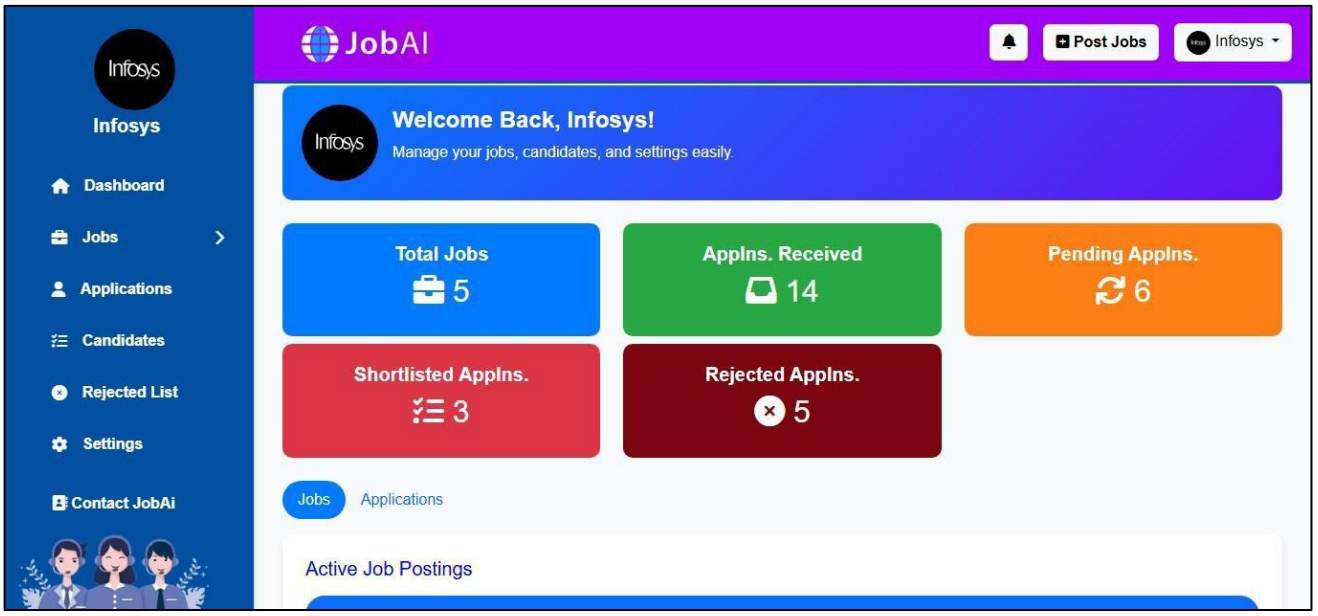

Fig 8.1.17 Company Page

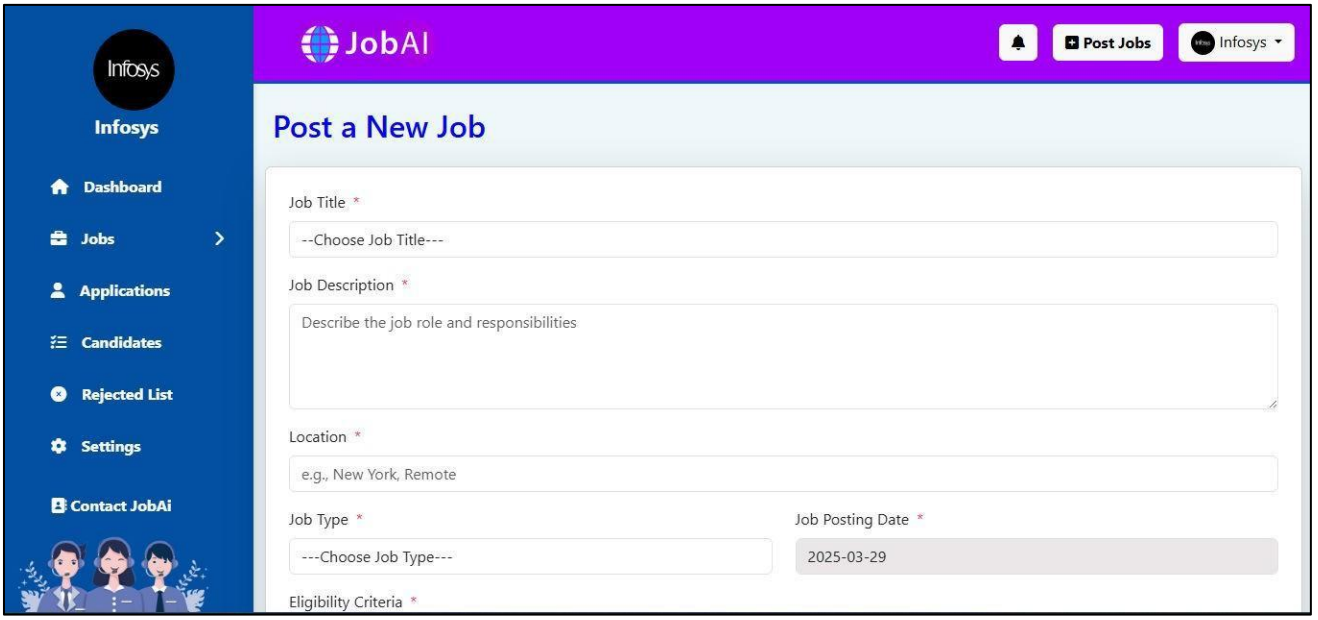

Fig 8.1.18 Post Jobs Page

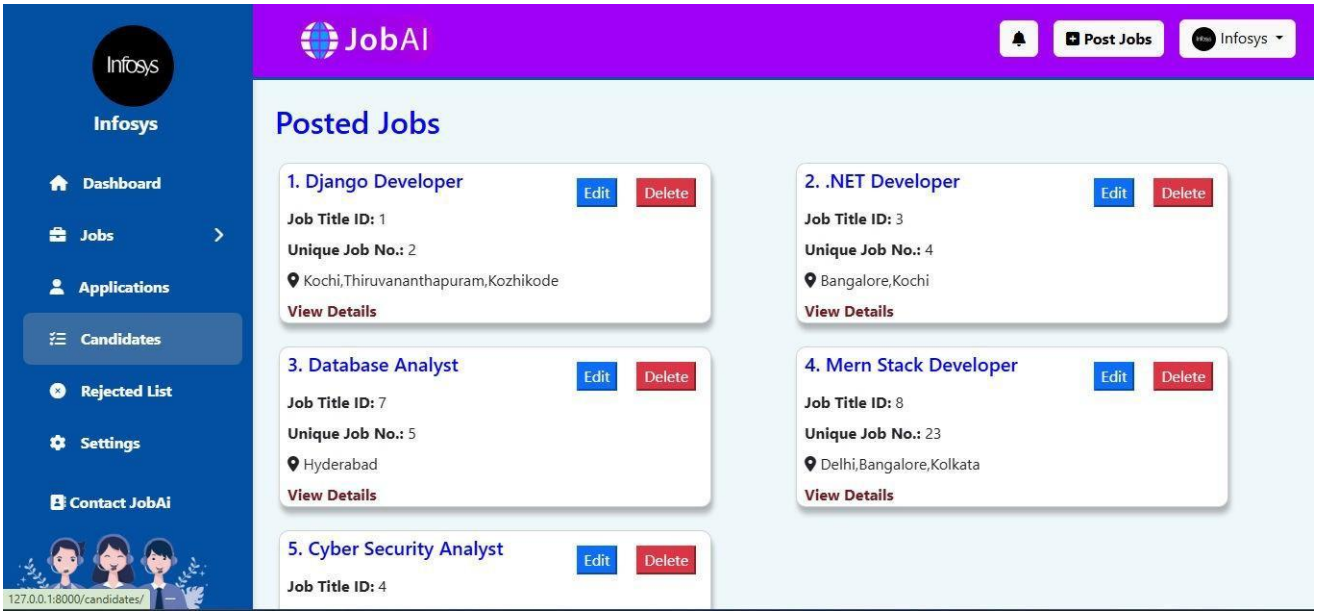

Fig 8.1.19 Posted Jobs List Page

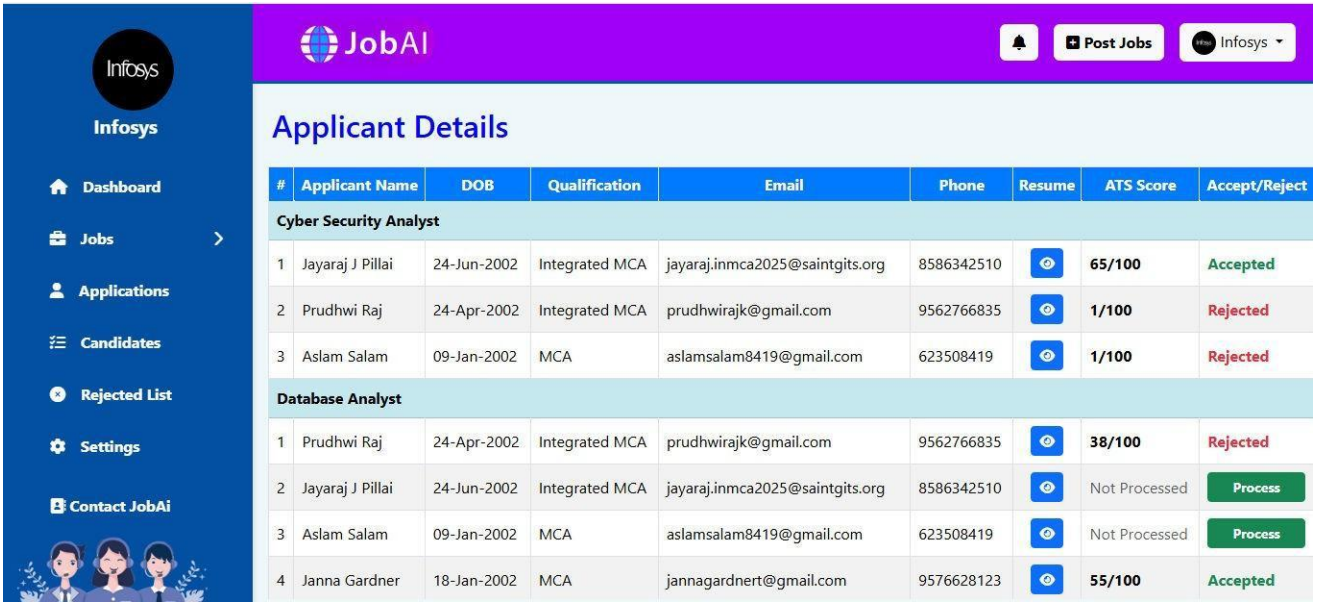

Fig 8.1.20 Landing Page

8.2 SAMPLE CODE

- **manage.py**

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'JobAi.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```

- **models.py**

```python
from django.db import models
# from fields import ComparableMixin

class Jobseeker_Registration(models.Model):
    name=models.CharField(max_length=255,default="Unknown")
    email=models.EmailField(unique=True)
    password=models.CharField(max_length=255)
    phone=models.CharField(max_length=20, blank=True, null=True)
    last_login = models.DateTimeField(null=True, blank=True)
    def __str__(self):
        return self.name

class jobseeker_profile(models.Model):
    user = models.OneToOneField(Jobseeker_Registration, on_delete=models.CASCADE, related_name='profile',null=True)  # ForeignKey to Jobseeker_Registration
    name = models.CharField(max_length=255, blank=True, null=True)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=20, blank=True, null=True)
    profile_img=models.ImageField(upload_to='images/',blank=True,null=True)
    dob = models.DateField(max_length=20,blank=True,null=True)
    highest_qualification = models.CharField(max_length=255, blank=True, null=True)
    percentage=models.TextField(blank=True,null=True)
    passoutyear=models.TextField(blank=True,null=True)
    job_preference = models.CharField(max_length=255, blank=True, null=True)
    university = models.CharField(max_length=255, blank=True, null=True)
    address = models.TextField(blank=True, null=True)
    skills = models.TextField(blank=True, null=True)
    resume = models.FileField(upload_to='documents/', blank=True, null=True)

    def __str__(self):
        return self.name if self.name else "Unnamed Jobseeker"

class jobseeker_resume(models.Model):
    user = models.ForeignKey(Jobseeker_Registration, on_delete=models.CASCADE, null=True, blank=True)  # user is required, cannot be null
    file = models.FileField(upload_to='documents/')
    uploaded_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"Document uploaded by {self.user.username} on {self.uploaded_at}"
```

```
class Company_Type_Master(models.Model):
    company_type=models.CharField(max_length=50)
    def __str__(self):
        return self.name if self.name else "Unnamed Company"

class Company(models.Model):
    password=models.CharField(max_length=255)
    # Links to Django's built-in User model
    name = models.CharField(max_length=255)
    description=models.CharField(max_length=12000,null=True)
    email = models.EmailField(unique=True)
    company_type = models.ForeignKey(Company_Type_Master,on_delete=models.CASCADE,default=None)
    address = models.CharField(max_length=255)
    profile_img=models.ImageField(upload_to='company_images/',blank=True,null=True)

    def __str__(self):
        return self.name
class job_title(models.Model):
    job_title=models.CharField(max_length=250,blank=True,null=True)
    def __str__(self):
        return self.name

class company_joblist(models.Model):
    company=models.ForeignKey(Company,on_delete=models.CASCADE,default=None)
    job_title = models.ForeignKey(job_title,on_delete=models.CASCADE,default=None)
    # job_number = models.CharField(max_length=100)
    job_description = models.CharField(max_length=12000, blank=True, null=True)
    job_type = models.CharField(max_length=255, blank=True, null=True)
    location=models.CharField(max_length=200,null=True)
    highest_qualification = models.CharField(max_length=255, blank=True, null=True)
    percent_criteria=models.TextField(blank=True,null=True)
    skills_required = models.TextField(blank=True, null=True)
    dateofpublish=models.DateField(max_length=20,null=True)
    Lastdate=models.DateField(max_length=20,null=True)
```

```
class JobNotification(models.Model):
    jobseeker_profile = models.ForeignKey(jobseeker_profile, on_delete=models.CASCADE, related_name='notifications')
    company_job = models.ForeignKey(company_joblist, on_delete=models.CASCADE, related_name='notifications')
    message = models.TextField()
    is_read = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Notification for {self.jobseeker.name} - {self.company_job.job_title}"


class UploadedFile(models.Model):
    file = models.FileField(upload_to='uploads/')
    uploaded_at = models.DateTimeField(auto_now_add=True)

### **New Job Application Table**
class JobApplication(models.Model):
    jobseeker = models.ForeignKey(jobseeker_profile, on_delete=models.CASCADE, related_name='applications')
    company_joblist = models.ForeignKey(company_joblist, on_delete=models.CASCADE, related_name='applications')
    applied_at = models.DateTimeField(auto_now_add=True)  # Tracks when applied
    ats_score = models.IntegerField(null=True, blank=True)  # Store ATS score
    status=models.TextField(blank=False,null=False,default="Applied")
    class Meta:
        unique_together = ('jobseeker', 'company_joblist')  # Ensures unique applications per jobseeker-job

    def __str__(self):
        return f"{self.jobseeker.name} applied for {self.job.job_title} at {self.job.company.name}"
```

- **Urls.py**

```python
from django.urls import path
from . import views
from .views import *
from django.conf.urls.static import static

urlpatterns = [

    path('',views.main,name="index"),
    path('profile/',views.jobseeker_home,name="home"),
    path('jobseeker_dashboard/', jobseeker_dashboard, name='jobseeker_dashboard'),
    path('jobseeker_login/',views.jobseeker_login,name="jobseeker_login"),
    path('jobseeker logout/',views.jobseeker_logout,name='jobseeker_logout'),
    path('jobseeker_profile/',views.jobseeker_profile_update,name="update_profile"),
    path('company_logout/',views.company_logout,name='company_logout'),
    path('delete_job/',views.delete_job,name='delete_job'),
    path("edit_job/", views.edit_job, name="edit_job"),
    path('base/',views.user_base,name="base"),
    path('delete-resume/',views.delete_resume,name='delete_resume'),
    path('company-change-password/',views.company_change_password,name='company_ch_pwd'),
    path("applicants/",views.applications,name='job_applications'),
    path('company_login/',views.company_login,name="company_login"),
    path('user-type/',views.user_type,name="user-type"),
    path("settings/", views.settings_view, name="settings_view"),
    path('delete_profile/',views.delete_user_profile,name='delete_profile'),
    path('search_job/',views.search_job,name='search-job'),
    path('search/',views.main_search,name="main_search"),
    path("send-cover-letter-email/", send_cover_letter_email, name="send_cover_letter_email"),
    path('reset_password/<int:user_id>/',views.reset_password,name="reset_password"),
    path('company_reset_password/<int:comp_id>/',views.company_reset_pwd,name="company_reset"),
    path('company_registration/',views.company_registration,name="company_registration"),
    path('company_dashboard/',views.company_dashboard,name="company_dashboard"),
    path('job_listing/',views.company_jobs,name="job_listing"),
    path('shortlisted/',views.shortlisted,name="shortlisted"),
    path('jobs/',views.jobs,name="jobs"),
    path('candidates/',views.candidates,name="candidates"),
    path('rejected/',views.rejected,name="rejected"),
    path('notification1/',views.mark_notification_as_read,name="mark_notification_as_read"),
    # path("check-ats-score/<int:application_id>/",views.check_ats_compatibility, name="check_ats_score"),
    path("auto-process-application/<int:application_id>/",views.auto_process_application, name="auto_process_application"),
    path('Post_Job/',views.company_postjob,name="Post Job"),
    path('company_settings/',views.company_settings,name="company_settings"),
    path('cover-letter/',views.coverletter,name='ai_cover_letter'),
    path('change_password/',views.change_password,name="j_change_pwd"),
    path('Register/',views.Register,name='Register'),
    path('company_type/',views.company_type,name="company_type"),
    path('Forgot_Password/',views.Forgot_pwd,name='Forgot Password'),
    path('user_support/',views.support,name='support'),
    path("contact/", contact_view, name="contact"),
    path("notifications/", jobseeker_notifications, name="jobseeker_notifications"),
    path("notifications/read/", mark_notifications_as_read, name="mark_notifications_as_read"), path('company_password_reset/',views.company_forgot_password,name='company_forgot_password'),
    path('auto-apply/',views.autoapply,name='auto-apply'),
    path('mock-interview/',views.mockinterview,name='mock-interview'),
    path('applied_jobs/',views.applied_jobs,name="applied-jobs")
]
if settings.DEBUG:
    urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT)
```

- **views.py**

```python
from datetime import date,datetime
from email import message
import openai
from django.shortcuts import get_object_or_404, render, redirect
import os
from .extract_resume import extract_resume_details
from .job_matching import match_jobs
from .auto_apply import auto_apply_jobs
from django.http import JsonResponse
from django.contrib.auth import logout
from django.core.files.storage import FileSystemStorage
from django.db.models import Q
# from weasyprint import HTML
from docx import Document
from django.conf import settings
from .models import * #noqa
from .ats_check_and_process_automation import auto_process_application#pylint:disable=unused-imports
from django.contrib import messages
from django.core.mail import send_mail
from itertools import groupby
from django.core.mail.message import EmailMessage
import tempfile
from django.utils.timezone import now
import json

openai.api_key = settings.OPENAI_API_KEY

def jobseeker_login(request):
    if request.method == "POST":
        Email = request.POST.get("email")
        Password = request.POST.get("password")
        try:
            jobseeker = Jobseeker_Registration.objects.get(email=Email, password=Password)
            request.session['jobseeker_name'] = jobseeker.name
            request.session['jobseeker_id'] = jobseeker.id
            request.session['jobseeker_email'] = jobseeker.email
            request.session['jobseeker_phone'] = jobseeker.phone
            messages.success(request, "Login Successfully")
            return redirect('jobseeker_dashboard')  # Redirect to a dashboard or home page after login
        except Jobseeker_Registration.DoesNotExist:
            messages.error(request, "Invalid Email or Password")
    return render(request, 'jobseeker_login.html')
```

```python
def jobseeker_profile_update(request):
    userid = request.session.get('jobseeker_id')
    dob = ""
    phone = ""
    address = ""
    qualification = ""
    skills = ""
    job_preference = ""
    if request.method == "POST":
        name = request.POST.get('name')
        email = request.POST.get('email')
        dob = request.POST.get('dob')
        phone = request.POST.get('phone')
        address = request.POST.get('Address')
        qualification = request.POST.get('Education')
        university = request.POST.get('University')
        percentage=request.POST.get('Percentage')
        passout=request.POST.get('passout')
        skills = request.POST.get('skills')
        image = request.FILES.get('image')
        job_preference = request.POST.get('job_preferences')
        # Fetch the first resume for the user (if multiple exist)
        resume = jobseeker_resume.objects.filter(user=userid).first()
        cv = resume.file if resume else None  # Handle case where no resume exists
        job_preference=job_title.objects.get(id=job_preference).job_title
        if jobseeker_profile.objects.filter(name=name).exists() and jobseeker_profile.objects.filter(email=email).exists():
            messages.error(request, "A Jobseeker with this name and/or email already exists. Please use different details.")
        else:
            jobseeker_obj = jobseeker_profile()
            jobseeker_obj.name = name
            jobseeker_obj.email = email
            jobseeker_obj.dob = dob
            jobseeker_obj.highest_qualification = qualification
            jobseeker_obj.percentage=percentage
            jobseeker_obj.passoutyear=passout
            jobseeker_obj.skills = skills
            jobseeker_obj.job_preference = job_preference
            jobseeker_obj.university = university
            jobseeker_obj.address = address
            jobseeker_obj.phone = phone
            jobseeker_obj.profile_img = image
            jobseeker_obj.user_id = userid
            jobseeker_obj.resume = cv
            jobseeker_obj.save()
            request.session['address'] = jobseeker_obj.address
            request.session['dob'] = jobseeker_obj.dob
            messages.success(request, "Updated profile successfully.Now you can use whole features of this portal.")
    return redirect('home')
```

- **forms.py**

```python
from django import forms
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth.models import User
from .models import Company


class UserSettingsForm(forms.ModelForm):
    class Meta:
        # model = UserSettings
        fields = ["dark_mode", "email_notifications", "two_factor_auth"]
        widgets = {
            "dark_mode": forms.CheckboxInput(attrs={"class": "form-check-input"}),
            "email_notifications": forms.CheckboxInput(attrs={"class": "form-check-input"}),
            "two_factor_auth": forms.CheckboxInput(attrs={"class": "form-check-input"}),
             }

class CustomPasswordChangeForm(PasswordChangeForm):
    old_password = forms.CharField(
        widget=forms.PasswordInput(attrs={"class": "form-control"}),
        label="Old Password"
       )
    new_password1 = forms.CharField(
        widget=forms.PasswordInput(attrs={"class": "form-control"}),
        label="New Password"
       )
    new_password2 = forms.CharField(
        widget=forms.PasswordInput(attrs={"class": "form-control"}),
        label="Confirm New Password"
       )

class UploadFileForm(forms.Form):
    file = forms.FileField()
```

- **Job_matching.py**

```python
from .models import company_joblist

def parse_list(text):
    """
    Parse a comma-separated string into a set of lowercase trimmed items.
    """
    if text:
        return set([item.strip().lower() for item in text.split(",") if item.strip()])
    return set()

def match_jobs(jobseeker):
    # Function to extract numeric percentage value from string
    def extract_percentage(value):
        try:
            return float(value.strip('%')) if value else 0.0
        except ValueError:
            return 0.0
    # Parse jobseeker skills and qualification
    seeker_skills = parse_list(jobseeker.skills)
    seeker_qual = jobseeker.highest_qualification.strip().lower() if jobseeker.highest_qualification else ""
    seeker_percentage = extract_percentage(jobseeker.percentage) if jobseeker.percentage else 0.0
    # Fetch all jobs (this could be optimized with a filter, but for now we'll do it in Python)
    all_jobs = company_joblist.objects.all()
    matching_jobs = []
    for job in all_jobs:
        job_skills = parse_list(job.skills_required)
        job_percentage_criteria = extract_percentage(job.percent_criteria)

        # Check for any common skills
        if not seeker_skills.intersection(job_skills):
            continue

        # Check eligibility: if job.highest_qualification exists, see if it matches the seeker qualification
        if job.highest_qualification:
            # Split eligibility criteria if multiple (assuming comma-separated)
            eligibility_set = parse_list(job.highest_qualification)
            if seeker_qual and seeker_qual not in eligibility_set:
                continue
            # Check if jobseeker's percentage meets or exceeds the job's percentage criteria
        if seeker_percentage < job_percentage_criteria:
            continue
        matching_jobs.append(job)
    return matching_jobs
```

- **extract_resume.py**

```python
import re


def extract_resume_details(content):
    """Extracts key details like name, skills, address, highest_qualification, job_preference, university name, date of birth, email, and phone number from the resume content."""
    details = {
        "name": "",
        "skills": "",
        "address": "",
        "highest_qualification": "",
        "passout_year": "",
        "percentage": "",
        "job_preference": "",
        "university": "",
        "dob": "",
        "email": "",
        "phone": ""
    }

    # Define patterns and keywords
    email_pattern = r"[a-zA-Z0-9+._-]+@[a-zA-Z0-9.-]+"
    phone_pattern = r"\+?\d{10,15}"
    # passout_year_pattern = r"\b(19\d{2}|20\d{2})\b"
    percentage_pattern = r"(\d{1,2}\.\d{1,2}\d{1,2})%"
    dob_patterns = [
        r"\b(\d{1,2})[-/ ](\d{1,2})[-/ ](\d{2,4})\b",  # Formats like DD/MM/YYYY, DD-MM-YYYY
        r"\b(\d{1,2})[-/ ](Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*[-/ ](\d{2,4})\b"  # Formats with month names
    ]
    qualification_keywords = ["Bachelor's Degree", "Integrated MCA", "MCA", "Master of Computer Application",
                              "PHD", "B.Sc Computer Science", "M.Sc Computer Science", "BTech Computer Science",
                              "M.Tech Computer Science"]
    job_preferences_keywords = ["Software Engineer", "Data Scientist", "Backend Developer",
                                "Frontend Developer","Project Manager"]
    university_keywords = ["University", "Institute", "College"]
    skills_keywords = ["Python", "Php", "Java", "C++", "Django", "SQL",
                       "Machine Learning", "Artificial Intelligence", "React",
                       "ReactJS", "Node", "NodeJS", "Bootstrap", "JavaScript",
                       "HTML", "CSS", "Git", "Data Analytics", ".Net", "Oracle"]
    address_keywords = ["State", "Country", "District"]
    indian_states = ["Andhra Pradesh", "Arunachal Pradesh", "Assam", "Bihar", "Chhattisgarh", "Goa", "Gujarat", "Haryana", "Himachal Pradesh", "Jharkhand", "Karnataka", "Kerala", "Madhya Pradesh",
                     "Maharashtra", "Manipur", "Meghalaya", "Mizoram", "Nagaland",
                     "Odisha", "Punjab", "Rajasthan", "Sikkim", "Tamil Nadu",
                     "Telangana", "Tripura", "Uttar Pradesh", "Uttarakhand",
                     "West Bengal"]

    # Use a set for skills to prevent duplicates
    skills_found = set()

    # Process content line-by-line
    lines = content.split("\n")
    for line in lines:
        line = line.strip()
        if not line:
            continue

        # Extract name: assume first occurrence of two or more capitalized words is the candidate's name
        if not details["name"]:
            name_match = re.match(r"^([A-Z][a-z]+(?:\s+[A-Z][a-z]+)+)", line)
            if name_match:
                details["name"] = name_match.group(1)

        # Extract email
        if not details["email"]:
            email_match = re.search(email_pattern, line)
            if email_match:
                details["email"] = email_match.group()

        # Extract phone
        if not details["phone"]:
            phone_match = re.search(phone_pattern, line)
            if phone_match:
                details["phone"] = phone_match.group()

        # # Extract date of birth
        # if not details["dob"]:
        #     dob_match = re.search(dob_patterns, line, flags=re.IGNORECASE)
        #     if dob_match:
        #         details["dob"] = dob_match.group(1)
        # Extract date of birth and format to YYYY-MM-DD (for HTML date input)
        if not details["dob"]:
            for pattern in dob_patterns:
                dob_match = re.search(pattern, line, flags=re.IGNORECASE)
                if dob_match:
                    try:
                        if len(dob_match.groups()) == 3:
                            day, month, year = dob_match.groups()
                            if month.isdigit():
                                formatted_date = f"{int(year):04d}-{int(month):02d}-{int(day):02d}"
                            else:
                                month_num = {
                                    "Jan": "01", "Feb": "02", "Mar": "03", "Apr": "04", "May": "05", "Jun": "06",
                                    "Jul": "07", "Aug": "08", "Sep": "09", "Oct": "10", "Nov": "11", "Dec": "12"
                                }.get(month[:3].capitalize(), "")
                                if month_num:
                                    formatted_date = f"{int(year):04d}-{month_num}-{int(day):02d}"
                                else:
                                    continue
                            details["dob"] = formatted_date
                            break
                    except ValueError:
                        continue

        # Extract address if line contains address-related keywords or Indian states
        if not details["address"]:
            if any(kw.lower() in line.lower() for kw in address_keywords) or any(state.lower() in line.lower()
                                                                                 for state in indian_states):
                details["address"] = line

        # Extract highest qualification
        if not details["highest_qualification"]:
            for qualification in qualification_keywords:
                if qualification.lower() in line.lower():
                    details["highest_qualification"] = qualification
                    break

        # Extract percentage
        if not details["percentage"]:
            percentage_match = re.search(percentage_pattern, line)
            if percentage_match:
                details["percentage"] = percentage_match.group()

        # Extract job preference
        if not details["job_preference"]:
            for job in job_preferences_keywords:
                if job.lower() in line.lower():
                    details["job_preference"] = job
                    break

        # Extract university: store the entire line if it contains any keyword
        if not details["university"]:
            for uni_kw in university_keywords:
                if uni_kw.lower() in line.lower():
                    details["university"] = line
                    break

        # Accumulate skills from the line
        for skill in skills_keywords:
            if skill.lower() in line.lower():
                skills_found.add(skill)

    details["skills"] = ", ".join(sorted(skills_found))
    return details
```

- **Auto-apply.py**

```python
from django.core.mail import send_mail
from django.utils.timezone import now

from .job_matching import match_jobs
from .models import JobApplication

def auto_apply_jobs(jobseeker, max_apply=4):
    today = now().date()

    # Get job seeker's preference (convert to lowercase for case-insensitive matching)
    job_preference = jobseeker.job_preference.lower() if jobseeker.job_preference else ""

    # Get applications already made today
    today_applied_jobs = JobApplication.objects.filter(jobseeker=jobseeker, applied_at__date=today)
    today_applied_jobs_count = today_applied_jobs.count()

    # **Enforce max 4 applications upfront**
    if today_applied_jobs_count >= max_apply:
        print(f"⚠️ Jobseeker {jobseeker.id} already applied for {today_applied_jobs_count} jobs today. No more applications allowed.")
        return 0, []

    # Get list of job IDs already applied for today (prevents duplicate applications for the same job)
    applied_job_ids = set(today_applied_jobs.values_list('company_joblist_id', flat=True))

    # Get matching jobs based on skills and eligibility
    matching_jobs = match_jobs(jobseeker)

    # Exclude jobs already applied for today and ensure job is still open
    available_jobs = [
        job for job in matching_jobs
        if job.id not in applied_job_ids  # Prevent duplicate job applications
        and job.Lastdate >= today  # Ensure job is still open
    ]

    # **Sort jobs based on priority**
    def job_sort_key(job):
        is_preferred = job_preference in job.job_title.job_title.lower()
        return (
            not is_preferred,  # Prioritize job preference
            job.Lastdate,       # Prioritize nearest deadline
            job.dateofpublish or ""  # Prioritize recently published jobs
             )

    sorted_jobs = sorted(available_jobs, key=job_sort_key)

    applications_made = 0
    applied_jobs_list = []

    for job in sorted_jobs:
        # **Strictly enforce max-apply limit**
        if (today_applied_jobs_count + applications_made) >= max_apply:
            print(f"Max applications ({max_apply}) reached. Stopping further applications.")
            break

        # **Final check before applying**
        if JobApplication.objects.filter(jobseeker=jobseeker, company_joblist=job).exists():
            print(f"Already applied to {job.job_title.job_title} at {job.company.name}. Skipping...")
            continue

        try:
            # Apply for the job
            JobApplication.objects.create(jobseeker=jobseeker, company_joblist=job)
            applications_made += 1  # Track successful applications
            applied_jobs_list.append(job)

            # Add this job to the exclusion list for future checks
            applied_job_ids.add(job.id)
            print(f"Applied to: {job.job_title.job_title} at {job.company.name}")

        except Exception as e:
            print(f"Error applying for {job.job_title.job_title}: {e}")
            continue

    # Send email notification if applications were made
    job_details_html = "".join([
    "<tr>"
    f"<td>🔹 <b>{job.job_title.job_title}</b></td>"
    f"<td>🏢 {job.company.name}</td>"
    f"<td>📍 {job.location}</td>"
    f"<td>💼 {job.job_type}</td>"
    f"<td>📅 {today}</td>"
    f"<td>⏳ {job.Lastdate}</td>"
   "</tr>"
    for job in applied_jobs_list
])

    email_body_html = f"""
<p>Dear {jobseeker.name},</p>
<p>You have successfully applied to the following jobs today via <b>JobAi</b>:</p>
<table border='1' cellspacing='0' cellpadding='5' style='border-collapse: collapse; text-align: left; width: 100%;'>
    <tr style='background-color: #f2f2f2;'>
        <th>Job Title</th>
        <th>Company</th>
        <th>Location</th>
        <th>Job Type</th>
        <th>Application Date</th>
        <th>Deadline</th>
    </tr>
    {job_details_html}
</table>
<p>🚀 <b>Next Steps:</b></p>
<ul>
    <li>📌 Check your application status in your <a href='http://127.0.0.1:8000/jobseeker_login/'>JobAi Profile</a>.</li>
    <li>📌 Prepare for interviews with our <b>AI Mock Interview Tool</b>.</li>
</ul>
<p>Thank you for using <b>JobAi</b>. We wish you success!</p>
<p>Best Regards,<br><b>JobAi Team</b><br>📧 support@jobai.com | 🌐 <a href='http://127.0.0.1:8000/jobseeker_login/'>www.jobai.com</a></p
"""
    if applications_made!=0:
        send_mail(
    subject="Job Application Confirmation - Your Applications for Today",
    message="This is an HTML email. Please enable HTML to view the content properly.",
    from_email="jobai.prksolutions@gmail.com",
    recipient_list=[jobseeker.email],
    fail_silently=False,
    html_message=email_body_html
)
    return applications_made, applied_jobs_list
```

- **ATS Checker and Process Automation.py**

```python
from django.conf import settings
from django.shortcuts import get_object_or_404
from django.http.response import JsonResponse
from django.contrib import messages
from django.core.mail.message import EmailMessage
import os
import json
import re
import openai
from docx import Document
from reportlab.lib.pagesizes import letter
import comtypes  # Windows only (Bundled below)
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.platypus import paragraph, Spacer, Table, TableStyle, Image, SimpleDocTemplate
from reportlab.lib import colors  # Fixed import
from .models import JobApplication, jobseeker_profile

def convert_docx_to_json(docx_path, filename):
    """Convert a .docx file to JSON format and save it to media folder"""
    document = Document(docx_path)
    data = {"paragraphs": []}

    for para in document.paragraphs:
        data["paragraphs"].append(para.text)

    json_path = os.path.join(settings.MEDIA_ROOT, "json", filename + ".json")

    try:
        with open(json_path, "w") as json_file:
            json.dump(data, json_file, indent=4)
    except Exception as e:
        print(f"Error saving JSON file: {e}")

    return json.dumps(data, indent=4)
# Create your views here.

def convert_doc_to_docx(doc_path):
    """Converts a DOC file to DOCX using comtypes (Windows only) """
    word = comtypes.client.CreateObject('Word.Application')#pylint:disable=undefined-variable
    doc = word.Documents.Open(doc_path)
    docx_path = doc_path + "x"
    doc.SaveAs(docx_path, FileFormat=16)  # 16 = docx format
    doc.Close()
    word.Quit()
    return docx_path

def extract_text_from_docx(docx_path):
    """Extracts text from a DOCX file."""
    doc = Document(docx_path)
    return "\n".join([para.text for para in doc.paragraphs])

def extract_resume_text(profile_id):
    """Fetches the resume path from the database and extracts text."""
    jobseeker = get_object_or_404(jobseeker_profile, id=profile_id)
    resume_path = os.path.join(settings.MEDIA_ROOT, str(jobseeker.resume))

    if resume_path.endswith(".docx"):
        return extract_text_from_docx(resume_path)
    elif resume_path.endswith(".doc"):
        docx_path = convert_doc_to_docx(resume_path)
        return extract_text_from_docx(docx_path)

    return None

def calculate_ats_score(resume_text, job_description):
    prompt = f"""
    You are an ATS resume screening AI. Evaluate the resume against the job description.

    **Scoring Criteria**:
    - **Skills and Qualifications match(50%)**
    - **Formatting & Readability (15%)**
      **Grammar & Clarity (40%)**

    minimum 1/3 of application must be accepted or increase ats score of 0 for same job

    **Resume**:
    {resume_text}

    **Job Description**:
    {job_description}

    Provide:
    1. **ATS Score** (out of 100)
    2. **Improvement Suggestions** (if score < 90)
    """

    response = openai.ChatCompletion.create( #pylint : disable=undefined-variable
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}],
        temperature=0
        )

    ats_result = response["choices"][0]["message"]["content"]
    ats_score = int(re.search(r'\b(\d{1,3})\b', ats_result).group(1))
    return ats_score

def generate_admit_card(jobseeker, job_title, application_id):
    """Generates an Admit Card PDF with jobseeker details, company logo, and profile image."""

    # File Path for PDF
    file_name = f"{jobseeker.name.replace(' ', '_')}_admit_card.pdf"
    file_path = os.path.join(settings.MEDIA_ROOT, "admit_cards", file_name)
    os.makedirs(os.path.dirname(file_path), exist_ok=True)

    application = get_object_or_404(JobApplication, id=application_id)

    # Create PDF Document
    pdf = SimpleDocTemplate(file_path, pagesize=letter)  # ✅ Now this works
    elements = []
    styles = getSampleStyleSheet()

    # Title
    title = Paragraph("<b>Aptitude Test Admit Card</b>", styles['Title'])

    # Load Company Logo
    logo_path = application.company_joblist.company.profile_img.path if application.company_joblist.company.profile_img else None
    company_logo = Image(logo_path, width=120, height=60) if logo_path and os.path.exists(logo_path) else Paragraph("<b>[Company Logo Not Available]</b>", styles['Normal'])

    # Load Jobseeker Profile Image
    profile_path = jobseeker.profile_img.path if jobseeker.profile_img else None
    applicant_image = Image(profile_path, width=100, height=100) if profile_path and os.path.exists(profile_path) else Paragraph("<b>[Applicant Image Not Available]</b>", styles['Normal'])

    # Table for Header with Company Logo & Title Alignment
    header_table = Table([[company_logo, title]], colWidths=[130, 400])
    header_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (0, 0), 'LEFT'),  # Align logo left
        ('ALIGN', (1, 0), (1, 0), 'CENTER')  # Align title center
    ]))
    elements.append(header_table)
    elements.append(Spacer(1, 20))  # Space between title and next section

    # Jobseeker Image Alignment (Top Right)
    applicant_image_table = Table([[applicant_image]], colWidths=[100], rowHeights=[100])
    applicant_image_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'RIGHT'),  # Align jobseeker image to the right
    ]))
    elements.append(applicant_image_table)
    elements.append(Spacer(1, 20))

    # Jobseeker Information Table
    data = [
        ["Applicant Name:", jobseeker.name],
        ["Email:", jobseeker.email],
        ["Phone:", jobseeker.phone],
        ["Qualification:", jobseeker.highest_qualification],
        ["Job Title:", job_title],
        ["Company:", application.company_joblist.company.name],
        ["Status:", "Selected"]
    ]

    table = Table(data, colWidths=[150, 300])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
        ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 8),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
    ]))
    elements.append(table)

    # Final Instructions
    instructions = Paragraph("""
        <br/><b>Instructions:</b><br/>
        Please bring this admit card along with required documents on the test day.
    """, styles['Normal'])
    elements.append(instructions)

    # Generate PDF
    pdf.build(elements)

    return file_path
```

CHAPTER 9
REFERENCES

9.1 REFERENCES

1. Ahmed, M., & Bakar, S. (2021). *Machine Learning-Based Resume Screening System for Recruitment*. International Journal of Advanced Computer Science and Applications, 12(5), 311–318. https://doi.org/10.14569/IJACSA.2021.0120540
2. Kulkarni, P., & Suryawanshi, S. (2020). *AI in Recruitment and Selection: Enhancing Talent Acquisition through Automation*. International Research Journal of Engineering and Technology (IRJET), 7(8), 508–513.
3. Zhang, Y., Zhao, L., & Chen, K. (2019). *A Deep Learning Based Applicant Tracking System for Intelligent Recruitment*. IEEE Access, 7, 159591–159602. https://doi.org/10.1109/ACCESS.2019.2951057
4. Indeed. (2023). *Job Search APIs & Automation Tools*. Retrieved from https://developer.indeed.com
5. Django Software Foundation. (2024). *Django Web Framework Documentation*. Retrieved from https://docs.djangoproject.com/
6. Stack Overflow Developer Survey. (2023). *Hiring Trends and Job Search Behavior*. Retrieved from https://survey.stackoverflow.co/2023
7. GitHub. (2023). *Sample Projects Using Django and AI in Recruitment Systems*. Retrieved from https://github.com
8. Kumar, V., & Singh, A. (2022). *An AI-Driven Framework for Smart Job Recommendation and Auto-Apply*. Journal of Emerging Technologies and Innovative Research (JETIR), 9(2), 451–460.
9. LinkedIn Talent Solutions. (2023). *The Future of Hiring: Using AI in the Recruitment Pipeline*. LinkedIn Reports. Retrieved from https://business.linkedin.com/talent-solutions