**ANUDIP FOUNDATION**

Internal Training Project Report

Project Title

**JOB DESCRIPTION AND RESUME MATCHER**

By

|  |  |
| --- | --- |
| Name | Enrollment No |
| Ashwini Pawar | AF0481829 |

Under Guidance

Of

Rajshri Thete

**ABSTRACT**

The recruitment process is a critical step for job seekers, particularly students and employers entering the job market. However, students often struggle to understand how well their resumes align with specific job descriptions. The Job Description and Resume Matcher System is a mini-project designed to assist students by analysing and comparing up to three resumes against a given job description. The system provides an easy-to-use platform where students can upload their resumes and receive a similarity score, helping them identify strengths and weaknesses in relation to specific job roles.

Developed using Django (Python web framework) with MySQL as the backend, the system utilizes Natural Language Processing (NLP) techniques to semantically analyse the content of resumes and job descriptions. By calculating cosine similarity, the system evaluates how well each resume matches the job requirements and displays a score out of 10. It also offers improvement suggestions to help students refine their resumes for better alignment with job expectations. The application features basic user functionalities such as registration, login, resume upload, and result visualization through a simple HTML interface.

Primarily designed for academic and career development purposes, this project benefits students and employees by offering a self-assessment tool for resume evaluation. It encourages them to tailor their resumes according to industry demands and gain insights into effective resume writing. Although limited to three resumes at a time, the system lays the foundation for scalable solutions in the future and showcases how machine learning can be applied to real-world career preparation.

**ACKNOWLEDGEMENT**

The project titled **“Job Description and Resume Matcher”** is a part of the academic requirements and has been successfully carried out by:

|  |  |
| --- | --- |
| Name | Enrollment No |
| Ashwini Pawar | AF0481829 |

Under the Guidance of **Ms. Rajshri Thete**

We are extremely grateful to our project guide, Ms. Rajshri Thete, for her constant support, guidance, and encouragement throughout the development of this project. Her expertise, constructive feedback, and valuable suggestions were crucial in shaping the direction of our work and ensuring its successful completion.

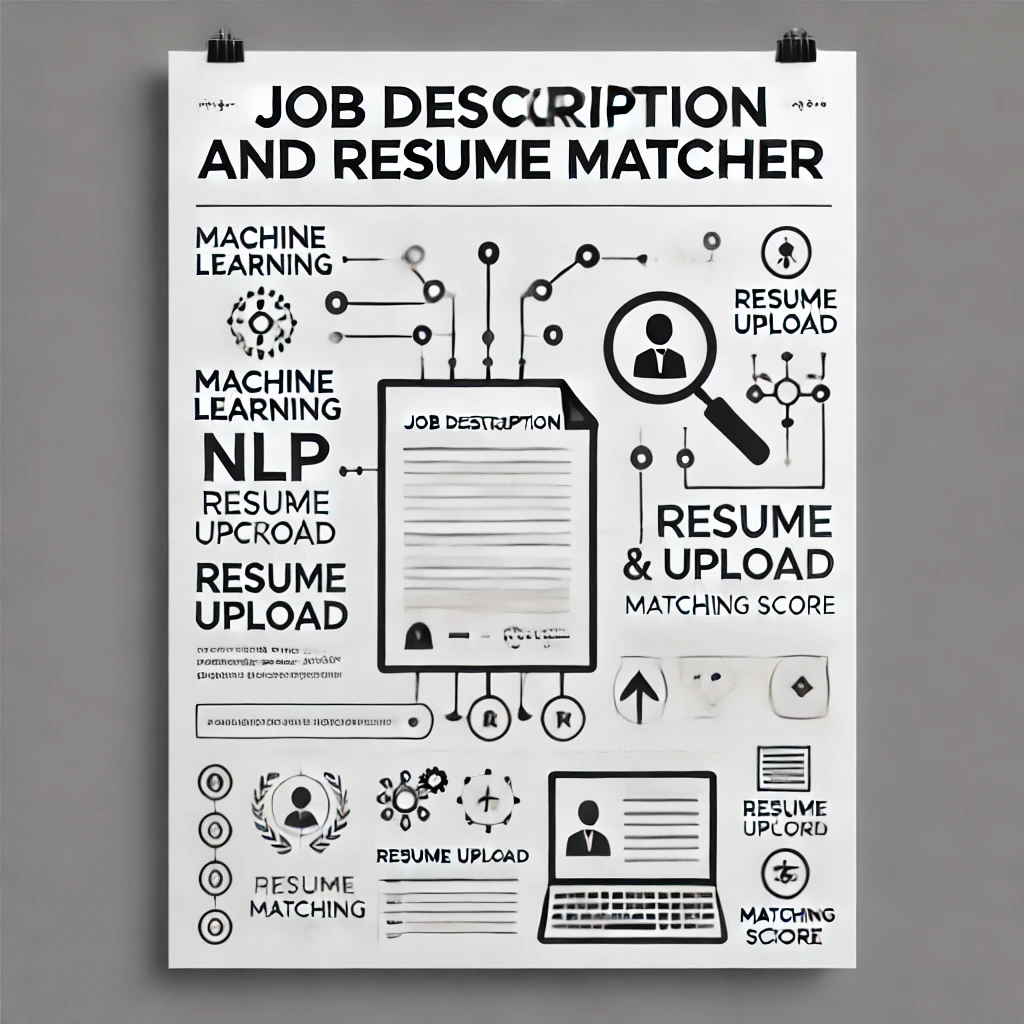
Furthermore, we extend our heartfelt gratitude to our families and friends for their continuous motivation and unwavering moral support during every phase of the project.

This project has been a significant learning experience, and we are thankful to all those who contributed to its successful execution.

**CONTENT OF THE PROJECT**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **TITLE** | **PAGE NO** |
| 1 | Title of Project | 1 |
| 2 | Abstract | 2 |
| 3 | Aknowledgement | 3 |
| 4 | Introduction /Objective | 6-7 |
|  | 4.1 Introduction | 6 |
|  | 4.2 Objective | 7 |
| 5 | System Analysis | 8 - 24 |
|  | 5.1 Problem Definition | 8 |
|  | 5.2 Prelimary Investigation | 9 |
|  | 5.3 Feasibility Study | 10 |
|  | 5.4 Project Planning | 13 |
|  | 5.5 Software Requirements Specifications | 14 |
|  | 5.6 Functional Requirements | 15 |
|  | 5.7 Data Model Design | 18 |
| 6 | System Design | 25 - 44 |
|  | 6.1 Database Design | 26 |
|  | 6.2 Procedural Design | 36 |
|  | 6.3 Outputs | 39 |
| 7 | Coding | 45 - 95 |
| 8 | Testing | 96 - 98 |
|  | 8.1 Testing Strategies | 97 |
| 9 | Reports | 99 |
| 10 | Future Scope | 100 |
| 11 | Conclusion | 101 |
| 12 | Bibilography and References | 102 |

**TITLE OF THE PROJECT**



**INTRODUCTION & OBJECTIVE OF THE PROJECT**

**2.1 INTRODUCTION**

In today’s digital and data-driven environment, the recruitment process is evolving rapidly with the integration of intelligent systems. Students seeking internships or jobs and employers looking for the right candidates often face the challenge of matching skills and qualifications efficiently. Manual screening of resumes against job descriptions can be time-consuming and inconsistent, especially when handling large volumes of applications.

The **Job Description and Resume Matcher** project addresses this problem by providing a web-based platform that allows users to upload resumes and compare them against a specified job description. The system uses Natural Language Processing (NLP) and Machine Learning (ML) techniques to semantically analyse the content of both resumes and job postings. By computing a similarity score out of 10, it offers an objective metric that reflects how well a resume fits a particular job. Additionally, the system suggests improvements, making it especially beneficial for students who are in the early stages of preparing for employment.

This project is developed using the Django framework with MySQL as the backend. The interface is simple and user-friendly, allowing users to register, log in, upload resumes, and instantly view matching results. Though the system currently supports uploading and comparing up to three resumes at a time, it serves as an effective tool for learning and self-assessment. Ultimately, this project demonstrates how ML and NLP can be applied to solve real-world problems in the field of career development and recruitment.

**2.2 OBJECTIVES**

The primary goal of this project is to automate the process of matching resumes with job descriptions. This system is particularly designed to assist students in assessing and improving their resumes based on job relevance. The objectives of the project are as follows:

1. To develop a web-based system that automates the matching of job descriptions and resumes.
2. To help students identify the best-fit job opportunities based on their skills and qualifications.
3. To assist employers in shortlisting candidates efficiently.
4. To provide a user-friendly interface for uploading and analysing resumes and job descriptions.
5. To implement a scoring algorithm that ranks candidates based on relevance to job criteria.
6. To integrate NLP tools for semantic analysis and keyword matching.
7. To improve the transparency, speed, and fairness of the recruitment process.
8. To suggest improvements to enhance resume quality based on the job description.
9. To encourage students to build industry-ready resumes and prepare effectively for job applications.

**SYSTEM ANALYSIS**

**3.1 PROBLEM DEFINITION**

Recruiters and students both struggle with efficiently matching qualifications to job requirements. Recruiters spend considerable time reviewing resumes manually, and students often apply for jobs without knowing their suitability. This results in inefficient hiring processes and missed opportunities.

This project addresses these problems by automating the resume screening process. Using NLP techniques, the system analyses and compares job descriptions with student resumes, producing a compatibility score. It helps employers filter candidates and allows students to target appropriate job postings more effectively.

* **Lack of Resume-Job Alignment Awareness for Students:**  
  Many students and job seekers are unaware of how effectively their resumes match specific job descriptions. This mismatch often results in missed opportunities, even for well-qualified candidates, due to the absence of structured feedback or guidance.
* **No Automated System to Assess Resume Relevance:**  
  There is no easy-to-use system available for students to automatically compare their resumes with job descriptions. Most tools fail to provide meaningful similarity scores that reflect actual job-relevance.
* **Recruitment Challenges Due to Language and Format Variations:**  
  Job descriptions and resumes often use different terminology and structures, making manual matching difficult. An automated tool using NLP and ML can bridge this gap by analysing and comparing both documents on a semantic level.

**3.2 PRELIMINARY INVESTIGATION**

**Purpose of the Project:**

The primary goal is to assist students and job seekers in evaluating how well their resumes align with specific job descriptions. This is done by creating an automated, web-based system that uses NLP and machine learning techniques to compare up to three resumes with a given job role and provide relevance scores along with improvement suggestions.

**Benefits to Users:**

* **Automated Matching:** Quickly identifies top candidates based on job requirements.
* **Objective Screening:** Reduces human bias by using consistent scoring metrics.
* **Skill Gap Analysis:** Students can receive feedback on missing or weak skills.
* **Time Efficiency:** Saves hours of manual resume scanning for employers.
* **Dual Access:** Caters to both students and company recruiters in one platform.

**3.3 PROPOSED SYSTEM :**

The proposed system is a **web-based application** designed to automate the process of comparing resumes with job descriptions using Natural Language Processing (NLP) and Machine Learning (ML) techniques. It is specifically tailored for students and early-career job seekers to help them assess and enhance their resumes based on the requirements of specific job roles.

**Key Features of the Proposed System:**

1. **Resume Upload Functionality:**
   * Users can upload up to **three resumes** in PDFor TXT format.
   * A job description is uploaded or pasted as a reference document.
2. **NLP-Based Text Extraction and Processing:**
   * The system uses NLP libraries (NLTK) to extract keywords, skills, and experience-related phrases from both the job description and the resumes.
3. **Semantic Similarity Analysis:**
   * Using **cosine similarity** and **TF-IDF (Term Frequency-Inverse Document Frequency)**, the system compares the textual content of resumes with the job description.
   * Each resume is assigned a **score out of 10** indicating how well it matches the job description.
4. **Result Visualization:**
   * Users receive an easy-to-understand output showing the score for each resume.
   * The system highlights **key matching sections** and provides **suggestions** for improvement.
5. **User Interface:**
   * Developed using **HTML/CSS with Django as the backend framework**, ensuring a smooth and responsive experience.
   * Basic functionality includes **registration**, **login**, **resume upload**, and **results display**.
6. **Database Support:**
   * User credentials, uploaded resumes, job descriptions, and similarity results are stored in a **MySQL database** for persistence and future analysis.

**3.4 FEASIBILITY STUDY**

A feasibility study evaluates the practicality and viability of the proposed system across multiple dimensions to ensure that it can be successfully developed and deployed.

**TYPES OF FEASIBILITY ANALYSIS**

* **Technical Feasibility**
* Built using Python (Django) for the backend and HTML, CSS for the frontend.
* NLP is handled using libraries like NLTK for parsing and keyword extraction.
* Resume parsing supported via PDF/Text extraction tools.
* Data stored in MySQL databases.
* **Economic Feasibility**
* Open-source tools and frameworks minimize development costs.
* Automated resume screening reduces recruitment overhead for companies.
* Scoring and skill analysis can help students improve employability without expensive coaching.
* **Operational Feasibility**
* Simple interface allows both tech-savvy and non-technical users to interact easily.
* Uploading job descriptions and resumes is straightforward.
* Match scores and recommendations make the platform informative and actionable.
* **Schedule Feasibility**
* Project timeline divided into design, development, testing, and deployment phases.
* Modular design allows for progressive feature additions like feedback systems or chatbots.
* Estimated development time: 8–10 weeks.
* **Social Feasibility**
* Helps bridge the gap between students and employers.
* Encourages career readiness and fair hiring practices.
* Reduces stress and uncertainty in the job application process.
* **Legal Feasibility**
* The system complies with data protection laws such as GDPR (if applicable) by ensuring secure storage and handling of personal information.
* No intellectual property issues arise from the use of open-source libraries.
* User agreements and privacy policies are drafted to inform users about data usage.
* **Is it technically feasible or not?**

The **Job Description and Resume Matcher** system is designed to be very user-friendly. Any user who understands simple English can easily operate the system without needing technical knowledge of frameworks, databases, or programming languages. The interface allows users to upload resumes and job descriptions with minimal effort. Therefore, we can conclude that the system is **operationally feasible**, as performing any operation on the platform is simple and does not involve any complex procedures.

* **Is it economically feasible or not?**

The project is **economically feasible** because it requires only minimal hardware and software resources. It can run on standard computers or servers without the need for expensive hardware upgrades. The software uses open-source or freely available technologies (e.g., Python, Flask, MySQL) and requires no costly licenses. Additionally, any necessary software dependencies will be installed automatically or are widely supported. Thus, the overall cost of development and deployment is low, making the project economically viable.

* **Is it schedule feasible or not?**
* The development of the **Job Description and Resume Matcher** system is estimated to take approximately **two months**. This timeline includes design, development, testing, and deployment phases. Given the project scope and available resources, this is a realistic and achievable timeframe, making the project **schedule feasible**.
* **Is it socially feasible or not?**
* Yes, the developed system holds **social feasibility** as it respects user privacy, does not perform any illegal operations, and complies with social and ethical standards. The system supports fair hiring practices by objectively matching resumes with job descriptions, thus contributing positively to recruitment processes without causing any social harm.

**ADVANTAGES OF FEASIBILITY STUDY**

Conducting a feasibility study offers several benefits before starting actual project development. For the **Job Description and Resume Matcher**, the following advantages were observed:

1. **Risk Identification and Management:**
   * Helped anticipate potential technical and operational challenges early, reducing the risk of failure.
2. **Improved Planning and Resource Allocation:**
   * Enabled structured planning of development stages, saving time and effort.
3. **Increased Confidence in Project Success:**
   * Gave stakeholders (students, guide, institution) assurance that the system was worth building.
4. **Clear Project Direction:**
   * Clarified system requirements, goals, and constraints, leading to focused and efficient development.
5. **Foundation for Decision Making:**
   * Provided an objective basis for moving forward with system design and implementation.
6. **Supports Scalability:**
   * Helped evaluate how the system can be extended in the future for larger resume datasets or recruiter access.

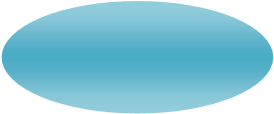
**3.5 PROJECT PLANNING**

**Purpose of Project Planning**

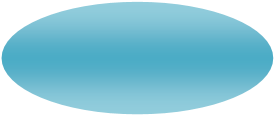
To ensure that the development of the Job Description and Resume Matcher proceeds in a structured and efficient manner. Planning covers tasks, timelines, risk management, and resource allocation.

**Phases Covered in the Plan**

1. **Preliminary Investigation** – Establishing goals and understanding core problems.
2. **System Analysis** – Gathering requirements, defining problems and user needs.
3. **System Design** – Creating database schemas, designing system architecture.
4. **Coding** – Implementing core features including NLP processing, matching algorithm, and user interfaces.
5. **Security** – Ensuring secure data handling for user resumes and job description.
6. **Testing** – Unit testing, integration testing, and user feedback testing.
7. **Implementation** – Hosting the system and onboarding initial users for real-world validation.



**Stop**



**Start**



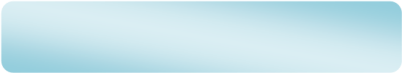
Preliminary



System Analysis



System Design



Coding



Testing



Security



Implementation

**3.4 SOFTWARE REQUIREMENTS APECIFICATIONS (SRS)**

The Software Requirement Specification (SRS) defines the core functionalities and technical details of the **Job Description and Resume Matcher** system. It ensures the system is scalable, maintainable, user-friendly, and effectively supports both **students** and **employees** in finding relevant job opportunities by matching resumes with job descriptions using NLP.

**System Overview**

This system provides a platform where **students** and **employees** can upload resumes and receive job matches based on their skills and experience. Employers can post job descriptions. The system analyses and compares the content using NLP techniques to calculate a **match score**.

The system is divided into the following two modules:

1. **User Module** – For students and employees to manage resumes and view job matches.
2. **Admin Module** – For system administrators to manage users, monitor activities, and view login/logout history.

**Software Requirement Specifications (SRS)**

**Software Requirements**

* **Frontend:** HTML, CSS, Bootstrap – For a responsive and intuitive interface.
* **Backend:** Python (Django) – For business logic and data processing.
* **Database:** MySQL– To store user profiles, resumes, job descriptions, logs, and scores.
* **NLP Tools:** NLTK – For resume and job description processing.
* **Resume Parser:** TDFID - For extracting structured text from resumes.

**Hardware Requirements**

* **Processor:** Intel i5 or above
* **RAM:** Minimum 8 GB
* **Storage:** At least 100 GB (to handle file storage and logs)
* **Internet:** Stable connection for real-time access and updates

**3.5 FUNCTIONAL REQUIREMENTS**

**1. User Module *(Students and Employees)***

Users can:

**User Registration and Authentication**

* New users can register by providing details such as name, email, phone number, user type (Student or Employee), and password.
* Email verification and strong password policies are enforced for security.
* Returning users can securely log in and log out of the platform.
* Forgot password and reset options are available.

**Resume Upload**

* Users can upload resumes in PDF, Text format.
* The system parses the resume to extract relevant information (skills, education, experience, etc.) for matching.

**Resume Matching and Score Generation**

* Once a job description is uploaded, users can select one or three resumes to compare against.
* The system uses NLP-based techniques to calculate **match scores** (e.g., percentage match, skill overlap).

**Match Report and Suggestions**

* Users receive a detailed **match report** that highlights:
  + Skills matched and unmatched
  + Missing qualifications

**2. Admin Module**

Admins have full control and visibility into user activity and system performance. Admins can:

* **Secure Login System**:
  + Authenticate and access the admin panel.
* **User Management**:
  + View, edit, or delete student/employee accounts.
  + Track total users, active users, and user types (student/employee).
* **Job & Resume Monitoring**:
  + View uploaded resumes and job descriptions
  + Manually validate or verify match scores
* **Login/Logout Tracking**:
  + Monitor user session history (login time, logout time, session duration).
  + Generate reports based on login frequency, user activity, or system usage.
* **System Monitoring**:
  + View number of matches made, most active users, most viewed jobs, etc.
* **Report Generation**:
  + Download usage reports.
  + Analyse system performance and user engagement trends.

**Software Engineering Paradigm**

The development of the **Job Description and Resume Matcher** system follows a structured and disciplined approach using the **Adapted Waterfall Model**. This model is chosen to ensure clarity, reliability, and phased progress throughout the system development lifecycle while allowing for iterative improvements based on feedback.

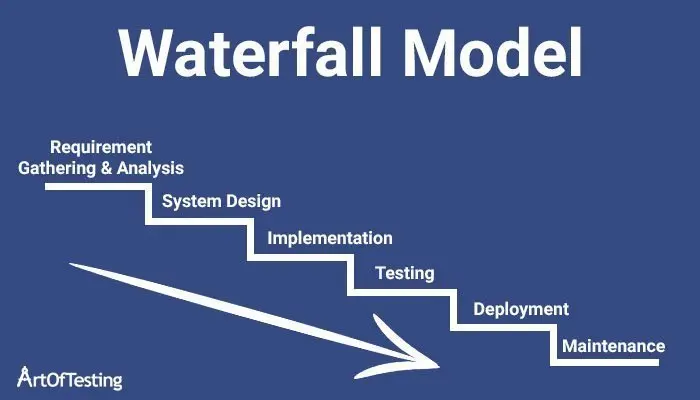
**Development Model: Adapted Waterfall Model**

Although the **Waterfall Model** traditionally involves a linear, phase-by-phase development process, this project introduces feedback loops particularly between the implementation and testing phases to refine features and fix issues promptly without derailing progress.

**Key Adaptations in the Waterfall Model:**

1. **Sequential Development Phases** – Each phase (requirement, design, coding, etc.) is completed before the next begins, maintaining a clear structure.
2. **Iterative Enhancements** – Testing feedback is looped back into the implementation to allow refinements.
3. **Defined Deliverables** – Each phase produces documentation, code, or modules that serve as input for the next stage.
4. **Practical Overlaps** – Minor overlaps are allowed between implementation and testing to reduce downtime and accelerate bug fixing.

Thus, the Waterfall Model with feedback mechanisms is chosen as the development paradigm for this application.



**Phase of Development**

**Requirement Gathering & Analysis**

* Identify stakeholders (students, employees, admin), and define system features such as resume upload, job matching, login history tracking, and report generation.

**System Design**

* Plan architecture using Django MVC, design MySQL database schema for resumes, jobs, users, and logs.
* Design user-friendly interfaces for each role.

**Implementation**

* Develop backend using Django and integrate MySQL. Create resume matching logic using NLP.
* Build responsive frontend with HTML, CSS, and Bootstrap.

**Testing**

* Perform unit and integration testing on modules like resume upload, match scoring, and login tracking.
* Validate data accuracy and system behaviour.

**Deployment & Maintenance**

* Hosting on a scalable environment.
* Continuous updates for feature enhancements.

**3.6 DATA FLOW DIAGRAM**

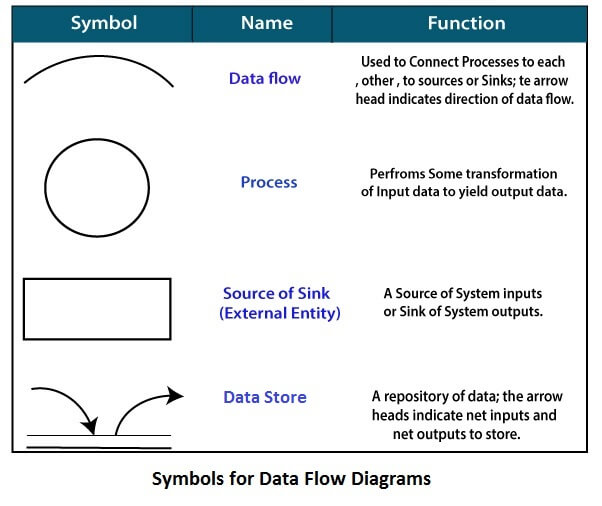
A Data Flow Diagram (DFD) is a traditional visual representation of the flow of data within the Job Description and Resume Matcher System. A neat and clear DFD can graphically depict the essential requirements of the system. It may include manual, automated, or hybrid processes.

It illustrates how data enters and exits the system, how information is processed, and where it is stored. This helps stakeholders understand the overall system structure. The objective of a DFD in this context is to show the boundaries and scope of the Job Matcher system. It serves as a communication tool between system analysts, developers, and users, and acts as a starting point for designing or redesigning the system. The DFD is also referred to as a *data flow graph* or *bubble chart*.

**The following observations about DFDs are essential:**

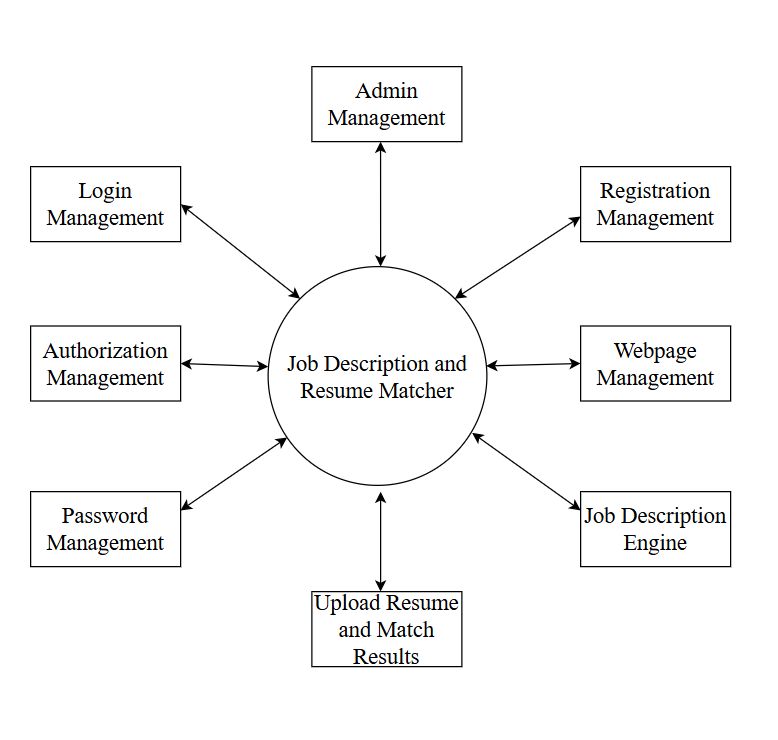
1. All names should be unique. This simplifies referencing and avoids confusion in identifying system elements.
2. DFD is not a flow chart. Unlike flowcharts, which show the order of execution, DFD arrows represent the flow of data only. The concept of sequence is not applicable in a DFD.
3. Suppress logical decisions. Do not include diamond-shaped decision boxes. Logical decision-making is implicit within the processing steps and should not be shown as conditional branches.
4. Avoid excessive detail. Focus on the key components. Defer handling of errors and exceptions until a later stage of system design.

**Standard DFD Symbols (adapted to this project):**

******

* **Circle (Process):** Represents a process that transforms input data into output data.  
  *Examples:* Job Description and Resume Matcher System
* **Curved Arrow (Data Flow):** Shows the flow of data into or out of a process or data store.  
  *Examples:* Resume Data, Job Description Data, Match Score
* **Parallel Lines (Data Store**): Represents a place where data is stored for retrieval and processing.  
  *Examples:* Resume Database, Job Description Repository, Match Result
* **Rectangle (Source or Sink):** External entity that acts as an input source or output destination.  
  *Examples:* Job Seeker (uploads resume).

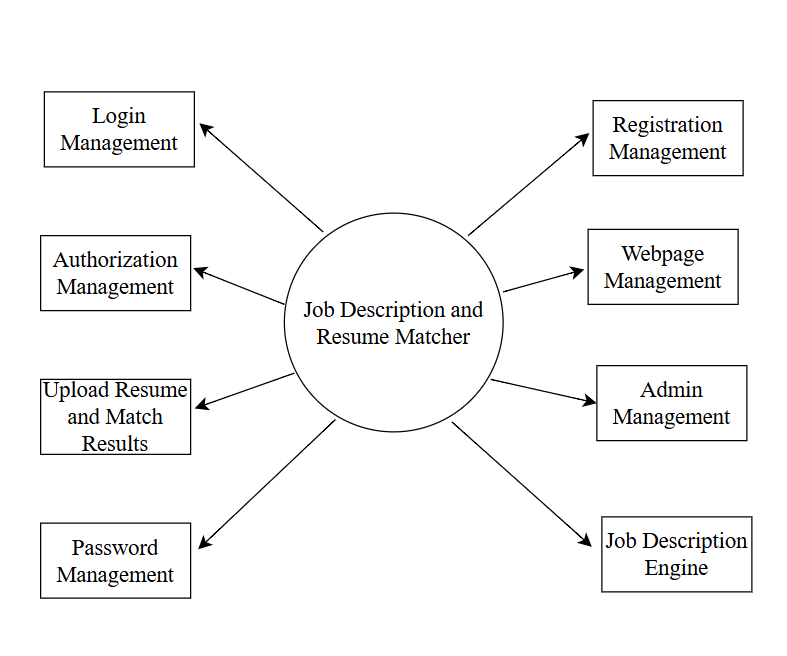
Zero Level DFD



The **Zero Level DFD of the Job Description and Resume Matcher System** illustrates the core functional processes that interact with the central system. These include **Admin Management, Registration Management, Login Management, Authorization Management, Password Management, Webpage Management, Job Description Engine, and Upload Resume and Match Results**.

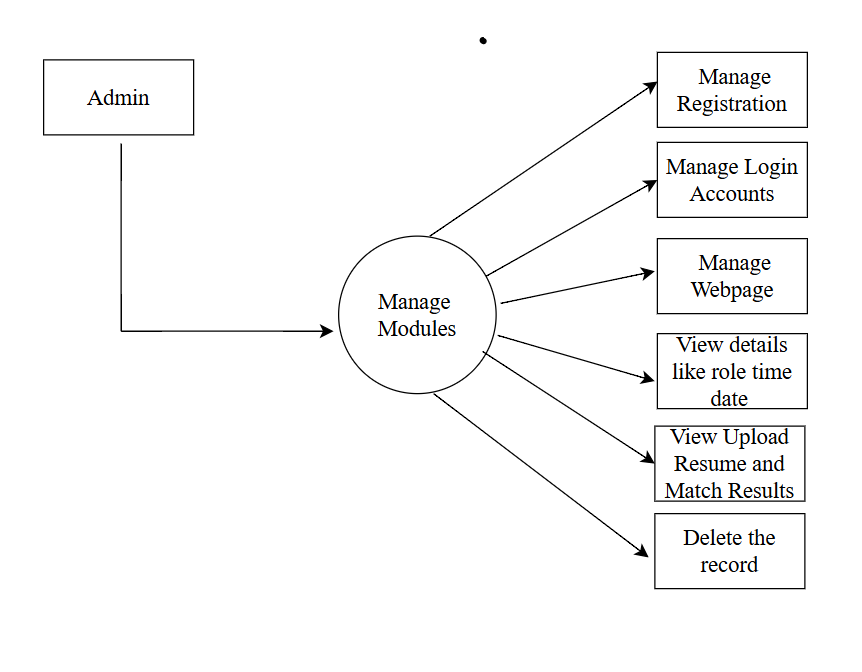
It highlights the bidirectional flow of data between the central process — Job Description and Resume Matcher — and its surrounding modules. Each external module either supplies essential input to the system or receives processed output.

First Level DFD



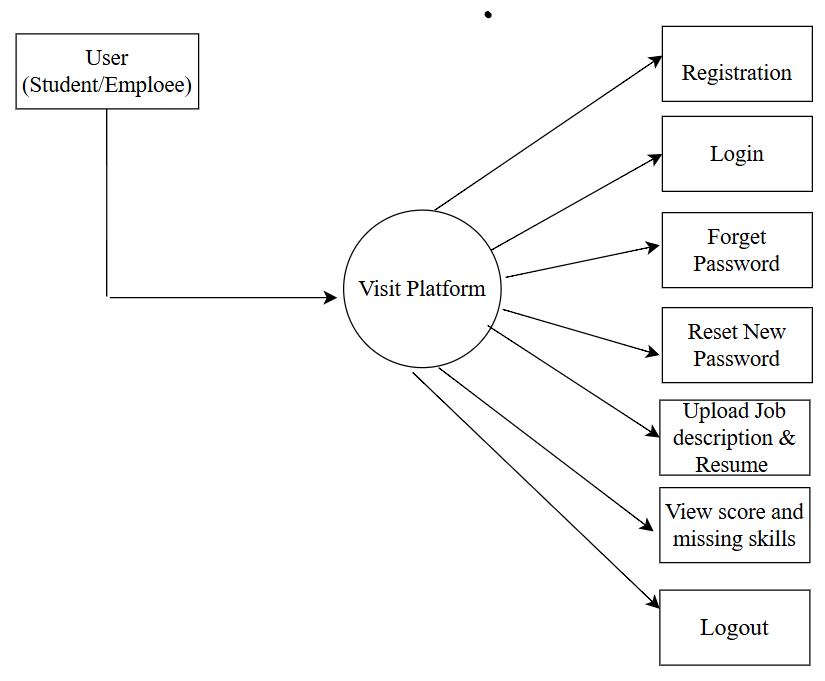
The First-Level DFD of the **Job Description and Resume Matcher** system illustrates how the system handles key functional modules such as **Login Management**, **Registration Management**, **Admin Management**, **Authorization Management**, and **Password Management**. It also connects to content-specific components including **Webpage Management**, **Job Description Engine**, and **Upload Resume and Match Results**. This level of the DFD provides a clear view of how data flows between these modules and the central process, ensuring a smooth and integrated workflow for managing and matching job descriptions with resumes effectively.

Second Level DFD



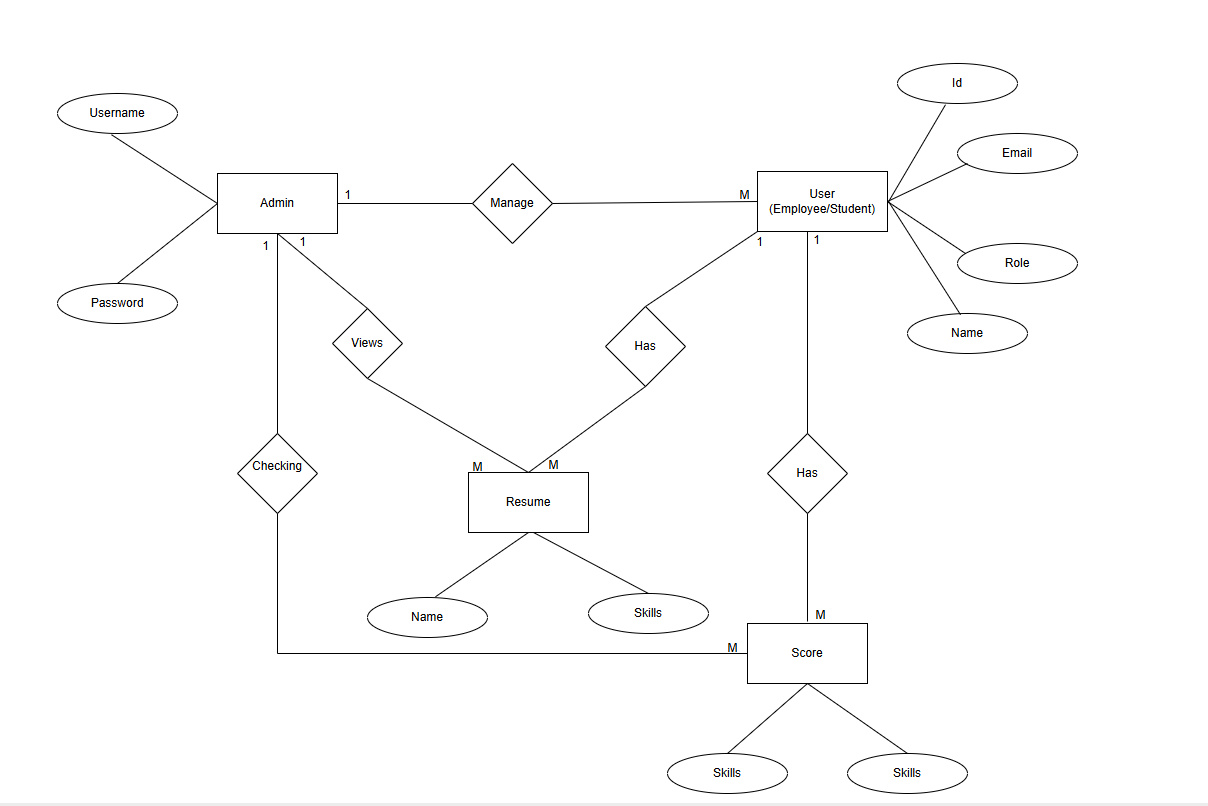
The Second-Level DFD for the Job Description and Resume Matcher system illustrates the internal operations performed by the Admin. After successful login, the Admin can manage user registrations, assign roles (student or employee), and view detailed match history including scores and skill gaps. The Admin also has control over managing login accounts, uploading resumes for job description matching, and monitoring the resulting match outcomes. Additionally, the Admin is responsible for maintaining public webpages and has the authority to update or delete user records and other data within the system to ensure efficient and secure portal management.

Second Level DFD



This **Second-Level DFD** illustrates the **User's interaction** with the system. When a user visits the website users can interact with various modules.

ER Diagram



1. **Users Module**

**SYSTEM DESIGN**

* **User Registration and Authentication**
* New users can register by providing details such as name, email, phone number, user type (Student or Employee), and password.
* **Resume Upload**
* Users can upload resumes in PDF, Text format.
* **Resume Matching and Score Generation**
* Once a job description is uploaded, users can select one or three resumes to compare against.
* **Match Report and Suggestions**
* Users receive a detailed **match report** that highlights:
  + Skills matched and unmatched
  + Missing qualifications

**2. Admin Module**

Admins have full control on user activity and system performance and can:

* **Secure Login System**: Authenticate and access the admin panel
* **User Management**:
  + Track total users, active users, and user types (student/employee)
* **Job & Resume Monitoring**:
  + View uploaded resumes and job descriptions
* **Login/Logout Tracking**:
  + Monitor user session history (login time, logout time, session duration)
* **System Monitoring**:
  + View number of matches made, most active users, most viewed jobs, etc.
* **Report Generation**:
  + Analyse system performance and user engagement trends.

**4.1 DATABASE STRUCTURE OF ALL MODULES**

The **Job Description and Resume Matcher System** contains **18 MySQL tables**: among these, **6 tables are custom-built** to handle core functionalities such as resumes, job descriptions, skills, matching scores, login history, and OTP verification. The other **12 tables are default tables automatically created by Django** for managing authentication, sessions, and permissions.

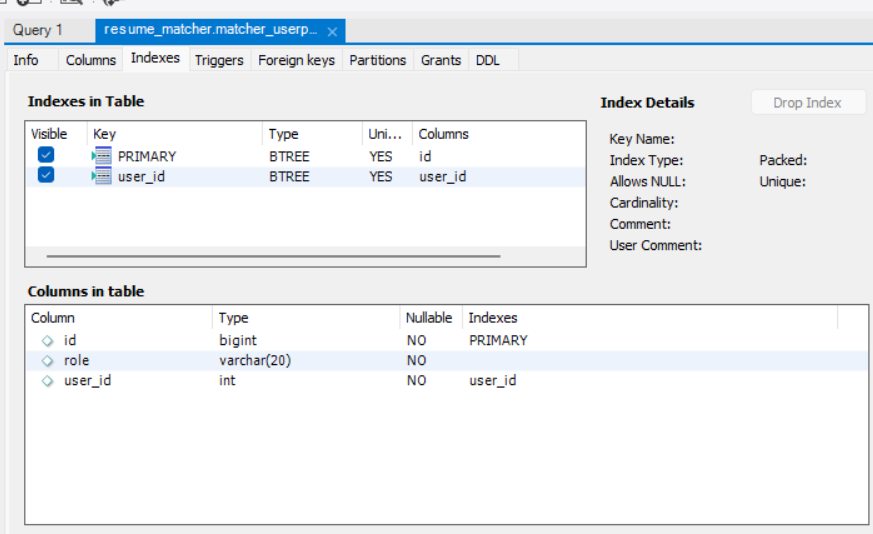
This database structure supports seamless CRUD (Create, Read, Update, Delete) operations on resumes and job postings, real-time matching processes, and secure user management, ensuring an effective and user-friendly experience.

**Customized Tables Details**

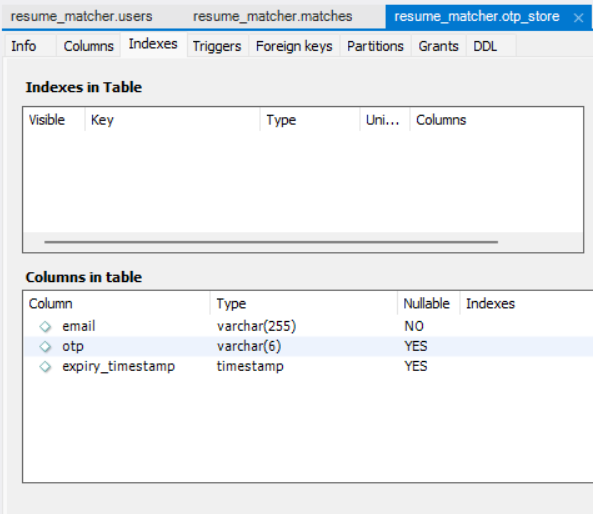
**Resume\_matcher.matches** : Stores match scores, resume name, missing skills



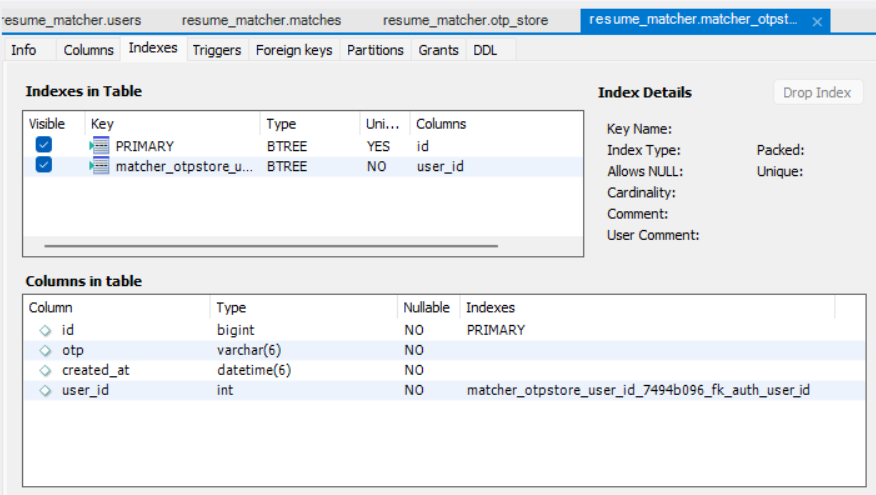
**Resume\_matcher.matcher\_userprofile** : Extends user with roles



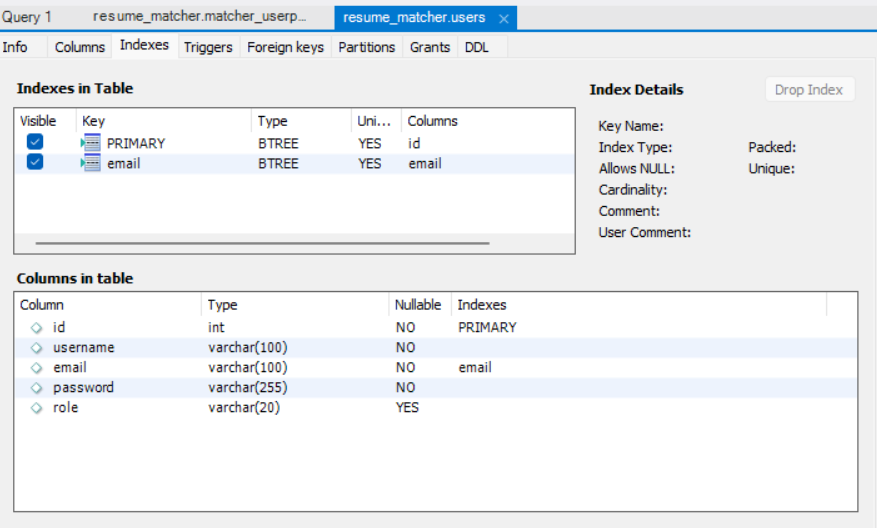
**Resume\_matcher. otp\_store :** Stores OTPs linked to users.



**Resume\_matcher. matcher\_otpstore** : Another OTP storage (maybe for admins or older logic)

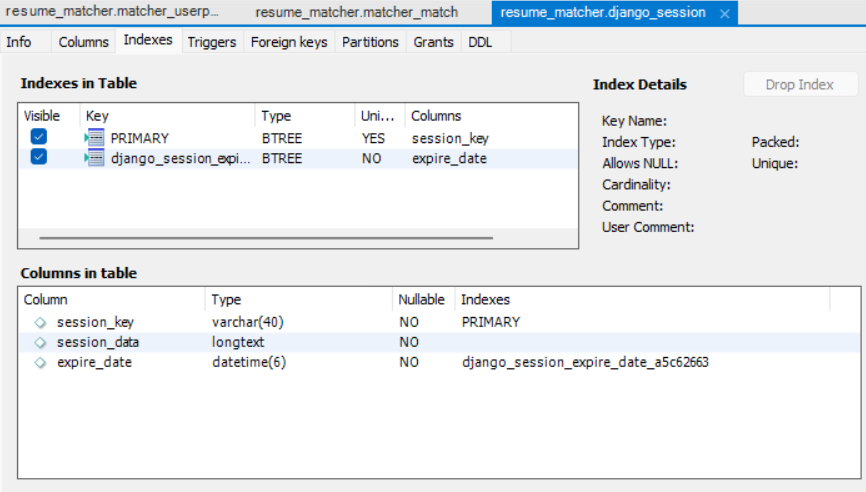


**Resume\_matcher.users :**



**Django Default Tables**

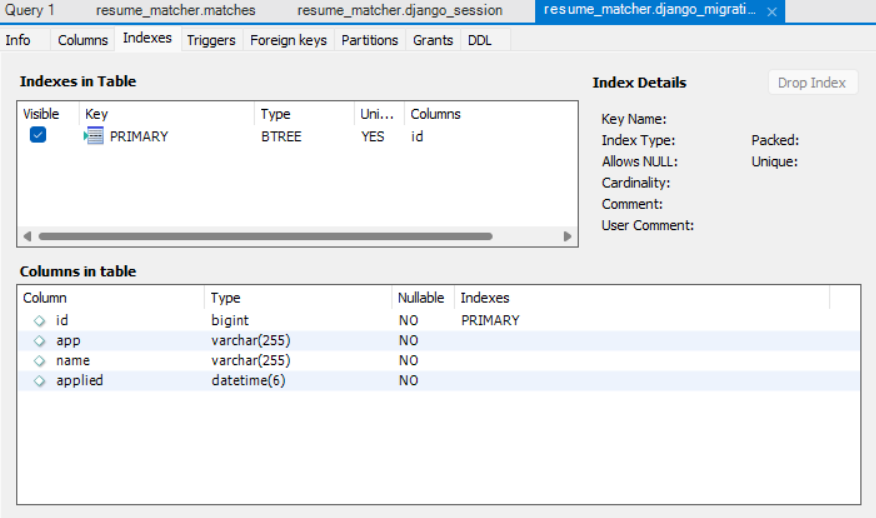
**Resume\_matcher.django\_session : Stores session data.**



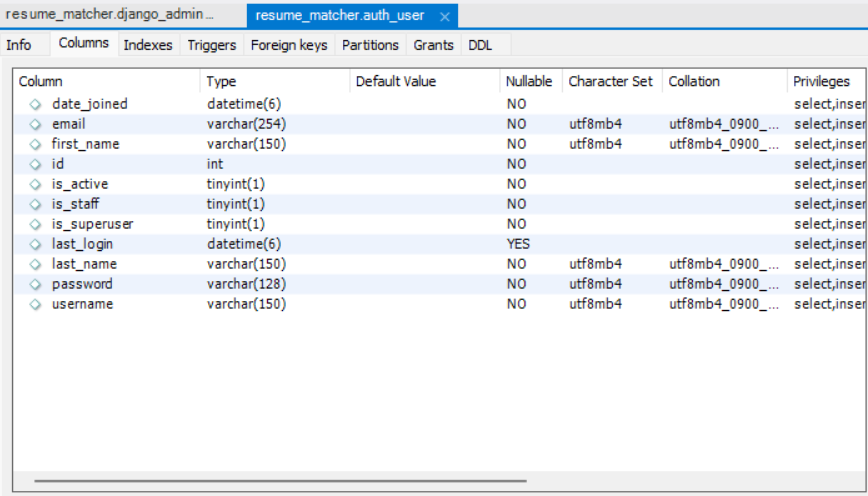
**Resume\_matcher.django\_admin\_log** : Logs admin actions.



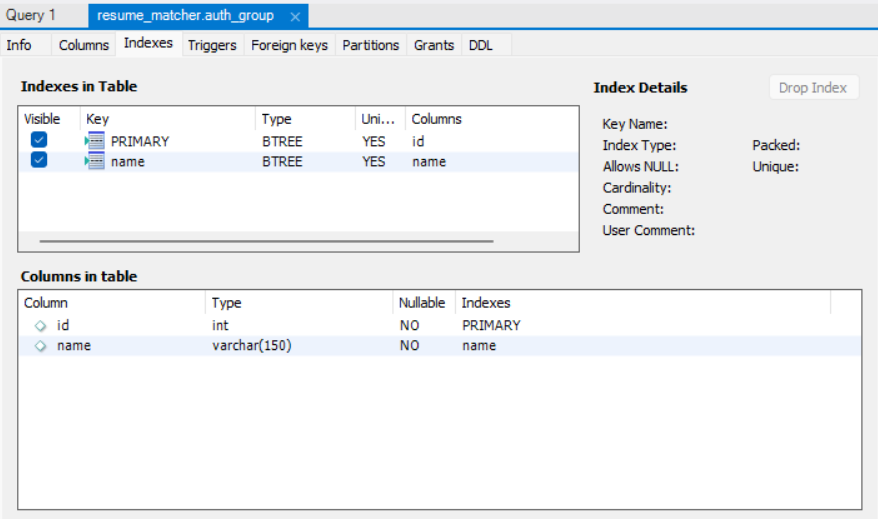
**Resume\_matcher.django\_migrations** : Tracks applied migrations.



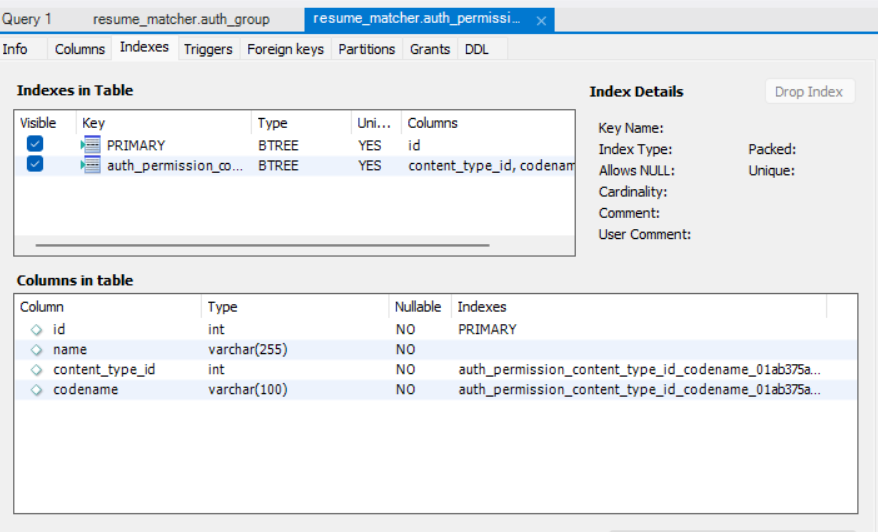
**Resume\_matcher.auth\_user** : Main user table used for login.



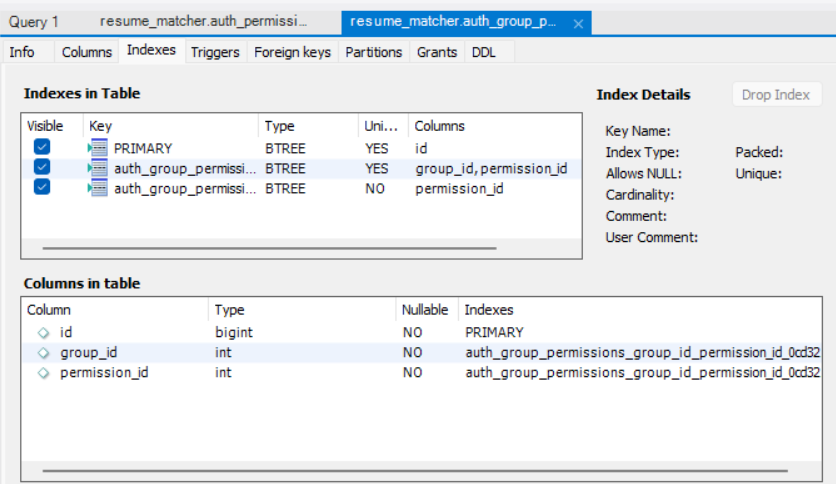
**Resume\_matcher.auth\_group** : User groups like admin, editor.



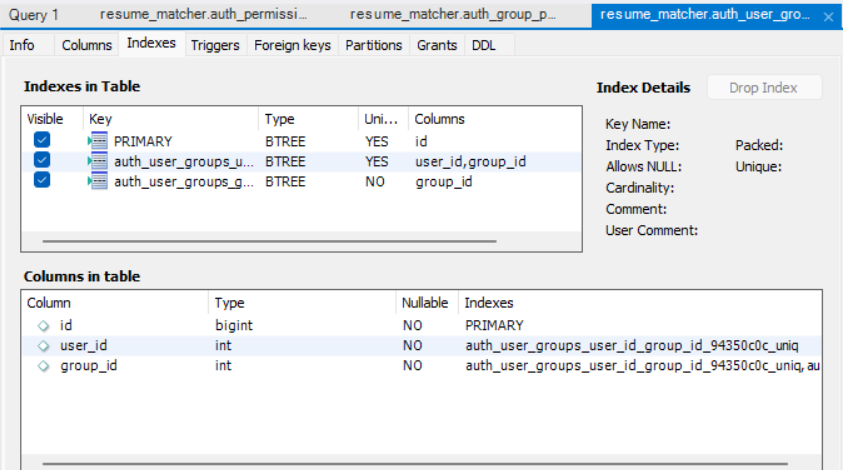
**Resume\_matcher.auth\_permission** : Permissions like add user, delete match.



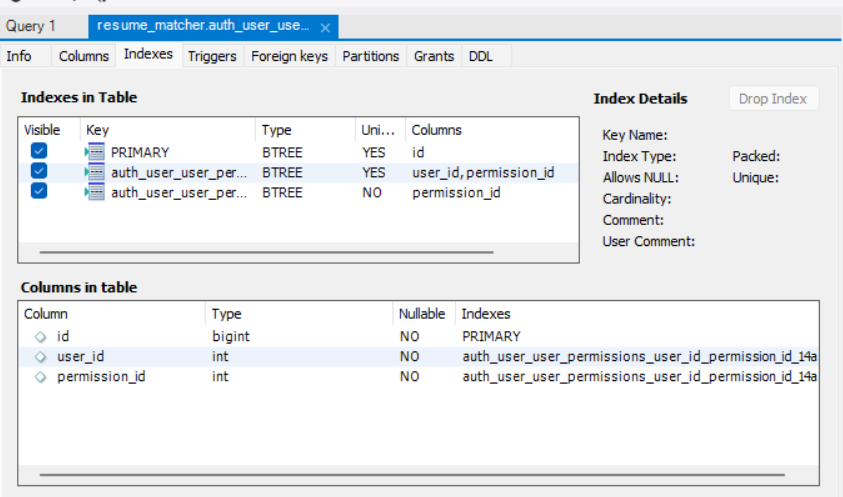
**Resume\_matcher.auth\_group\_permissions** : Join table for groups & permissions.



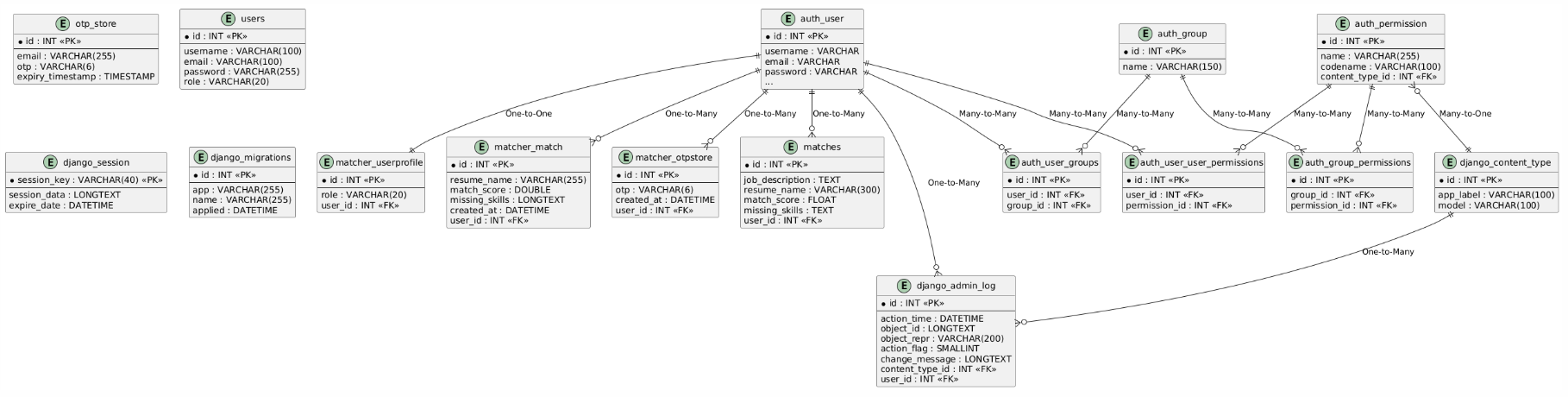
**Resume\_matcher.auth\_user\_groups** : Join table for users & groups.

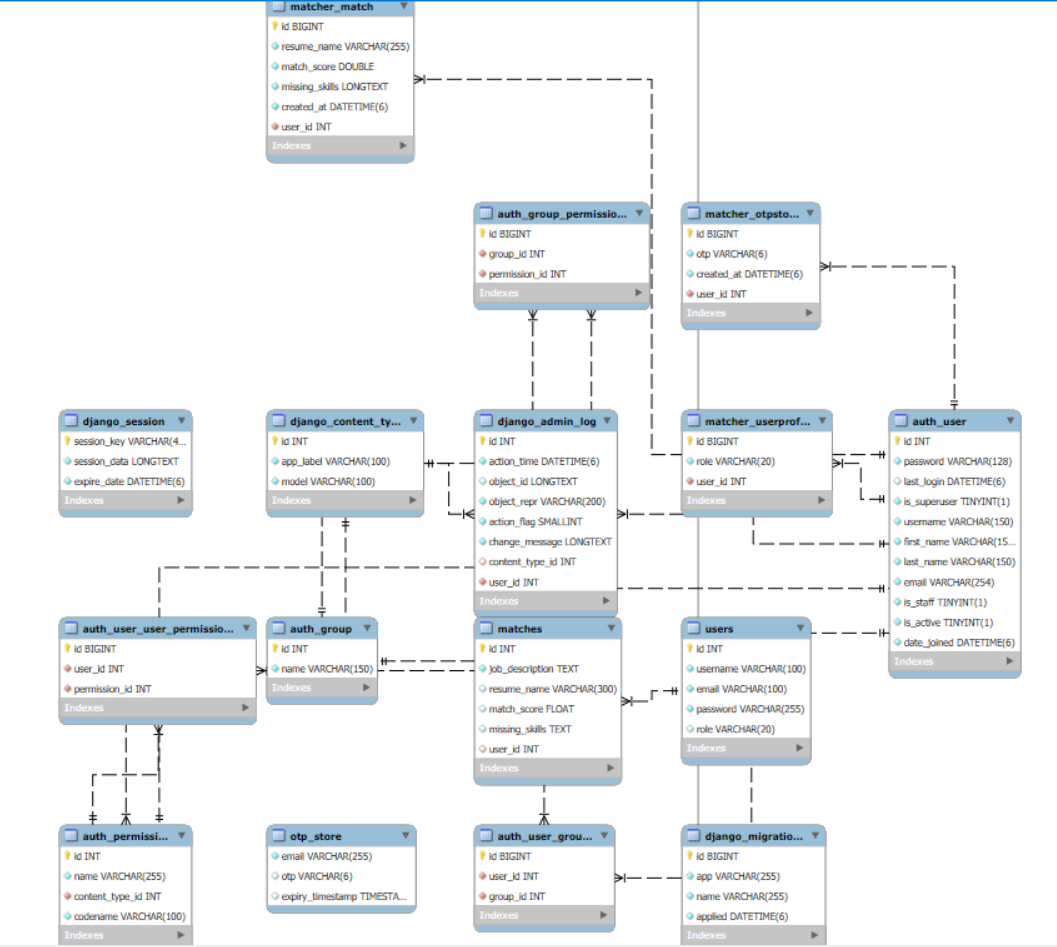


**Resume\_matcher.auth\_user\_user\_permissions :** Join table for users & permissions.

****

**Relationship between tables(Class Diagram)**





**4.2 PROCEDURAL DESIGN**

**4.3.1 User Panel Design**

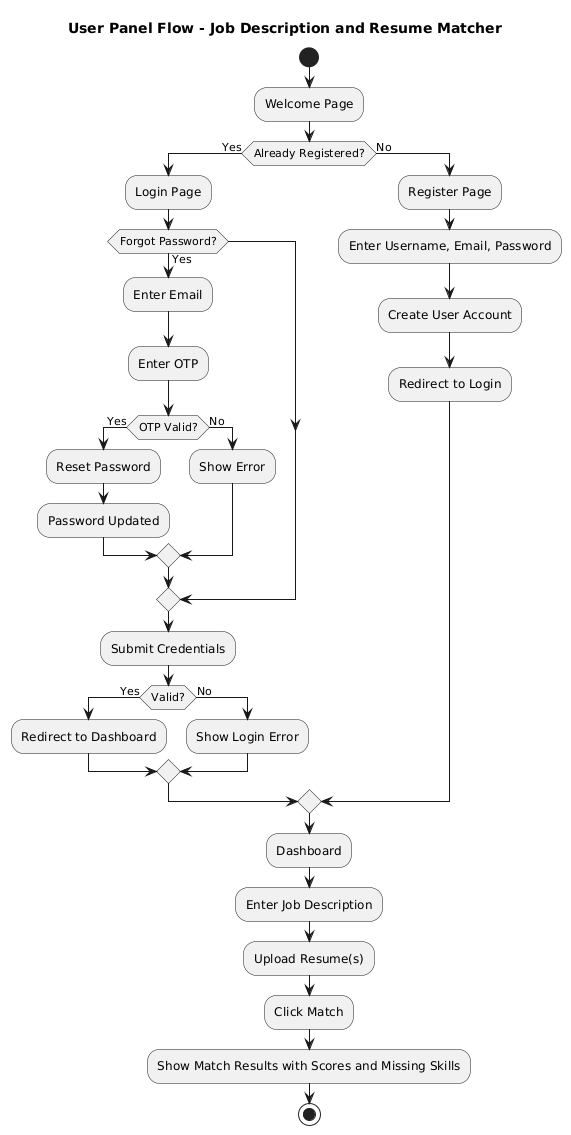
In the **user panel design** of our project, we have completed all the necessary functionalities for the user. Here, we provide a facility where students and employees can match their resumes with job descriptions. On the **index page**, the user can enter a job description and upload one or three resumes in PDF format. After submitting, the system shows the match score and highlights the missing skills from the resume compared to the job description.

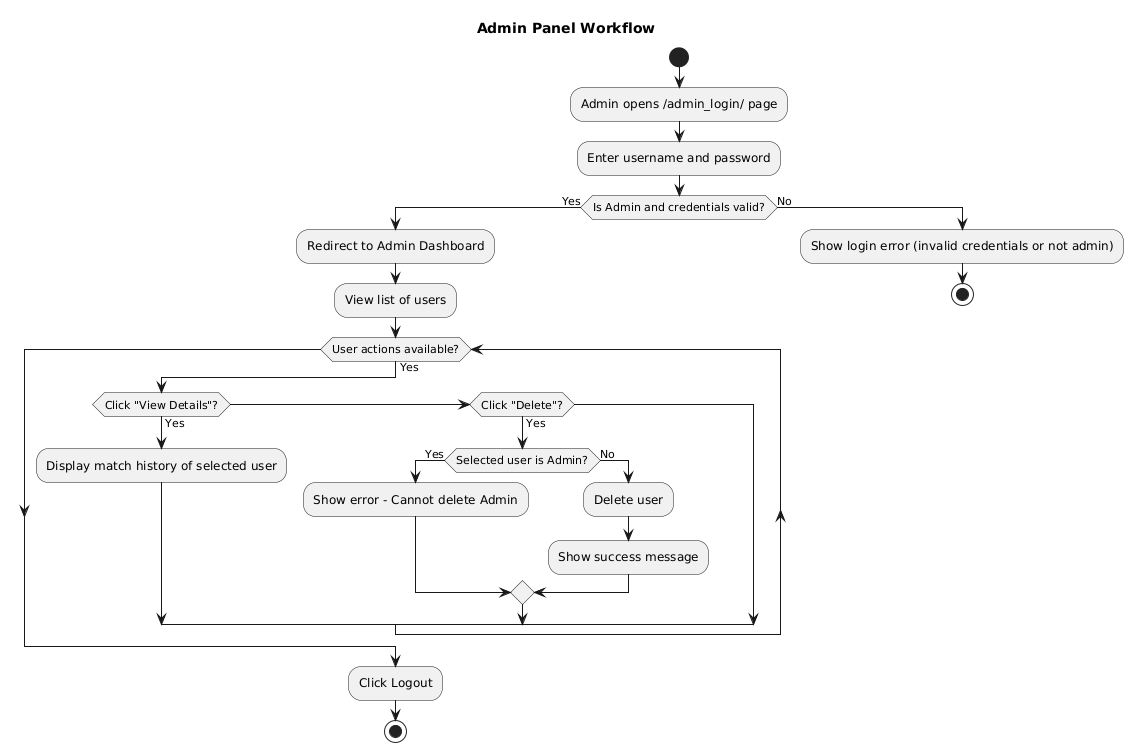
By selecting the available options, the user can also view missed skills, update their profile, or reset their password if needed. Through a clean and easy-to-use interface, users can access all the important features and get useful insights about how well their resume fits the job requirements. This helps users improve their resumes and increase their chances of getting hired.

**4.3.2 Admin Panel Design**

In the **admin panel design**, we have provided all essential functionalities to manage the system efficiently. The admin can log in through a dedicated admin login page. After successful authentication, the admin is redirected to the **admin dashboard** where they can view all registered users and the match history of each user. This gives the admin a clear overview of how the system is being used.

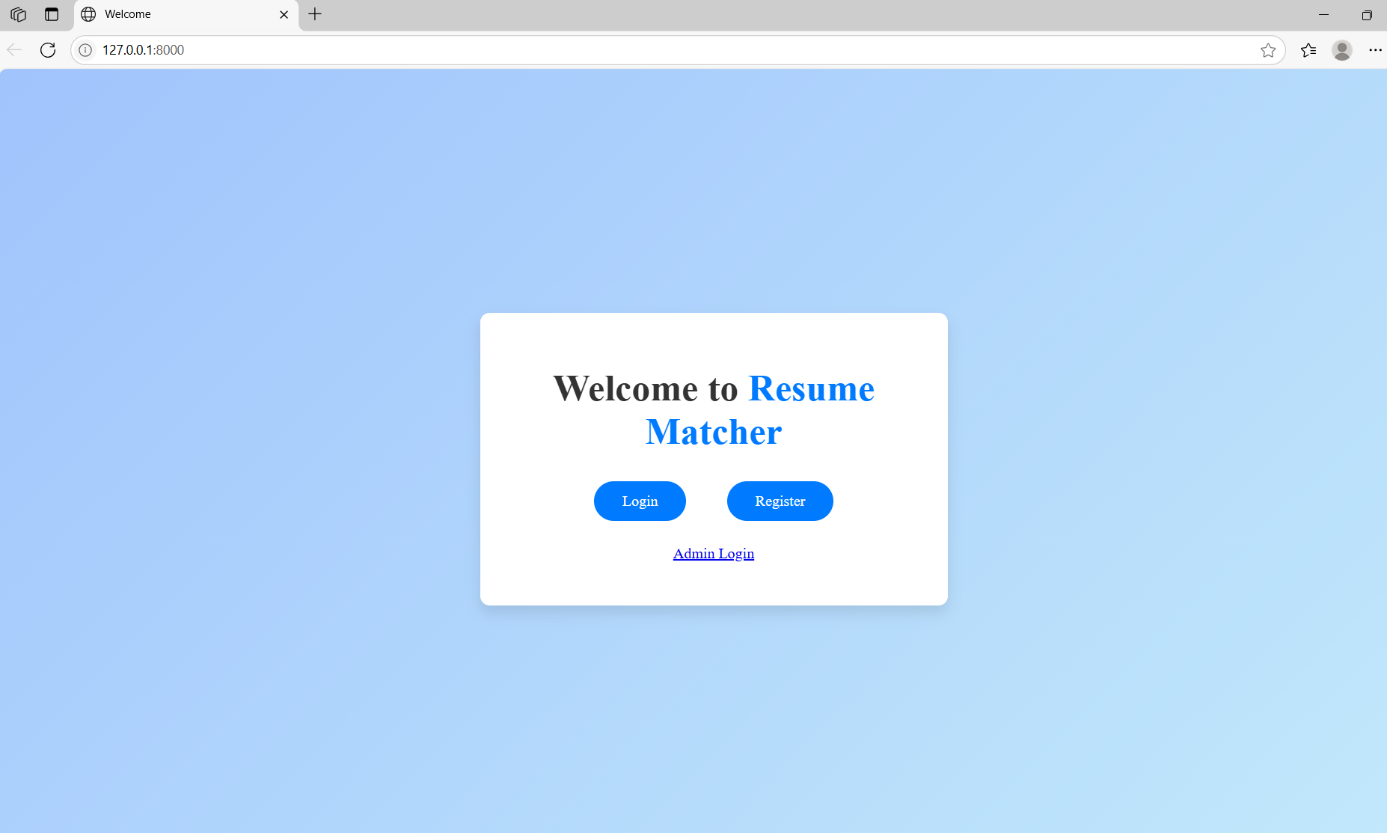
From the dashboard, the admin can also view individual user details, monitor their resume matching activities, and delete users if necessary (except other admins). This panel helps ensure smooth administration and maintenance of the platform. Through the admin interface, all data is presented in an organized way, allowing effective control over the user base and resume matching records.



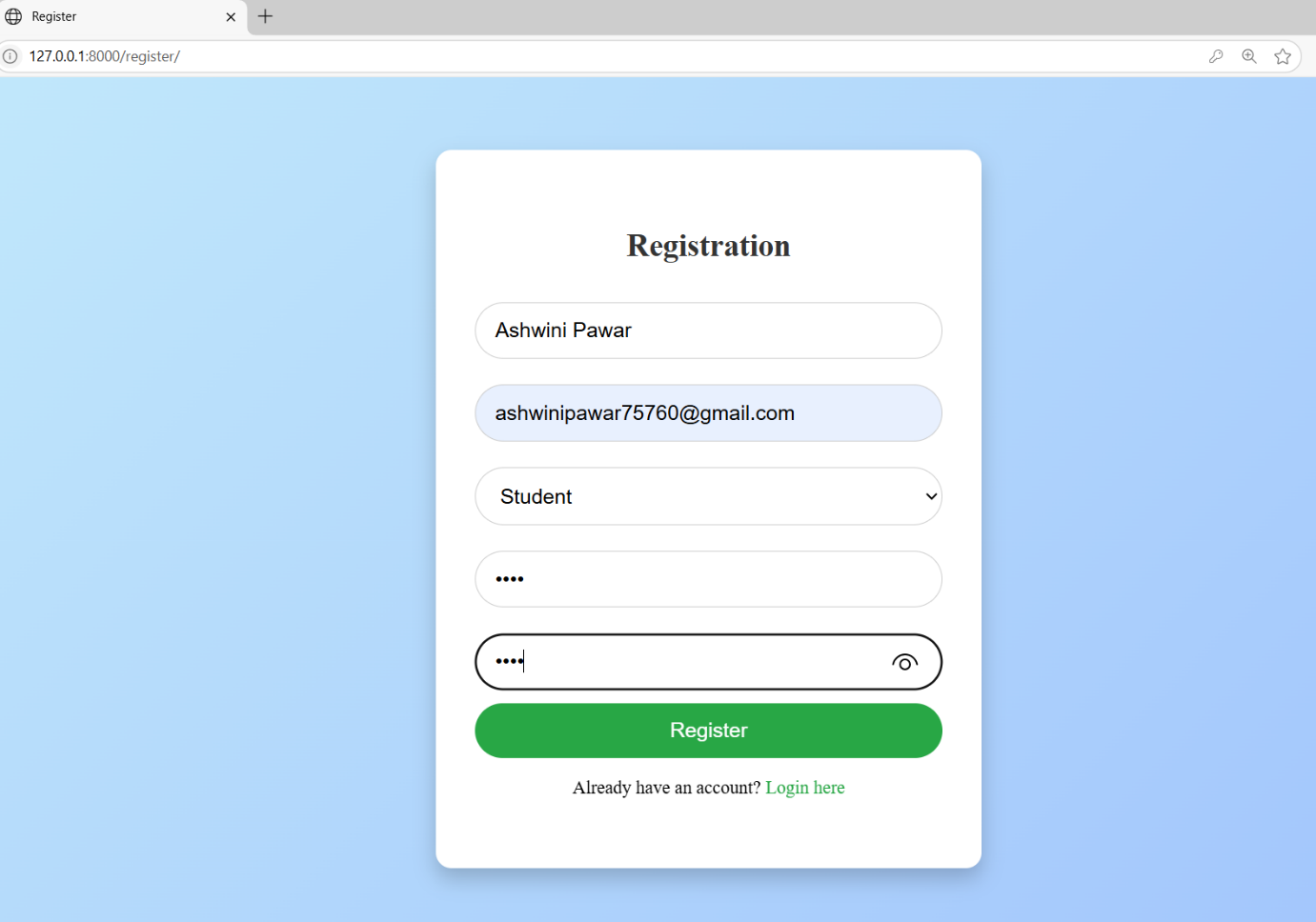


**4.3 OUTPUTS**

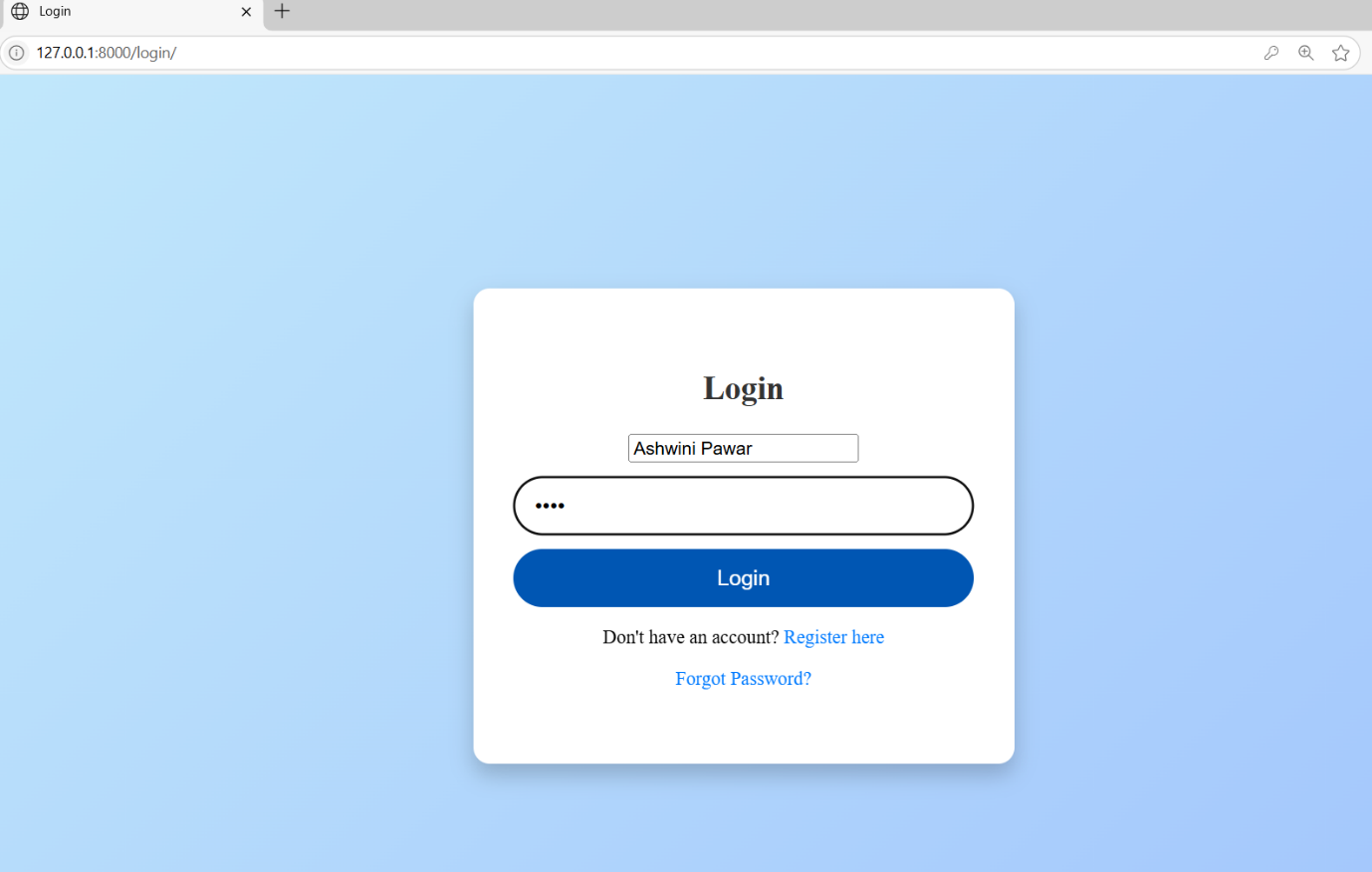
**Main Dashboard:**



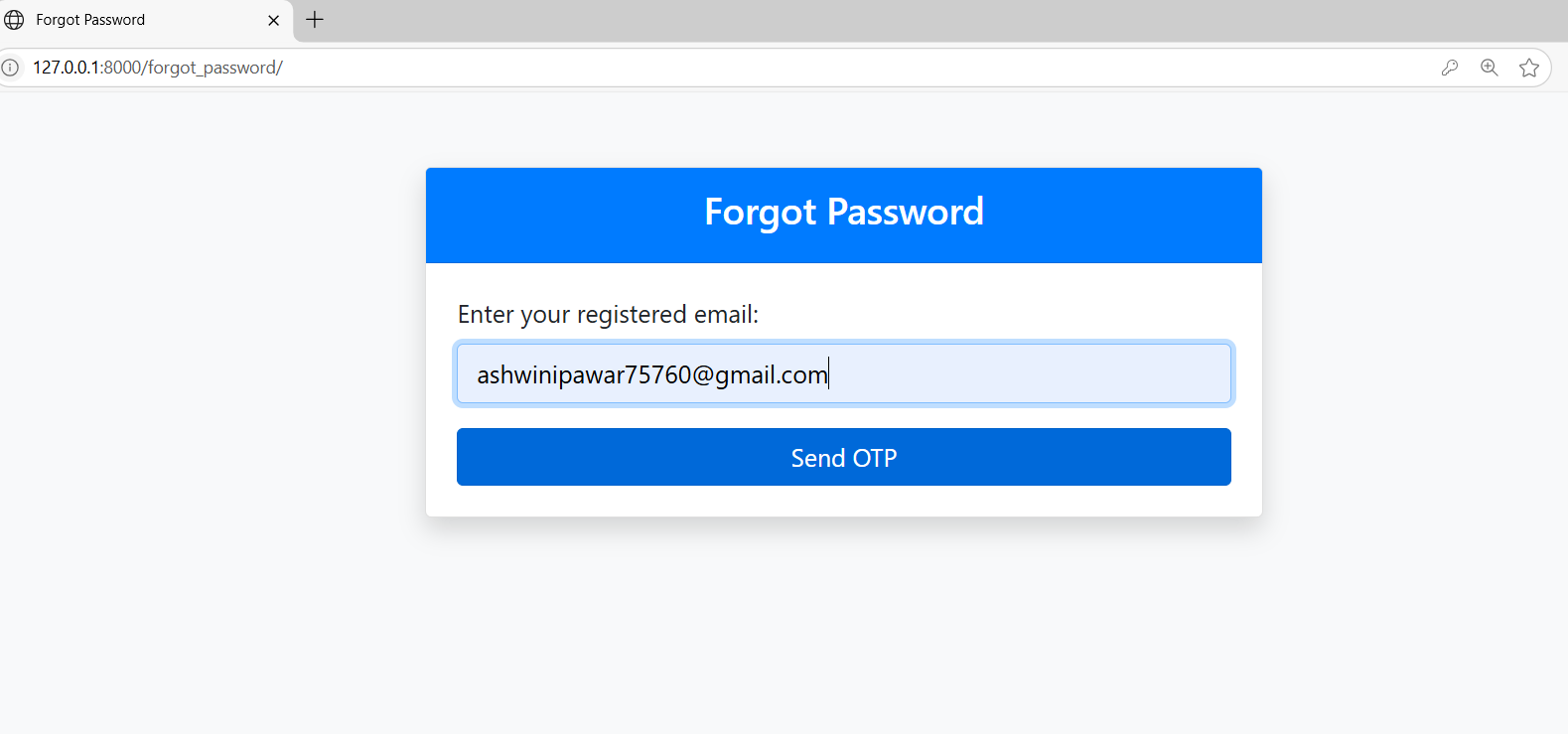
**Registration Page:**



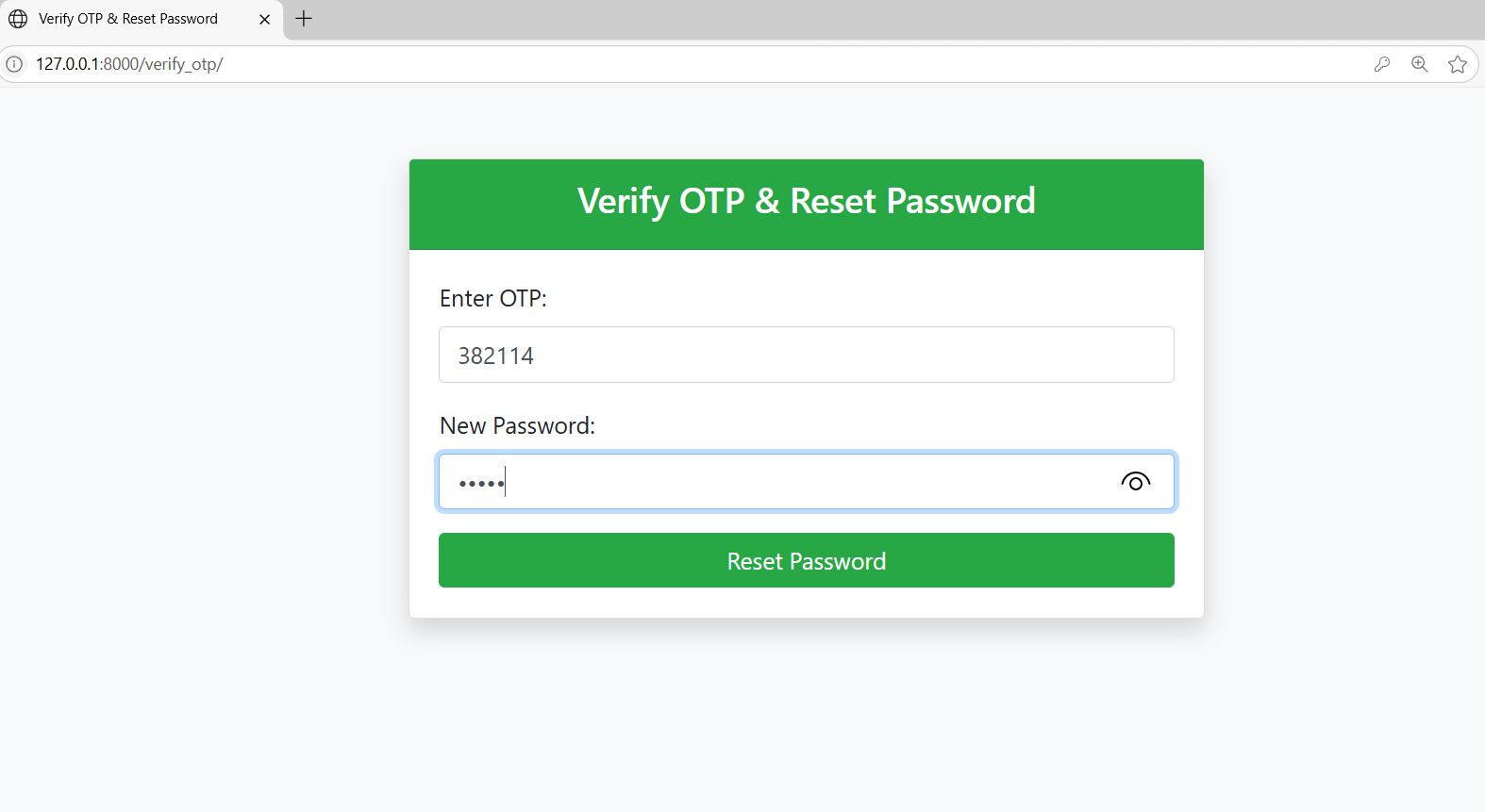
**Login Page:**



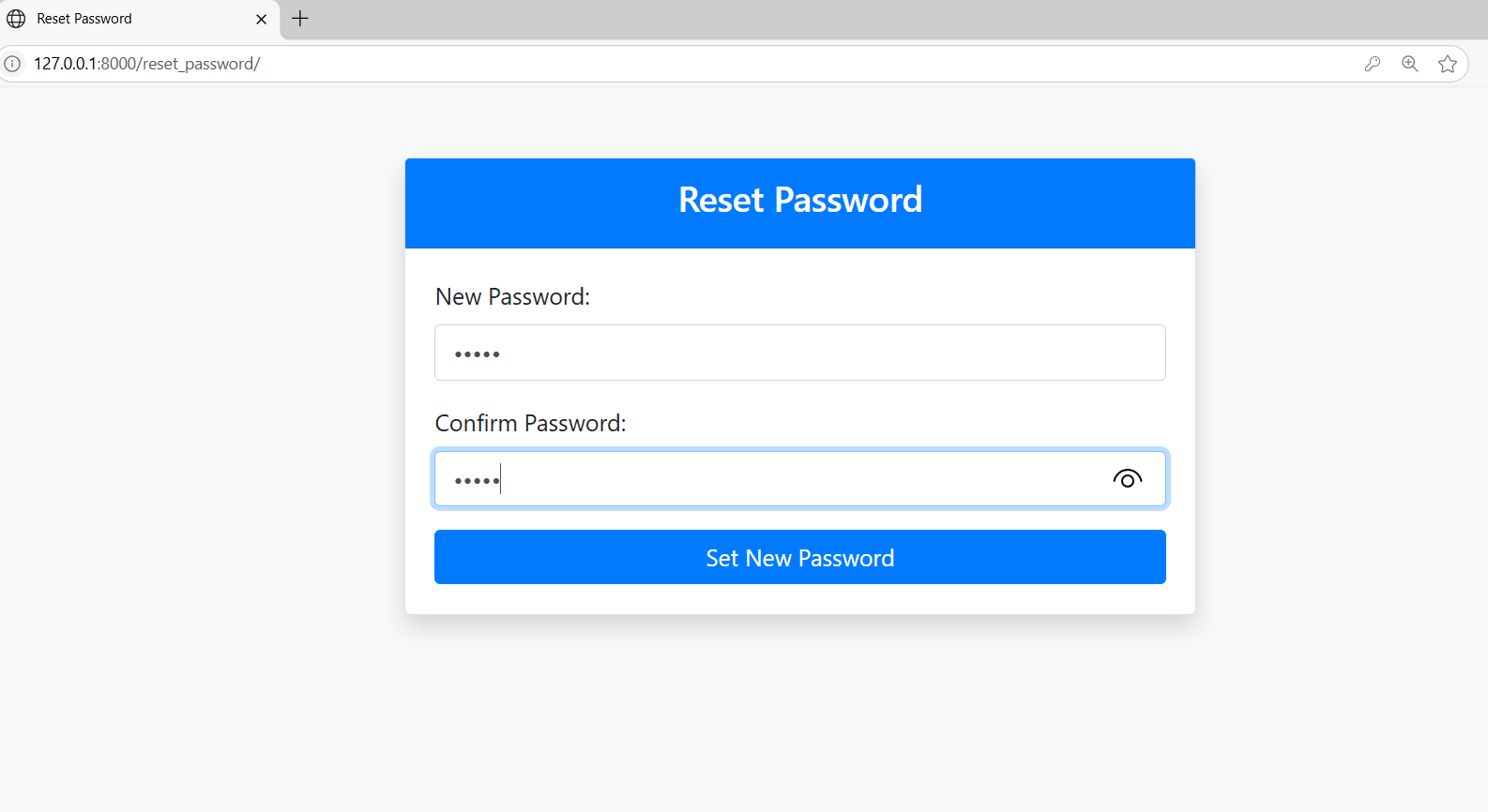
**Forget Password:**



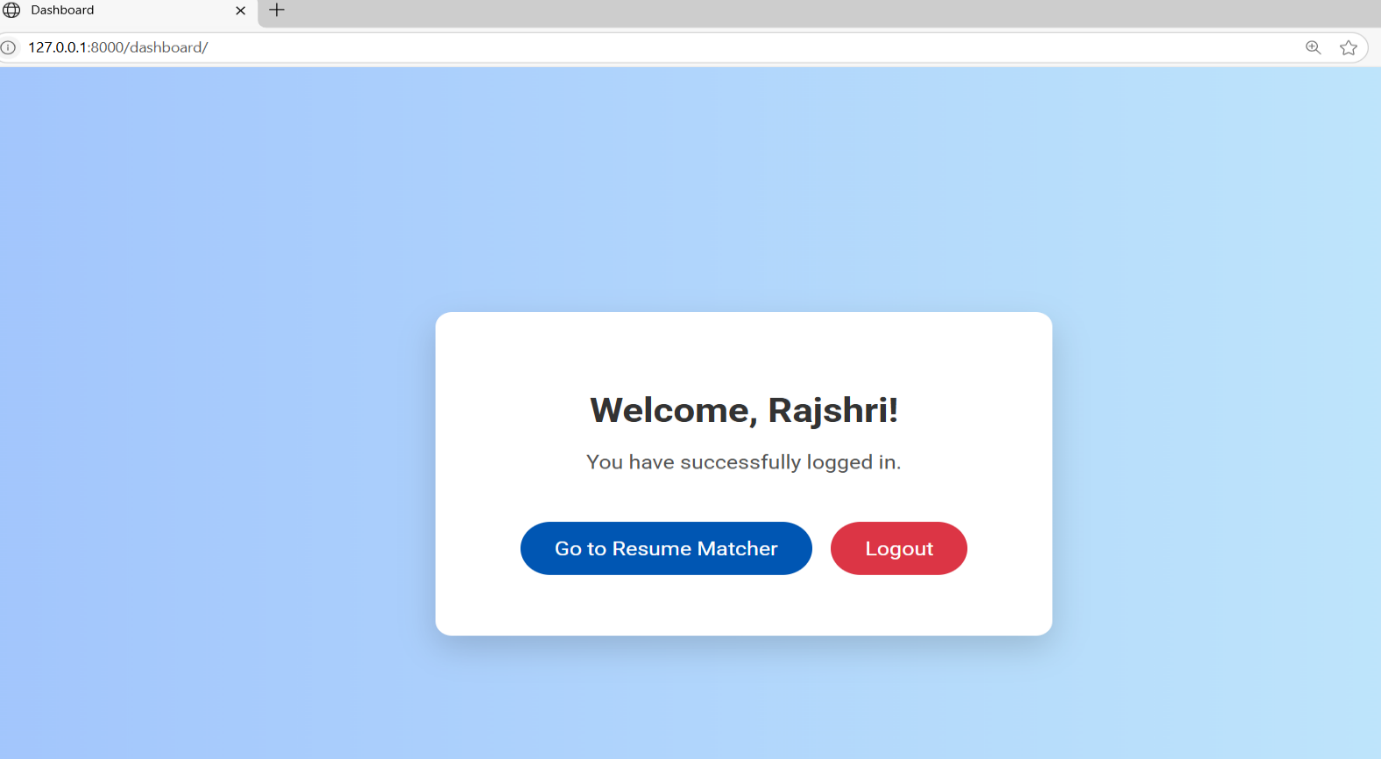
**Verify OTP:**



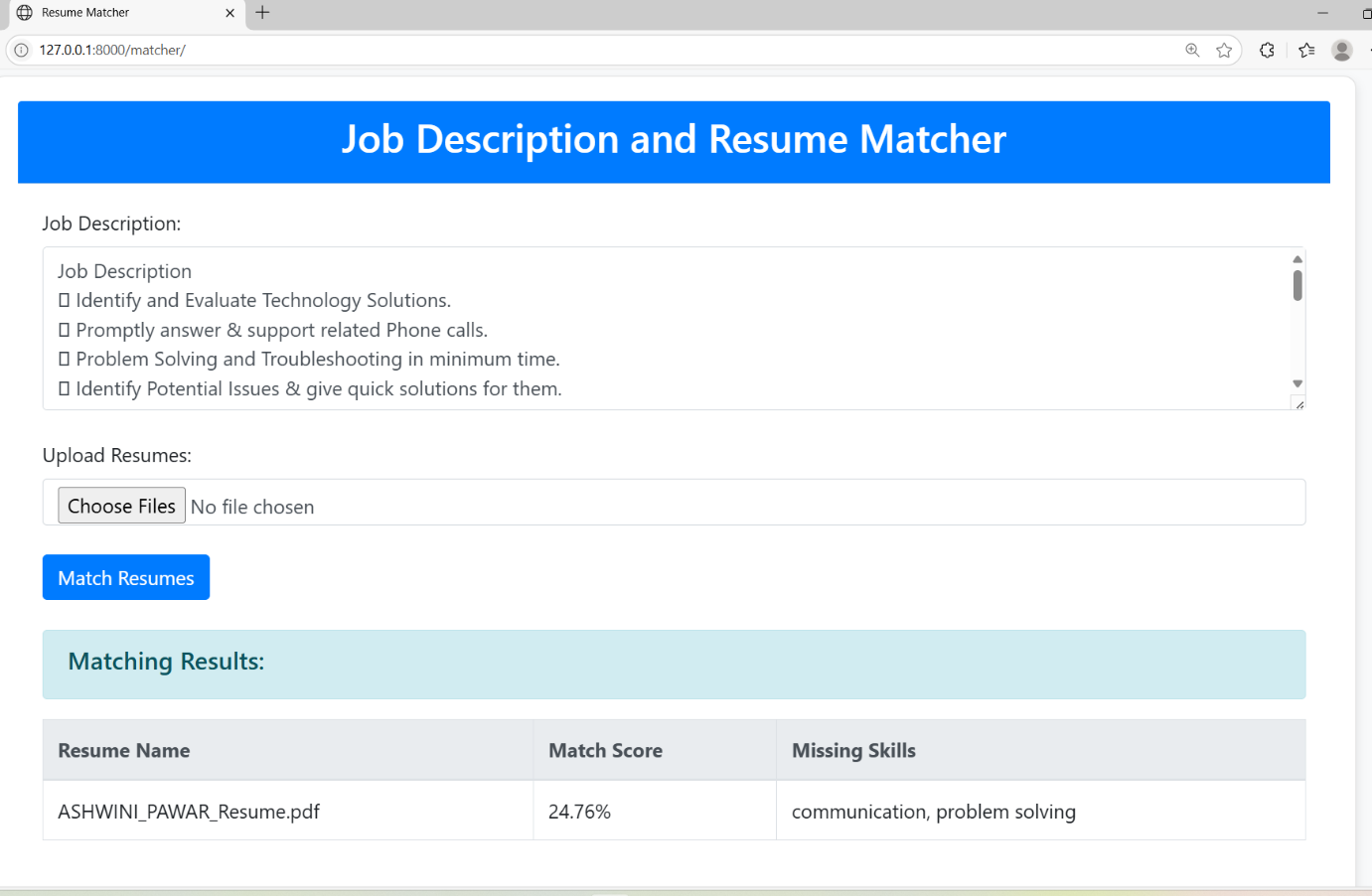
**Reset Password:**



**Welcome User Dashboard:**



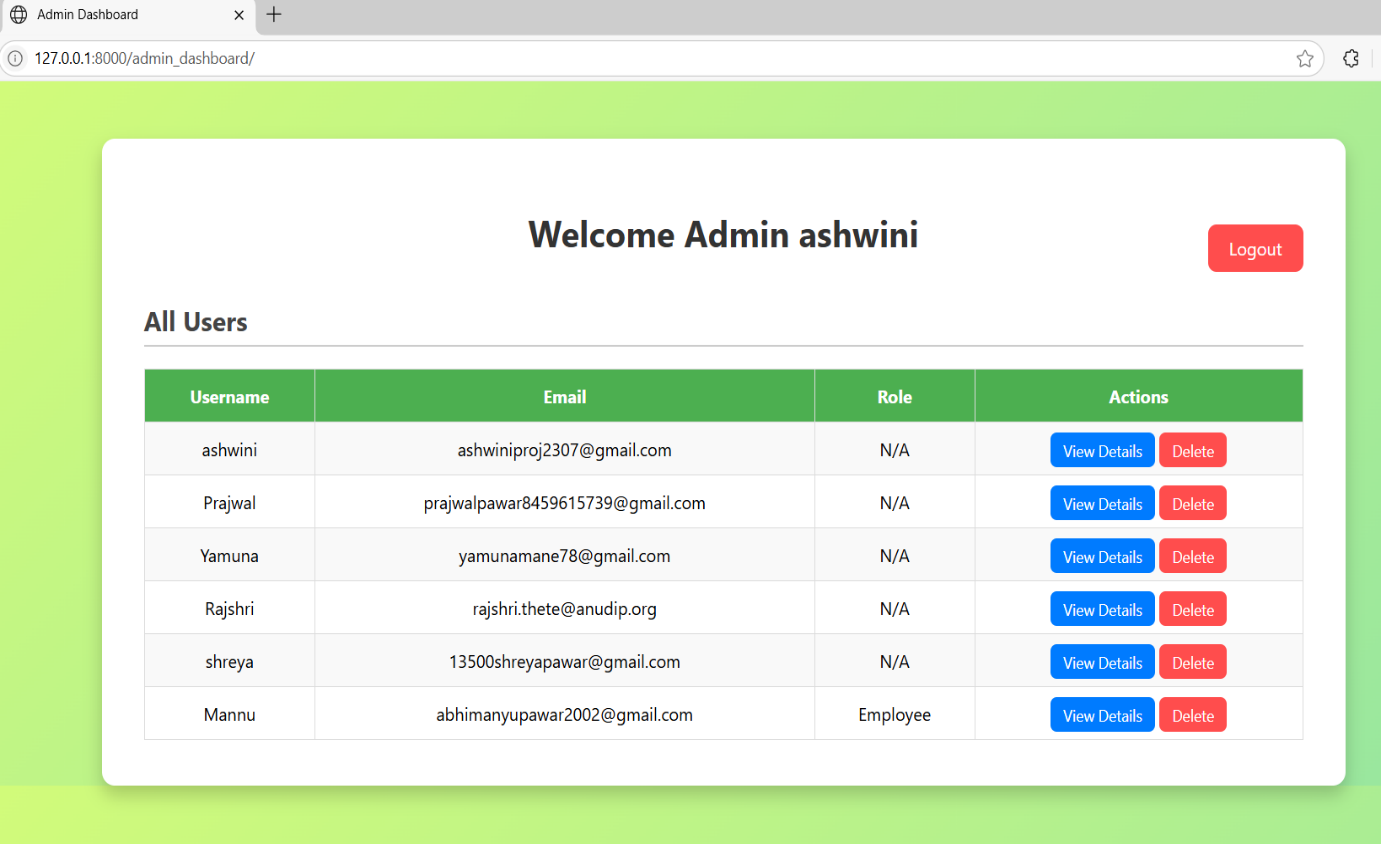
**Job Description and Resume Matcher Dashboard:**



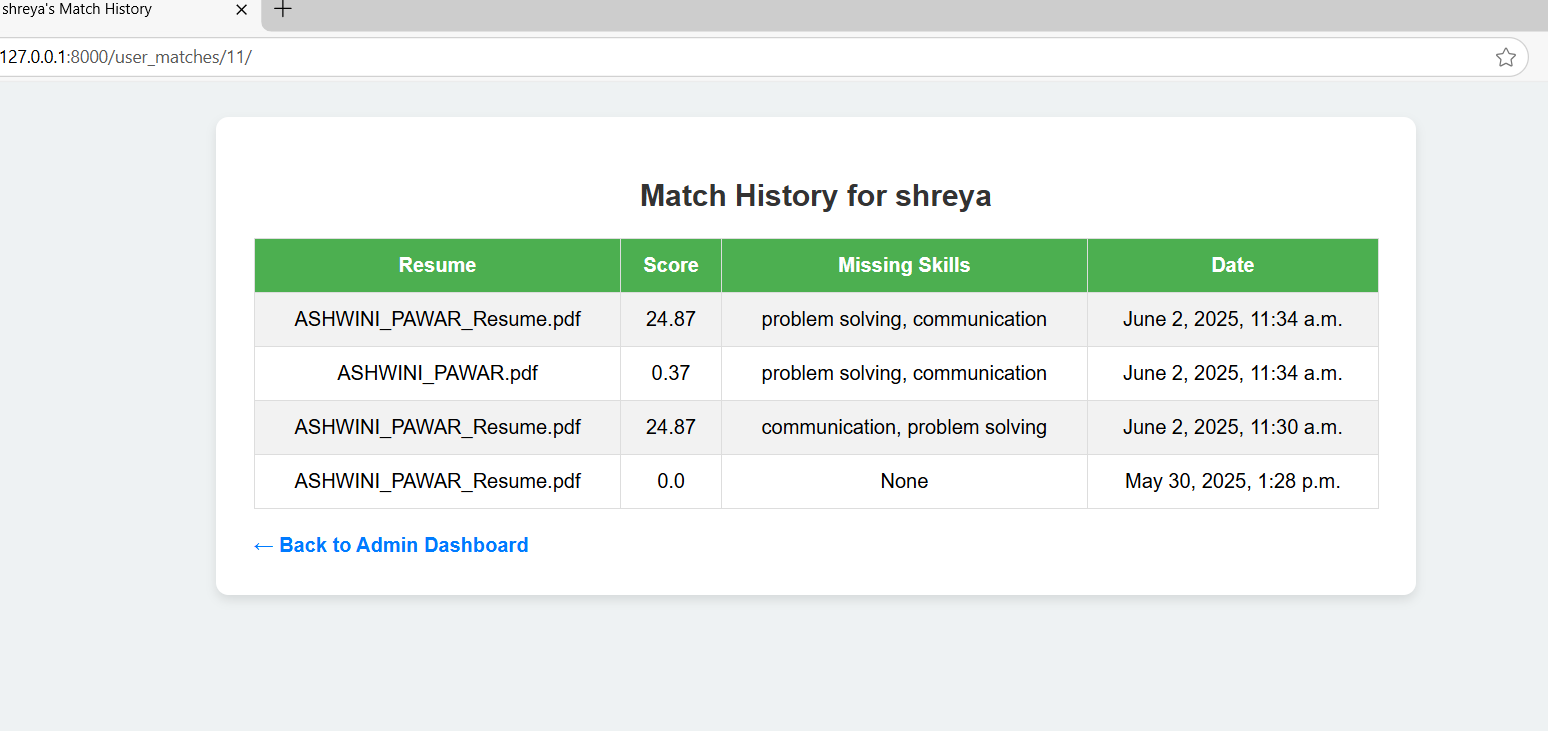
**Admin Login:**



**Admin Dashboard:**



**Match History:**



**CODING**

**Welcome.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Welcome</title>

    <style>

        body {

            font-family: 'Times New Roman', Times, serif;

            background-color: #f4f7fc;

            margin: 0;

            padding: 0;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            background: linear-gradient(135deg, #a1c4fd, #c2e9fb);

        }

        .container {

            text-align: center;

            background: white;

            padding: 30px 50px;

            border-radius: 10px;

            box-shadow: 0 8px 15px rgba(0, 0, 0, 0.1);

            width: 100%;

            max-width: 400px;

            animation: fadeIn 1s ease-out;

        }

        h1 {

            font-size: 2.5em;

            color: #333;

            margin-bottom: 20px;

        }

        .highlight {

            color: #007bff;

            font-weight: 700;

        }

        .btn {

            display: inline-block;

            margin: 10px 20px;

            padding: 12px 30px;

            font-size: 16px;

            text-decoration: none;

            color: white;

            background-color: #007bff;

            border-radius: 30px;

            transition: background-color 0.3s;

        }

        .btn:hover {

            background-color: #0056b3;

        }

        .button-group {

            margin-top: 20px;

        }

        @keyframes fadeIn {

            0% {

                opacity: 0;

                transform: translateY(20px);

            }

            100% {

                opacity: 1;

                transform: translateY(0);

            }

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Welcome to <span class="highlight">Resume Matcher</span></h1>

        <div class="button-group">

            <a href="{% url 'login' %}" class="btn">Login</a>

            <a href="{% url 'register' %}" class="btn">Register</a>

            <p><a href="{% url 'admin\_login' %}">Admin Login</a></p>

        </div>

    </div>

</body>

</html>

**Register.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Register</title>

    <style>

        \* {

            box-sizing: border-box;

        }

        body {

            margin: 0;

            padding: 0;

            font-family: 'Times New Roman', Times, serif;

            background: linear-gradient(135deg, #c2e9fb, #a1c4fd);

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

        }

        .container {

            background: white;

            padding: 40px 30px;

            border-radius: 12px;

            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

            width: 100%;

            max-width: 420px;

            text-align: center;

            animation: slideFade 0.7s ease;

        }

        h2 {

            margin-bottom: 20px;

            color: #333;

        }

        input[type="text"],

        input[type="email"],

        input[type="password"],

        select {

            width: 100%;

            padding: 12px 15px;

            margin: 10px 0;

            border: 1px solid #ccc;

            border-radius: 30px;

            font-size: 16px;

        }

        button {

            width: 100%;

            padding: 12px;

            background-color: #28a745;

            border: none;

            border-radius: 30px;

            color: white;

            font-size: 16px;

            cursor: pointer;

            transition: background-color 0.3s ease;

        }

        button:hover {

            background-color: #218838;

        }

        p {

            margin-top: 15px;

            font-size: 14px;

        }

        a {

            color: #28a745;

            text-decoration: none;

            font-weight: 500;

        }

        a:hover {

            text-decoration: underline;

        }

        @keyframes slideFade {

            from {

                opacity: 0;

                transform: translateY(30px);

            }

            to {

                opacity: 1;

                transform: translateY(0);

            }

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Registration</h2>

        <form method="POST">

                {% csrf\_token %}

            <input type="text" name="username" placeholder="Your Name" required>

            <input type="email" name="email" placeholder="Your Email" required>

            <select name="role" required>

                <option value="">Select Role</option>

                <option value="Student">Student</option>

                <option value="Employee">Employee</option>

            </select>

            <input type="password" name="password" placeholder="Password" required>

            <input type="password" name="confirm\_password" placeholder="Confirm Password" required>

            <button type="submit">Register</button>

        </form>

        <p>Already have an account? <a href="{% url 'login' %}">Login here</a></p>

    </div>

</body>

</html>

**Login.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Login</title>

    <style>

        \* {

            box-sizing: border-box;

        }

        body {

            margin: 0;

            padding: 0;

            font-family: 'Times New Roman', Times, serif;

            background: linear-gradient(135deg, #c2e9fb, #a1c4fd);

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

        }

        .container {

            background: white;

            padding: 40px 30px;

            border-radius: 12px;

            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

            width: 100%;

            max-width: 400px;

            text-align: center;

            animation: slideFade 0.7s ease;

        }

        h2 {

            margin-bottom: 20px;

            color: #333;

        }

        input[type="email"],

        input[type="password"] {

            width: 100%;

            padding: 12px 15px;

            margin: 10px 0;

            border: 1px solid #ccc;

            border-radius: 30px;

            font-size: 16px;

        }

        button {

            width: 100%;

            padding: 12px;

            background-color: #007bff;

            border: none;

            border-radius: 30px;

            color: white;

            font-size: 16px;

            cursor: pointer;

            transition: background-color 0.3s ease;

        }

        button:hover {

            background-color: #0056b3;

        }

        p {

            margin-top: 15px;

            font-size: 14px;

        }

        a {

            color: #007bff;

            text-decoration: none;

            font-weight: 500;

        }

        a:hover {

            text-decoration: underline;

        }

        @keyframes slideFade {

            from {

                opacity: 0;

                transform: translateY(30px);

            }

            to {

                opacity: 1;

                transform: translateY(0);

            }

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Login</h2>

        <form method="POST" action="{% url 'login' %}">

    {% csrf\_token %}  <!-- Add this line -->

    <input type="name" name="username" placeholder="Username" required>

    <input type="password" name="password" placeholder="Password" required>

    <button type="submit">Login</button>

</form>

        <p>Don't have an account? <a href="{% url 'register' %}">Register here</a></p>

        <p><a href="{% url 'forgot\_password' %}">Forgot Password?</a></p>

    </div>

</body>

</html>

**Forget\_password.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Forgot Password</title>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body class="bg-light">

    <div class="container mt-5">

        <div class="row justify-content-center">

            <div class="col-md-6">

                <div class="card shadow rounded">

                    <div class="card-header bg-primary text-white text-center">

                        <h4>Forgot Password</h4>

                    </div>

                    <div class="card-body">

                        <form method="POST">

                            {% csrf\_token %}

                            <div class="form-group">

                                <label for="email">Enter your registered email:</label>

                                <input type="email" class="form-control" name="email" required>

                            </div>

                            <button type="submit" class="btn btn-primary btn-block">Send OTP</button>

                        </form>

                        {% if message %}

                            <div class="alert alert-info mt-3">

                                {{ message }}

                            </div>

                        {% endif %}

                    </div>

                </div>

            </div>

        </div>

    </div>

</body>

</html>

**Verify\_otp.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Verify OTP & Reset Password</title>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body class="bg-light">

    <div class="container mt-5">

        <div class="row justify-content-center">

            <div class="col-md-6">

                <div class="card shadow rounded">

                    <div class="card-header bg-success text-white text-center">

                        <h4>Verify OTP & Reset Password</h4>

                    </div>

                    <div class="card-body">

                        <form method="POST" action="{% url 'verify\_otp' %}">

                            {% csrf\_token %}

                            <input type="hidden" name="email" value="{{ email }}">

                            <div class="form-group">

                                <label for="otp">Enter OTP:</label>

                                <input type="text" class="form-control" name="otp" required>

                            </div>

                            <div class="form-group">

                                <label for="new\_password">New Password:</label>

                                <input type="password" class="form-control" name="new\_password" required>

                            </div>

                            <button type="submit" class="btn btn-success btn-block">Reset Password</button>

                        </form>

                        {% if message %}

                            <div class="alert alert-info mt-3">

                                {{ message }}

                            </div>

                        {% endif %}

                    </div>

                </div>

            </div>

        </div>

    </div>

</body>

</html>

**Reset\_password.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Reset Password</title>

    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

<body class="bg-light">

    <div class="container mt-5">

        <div class="row justify-content-center">

            <div class="col-md-6">

                <div class="card shadow rounded">

                    <div class="card-header bg-primary text-white text-center">

                        <h4>Reset Password</h4>

                    </div>

                    <div class="card-body">

                        <form method="POST" action="{% url 'reset\_password' %}">

                            {% csrf\_token %}

                            <div class="form-group">

                                <label for="new\_password">New Password:</label>

                                <input type="password" class="form-control" name="new\_password" required>

                            </div>

                            <div class="form-group">

                                <label for="confirm\_password">Confirm Password:</label>

                                <input type="password" class="form-control" name="confirm\_password" required>

                            </div>

                            <button type="submit" class="btn btn-primary btn-block">Set New Password</button>

                        </form>

                        {% if message %}

                            <div class="alert alert-info mt-3">

                                {{ message }}

                            </div>

                        {% endif %}

                    </div>

                </div>

            </div>

        </div>

    </div>

</body>

</html>

**Dashboard.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Dashboard</title>

    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

    <style>

        \* {

            box-sizing: border-box;

        }

        body {

            margin: 0;

            padding: 0;

            font-family: 'Roboto', sans-serif;

            background: linear-gradient(to right, #a1c4fd, #c2e9fb);

            height: 100vh;

            display: flex;

            align-items: center;

            justify-content: center;

        }

        .container {

            background: #fff;

            padding: 40px 30px;

            border-radius: 12px;

            box-shadow: 0 10px 25px rgba(0, 0, 0, 0.15);

            width: 90%;

            max-width: 450px;

            text-align: center;

            animation: fadeInUp 0.6s ease-out;

        }

        h2 {

            margin-bottom: 10px;

            font-size: 28px;

            color: #333;

        }

        p {

            font-size: 16px;

            color: #555;

            margin-bottom: 30px;

        }

        .btn {

            padding: 12px 25px;

            font-size: 16px;

            border-radius: 25px;

            background-color: #007bff;

            border: none;

            color: white;

            cursor: pointer;

            transition: background-color 0.3s ease;

            text-decoration: none;

            margin: 10px 5px;

            display: inline-block;

        }

        .btn:hover {

            background-color: #0056b3;

        }

        .btn-logout {

            background-color: #dc3545;

        }

        .btn-logout:hover {

            background-color: #b02a37;

        }

        @keyframes fadeInUp {

            from {

                opacity: 0;

                transform: translateY(30px);

            }

            to {

                opacity: 1;

                transform: translateY(0);

            }

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Welcome, {{ username }}!</h2>

        <p>You have successfully logged in.</p>

        <a href="{% url 'index' %}" class="btn">Go to Resume Matcher</a>

        <a href="{% url 'logout' %}" class="btn btn-logout">Logout</a>

    </div>

</body>

</html>

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Resume Matcher</title>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

    <style>

        body {

            background-color: #f8f9fa;

        }

        .container {

            margin-top: 50px;

        }

        .card {

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 4px 8px rgba(0,0,0,0.1);

        }

        .card-header {

            background-color: #007bff;

            color: white;

            border-radius: 10px 10px 0 0;

            padding: 10px 20px;

        }

        .card-body {

            padding: 20px;

        }

        .form-group {

            margin-bottom: 1.5rem;

        }

        .user-message {

            text-align: right;

            color: blue;

            margin: 5px 0;

        }

    </style>

</head>

<body>

    <div class="container">

        <!-- Resume Matcher Form -->

        <div class="card">

            <div class="card-header text-center">

                <h2>Job Description and Resume Matcher</h2>

            </div>

            <div class="card-body">

                <form method="POST" action="{% url 'matcher' %}" enctype="multipart/form-data">

                    {% csrf\_token %}

                    <div class="form-group">

                        <label for="job\_description">Job Description:</label>

                        <textarea class="form-control" id="job\_description" name="job\_desc\_text" rows="5" required>{{ job\_desc }}</textarea>

                    </div>

                    <div class="form-group">

                        <label for="resumes">Upload Resumes:</label>

                        <input type="file" class="form-control" id="resumes" name="resumes" multiple required accept=".pdf, .docx, .txt">

                    </div>

                    <button type="submit" class="btn btn-primary">Match Resumes</button>

                </form>

                <!-- Show Uploaded Files -->

                {% if uploaded\_filenames %}

                    <div class="mt-4">

                        <h5>Uploaded Resumes:</h5>

                        <ul>

                            {% for name in uploaded\_filenames %}

                                <li>{{ name }}</li>

                            {% endfor %}

                        </ul>

                    </div>

                {% endif %}

                <!-- Show Matching Results -->

                {% if results %}

                    <div class="alert alert-info mt-4">

                        <h5>Matching Results:</h5>

                    </div>

                    <table class="table table-bordered mt-3">

                        <thead class="thead-light">

                            <tr>

                                <th>Resume Name</th>

                                <th>Similarity Score</th>

                                <th>Missing Skills</th>

                            </tr>

                        </thead>

                        <tbody>

                            {% for resume in results %}

                                <tr>

                                    <td>{{ resume.resume\_name }}</td>

                                    <td>{{ resume.similarity\_score }}</td>

                                    <td>{{ resume.missing\_skills }}</td>

                                </tr>

                            {% endfor %}

                        </tbody>

                    </table>

                {% endif %}

            </div>

        </div>

    </div>

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**User\_match\_history.html**

<!DOCTYPE html>

<html>

<head>

    <title>{{ user.username }}'s Match History</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background: #eef2f3;

            padding: 20px;

        }

        .container {

            background: white;

            padding: 30px;

            border-radius: 10px;

            max-width: 900px;

            margin: auto;

            box-shadow: 0 4px 8px rgba(0,0,0,0.1);

        }

        h2 {

            text-align: center;

            color: #333;

        }

        table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 20px;

        }

        table, th, td {

            border: 1px solid #ddd;

        }

        th, td {

            padding: 12px;

            text-align: center;

        }

        th {

            background-color: #4CAF50;

            color: white;

        }

        tr:nth-child(even) {

            background-color: #f2f2f2;

        }

        a.back-link {

            display: inline-block;

            margin-top: 20px;

            text-decoration: none;

            color: #007bff;

            font-weight: bold;

        }

        a.back-link:hover {

            text-decoration: underline;

        }

    </style>

</head>

<body>

<div class="container">

    <h2>Match History for {{ user.username }}</h2>

    <table>

        <tr>

            <th>Resume</th>

            <th>Score</th>

            <th>Missing Skills</th>

            <th>Date</th>

        </tr>

        {% for match in matches %}

        <tr>

            <td>{{ match.resume\_name }}</td>

            <td>{{ match.match\_score }}</td>

            <td>{{ match.missing\_skills }}</td>

            <td>{{ match.created\_at }}</td>

        </tr>

        {% endfor %}

    </table>

    <a class="back-link" href="{% url 'admin\_dashboard' %}">&larr; Back to Admin Dashboard</a>

</div>

</body>

</html>

**Admin\_login.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Admin Login</title>

    <style>

        body {

            font-family: 'Times New Roman', Times, serif;

            background: linear-gradient(135deg, #c2e9fb, #a1c4fd);

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            margin: 0;

        }

        .container {

            background: white;

            padding: 40px 30px;

            border-radius: 12px;

            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);

            width: 100%;

            max-width: 400px;

            text-align: center;

        }

        h2 {

            margin-bottom: 20px;

            color: #333;

        }

        input[type="text"],

        input[type="password"] {

            width: 100%;

            padding: 12px 15px;

            margin: 10px 0;

            border: 1px solid #ccc;

            border-radius: 30px;

            font-size: 16px;

        }

        button {

            width: 100%;

            padding: 12px;

            background-color: #007bff;

            border: none;

            border-radius: 30px;

            color: white;

            font-size: 16px;

            cursor: pointer;

            transition: background-color 0.3s ease;

        }

        button:hover {

            background-color: #0056b3;

        }

    </style>

</head>

<body>

    <div class="container">

        <h2>Admin Login</h2>

        <form method="POST" action="{% url 'admin\_login' %}">

                {% csrf\_token %}

            <input type="text" name="username" placeholder="Admin Username" required>

            <input type="password" name="password" placeholder="Admin Password" required>

            <button type="submit">Login as Admin</button>

        </form>

    </div>

</body>

</html>

**Admin\_dashboard.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Admin Dashboard</title>

    <style>

        body {

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

            background: linear-gradient(135deg, #d4fc79, #96e6a1);

            margin: 0;

            padding: 0;

        }

        .container {

            max-width: 1100px;

            margin: auto;

            padding: 40px;

            background: white;

            margin-top: 50px;

            border-radius: 12px;

            box-shadow: 0 8px 16px rgba(0,0,0,0.2);

        }

        h1 {

            text-align: center;

            color: #333;

        }

        a.logout-btn {

            display: inline-block;

            padding: 10px 20px;

            background: #ff4d4d;

            color: white;

            text-decoration: none;

            border-radius: 8px;

            float: right;

            margin-top: -50px;

        }

        h2 {

            margin-top: 40px;

            color: #444;

            border-bottom: 2px solid #ccc;

            padding-bottom: 5px;

        }

        table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 15px;

        }

        table, th, td {

            border: 1px solid #ddd;

        }

        th, td {

            padding: 12px;

            text-align: center;

        }

        th {

            background-color: #4CAF50;

            color: white;

        }

        tr:nth-child(even) {

            background-color: #f9f9f9;

        }

        tr:hover {

            background-color: #f1f1f1;

        }

        .btn {

            padding: 6px 12px;

            border-radius: 6px;

            text-decoration: none;

            color: white;

            font-size: 14px;

        }

        .view-btn {

            background-color: #007bff;

        }

        .view-btn:hover {

            background-color: #0056b3;

        }

        .delete-btn {

            background-color: #ff4d4d;

        }

        .delete-btn:hover {

            background-color: #cc0000;

        }

    </style>

</head>

<body>

<div class="container">

    <h1>Welcome Admin {{ request.user.username }}</h1>

    <a href="{% url 'logout' %}" class="logout-btn">Logout</a>

    <h2>All Users</h2>

    <table>

        <tr>

            <th>Username</th>

            <th>Email</th>

            <th>Role</th>

            <th>Actions</th>

        </tr>

        {% for user in users %}

        <tr>

            <td>{{ user.username }}</td>

            <td>{{ user.email }}</td>

            <td>{% if user.is\_staff %}Admin{% else %}User{% endif %}</td>

            <td>

                <a href="{% url 'user\_match\_history' user.id %}" class="btn view-btn">View Details</a>

                <a href="{% url 'delete\_user' user.id %}" class="btn delete-btn" onclick="return confirm('Are you sure you want to delete this user?');">Delete</a>

            </td>

        </tr>

        {% endfor %}

    </table>

</div>

</body>

</html>

**View\_user.html**

<!DOCTYPE html>

<html>

<head>

    <title>User Details</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            background: linear-gradient(135deg, #f6d365, #fda085);

            padding: 40px;

        }

        .card {

            max-width: 600px;

            margin: auto;

            background: white;

            padding: 30px;

            border-radius: 12px;

            box-shadow: 0 8px 16px rgba(0,0,0,0.2);

        }

        h2 {

            text-align: center;

            color: #333;

        }

        p {

            font-size: 16px;

            margin: 10px 0;

        }

        table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 20px;

        }

        th, td {

            padding: 10px;

            border: 1px solid #ddd;

            text-align: center;

        }

        th {

            background: #4CAF50;

            color: white;

        }

        a.back-btn {

            display: inline-block;

            margin-top: 20px;

            padding: 10px 20px;

            background: #007bff;

            color: white;

            text-decoration: none;

            border-radius: 6px;

        }

        a.back-btn:hover {

            background: #0056b3;

        }

    </style>

</head>

<body>

<div class="card">

    <h2>User Details</h2>

    <p><strong>Username:</strong> {{ user\_obj.username }}</p>

    <p><strong>Email:</strong> {{ user\_obj.email }}</p>

    <p><strong>Role:</strong> {% if user\_obj.is\_staff %}Admin{% else %}User{% endif %}</p>

    <h3>Resume Matches</h3>

    {% if matches %}

    <table>

        <tr><th>Resume</th><th>Score</th><th>Missing Skills</th><th>Date</th></tr>

        {% for match in matches %}

        <tr>

            <td>{{ match.resume\_name }}</td>

            <td>{{ match.match\_score }}</td>

            <td>{{ match.missing\_skills }}</td>

            <td>{{ match.created\_at }}</td>

        </tr>

        {% endfor %}

    </table>

    {% else %}

    <p>No matches found.</p>

    {% endif %}

    <a href="{% url 'admin\_dashboard' %}" class="back-btn">← Back to Dashboard</a>

</div>

</body>

</html>

**Admin.py**

from django.contrib import admin

from .models import UserProfile, Match, OTPStore

admin.site.register(UserProfile)

admin.site.register(Match)

admin.site.register(OTPStore)

# Register your models here.

**Models.py**

from django.db import models

from django.contrib.auth.models import User

class UserProfile(models.Model):

    ROLE\_CHOICES = (

        ('Student', 'Student'),

        ('Employee', 'Employee'),

    )

    user = models.OneToOneField(User, on\_delete=models.CASCADE)

    role = models.CharField(max\_length=20, choices=ROLE\_CHOICES)

    def \_str\_(self):

        return f'{self.user.username} - {self.role}'

class Match(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    resume\_name = models.CharField(max\_length=255)

    match\_score = models.FloatField()

    missing\_skills = models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_str\_(self):

        return f'{self.user.username} - {self.resume\_name} - {self.match\_score}'

class OTPStore(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    otp = models.CharField(max\_length=6)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_str\_(self):

        return f'{self.user.username}-{self.otp}'

**urls.py**

from django.urls import path

from . import views

urlpatterns = [

    path('', views.welcome, name='welcome'),

    path('login/', views.login\_view, name='login'),

    path('register/', views.register, name='register'),

    path('forgot\_password/', views.forgot\_password, name='forgot\_password'),

    path('verify\_otp/', views.verify\_otp, name='verify\_otp'),

    path('reset\_password/', views.reset\_password, name='reset\_password'),

    path('dashboard/', views.dashboard, name='dashboard'),

    path('index/', views.index, name='index'),

    path('logout/', views.logout\_view, name='logout'),

    path('admin\_login/', views.admin\_login, name='admin\_login'),

    path('admin\_dashboard/', views.admin\_dashboard, name='admin\_dashboard'),

    path('matcher/', views.matcher, name='matcher'),

    path('view\_user/<int:user\_id>/', views.view\_user, name='view\_user'),

    path('delete\_user/<int:user\_id>/', views.delete\_user, name='delete\_user'),

    path('user\_matches/<int:user\_id>/', views.user\_match\_history, name='user\_match\_history'),

]

**Views.py**

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.contrib.auth.models import User

from django.contrib import messages

from django.contrib.auth import authenticate, login as auth\_login, logout as auth\_logout

from django.core.mail import send\_mail

from django.conf import settings

import random

from django.contrib.auth.decorators import login\_required

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import re

import os

from PyPDF2 import PdfReader

from .skill\_set import SKILL\_KEYWORDS

from .models import Match

from django.views.decorators.csrf import csrf\_protect

def clean\_text(text):

    text = text.lower()

    text = re.sub(r'\W', ' ', text)

    text = re.sub(r'\s+', ' ', text)

    return text

def extract\_text\_from\_pdf(pdf\_file):

    pdf\_reader = PdfReader(pdf\_file)

    text = ""

    for page in pdf\_reader.pages:

        text += page.extract\_text()

    return text

# View user details

def view\_user(request, user\_id):

    user = get\_object\_or\_404(User, id=user\_id)

    user\_matches = Match.objects.filter(user=user)

    return render(request, 'view\_user.html', {

        'user\_obj': user,

        'matches': user\_matches

    })

# Delete user

def delete\_user(request, user\_id):

    user = get\_object\_or\_404(User, id=user\_id)

    if user.is\_staff:

        messages.error(request, "Cannot delete another Admin!")

        return redirect('admin\_dashboard')

    user.delete()

    messages.success(request, "User deleted successfully.")

    return redirect('admin\_dashboard')

# Helper function to generate 6-digit OTP

def generate\_otp():

    return str(random.randint(100000, 999999))

# Welcome page

def welcome(request):

    return render(request, 'welcome.html')

# Register

def register(request):

    if request.method == 'POST':

        username = request.POST['username']

        email = request.POST['email']

        password = request.POST['password']

        if User.objects.filter(username=username).exists():

            messages.error(request, "Username already exists")

            return redirect('register')

        if User.objects.filter(email=email).exists():

            messages.error(request, "Email already registered")

            return redirect('register')

        user = User.objects.create\_user(username=username, email=email, password=password)

        user.save()

        messages.success(request, "Registration successful. Please login.")

        return redirect('login')

    return render(request, 'register.html')

# Login

@csrf\_protect

def login\_view(request):

    if request.method == 'POST':

        username = request.POST['username']

        password = request.POST['password']

        user = authenticate(request, username=username, password=password)

        if user is not None:

            auth\_login(request, user)

            return redirect('dashboard')

        else:

            messages.error(request, "Invalid credentials")

            return redirect('login')

    return render(request, 'login.html')

# Forgot Password

def forgot\_password(request):

    if request.method == 'POST':

        email = request.POST['email']

        try:

            user = User.objects.get(email=email)

            otp = generate\_otp()

            request.session['reset\_email'] = email

            request.session['otp'] = otp

            send\_mail(

                subject='Your OTP for Password Reset',

                message=f'Your OTP is {otp}',

                from\_email=settings.EMAIL\_HOST\_USER,

                recipient\_list=[email],

            )

            messages.success(request, 'OTP sent to your email.')

            return redirect('verify\_otp')

        except User.DoesNotExist:

            messages.error(request, 'Email not registered')

            return redirect('forgot\_password')

    return render(request, 'forgot\_password.html')

# Verify OTP

def verify\_otp(request):

    if request.method == 'POST':

        entered\_otp = request.POST['otp']

        actual\_otp = request.session.get('otp')

        if entered\_otp == actual\_otp:

            messages.success(request, 'OTP verified. Please set a new password.')

            return redirect('reset\_password')

        else:

            messages.error(request, 'Invalid OTP')

            return redirect('verify\_otp')

    return render(request, 'verify\_otp.html')

# Reset Password

def reset\_password(request):

    if request.method == 'POST':

        new\_password = request.POST['new\_password']

        confirm\_password = request.POST['confirm\_password']

        email = request.session.get('reset\_email')

        if new\_password != confirm\_password:

            messages.error(request, 'Passwords do not match')

            return redirect('reset\_password')

        try:

            user = User.objects.get(email=email)

            user.set\_password(new\_password)

            user.save()

            # Clear session data

            request.session.pop('reset\_email', None)

            request.session.pop('otp', None)

            messages.success(request, 'Password reset successful. Please login.')

            return redirect('login')

        except User.DoesNotExist:

            messages.error(request, 'Error resetting password')

            return redirect('forgot\_password')

    return render(request, 'reset\_password.html')

# Dashboard

@login\_required

def dashboard(request):

    return render(request, 'dashboard.html', {'username': request.user.username})

# Index

@login\_required

def index(request):

    return render(request, 'index.html')

# ✅ Matcher (updated to save match to database)

@login\_required

def matcher(request):

    results = None

    job\_desc\_text = ''

    if request.method == 'POST':

        job\_desc\_text = request.POST.get('job\_desc\_text', '')

        job\_desc\_clean = clean\_text(job\_desc\_text)

        uploaded\_files = request.FILES.getlist('resumes')

        results = []

        for file in uploaded\_files:

            resume\_text = extract\_text\_from\_pdf(file)

            resume\_clean = clean\_text(resume\_text)

            # TF-IDF Similarity

            vectorizer = TfidfVectorizer()

            tfidf\_matrix = vectorizer.fit\_transform([resume\_clean, job\_desc\_clean])

            similarity = cosine\_similarity(tfidf\_matrix[0:1], tfidf\_matrix[1:2])[0][0]

            similarity\_score = round(similarity, 2)

            # Skill Matching

            job\_skills = [skill for skill in SKILL\_KEYWORDS if skill in job\_desc\_clean]

            resume\_skills = [skill for skill in SKILL\_KEYWORDS if skill in resume\_clean]

            missing\_skills = list(set(job\_skills) - set(resume\_skills))

            missing\_skills\_str = ', '.join(missing\_skills) if missing\_skills else 'None'

            # ✅ Save match record to database

            Match.objects.create(

                user=request.user,

                resume\_name=file.name,

                match\_score=similarity\_score,

                missing\_skills=missing\_skills\_str

            )

            # Add to display results

            results.append({

                'resume\_name': file.name,

                'similarity\_score': similarity\_score,

                'missing\_skills': missing\_skills\_str

            })

        results = sorted(results, key=lambda x: x['similarity\_score'], reverse=True)

    return render(request, 'index.html', {'results': results, 'job\_desc': job\_desc\_text})

# Logout

def logout\_view(request):

    auth\_logout(request)

    return redirect('welcome')

# Admin Login

def admin\_login(request):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        if username and password:

            user = authenticate(request, username=username, password=password)

            if user is not None and user.is\_staff:

                auth\_login(request, user)

                return redirect('admin\_dashboard')

            else:

                messages.error(request, "Invalid admin credentials or user is not an admin.")

                return redirect('admin\_login')

        else:

            messages.error(request, "Please provide both username and password.")

            return redirect('admin\_login')

    return render(request, 'admin\_login.html')

# Admin Dashboard

@login\_required

def admin\_dashboard(request):

    if not request.user.is\_staff:

        return redirect('admin\_login')

    users = User.objects.all()

    matches = Match.objects.all()

    return render(request, 'admin\_dashboard.html', {

        'admin\_name': request.user.username,

        'users': users,

        'matches': matches

    })

# User Match History

def user\_match\_history(request, user\_id):

    user = get\_object\_or\_404(User, id=user\_id)

    matches = Match.objects.filter(user=user).order\_by('-created\_at')

    return render(request, 'user\_match\_history.html', {

        'user': user,

        'matches':matches

        })

**TESTING PROCEDURES**

To ensure the quality and reliability of the **Job Description and Resume Matcher** system, we followed a layered testing approach that included unit testing, module testing, and full system validation. Below is a summary of the testing strategies used:

**Unit Testing**

Unit testing was performed on individual components such as text preprocessing, skill extraction, and similarity scoring functions. Each method was tested independently to verify its correctness.

**Module Testing**

Module testing involved grouping 4–6 related units (like the resume matcher and skill comparison logic) to ensure proper integration and functionality within a specific module. Interactions between views, forms, and the database.

**Sub-system Testing**

Sub-systems such as the **user account management**, **authentication system**, and **resume matching engine** were tested in combined scenarios. This allowed us to validate the data flow between modules and ensure that intermediate outputs were handled correctly.

**System Testing**

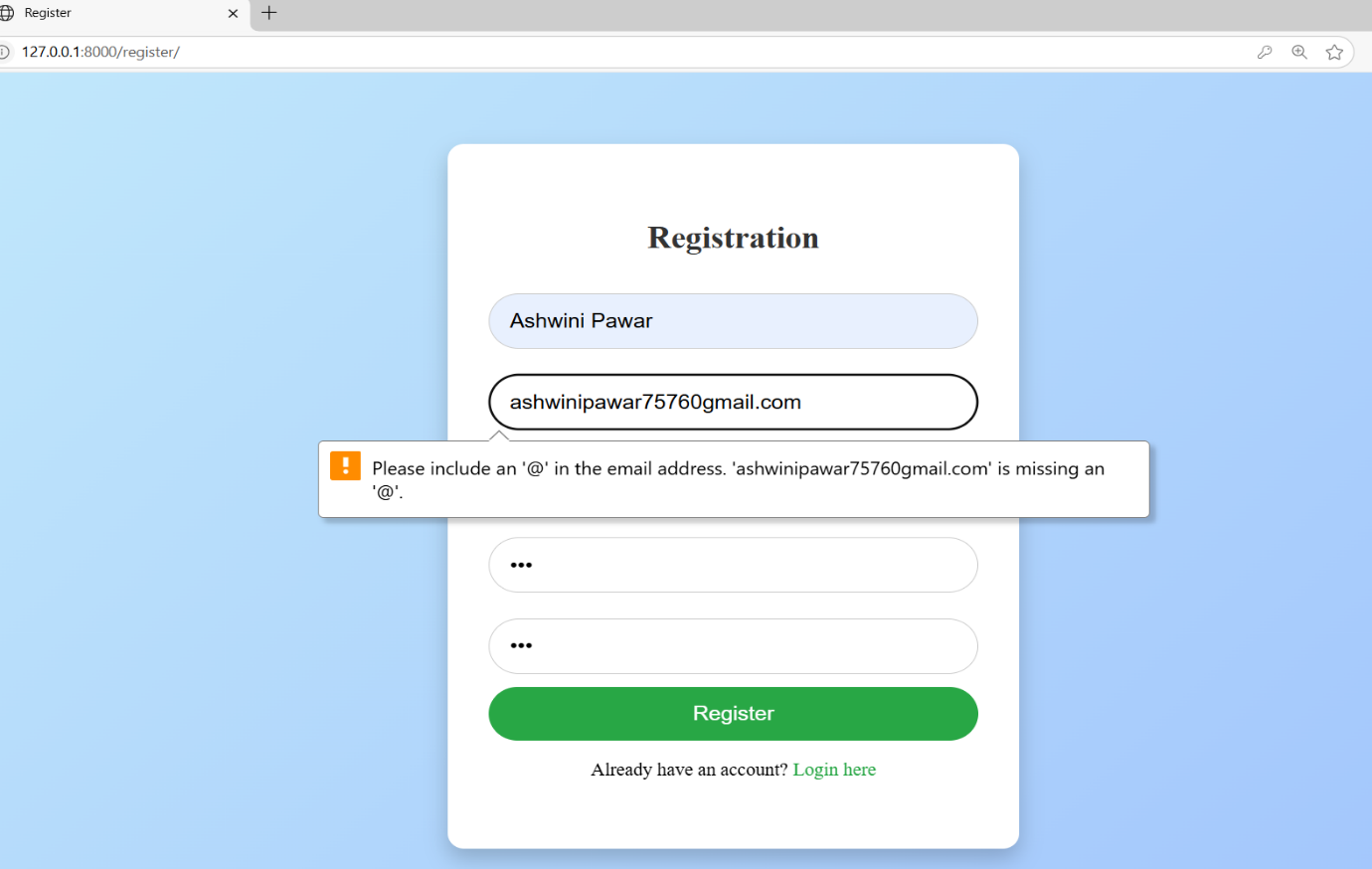
System testing was carried out on the fully integrated application. We tested the system from the user's perspective, starting from registration and login to resume upload and match result generation. Admin functionalities like managing users and viewing match histories were also validated thoroughly to ensure system-wide consistency.

**Acceptance Testing**

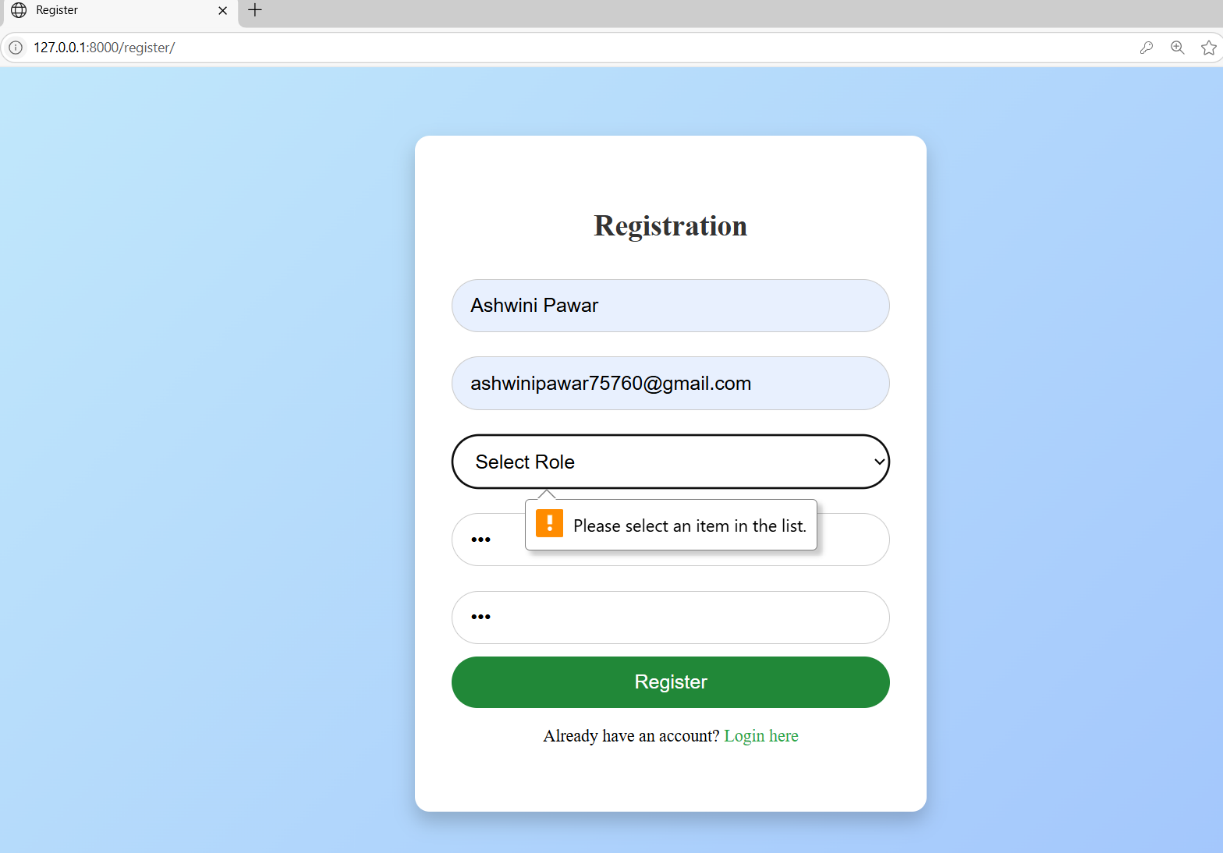
Acceptance testing was conducted to evaluate the application from an end-user point of view. After developing the application, we deployed it in a controlled environment and allowed target users (students/employees) to interact with it. Based on their feedback, the application was refined and validated to ensure that it met all predefined requirements.

**6.1 UNIT TESTING**

* **Email address Field Validation:**

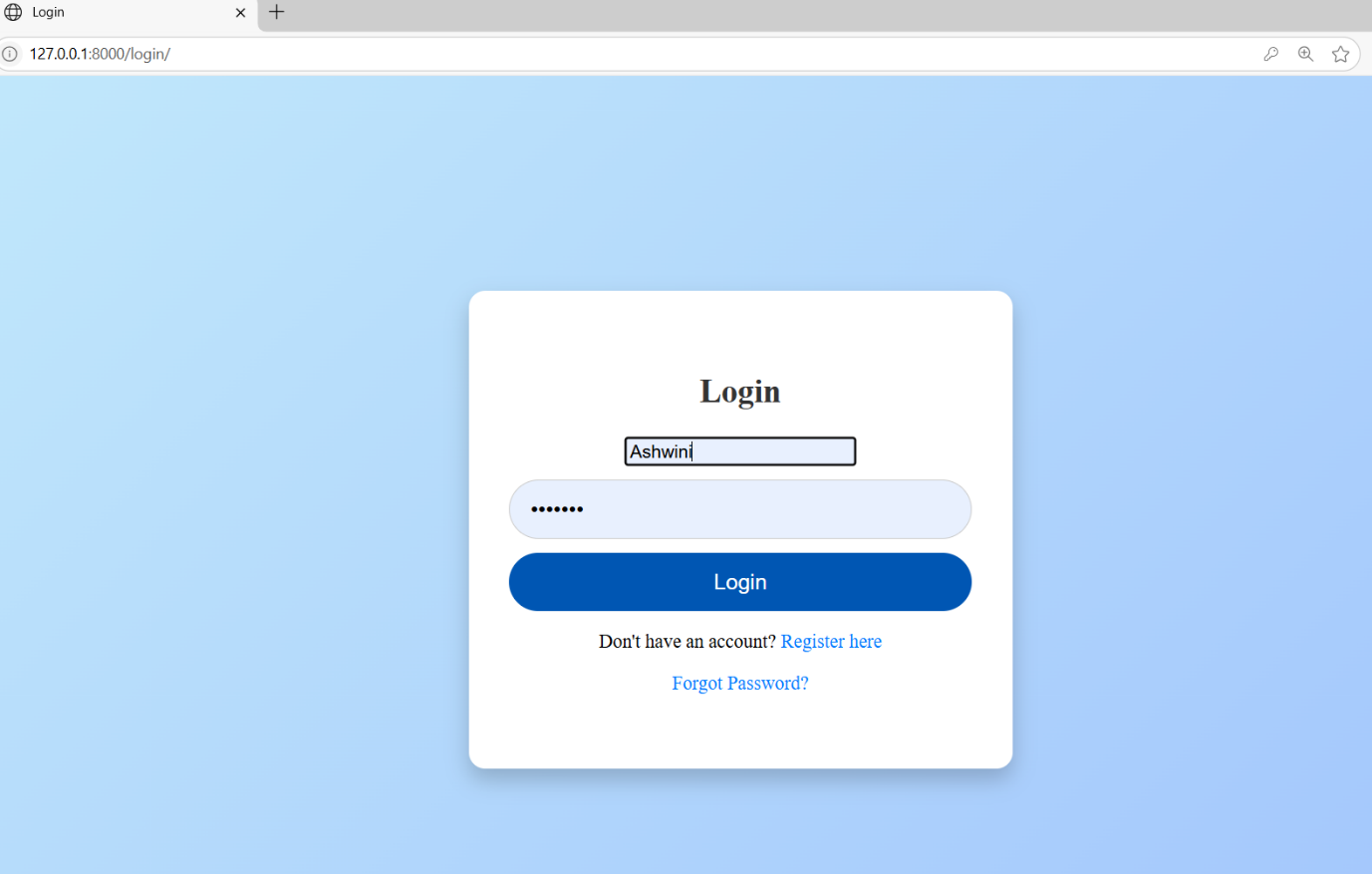


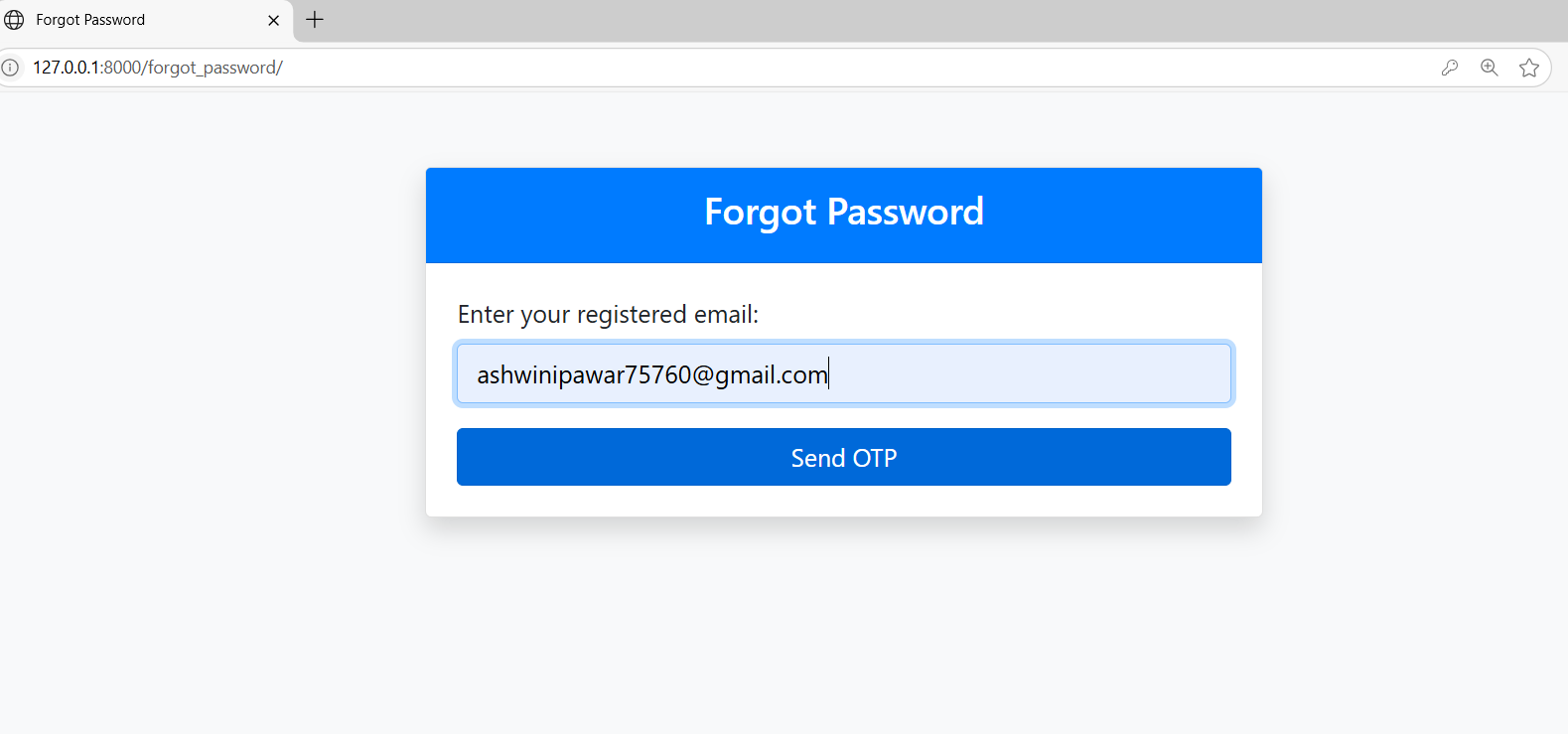
* **Empty Field Validation:**

****

**6.2 INTEGRATION TESTING**

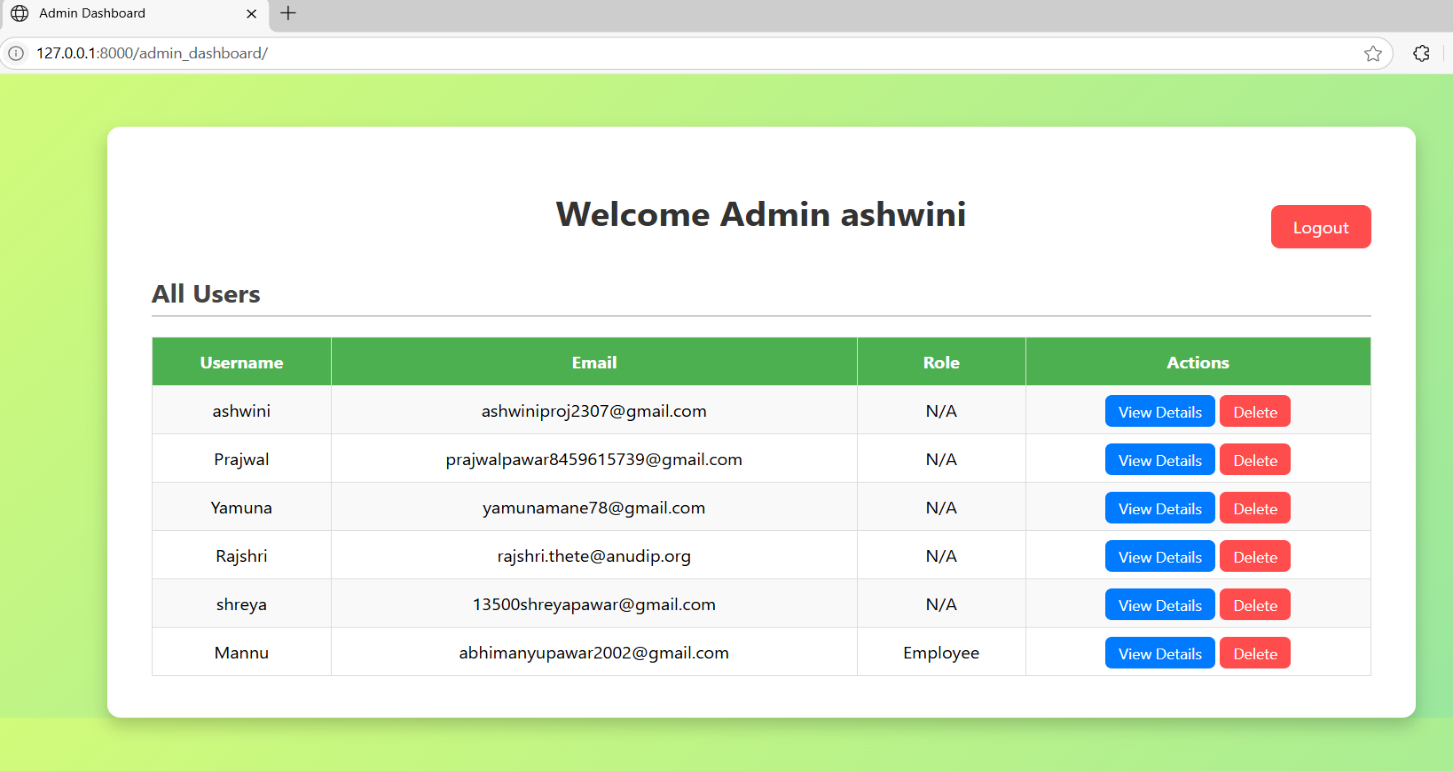
* **Hyperlink Testing:**

****

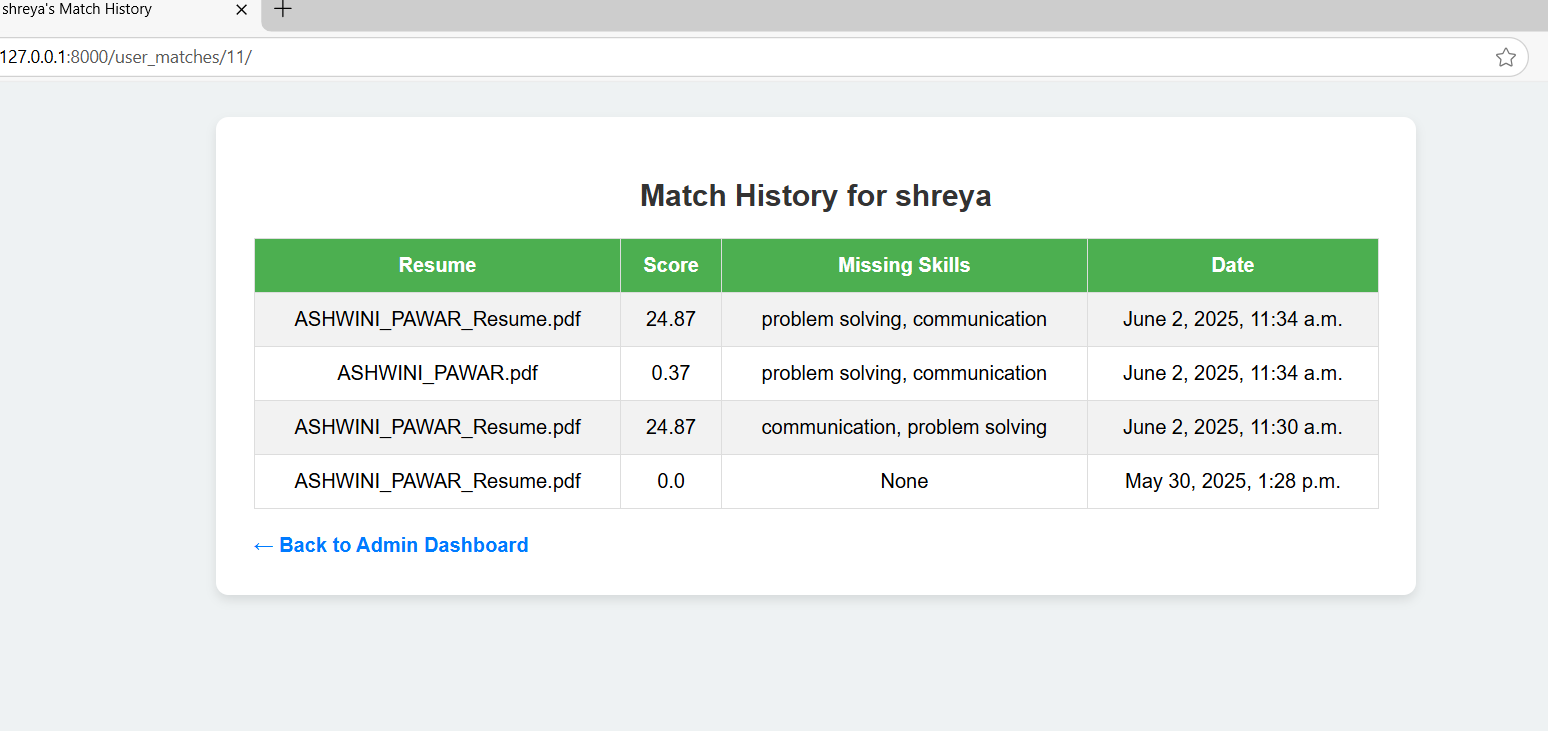


**REPORTS**

**Admin Report**



**Match History of user:**

****

**FUTURE SCOPE**

The **Job Description and Resume Matcher** system has been developed to simplify the process of comparing resumes against job descriptions. While the current version fulfils core functionality, there are several enhancements that can be implemented to make it more effective and user-friendly in the future:

1. **Interactive Dashboard with Visual Analytics**

* Incorporate a dynamic dashboard that uses pie charts, bar graphs, and scoring meters to visually represent:
* Resume match percentage with selected job descriptions.
* Distribution of skills matched vs. missing.
* Section-wise analysis (e.g., Education, Experience, Skills) contributing to the overall score.

1. **Score-Boosting Suggestions**

Offer real-time, personalized feedback with:

* Actionable suggestions (e.g., “Add more experience with X tools,” “Include leadership roles,” etc.).
* Recommended keywords to include based on job description.
* Estimated score improvement after implementing suggestions.
* Option to simulate score changes before editing the resume.

1. **AI Chat Assistant**

* Integrate a chatbot (like a fine-tuned GPT model) to:
* Answer career-related questions.
* Guide users through the platform's features.

1. **Cloud Integration & Scalability**

* Deploy the system to cloud platforms for greater scalability, security, and global access.

1. **Multilingual Support**

* Enable users to upload and match resumes in different languages, making the tool accessible to a wider audience.

**CONCLUSION**

The *Job Description and Resume Matcher* system is developed to bridge the gap between job seekers' qualifications and job requirements in a simplified and automated way. This project streamlines the comparison process by analysing resumes against job descriptions using text similarity and keyword matching techniques.

The system ensures that students and employees can easily identify how well their profiles align with specific job roles, and also highlights missing skills, which can guide further skill development. With additional features like resume uploads, similarity scoring, and match history tracking, the system offers a practical solution for career preparation.

Overall, this project not only enhances the job application process but also serves as a valuable tool for self-assessment and improvement. It lays a strong foundation for further enhancements in career support systems.

**BIBILOGRAPHY AND REFERENCE**

**BIBILOGRAPHY**

**William S. Vincent** (2018). *Django for Beginners*. WelcomeToCode.

**Mark Lutz** (2013). *Learning Python*. O'Reilly Media.

**Sebastian Raschka** (2015). *Python Machine Learning*. Packt Publishing.

**REFERENCES**

Django Documentation – <https://docs.djangoproject.com/>

MySQL Documentation – <https://dev.mysql.com/doc/>

Scikit-learn Documentation – <https://scikit-learn.org/stable/>

W3Schools – <https://www.w3schools.com/>

**YouTube – Programming with Mosh** – Web & backend development   
Channel: <https://www.youtube.com/c/programmingwithmosh>

**GeeksforGeeks** – Python, Django, MySQL, and algorithm tutorials  
Website: <https://www.geeksforgeeks.org/>

**TutorialsPoint – Python & Django**  
Website: <https://www.tutorialspoint.com/django/index.htm>