

REAL – TIME EMPLOYMENT GATEWAY USING DJANGO

A PROJECT REPORT SUBMITTED TO

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

AWARD OF THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

BY

MANIKANDAN V (REG NO. RA2332241010127)

MANIKANDAN V (REG NO. RA2332241010128)

MOHAMED HASIM N (REG NO. RA2332241010129)

KIRTHIK VYSHNAV S S A (REG NO. RA2332241010135)

UNDER THE GUIDANCE OF

Mrs. R. THILAGAVATHY M.C.A., M.Phil., Ph.D.,



DEPARTMENT OF COMPUTER APPLICATIONS

FACULTY OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Kattankulathur – 603 203

Chennai, Tamilnadu

OCTOBER – 2024

BONAFIDE CERTIFICATE

This is to certify that the project report titled “**REAL - TIME EMPLOYMENT GATEWAY USING DJANGO**” is a bonafide work carried out by **MANIKANDAN V** (RA2332241010127), **MANIKANDAN V** (RA2332241010128), **MOHAMED HASIM N** (RA2332241010129), **KIRTHIK VYSHNAV S S A** (RA2332241010135) under my supervision for the award of the Degree of Master of Computer Applications. To my knowledge the work reported herein is the original work done by these students.

Mrs. R. Thilagavathy

Assistant Professor,
Department of Computer Applications

(GUIDE)

Dr. R. Jayashree

Associate Professor & Head,
Department of Computer Applications

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

With profound gratitude to the ALMIGHTY, I take this chance to thank the people who helped me to complete this project.

We take this as a right opportunity to say THANKS to my parents who are there to stand with me always with the words “YOU CAN”.

We are thankful to **Dr. T. R. Paarivendhar**, Chancellor, and **Prof. A. Vinay Kumar**, Pro Vice-Chancellor (SBL), SRM Institute of Science & Technology, who gave us the platform to establish me to reach greater heights.

We earnestly thank **Dr. A. Duraisamy**, Dean, Faculty of Science and Humanities, SRM Institute of Science & Technology, who always encourage us to do novel things.

A great note of gratitude to **Dr. S. Albert Antony Raj**, Deputy Dean, Faculty of Science and Humanities for his valuable guidance and constant Support to do this Project.

We express our sincere thanks to **Dr. R. Jayashree**, Associate Professor & Head, for her support to execute all incline in learning.

It is our delight to thank our project guide **Mrs. R. Thilagavathy**, Assistant Professor, Department of Computer Applications, for his help, support, encouragement, suggestions, and guidance throughout the development phases of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

Our gratitude to friends and people who are known and unknown to me who helped in carrying out this project work a successful one.

MANIKANDAN V

MANIKANDAN V

MOHAMED HASIM N

KIRTHIK VYSHNAV S S A

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
2.	SOFTWARE REQUIREMENT ANALYSIS.....	2
2.1	HARDWARE SPECIFICATION	2
2.2	SOFTWARE SPECIFICATION	2
2.3	ABOUT THE SOFTWARE AND ITS FEATURE.....	3
3.	SYSTEM ANALYSIS.....	4
3.1	EXISTING SYSTEM	4
3.2	PROPOSED SYSTEM.....	5
3.3	FEASIBILITY STUDY	6
4.	SYSTEM DESIGN.....	7
4.1	DATA FLOW DIAGRAM	7
4.2	UML DIAGRAMS	9
4.3	DATABASE DESIGN	14
5.	CODE TEMPLATES.....	16
5.1	MODULE DESCRIPTION	16
5.2	LIST OF TABLES	17
6.	TESTING.....	20
6.1	TESTING METHODOLOGIES	20
7.	OUTPUT SCREENS.....	21
8.	CONCLUSION.....	26

9.	FURTHER ENHANCEMENTS.....	27
10.	REFERENCES.....	28
11.	APPENDICES.....	29
11.1	USER DOCUMENTATION	29
11.2	README.....	29
11.3	SAMPLE SOURCE PROGRAM.....	31

LIST OF TABLES

Table 5.2.1 Employer Table	17
Table 5.2.2 Employee Table	18
Table 5.2.3 Job Table	19

LIST OF FIGURES CAN BE PROVIDED

Figure 4.1.1 Level 0 DFD	7
Figure 4.1.2 Level 1 DFD	8
Figure 4.2.1 Use Case Diagram	9
Figure 4.2.2. Activity Diagram	10
Figure 4.2.3 Sequence Diagram.....	13
Figure 4.3.1 ER Diagram	14

PLAGIARISM CERTIFICATE

DETECTING EMOTION BY TEXT ANALYSIS

ORIGINALITY REPORT

0%

SIMILARITY INDEX

0%

INTERNET SOURCES

0%

PUBLICATIONS

%

STUDENT PAPERS

PRIMARY SOURCES

Exclude quotes Off

Exclude bibliography Off

Exclude matches < 10 words

ABSTRACT

The Real - Time Employment Gateway using Django is a dynamic web application developed using the Django framework, aimed at bridging the gap between job seekers and employers by providing a centralized platform for recruitment. In today's digital world, traditional methods of job hunting and recruitment have become increasingly inefficient and time-consuming. This portal provides a comprehensive solution by automating the process of job search, application submission, and employer-candidate interactions. The system is designed with a user-friendly interface that simplifies the hiring process, catering to both job seekers looking for suitable opportunities and employers seeking qualified candidates. The portal offers distinct features for its two primary user types: job seekers and employers. Job seekers can create personalized profiles, browse job listings, and apply for jobs that match their skills and interests. An intelligent search algorithm ensures that users can filter opportunities by location, industry, experience level, and more. Additionally, job seekers receive real-time updates on application status and relevant new job postings. On the employer side, the platform allows organizations to post job openings, review applications, shortlist candidates. The system integrates advanced features to help employers make informed hiring decisions faster. Employers can also manage job postings and track candidate progress through an intuitive dashboard, reducing the administrative burden of the hiring process. The back-end of the application is developed using Django, a robust Python-based web framework, which ensures a secure, scalable, and maintainable system. Django's ORM (Object-Relational Mapping) streamlines database management, ensuring that data such as user profiles, job listings, and applications are stored and retrieved efficiently. Furthermore, Django's security features, including user authentication and data protection, safeguard user information and ensure privacy. To enhance the user experience, the portal includes an admin panel for managing the overall system. The use of HTML, CSS, and JavaScript for front-end development ensures a responsive and interactive design, making the portal accessible across various devices. The portal is built to handle multiple users simultaneously, leveraging Django's scalability, ensuring smooth operation even under heavy load.

1. INTRODUCTION

In the modern digital age, the job market has undergone a profound transformation, largely driven by advancements in technology and the growing reliance on online platforms for job searches and recruitment. The Real - Time Employment Gateway using Django serves as a vital bridge connecting job seekers with potential employers, streamlining the hiring process and making it more efficient for all parties involved. As the demand for user-friendly and intuitive job search solutions continues to rise, the development of such platforms has become increasingly important. This project aims to create a comprehensive Real - Time Employment Gateway using Django, a powerful and flexible web framework that promotes rapid development and clean, pragmatic design. Django's robust architecture and extensive libraries facilitate the creation of a highly scalable platform capable of handling a large number of users and job listings. The use of Django also ensures security features, which are essential in maintaining user privacy and safeguarding sensitive information. The primary objective of this Portal is to provide a seamless experience for both job seekers and employers. Job seekers can easily create profiles and search for job opportunities based on various criteria, including location, industry, and job type. Additionally, the portal will feature advanced search functionalities and filters, enabling users to refine their searches effectively. Employers, on the other hand, can leverage the platform to post job openings, manage applications, and communicate with candidates. The admin dashboard will empower employers to track application statuses and view analytics on job postings, enhancing their ability to make informed hiring decisions. Furthermore, the portal will support notifications and alerts, keeping both job seekers and employers informed about relevant updates, new job postings, and application statuses. The significance of developing an Real - Time Employment Gateway using Django extends beyond simply providing a platform for job search; it addresses the pressing need for efficient recruitment solutions in an increasingly competitive job market. By integrating various features such as real-time job alerts, user authentication, and application tracking, the portal aims to create an engaging and interactive user experience.

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 HARDWARE SPECIFICATION

OS	Windows: 10 or newer MAC: OS X v10.7 or higher Linux: Ubuntu
Processor	2GHz or more
RAM	2.00 GB or above
Hard Disk	64 GB or more
Network	Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)

2.2 SOFTWARE SPECIFICATION

IDE	Visual Studio Code
Web Server	Localhost:8000
Server-Side Scripting	Django (Python)
Client-Side Scripting	JS & jQuery
Front End	HTML & CSS
Database	MySQL
Browser	Chrome or Edge

2.3 ABOUT THE SOFTWARE AND ITS FEATURE

2.3.1 REGISTRATION

In the registration module job seeker have to include all the details like personal details, contact details, etc. Also job seeker has to add his experience details, job requirements. While job recruiter has to add his contact details and organization details for the registration and upload company profile.

2.3.2 JOB POST

Employer can post a job by providing all the job details like qualifications details, requirements for the job, designation details, job salary details and also provide type of jobs. They also can delete the jobs whenever they want. After successfully posted a job it will be available for all the job seekers who are searching for a job. And it will be available on home page as recently posted job.

2.3.3 SEARCH

Employee Can Search job according to their interest. And also apply for that job or they can add into bookmark for future whenever they find for job for that company then they easily find out company from bookmark. Employer search candidates for their requirements using keyword like technology. And also can communicate with employee for their any other query or information via send message .and also employer see the resume of applicants.

2.3.4 MANAGE ACCOUNT

While employers can manage their job postings. And providing all the job details like qualifications details, requirements for the job, designation details, job salary details and also provide type of jobs. They also can delete the jobs whenever they want. While employee can manage their bookmark, applied for job and also getting full details of employer. Employees can delete their account anytime. Also they can apply for the different jobs according to their interests.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In recent years, online job portals have become integral to the hiring and job-seeking processes for employers and applicants alike. These systems are designed to facilitate connections between job seekers and companies, enabling applicants to search and apply for job postings while employers gain access to a pool of candidates. However, many existing job portal systems still face limitations that prevent them from fully meeting the needs of all users involved. Here's a detailed overview of the existing system. Many current portals lack personalized recommendations that match job seekers with roles based on their unique skill sets, qualifications, and job preferences. This forces applicants to search manually through hundreds of listings, which can be time-consuming and inefficient. Employers on existing platforms often struggle to filter out irrelevant applications and identify high-quality candidates quickly. Current systems usually rely on generic filters that may not be sophisticated enough to narrow down to a smaller, more qualified pool. Many job portals use basic search algorithms that rely on keywords rather than deeper insights into the job seeker's profile. This results in mismatches where irrelevant job listings appear for applicants, diminishing the overall experience. Advanced job portals that do employ machine learning and AI for recommendation algorithms are still limited, as they may not fully account for a candidate's soft skills, work style, or other critical attributes that influence job fit. Job seekers and employers alike face difficulties managing applications effectively. Job seekers cannot easily track which roles they've applied for, making it hard to follow up or withdraw applications. Employers are often forced to sift through a high volume of applications without suitable sorting tools, leading to longer hiring timelines and increased administrative burdens. Many job portals have outdated, cluttered interfaces that lack intuitive navigation and design flexibility. The one-size-fits-all approach of these platforms makes it challenging for users to efficiently access information or find specific functionalities. This user experience limitation affects both mobile and desktop users, creating a frustrating experience for job seekers and employers.

3.2 PROPOSED SYSTEM

The To overcome the limitations of existing job portals and provide a comprehensive platform that serves the needs of job seekers, employers, and administrators, the proposed system incorporates advanced features focused on usability, efficiency, and intelligence. By leveraging modern technologies such as a responsive user interface, and robust data management tools, this proposed system aims to streamline the hiring process, enhance user engagement, and ensure data security. The system will also allow users to create multiple tailored profiles for different roles, enhancing their appeal to various employers. Employers can establish comprehensive company profiles, with sections for company background, industry. This feature provides job seekers with context on prospective employers, helping them make informed application decisions. Employers will have access to tools that allow them to review, sort, and manage applications in an organized way. They can categorize candidates by application status and shortlist or reject candidates with ease. Both job seekers and employers benefit from intuitive dashboards tailored to their needs. Job seekers have easy access to their profile, job recommendations, and application statuses, while employers have tools for posting jobs, managing candidates, and viewing analytics. The system offers analytics tools for employers to track job post performance, including application numbers, views, and engagement metrics. This data provides valuable insights, helping employers improve job listings and target the right audience. The system allows users to manage their visibility and decide who can view their profiles or personal details, providing an additional layer of privacy control. The proposed system implements industry-standard encryption to safeguard user data. Sensitive information such as job history, and contact details are securely stored, ensuring compliance with data protection regulations. Different user types (e.g., job seekers, employers, admins) have specific access levels, ensuring only authorized individuals can view or edit sensitive information.

3.3 FEASIBILITY STUDY

As the name implies, a feasibility study is used to determine the viability of an idea, such ensuring this project is legally and technically feasible as well as economically justifiable. This indicated whether this project is worth the investment-in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

3.3.1 FEASIBILITY SYSTEM

The objective of feasibility study is not only to solve the problem but also to acquire a sense of its scope. During the study, the problem definition is crystallized and aspects of the problem to be included in the system are determined. Consequently, benefits are estimated with greater accuracy at this stage. The key considerations are:

- Economic feasibility
- Technical feasibility
- Operational feasibility

3.3.2 ECONOMIC FEASIBILITY

Economic feasibility studies not only the cost of hardware, software is included but also the benefits in the form of reduced costs are considered here. This project, will certainly be beneficial since there will be reduction in manual work due to the recommendation system and increase in the productivity. It need not require any additional hardware resources as well as it will be saving lot of time.

3.3.3 TECHNICAL FEASIBILITY

Technical feasibility evaluates the hardware requirements, software technology, available personnel etc., This project is very much technically feasible. This project is very much concerned with specifying equipment and the project will successfully satisfy almost all the user's requirements.

Therefore, the basic input/output of all data is identified. So, the project can easily be built up and it will also be technically feasible

3.3.4 OPERATIONAL FEASIBILITY

Proposed system is beneficial only if they can be turned into information systems that will meet the organizational requirements. This system supports in producing good results and reduces manual work and aids in decision-making on the project.

4. SYSTEM DESIGN

4.1 DATA FLOW DIAGRAM

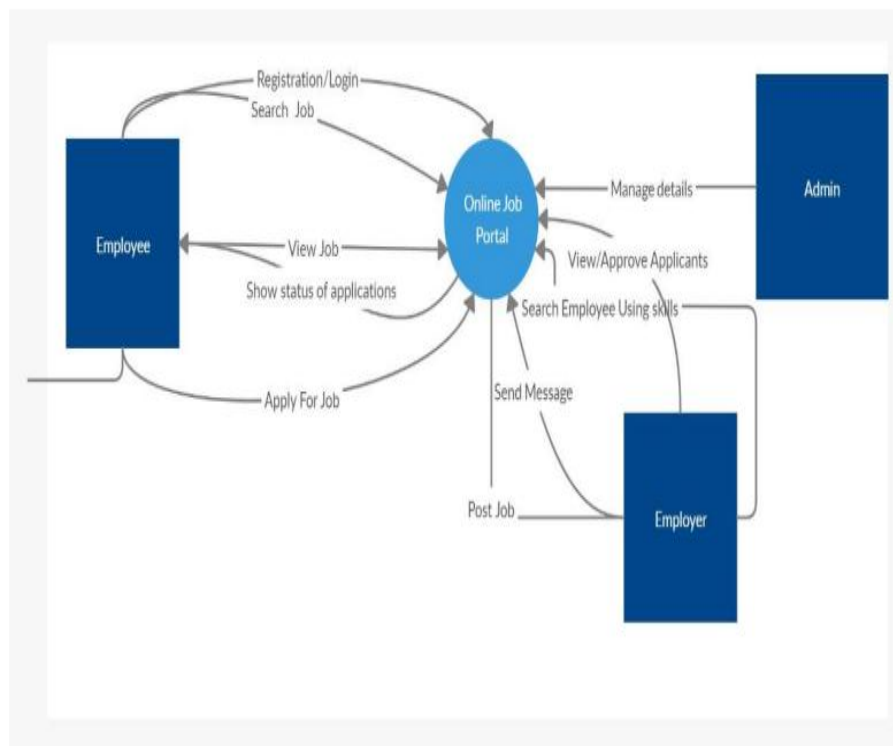


Figure 4.1.1 Level 0 DFD

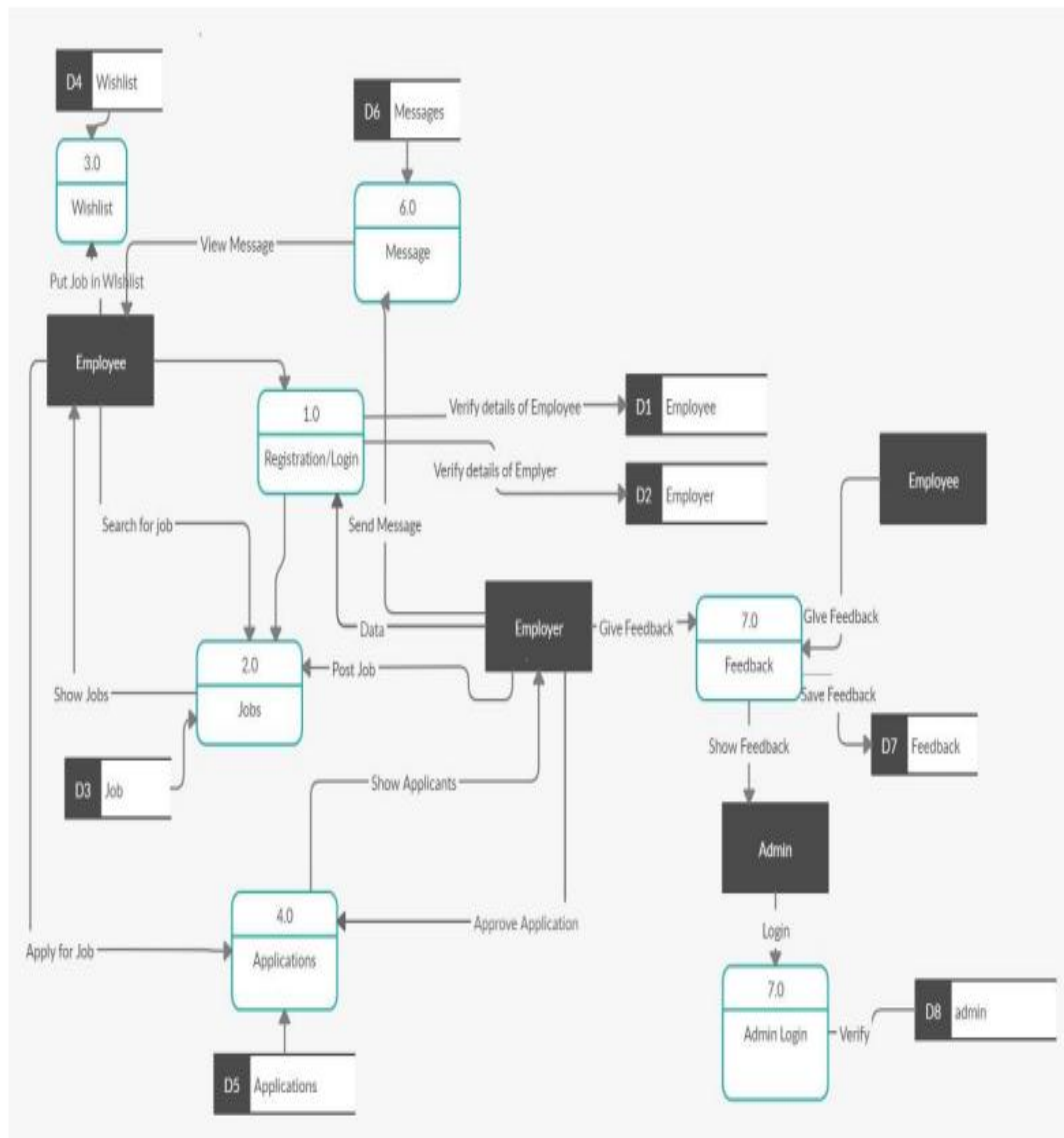


Figure 4.1.2 Level 1 DFD

4.2 UML DIAGRAMS

4.2.1 USE CASE DIAGRAM

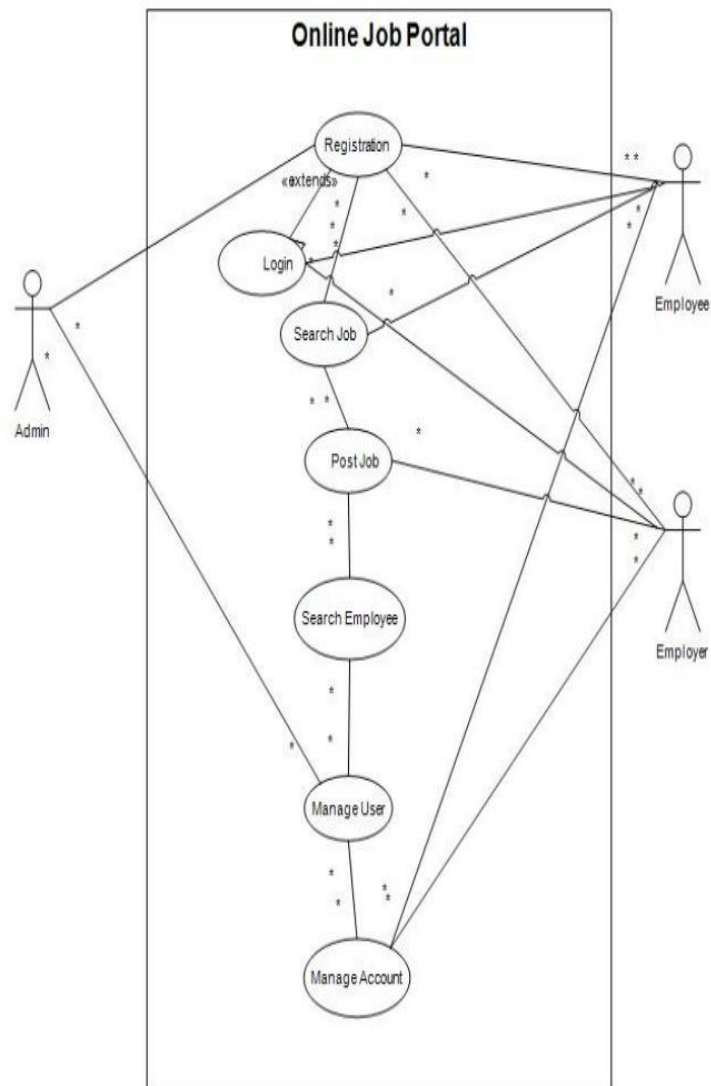


Figure 4.2.1.1 Use Case Diagram

4.2.2 ACIVITY DIAGRAM

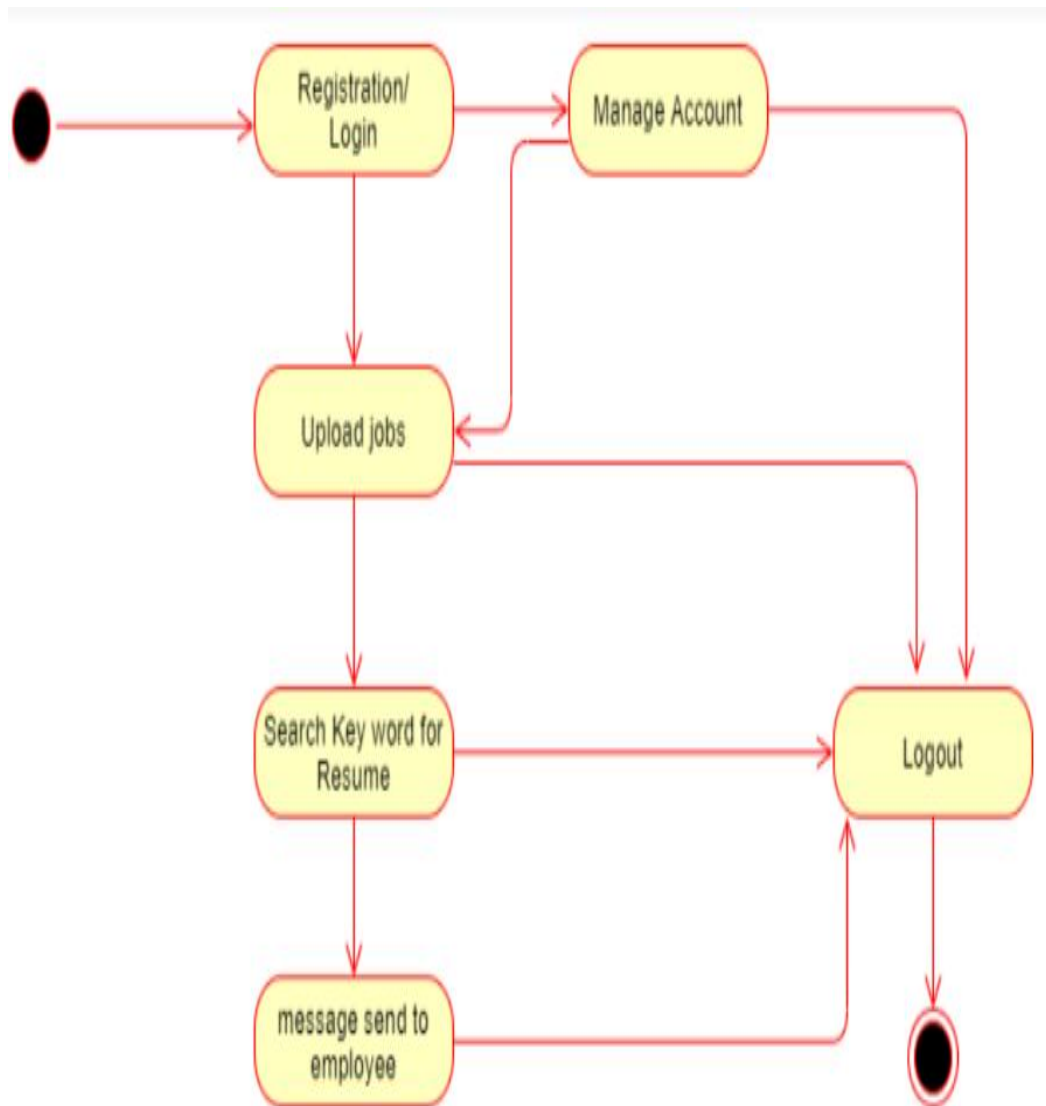


Figure 4.2.2.1 Employer Activity Diagram

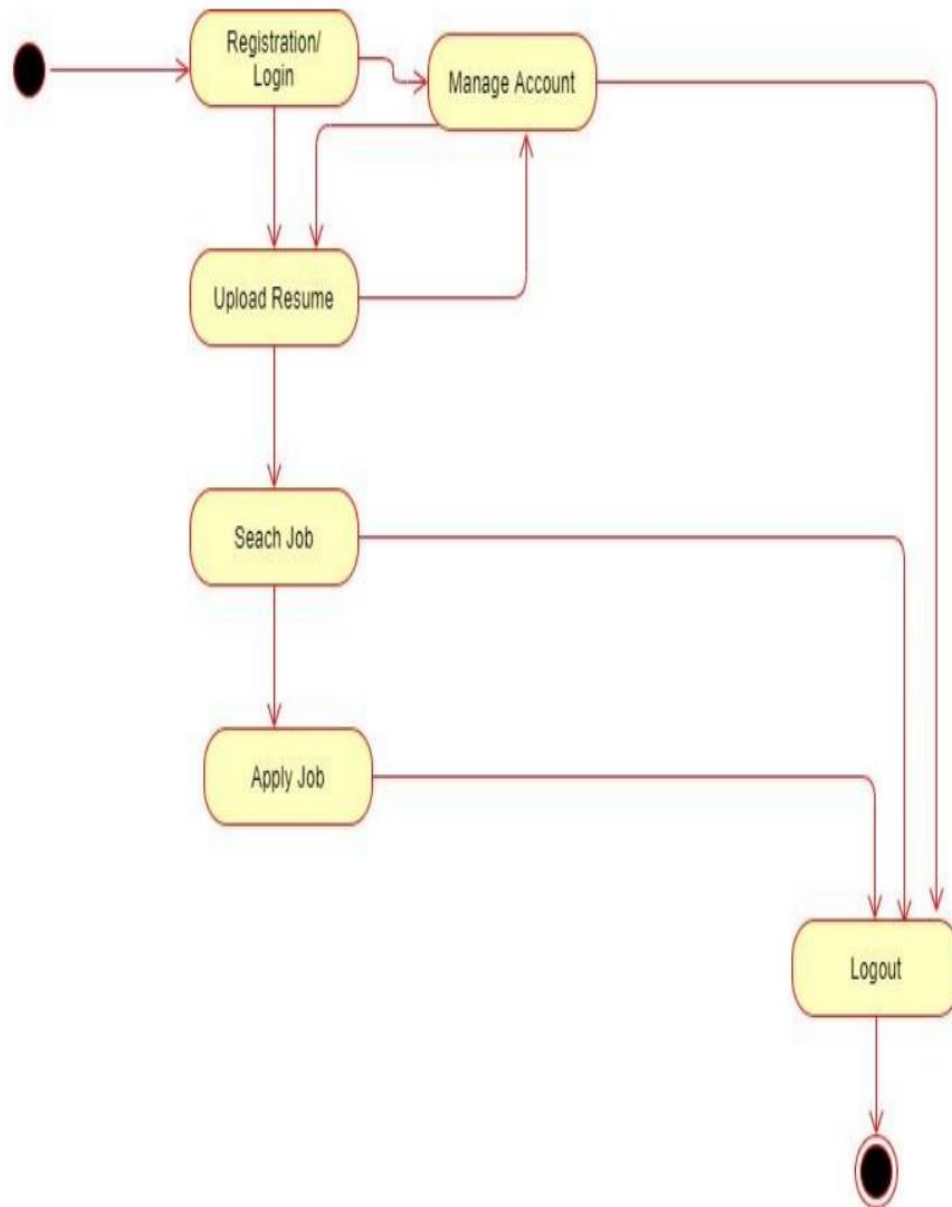


Figure 4.2.2.2 Employee Activity Diagram

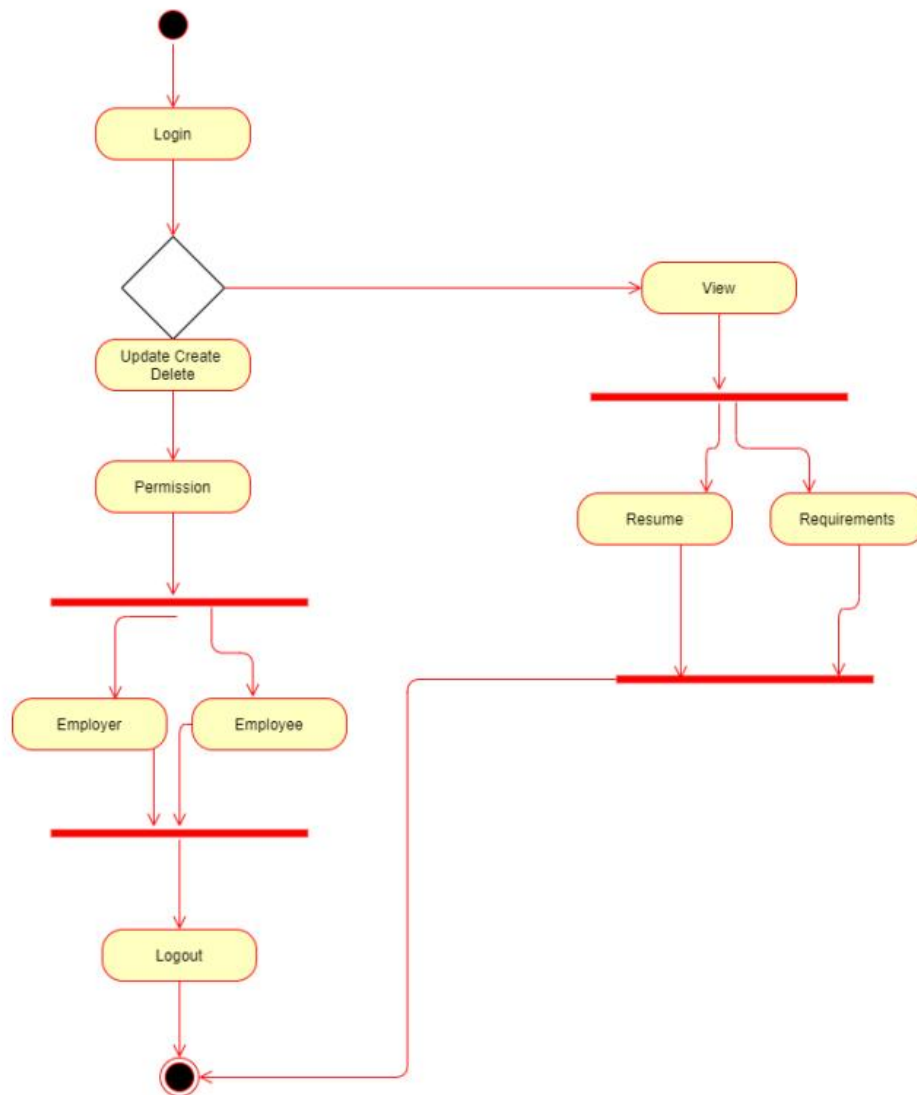


Figure 4.2.2.3 Admin Activity Diagram

4.2.3 SEQUENCE DIAGRAM

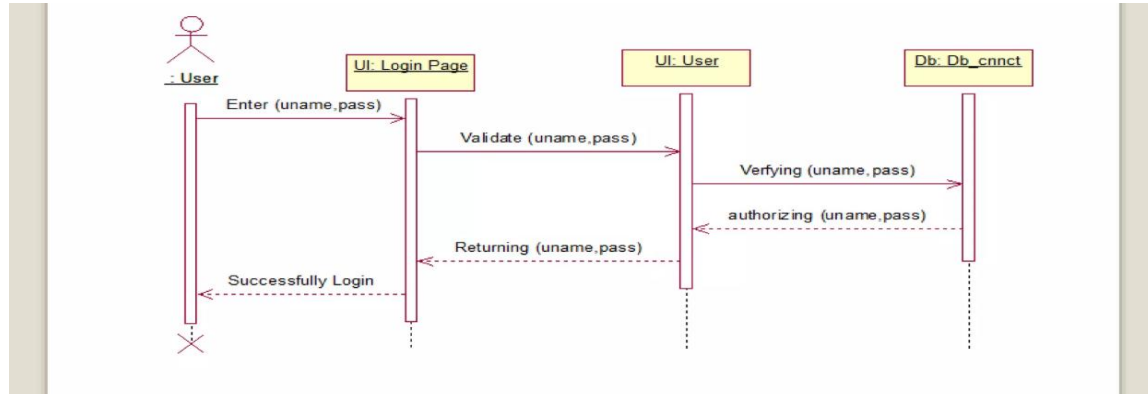


Figure 4.2.3.1 Employee Sequence Diagram

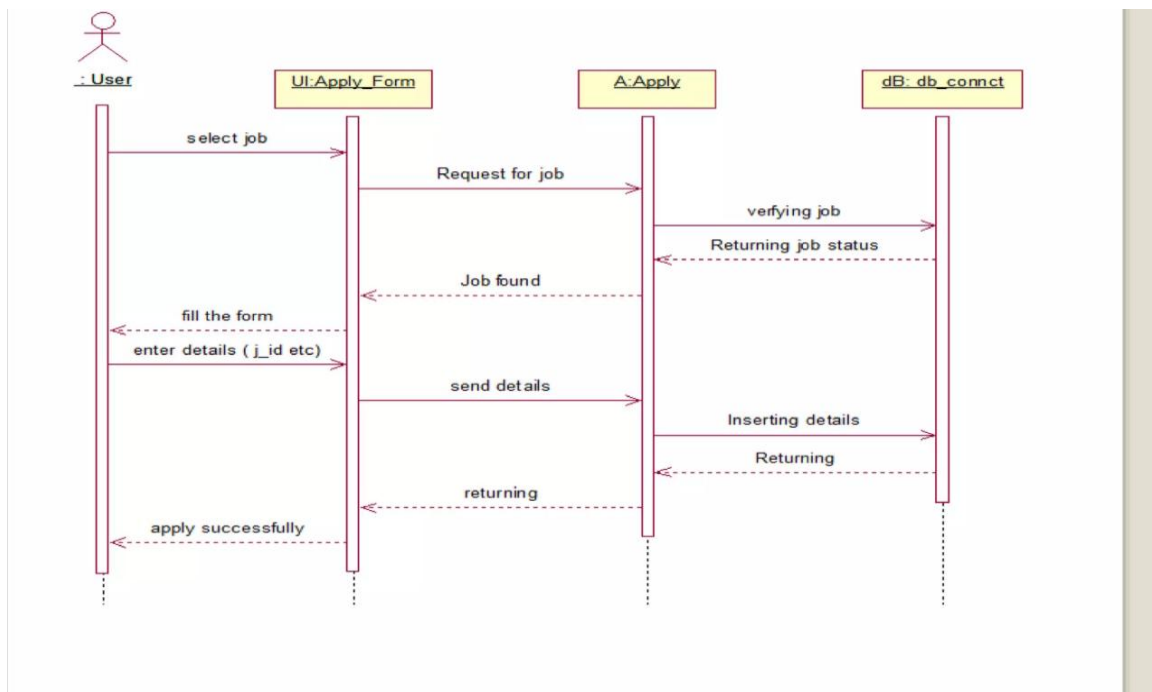


Figure 4.2.3.2 Job Apply Sequence Diagram

4.3 DATABASE DESIGN

4.3.1 ER DIAGRAM

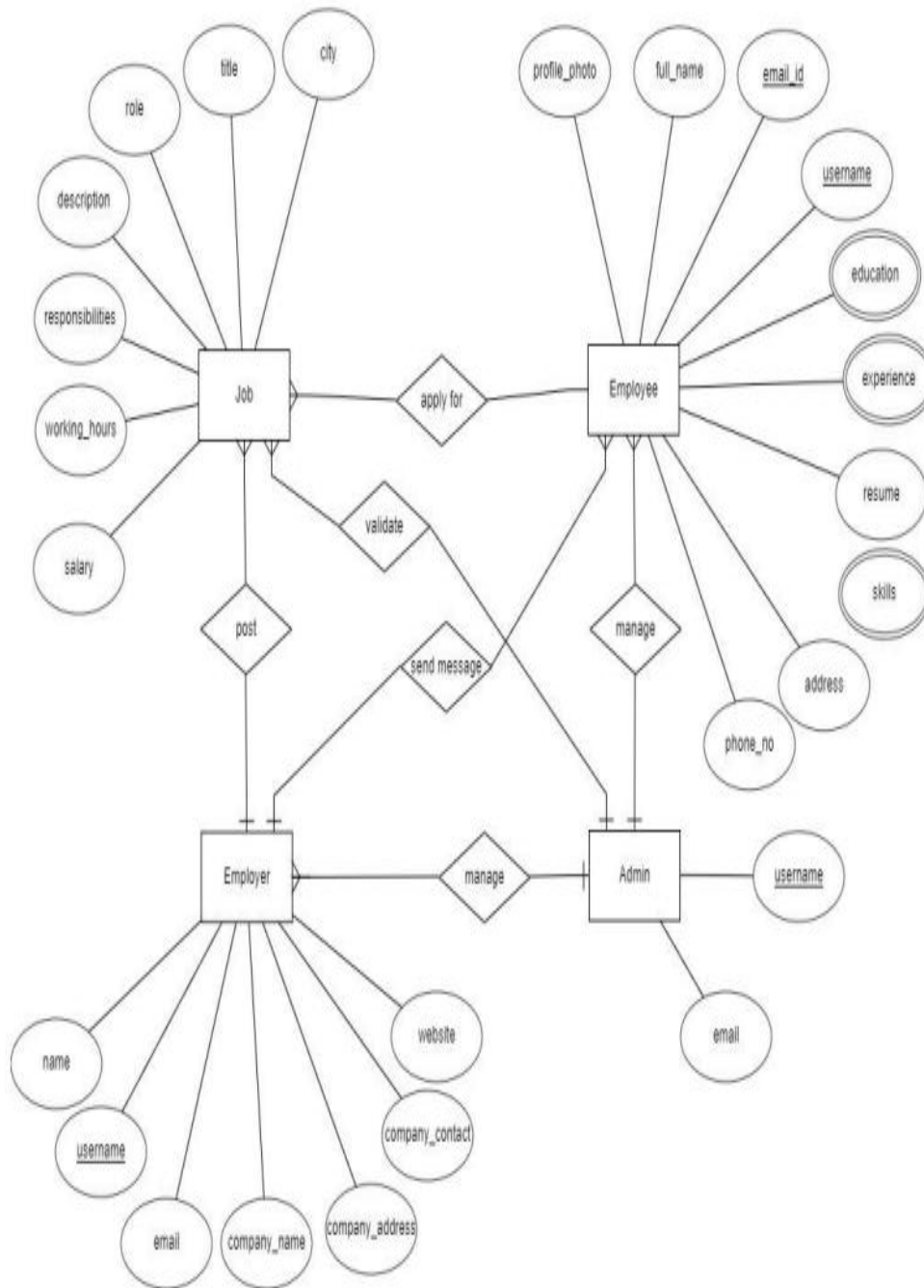


Figure 4.3.1.1 ER Diagram

4.3.2 FDs AND NORMALIZATION

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

4.3.2.1 Third Normal Form (3NF)

A table is said to be in 3NF if it is in 2NF and every non-key attribute is not dependent on any other non-key attribute. All non-key attributes dependent on other non-key attributes should be isolated to form a new entity

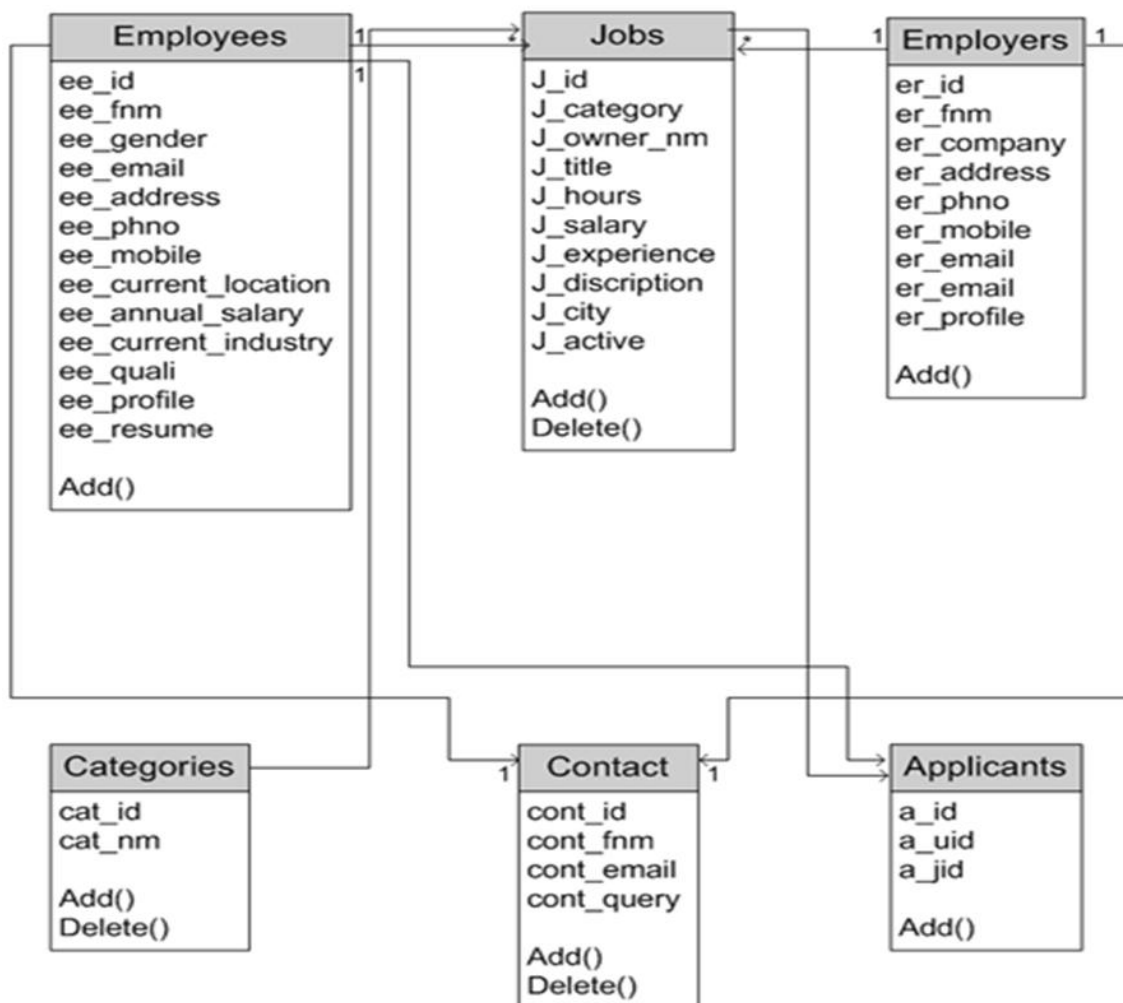


Figure 4.3.2.1.1 Database Schema

5. CODE TEMPLATES

5.1.1.2 JOB SEARCH

A robust search functionality allows employees to filter jobs based on criteria such as location, job type, industry, and salary.

5.1 MODULE DESCRIPTION

This Online Fitness Client Manager provides 3 main modules to operate.

5.1.1 EMPLOYEE MODULE

3.1.1.1 PROFILE CREATION

Employees can create detailed profiles showcasing their skills, experience, education, and resumes.

5.1.1.3 APPLICATION TRACKING

Employees can track their job applications, view the status of their applications, and receive notifications about interview schedules or updates from employers.

5.1.2 EMPLOYER MODULE

5.1.2.1 JOB POSTING

Employers can create and post job listings with detailed descriptions, requirements, and company information.

5.1.2.2 APPLICATION MANAGEMENT

Employers can manage job applications, schedule interviews, and communicate with candidates through an integrated messaging system.

5.1.2.3 ANALYTICS DASHBOARD

An analytics dashboard provides insights into job posting performance, including views, applications received, and candidate engagement metrics.

5.1.3 ADMIN MODULE

5.1.3.1 USER MANAGEMENT

Admins can manage employee and employer accounts, including verification, approval, and the ability to suspend or delete accounts if necessary.

5.1.3.2 DATA MANAGEMENT

Ability to manage and analyze data related to user activity, job postings, and application trends to inform strategic decisions.

5.2 LIST OF TABLES

5.2.1 EMPLOYER TABLE

Table 1:				
Name:		employer		
Description:		Records information regarding employer.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	er_id	int	primary key	Holds unique id of employer.
2	er_full_name	varchar(50)	Null	Holds full name of the employer.
3	er_user_name	varchar(50)	Null	Holds user name of the employer.
4	er_email	varchar(50)	Null	Holds e-mail id of the employer.
5	er_pwd	varchar(50)	Null	Holds password of employer.
6	er_comp_name	varchar(50)	Null	Holds company name.
7	er_comp_address	text	Null	Holds company address.
8	er_comp_phone	varchar(20)	Null	Holds company phone number.
9	er_comp_web_add	varchar(50)	Null	Holds company website URL.
10	er_comp_fax	varchar(50)	Null	Holds company fax number.
11	er_comp_logo	varchar(max)	Null	Holds company logo.
12	er_comp_profile	varchar(max)	Null	Holds company profile
13	er_sec_que	varchar(50)	Null	Holds security question
14	er_sec_ans	varchar(50)	Null	Holds security answer

Table 5.2.1 Employer Table

5.2.2 EMPLOYEE TABLE

Table 2:				
Name:		employee		
Description:		Records information regarding employee.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	ee_id	int	primary key	Holds the employee-id.
2	ee_full_name	varchar(50)	Null	Holds full name of employee.
3	ee_user_name	varchar(50)	Null	Holds user name of employee.
4	ee_email	varchar(50)	Null	Holds email id of employee.
5	ee_pwd	varchar(250)	Null	Holds password of employee.
6	ee_education	varchar(50)	Null	Holds education detail of employee.
7	ee_experience	Int	Null	Holds experience detail of employee.
8	ee_past_work	Text	Null	Holds past work detail of employee.
9	ee_photo	varchar(255)	Null	Holds photograph of employee.
10	ee_resume	varchar(255)	Null	Holds resume of employee.
11	ee_phone_no	varchar(50)	Null	Holds phone no of employee.
12	ee_address	Text	Null	Holds address of employee.
13	ee_gender	Char(1)	Null	Holds gender of employee.

Table 5.2.2 Employee Table

5.2.3 JOB TABLE

Table 4:				
Name:		Job		
Description:		Records information about job posted by recruiter.		
Fields				
Sr. No.	Field Name	Field Type	Constraints	Description
1	j_id	int	primary key	Holds id of job.
2	j_er_id	int	foreign key	Holds id of employer.
3	j_jc_id	int	foreign key	Holds id of job category.
4	j_sub_cat_id	Int	Foreign key	Holds id of job sub category
5	j_title	varchar(150)	Null	Holds title of the job.
6	j_exp_required	varchar(50)	Null	Holds job experienced required
7	j_min_edu	varchar(50)	Null	Holds job minimum education
8	j_desc	Text	Null	Holds description of job.
9	j_role	Text	Null	Holds role of job.
10	j_respo	Text	Null	Holds responsibilities of job.
11	j_city	varchar(50)	Null	Holds city name of job.
12	j_work_hours	Int	Null	Holds working hours detail of job
13	j_exp_salary	varchar(50)	Null	Holds expected salary of job.
14	j_has_bond	varchar(50)	Null	Holds information of bond information.

Table 5.2.3 Job Table

6. TESTING

6.1 TESTING METHODOLOGIES

6.1.1 FUNCTIONALITY TESTING

The main goal of functional testing is to make sure that all the functions within this web app are working smoothly without any technical glitches. In this web application, functional testing can cover different things like whether all the links are working properly or not, testing forms in all the pages, validating HTML or CSS, testing database for the security etc.

6.1.2 USABILITY TESTING

When it comes to making this application user-friendly and effective, its user interface should comply with the standards. Menus, buttons and links to different pages should be visible and consistent on all webpages, content should be legible with no spelling mistakes as well as grammatical errors, images (if present) should contain an "alt" text.

6.1.3 WEB UI TESTING

The most important interfaces within a web application are web server, application server interface and database server interface. Web UI testing will ensure that all the individual components in the web application are connected appropriately and checks whether the interaction between these servers are executed properly or not.

6.1.4 COMPATIBILITY TESTING

Compatibility of this web application is one of the most crucial things to be considered while testing the application. Compatibility testing will check this web application for browser compatibility, and operating system compatibility.

6.1.5 PERFORMANCE TESTING

Performance testing will help in determining the performance of this web application under various scenarios. Performance testing usually includes stress testing, scalability testing and load testing. In this method, website is usually tested for its functionality on different operating system, hardware platforms etc.

7. OUTPUT SCREENS

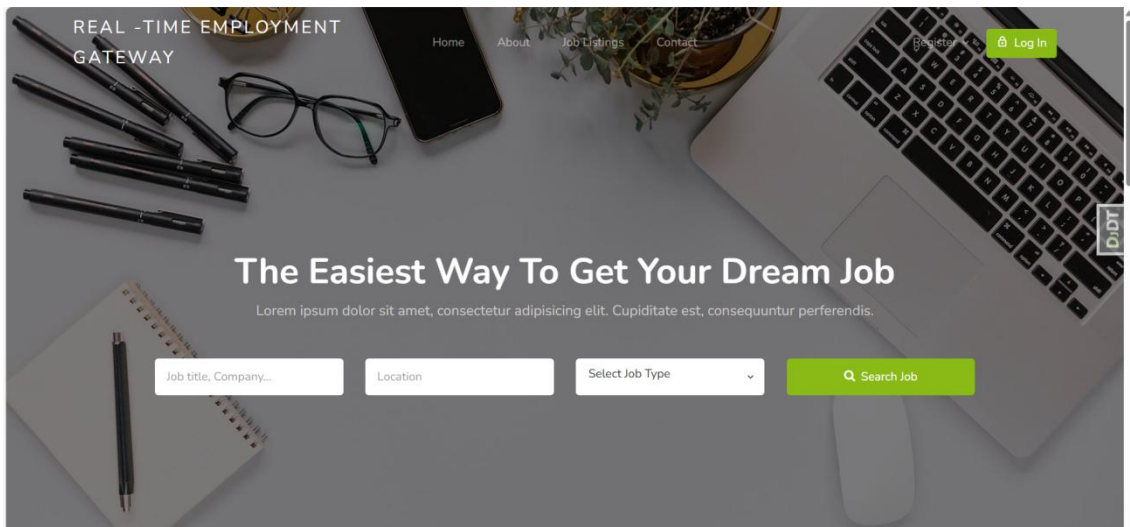


Photo 7.1 Home Page

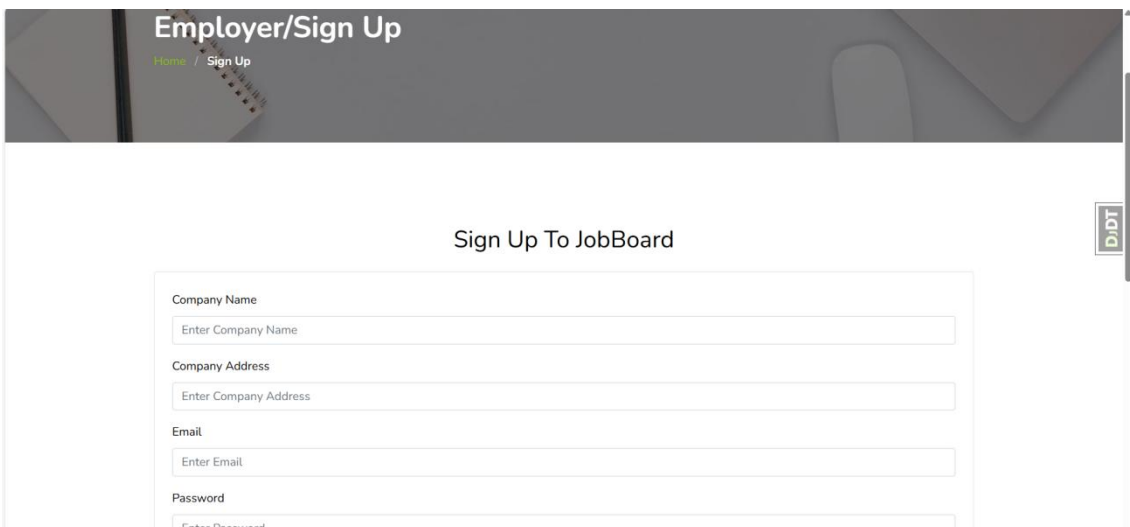
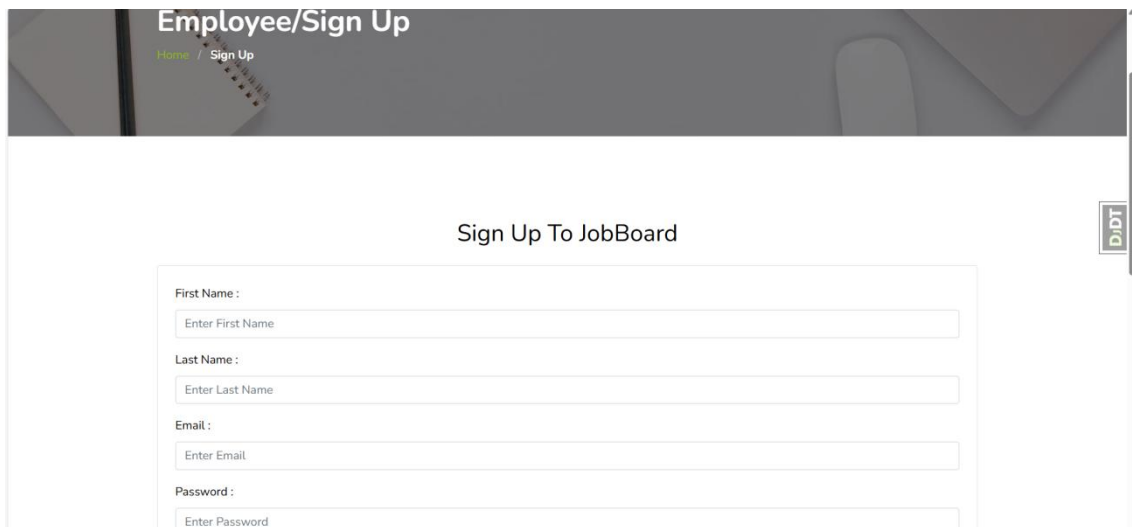


Photo 7.2 Employer Sign up



The image shows a web page for signing up as an employee. The header has a dark background with the text 'Employee/Sign Up' and a breadcrumb trail 'Home / Sign Up'. The main content area is white and features a centered heading 'Sign Up To JobBoard'. Below this is a form with four input fields: 'First Name', 'Last Name', 'Email', and 'Password', each with a placeholder text 'Enter [Field Name]'. A vertical sidebar on the right contains a 'D/DT' button.

Employee/Sign Up

[Home](#) / [Sign Up](#)

Sign Up To JobBoard

First Name :

Last Name :

Email :

Password :

D/DT

Photo 7.3 Employee Sign up



The image shows a web page for logging in. The header has a dark background with the text 'Log In' and a breadcrumb trail 'Home / Log In'. The main content area is white and features a centered heading 'Log In'. Below this is a form with two input fields: 'Email' and 'Password', each with a placeholder text 'Email' and 'Password' respectively. A green 'Sign In' button is located below the password field. A vertical sidebar on the right contains a 'D/DT' button.

Log In

[Home](#) / [Log In](#)

Log In

Email

Password

D/DT

Photo 7.4 Login Page

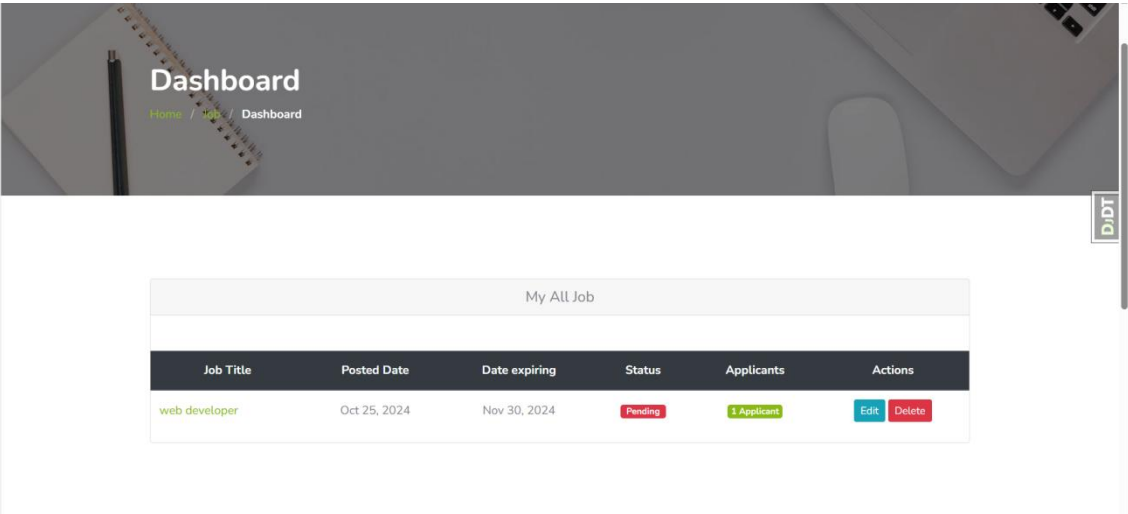


Photo 7.5 Employer Dashboard

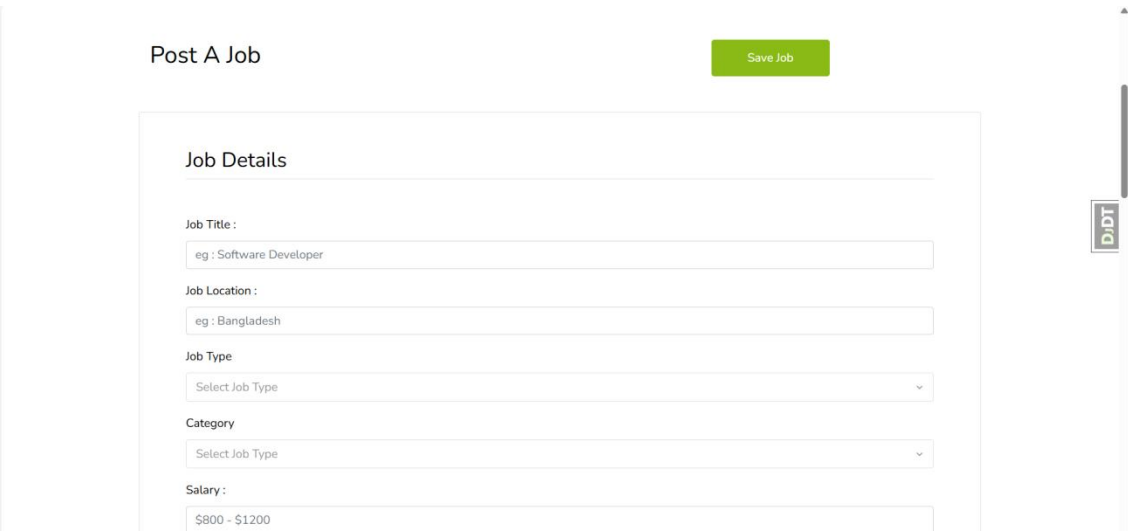


Photo 7.6 Posting a Job

Update Profile

[Home](#) / [Update Profile](#)

Update Profile

First name

mani

Last name

v

☒ Male ☐ Female

Update

Photo 7.7 Employee Profile Update

Dashboard

[Home](#) / [Jobs](#) / [Dashboard](#)

Bookmarked Job

Aplied Job

Bookmarked Posts

Job Title	Posted Date	Date expiring	Actions
web developer	Oct 25, 2024	Nov 30, 2024	Delete

Photo 7.8 Employee Dashboard

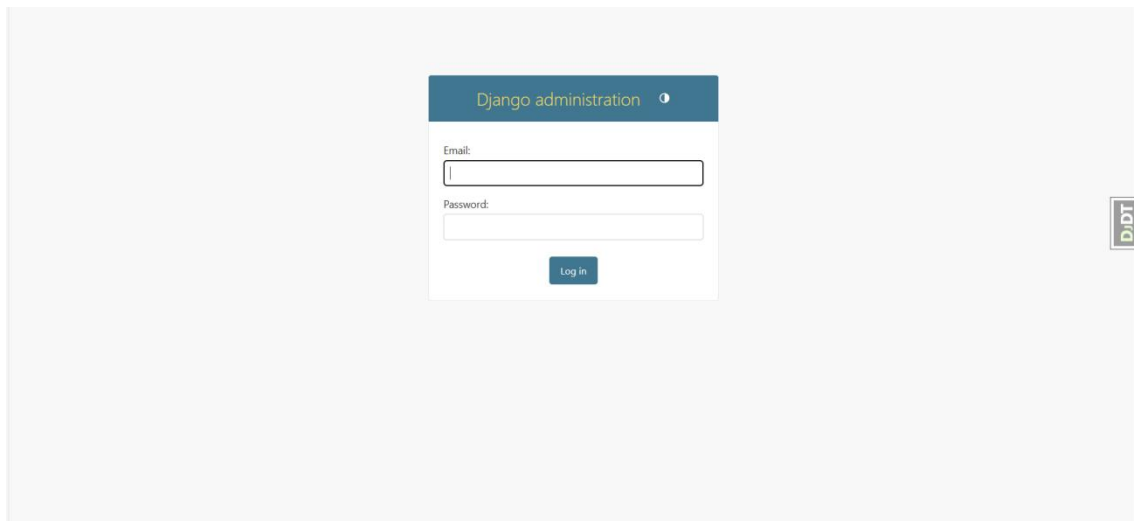


Photo 7.9 Admin Login

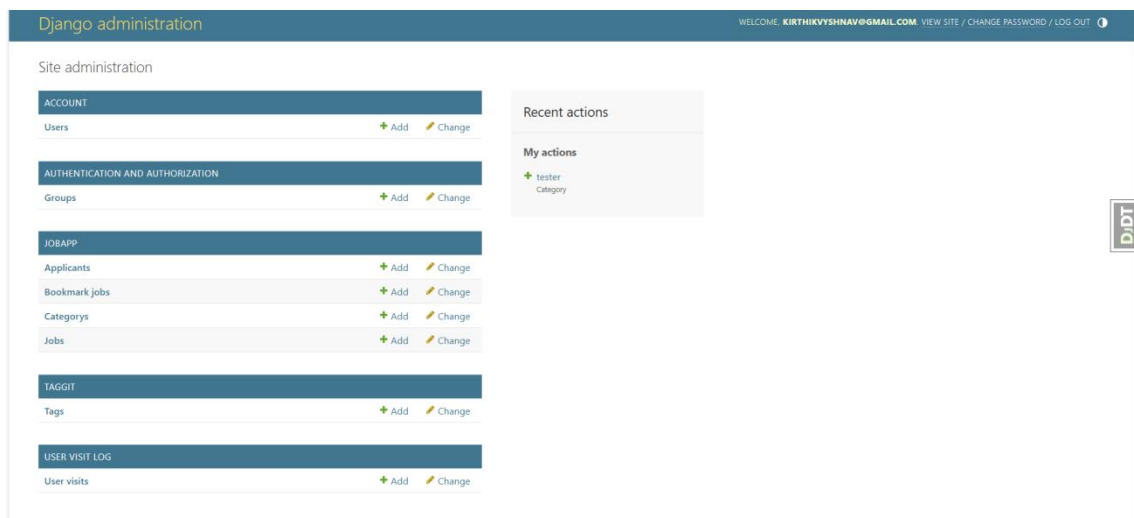


Photo 7.10 Admin Dashboard

8. CONCLUSION

The Real-Time Employment Gateway project, developed using Django, concludes as a high-impact, dynamic platform designed to streamline the job application and recruitment processes for both employers and job seekers. The project leverages Django's robust and scalable framework, which enables efficient handling of complex functionalities, including user authentication, session management, and permissions. By segmenting users into job seekers and employers, the platform offers tailored experiences with role-based access that guarantees security and user privacy. The employment gateway features a dynamic job posting module, allowing employers to create, update, and manage job listings in real-time. This functionality ensures that job listings are instantly visible to job seekers, making the application process immediate and efficient. Employers can also track applicant details, utilize intuitive dashboards for tracking recruitment, and benefit from tools to update or delete postings effortlessly. This comprehensive dashboard system supports seamless user navigation and management, all optimized through Django's ORM for fast data retrieval and minimal downtime, which enhances the platform's responsiveness and reliability. Job seekers benefit from an intuitive interface that includes real-time job search filters, allowing users to find positions that match their skills and career goals. Additionally, the bookmarking feature enables users to save interesting job postings, while the real-time application submission system ensures that applications are delivered immediately. This platform also supports profile management where job seekers can easily update their personal information, work experience, and skills, thereby improving their visibility to potential employers. The platform's underlying technical architecture is optimized for scalability, with Django's ORM facilitating smooth data transactions and handling large volumes of job and applicant data. This adaptability enables future scalability, positioning the project well for additional features, such as analytical dashboards to provide insights into hiring trends and applicant demographics. Through these potential enhancements, the platform could analyze user behavior and employment data, making it a versatile tool for both corporate recruitment and workforce development. This project has the potential to significantly improve recruitment efficiency and user engagement, making it a valuable resource in today's competitive job market.

9. FURTHER ENHANCEMENTS

This platform will be enhanced later by including more facilities, which will enable the For further enhancements to the Real-Time Employment Gateway project, several key improvements can be implemented to elevate its functionality, user engagement, and overall effectiveness in the job market. One significant enhancement would be the integration of machine learning algorithms to provide personalized job recommendations. By analyzing user profiles, application history, and job market trends, the system could suggest roles that align with a user's skills and preferences. This tailored approach not only increases the likelihood of job matches but also enhances user satisfaction and engagement. Additionally, incorporating real-time analytics would empower both employers and job seekers to gain insights into hiring trends and market demands. An analytical dashboard could showcase data visualizations of job postings, application rates, and candidate demographics, allowing employers to refine their recruitment strategies based on actionable insights. Job seekers could benefit from understanding the competitiveness of their applications in relation to others in their field. Expanding the platform's mobile accessibility would also greatly enhance user experience. A mobile application or a responsive web design could facilitate job searching and applications on-the-go, catering to the increasing number of users who prefer mobile devices for job hunting. Push notifications could alert users about new job postings that match their criteria or updates on their application status, fostering real-time engagement. Implementing a social media integration feature could further enhance the platform's reach. Allowing users to share job postings on their social networks would increase visibility and potentially attract a larger audience of job seekers. Employers could also benefit from promoting their company culture through integrated social media content, showcasing their work environment and values, which could help attract top talent. By integrating these enhancements, the Real-Time Employment Gateway could significantly elevate its role in the recruitment process, adapting to evolving job market needs and user expectations while establishing itself as a leading employment solution. These developments not only aim to improve functionality but also enhance user satisfaction, engagement, and trust, ensuring the platform remains relevant in a competitive landscape.

10.REFERENCES

Django for Beginners: Build websites with Python and Django by William S. Vincent - A great introduction to Django that guides you through building projects step-by-step.

Django for APIs: Build web APIs with Python & Django by William S. Vincent - Focuses on building RESTful APIs with Django, which is useful for real-time features.

JavaScript: The Good Parts by Douglas Crockford - A classic book that dives into the best features of JavaScript, essential for front-end development.

HTML and CSS: Design and Build Websites by Jon Duckett - Provides a visually engaging introduction to web design with HTML and CSS.

Eloquent JavaScript: A Modern Introduction to Programming by Marijn Haverbeke - Offers in-depth insights into JavaScript, enhancing your front-end skills.

HTML - <https://www.w3schools.com/html/>

jQuery - <https://jquery.com/>

CSS - <https://css-tricks.com/>

Django - <https://www.djangoproject.com/start/>

MySQL - <https://www.mysqltutorial.org/>

11.APPENDICES

11.1 USER DOCUMENTATION

11.1.1 EXECUTION INSTRUCTIONS

1. Open the Windows Power Shell
2. Direct the Project Folder using **cd** command
3. Create the virtual environment using **venu\Scripts\activate** command
4. Start the server using **python manage.py runserver** command
5. The project was hosted in the **localhost:8000** on the web browser

11.2 README

11.2.1 EMPLOYER MODULE GUIDE

- Employers using this job portal can streamline recruitment with a user-friendly interface tailored to manage job listings and candidate applications.
- After registering with company details, employers gain access to a secure dashboard where they can create detailed job postings, specifying roles, responsibilities, qualifications, salary, and location.
- The dashboard also allows employers to track applicants for each position, with options to review resumes, bookmark potential candidates, and update job statuses to either open or filled.
- Employers can update or delete postings to maintain relevant listings and enhance engagement.
- Additionally, the platform supports a dedicated company profile page, which allows businesses to showcase their culture, mission, and branding, providing candidates with a comprehensive view of the organization.
- This setup not only simplifies recruitment but also enhances the company's presence, attracting candidates aligned with its values.

11.2.2 EMPLOYEE MODULE GUIDE

- Job seekers on this platform benefit from a comprehensive tool for discovering and applying for jobs suited to their qualifications and career goals.
- Once registered, candidates can complete a profile featuring their skills, experience, and preferred job criteria, which can be edited at any time.
- The portal offers a powerful job search feature with filters for job type, location, and industry, enabling users to identify opportunities that align with their profile.
- Users can apply directly through the platform, which streamlines the application process and updates their application status on the dashboard.
- A bookmarking feature allows job seekers to save preferred listings for future review and easy access.
- The platform's interface encourages active job hunting while maintaining a personal profile that enhances visibility to employers.
- With its intuitive design, the platform not only makes job applications more manageable but also helps candidates track their progress, offering a well-rounded and efficient job search experience.

11.2.3 ADMIN MODULE GUIDE

- The Admin section of this job portal offers comprehensive control over the platform's content and user management.
- Upon logging in, admins can manage employer and employee accounts, monitor their activities, and ensure all profiles adhere to platform guidelines.
- The dashboard allows admins to oversee job postings, with options to approve, edit, or remove listings to maintain quality and relevance.
- Additionally, admins can view application metrics and usage data to monitor platform engagement.
- Other features include access to analytics, enabling admins to track hiring trends, user activity, and system performance.

11.3 SAMPLE SOURCE PROGRAM

11.3.1 ADMIN.PY

```
from django import forms

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin as BaseUserAdmin

from django.contrib.auth.forms import ReadOnlyPasswordHashField


from .models import User


class AddUserForm(forms.ModelForm):
    """
    New User Form. Requires password confirmation.
    """
    password1 = forms.CharField(
        label='Password', widget=forms.PasswordInput
    )
    password2 = forms.CharField(
        label='Confirm password', widget=forms.PasswordInput
    )

    class Meta:
        model = User

        fields = ('email', 'first_name', 'last_name', 'gender', 'role', )

    def clean_password2(self):
```



```

# Check that the two password entries match

password1 = self.cleaned_data.get("password1")

password2 = self.cleaned_data.get("password2")

if password1 and password2 and password1 != password2:

    raise forms.ValidationError("Passwords do not match")

    return password2

```

```

def save(self, commit=True):

    # Save the provided password in hashed format

    user = super().save(commit=False)

    user.set_password(self.cleaned_data["password1"])

    if commit:

        user.save()

    return user

```

```

class UpdateUserForm(forms.ModelForm):

    """

    Update User Form. Doesn't allow changing password in the Admin.

    """

    password = ReadOnlyPasswordHashField()

```

```

class Meta:

    model = User

    fields = (

```

```
'email', 'password', 'first_name', 'gender', 'role', 'last_name', 'is_active',  
'is_staff'  
)
```

```
def clean_password(self):  
  
# Password can't be changed in the admin  
  
return self.initial["password"]
```

```
class UserAdmin(BaseUserAdmin):  
  
form = UpdateUserForm  
  
add_form = AddUserForm
```

```
list_display = ('email', 'first_name', 'last_name', 'gender', 'role', 'is_staff')  
  
list_filter = ('is_staff', )  
  
fieldsets = (  
  
(None, {'fields': ('email', 'password')}),  
  
('Personal info', {'fields': ('first_name', 'last_name', 'gender', 'role', )}),  
  
('Permissions', {'fields': ('is_active', 'is_staff')}),  
  
)  
  
add_fieldsets = (  
  
(  
  
None,  
  
{  
  
'classes': ('wide',),
```

```

'fields': (
    'email', 'first_name', 'last_name', 'gender', 'role', 'password1',
    'password2'
)
},
)

search_fields = ('email', 'first_name', 'last_name')

ordering = ('email', 'first_name', 'last_name')

filter_horizontal = ()

admin.site.register(User, UserAdmin)

```

11.3.2 FORMS.PY

```

from django import forms

from django.contrib.auth import authenticate

from django.contrib.auth.forms import UserCreationForm

from account.models import User

class EmployeeRegistrationForm(UserCreationForm):

    def __init__(self, *args, **kwargs):

        UserCreationForm.__init__(self, *args, **kwargs)

        self.fields['gender'].required = True

```

```
self.fields['first_name'].label = "First Name :"  
self.fields['last_name'].label = "Last Name :"  
self.fields['password1'].label = "Password :"  
self.fields['password2'].label = "Confirm Password :"  
self.fields['email'].label = "Email :"  
self.fields['gender'].label = "Gender :"
```

```
self.fields['first_name'].widget.attrs.update(  
    {  
        'placeholder': 'Enter First Name',  
    }  
)
```

```
self.fields['last_name'].widget.attrs.update(  
    {  
        'placeholder': 'Enter Last Name',  
    }  
)
```

```
self.fields['email'].widget.attrs.update(  
    {  
        'placeholder': 'Enter Email',  
    }  
)
```

```
self.fields['password1'].widget.attrs.update(  
    {
```

```

        'placeholder': 'Enter Password',
    }
)

self.fields['password2'].widget.attrs.update(
    {
        'placeholder': 'Confirm Password',
    }
)

class Meta:

    model=User

    fields = ['first_name', 'last_name', 'email', 'password1', 'password2', 'gender']

    def clean_gender(self):

        gender = self.cleaned_data.get('gender')

        if not gender:

            raise forms.ValidationError("Gender is required")

        return gender

    def save(self, commit=True):

        user = UserCreationForm.save(self,commit=False)

        user.role = "employee"

```

```
if commit:

    user.save()

return user
```

```
class EmployerRegistrationForm(UserCreationForm):

    def __init__(self, *args, **kwargs):

        UserCreationForm.__init__(self, *args, **kwargs)

        self.fields['first_name'].required = True

        self.fields['last_name'].required = True

        self.fields['first_name'].label = "Company Name"

        self.fields['last_name'].label = "Company Address"

        self.fields['password1'].label = "Password"

        self.fields['password2'].label = "Confirm Password"


        self.fields['first_name'].widget.attrs.update(

            {

                'placeholder': 'Enter Company Name',

            }

        )

        self.fields['last_name'].widget.attrs.update(

            {

                'placeholder': 'Enter Company Address',

            }

        )
```

```

self.fields['email'].widget.attrs.update(
    {
        'placeholder': 'Enter Email',
    }
)

self.fields['password1'].widget.attrs.update(
    {
        'placeholder': 'Enter Password',
    }
)

self.fields['password2'].widget.attrs.update(
    {
        'placeholder': 'Confirm Password',
    }
)

class Meta:

    model=User

    fields = ['first_name', 'last_name', 'email', 'password1', 'password2',]

    def save(self, commit=True):
        user = UserCreationForm.save(self,commit=False)

        user.role = "employer"

```

```

    if commit:

        user.save()

    return user


class UserLoginForm(forms.Form):

    email = forms.EmailField(

        widget=forms.EmailInput(attrs={ 'placeholder':'Email',})

    )

    password = forms.CharField(strip=False,widget=forms.PasswordInput(attrs={

        'placeholder':'Password',

    })))


    def clean(self, *args, **kwargs):

        email = self.cleaned_data.get("email")

        password = self.cleaned_data.get("password")


        if email and password:

            self.user = authenticate(email=email, password=password)

            try:

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

            except User.DoesNotExist:

                raise forms.ValidationError("User Does Not Exist.")

```



```

        if not user.check_password(password):

            raise forms.ValidationError("Password Does not Match.")


        if not user.is_active:

            raise forms.ValidationError("User is not Active.")


    return super(UserLoginForm, self).clean(*args, **kwargs)


    def get_user(self):

        return self.user


class EmployeeProfileEditForm(forms.ModelForm):


    def __init__(self, *args, **kwargs):

        super(EmployeeProfileEditForm, self).__init__(*args, **kwargs)

        self.fields['first_name'].widget.attrs.update(

            {

                'placeholder': 'Enter First Name',

            }

        )

        self.fields['last_name'].widget.attrs.update(

            {

                'placeholder': 'Enter Last Name',

            }

```

```
)
```

```
class Meta:
```

```
    model = User
```

```
    fields = ["first_name", "last_name", "gender"]
```

11.3.3 MANAGERS.PY

```
from django.contrib.auth.base_user import BaseUserManager
```

```
from django.utils.translation import gettext_lazy as _
```

```
class CustomUserManager(BaseUserManager):
```

```
    use_in_migrations = True
```

```
    """
```

```
    Custom user model manager where email is the unique identifiers
    for authentication instead of usernames.
```

```
    """
```

```
    def create_user(self, email, password, **extra_fields):
```

```
        """
```

```
        Create and save a User with the given email and password.
```

```
        """
```

```
        if not email:
```

```
            raise ValueError('The Email must be set')
```

```
        if not password:
```

```

        raise ValueError('The Password must be set')

    email = self.normalize_email(email)

    user = self.model(email=email, **extra_fields)

    user.set_password(password)

    user.save()

    return user

def create_superuser(self, email, password, **extra_fields):
    """
    Create and save a SuperUser with the given email and password.
    """
    extra_fields.setdefault('is_staff', True)

    extra_fields.setdefault('is_superuser', True)

    extra_fields.setdefault('is_active', True)

    if extra_fields.get('is_staff') is not True:
        raise ValueError('Superuser must have is_staff=True.')

    if extra_fields.get('is_superuser') is not True:
        raise ValueError('Superuser must have is_superuser=True.')

    return self.create_user(email, password, **extra_fields)

```

11.3.4 MODELS.PY

```

from django.contrib.auth.models import AbstractUser

from django.db import models

```

```
from account.managers import CustomUserManager
```

```
JOB_TYPE = (  
    ('M', "Male"),  
    ('F', "Female"),  
  
)
```

```
ROLE = (  
    ('employer', "Employer"),  
    ('employee', "Employee"),  
  
)
```

```
class User(AbstractUser):  
    username = None  
    email = models.EmailField(unique=True, blank=False,  
                               error_messages={  
                                   'unique': "A user with that email already exists.",  
                               })  
    role = models.CharField(choices=ROLE, max_length=10)  
    gender = models.CharField(choices=JOB_TYPE, max_length=1)  
  
    USERNAME_FIELD = "email"  
    REQUIRED_FIELDS = []
```

```

def __str__(self):
    return self.email

def get_full_name(self):
    return self.first_name+ ' ' + self.last_name

objects = CustomUserManager()

```

11.3.5 VIEWS.PY

```

from django.contrib import auth

from django.contrib import messages

from django.contrib.auth.decorators import login_required

from django.http import HttpResponseRedirect

from django.shortcuts import render, redirect , get_object_or_404

from django.urls import reverse, reverse_lazy


from account.forms import *

from jobapp.permission import user_is_employee


def get_success_url(request):

    """

    Handle Success Url After LogIN

```

```
"""
```

```
if 'next' in request.GET and request.GET['next'] != ":
```

```
    return request.GET['next']
```

```
else:
```

```
    return reverse('jobapp:home')
```

```
def employee_registration(request):
```

```
"""
```

```
Handle Employee Registration
```

```
"""
```

```
form = EmployeeRegistrationForm(request.POST or None)
```

```
if form.is_valid():
```

```
    form = form.save()
```

```
    return redirect('account:login')
```

```
    context={
```

```
        'form':form
```

```
    }
```

```
    return render(request,'account/employee-registration.html',context)
```

```
def employer_registration(request):
```

```
"""
```

Handle Employee Registration

```
"""
```

```
form = EmployerRegistrationForm(request.POST or None)
```

```
if form.is_valid():
```

```
form = form.save()
```

```
return redirect('account:login')
```

```
context={
```

```
'form':form
```

```
}
```

```
return render(request,'account/employer-registration.html',context)
```

```
@login_required(login_url=reverse_lazy('accounts:login'))
```

```
@user_is_employee
```

```
def employee_edit_profile(request, id=id):
```

```
"""
```

Handle Employee Profile Update Functionality

```
"""
```

```
user = get_object_or_404(User, id=id)

form = EmployeeProfileEditForm(request.POST or None, instance=user)

if form.is_valid():
    form = form.save()

    messages.success(request, 'Your Profile Was Successfully Updated!')

    return redirect(reverse("account:edit-profile", kwargs={
        'id': form.id
    })))

context={

    'form':form
}

return render(request,'account/employee-edit-profile.html',context)
```

```
def user_logIn(request):
```

```
"""
```

```
Provides users to logIn
```

```
"""
```



```
form = UserLoginForm(request.POST or None)
```

```
if request.user.is_authenticated:
```

```
    return redirect('/')
```

```
else:
```

```
    if request.method == 'POST':
```

```
        if form.is_valid():
```

```
            auth.login(request, form.get_user())
```

```
            return HttpResponseRedirect(get_success_url(request))
```

```
            context = {
```

```
                'form': form,
```

```
            }
```

```
            return render(request, 'account/login.html', context)
```

```
def user_logOut(request):
```

```
    """
```

```
    Provide the ability to logout
```

```
    """
```

```
    auth.logout(request)
```

```
    messages.success(request, 'You are Successfully logged out')
```

```
    return redirect('account:login')
```

11.3.6 SETTINGS.PY

```
import os

import django_heroku

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'etdq)uvq=t0rc&ams5_ovn6w8bcwknjj0u97*(#n^(76x*+dr1'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['127.0.0.1:8000, djobportal.herokuapp.com']

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
```

```
'django.contrib.contenttypes',
```

```
'django.contrib.sessions',
```

```
'django.contrib.messages',
```

```
'django.contrib.staticfiles',
```

```
'jobapp.apps.JobappConfig',
```

```
'account.apps.AccountConfig',
```

```
#3rd Party App
```

```
'ckeditor',
```

```
'taggit',
```

```
'user_visit',
```

```
'debug_toolbar',
```

```
]
```

```
MIDDLEWARE = [
```

```
'django.middleware.security.SecurityMiddleware',
```

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

```
'django.middleware.common.CommonMiddleware',
```

```
'django.middleware.csrf.CsrfViewMiddleware',
```

```
'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
'django.contrib.messages.middleware.MessageMiddleware',
```

```
'django.middleware.clickjacking.XFrameOptionsMiddleware',
```

```
'whitenoise.middleware.WhiteNoiseMiddleware',
```

```
'user_visit.middleware.UserVisitMiddleware',
```

```
'debug_toolbar.middleware.DebugToolbarMiddleware',  
]  
  
ROOT_URLCONF = 'job.urls'  
  
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'template')],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    ],  
]  
  
WSGI_APPLICATION = 'job.wsgi.application'
```

Database

<https://docs.djangoproject.com/en/3.0/ref/settings/#databases>

```

# DATABASES = {

#     'default': {

#         'ENGINE': 'django.db.backends.postgresql',

#         'NAME': 'jobportal',

#         'USER': 'postgres',

#         'PASSWORD': '1234',

#         'HOST': 'localhost',

#         'PORT': '',

#     }

# }

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),

    }

}

#for debug toolbar

INTERNAL_IPS = [

    # ...

    "127.0.0.1",

    # ...

]

# CACHES

```

```

# -----

# CACHES = {
#     "default": {
#         'BACKEND': 'django.core.cache.backends.redis.RedisCache',
#         'LOCATION': 'redis://127.0.0.1:6379',
#         "OPTIONS": {
#             "CLIENT_CLASS": "django_redis.client.DefaultClient"
#         },
#         "KEY_PREFIX": "config"
#     }
# }

# Password validation

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },

```

```
{
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
},
]

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
```

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

```
MEDIA_URL = "/media/"
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

```
AUTH_USER_MODEL = 'account.User'
```

```
# CKeditor Config
```

```
CKEDITOR_CONFIGS = {
```

```
    'default': {
```

```
        'width': '100%',
```

```
        'tabSpaces': 4,
```

```
    }
```

```
}
```

```
from django.contrib.messages import constants as messages
```

```
MESSAGE_TAGS = {
```

```
    messages.DEBUG: 'alert-info',
```

```
    messages.INFO: 'alert-info',
```

```
    messages.SUCCESS: 'alert-success',
```

```
    messages.WARNING: 'alert-warning',
```

```
    messages.ERROR: 'alert-danger',
```



```
}
```

```
# Activate Django-Heroku.
```

```
django_heroku.settings(locals())
```

```
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
```