Final Year Design Project

GradHire

By

Muhib Arshad	BSEF21M540
Shafqat Abbas	BSEF21M520
Muhammad Bilal	BSEF21M528
Faiz Muhammad	BSEF21M554

Under the supervision of

Mr. Farhan Ahmad Ch.

Bachelor of Science in Software Engineering (2021-2025)

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE.

GradHire

A project presented to

University of the Punjab, Lahore

In partial fulfilment

of the requirement for the degree of

Bachelors of Science in Software Engineering (2021-2025)

By

Muhib Arshad	**BSEF21M540**
Shafqat Abbas	**BSEF21M520**
Muhammad Bilal	**BSEF21M528**
Faiz Muhammad	**BSEF21M554**

FACULTY OF COMPUTING &

INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE

DECLARATION

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied the report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other, we will stand by the consequences. No portion of the work presented has been submitted for any application for any other degree or qualification of this or any other university or institute of learning.

Signature: Muhib Arshad ---------------------------

Muhib Arshad [BSEF21M540]

Signature: Muhammad Bilal ------------------------

Muhammad Bilal [BSEF21M528]

Signature: Shafqat Abbas ---------------------------

Shafqat Abbas [BSEF21M520]

Signature: Faiz Muhammad ----------------------

Faiz Muhammad [BSEF21M554]

CERTIFICATE OF APPROVAL

It is to certify that the final year design project (FYDP) of BSSE “GradHire” was developed by **MuhibArshad[BSEF21M540],ShafqatAbbas[BSEF21M520],MuhammadBilal[BSEF21M528] and Faiz Muhammad[BSEF21M554]** under the supervision of “ **Mr. Farhan Ahmad Ch.**” in my opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Software Engineering.

Signature: -

FYDP Supervisor:

Signatures (Faculty Advisory Committee (FAC)

Signatures			
Name	Mr. Farhan Ahmed Ch.	Dr. Shahid Farid	Ms. Tayyaba Tariq
	FAC1	FAC2	FAC3

Signature: -

Head of FYDP Coordination Office:

Signature:--------------------------------------- Dated: ________________

Chairperson, Department of Software Engineering

Executive Summary

The GradHire project proposal outlines the development of a job portal aimed at addressing inefficiencies in the recruitment process. It highlights the platform's objectives, including personalized job recommendations, automated resume filtering, and seamless communication between job seekers and recruiters. Designed using the MERN stack, GradHire emphasizes scalability, user-friendliness, and modern web technologies to streamline hiring and job-seeking experiences. The proposal identifies key stakeholders, problem areas in traditional recruitment, and the project's scope, emphasizing its role as a comprehensive and efficient solution.

The SRS document specifies the system's functional and non-functional requirements, detailing features like job posting, profile management, applicant tracking, and dual-mode operation for recruiters and job seekers. Non-functional requirements focus on usability, scalability, security, and performance. The SDS expands on this by presenting system architecture, data design, and component interactions, supported by UML diagrams and a requirements traceability matrix. The prototype demonstrates the platform's core functionalities, such as intuitive dashboards, job search filters, and profile management tools, providing a tangible representation of the envisioned system's capabilities. Together, these documents ensure a structured and clear approach to designing and implementing the GradHire platform.

Keywords: Resume Filtering,Job Recommendations,Communication

Acknowledgement

I would like to express my sincere gratitude to my FYDP supervisor, [**Mr. Farhan Ahmad Ch.**], for their valuable guidance and support throughout this project. I also thank the faculty and staff of [Software Engineering/FCIT] for their assistance and resources. Appreciation is extended to my team members for their collaboration, and to my family and friends for their continued encouragement.

Signature: Muhib Arshad ---------------------------	*Signature:* Muhammad Bilal ------------------------
Muhib Arshad [BSEF21M540]	Muhammad Bilal [BSEF21M528]
Signature: Shafqat Abbas ---------------------------	*Signature:* Faiz Muhammad ----------------------
Shafqat Abbas [BSEF21M520]	Faiz Muhammad [BSEF21M554]

Abbreviations

Table of all the abbreviations and acronyms used in your FYP along with their respective brief description. The abbreviation list must be ordered alphabetically.

BL	Backlog list : List of the requirements under consideration
FCIT	Faculty of Computing and Information Technology
UT	Unit Testing
IT	Integration Testing
FT	Functional Testing
API	Application Programming Interface
MERN	MongoDB, Express.js, React.js, Node.js

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Introduction

The chapter provides an overview of the GradHire project, emphasizing its goal to streamline recruitment through features like personalized job recommendations, automated resume filtering, and efficient employer-candidate communication. It highlights challenges such as inefficient job matching and high recruitment costs while proposing a scalable, user-friendly solution built using the MERN stack. The chapter outlines the project's scope and objectives, focusing on modernizing and improving the hiring process for both job seekers and employers.

1.1 Problem Statement

In today’s competitive job market, the recruitment process is plagued by inefficiencies that negatively affect both employers and job seekers. Companies often face challenges in filtering through large volumes of irrelevant applications, resulting in extended hiring timelines and increased costs. On average, it takes 42 days and 100k to fill a single position, highlighting the inefficiencies in traditional hiring methods. Existing recruitment platforms typically lack intelligent filtering mechanisms, leading to mismatches between candidate profiles and job requirements. This not only increases recruiter workload but also causes frustration among applicants who do not receive relevant opportunities. Approximately 75% of resumes are rejected without proper evaluation due to poor keyword matching or lack of relevance. Moreover, job seekers report a 56% dissatisfaction rate with current job portals, citing non-personalized job recommendations and poor communication with recruiters. There is a clear gap in the market for a system that bridges these disconnects efficiently and intelligently.

GradHire is being developed to address these problems by offering a software solution that automates and streamlines the recruitment process. The system aims to minimize manual effort, improve candidate-job fit, and enhance the overall hiring experience. It will reduce application review time, lower recruitment costs, and improve satisfaction on both ends of the hiring process. By leveraging intelligent algorithms, GradHire will match job seekers with roles aligned to their skills, preferences, and career goals, while enabling employers to filter suitable candidates more effectively.

1.2 Problem Solution

GradHire is a smart recruitment platform designed to modernize and automate the job-matching process. The primary objective of the software is to reduce inefficiencies in recruitment by using intelligent filtering, machine learning, and personalized recommendation techniques. The platform will serve both employers and job seekers by offering tools that simplify job posting, resume screening, and application management. It aims to significantly reduce time-to-hire and cost-per-hire by automating resume evaluations and filtering out irrelevant applications.

The system will analyze candidate profiles in depth and suggest job openings based on skills, experience, and career preferences. For recruiters, it will provide dashboards to manage applicants, schedule interviews, and communicate directly with shortlisted candidates. GradHire will include features such as AI-powered job matching, resume parsing, applicant tracking, and real-time notifications. By integrating communication tools within the platform, it enhances transparency and engagement between recruiters and applicants.

The software will support both web and mobile platforms to ensure accessibility and usability. It will also incorporate analytics to help employers evaluate the effectiveness of their recruitment campaigns. The long-term goal of GradHire is to transform traditional hiring into a seamless, data-driven experience that benefits all stakeholders.

1.3 Objectives of the Proposed System

Quantifiable objectives to be achieved using this system. Please follow the do’s and don’ts of writing objectives

1. **Enable Dual-Role User Functionality**
   Allow users to switch seamlessly between job seeker and recruiter roles within a single account, minimizing login friction and supporting simplified user management.

2. **Support Responsive Design across Devices:**
   Ensure the application is fully responsive and offers a consistent user experience on desktops, tablets, and smartphones (min screen width: 320px).

3. **Accelerate Job Search and Filtering**
   Implement an efficient job search system with multiple filters (e.g., location, skills, type, salary) that returns results within **5–7 seconds**, ensuring user satisfaction and reduced bounce rate.

4. **Automate Resume Filtering and Matching**
   Enable recruiters to filter resumes based on experience, skills, and qualifications using predefined criteria, discarding irrelevant applications with at least **90% precision** to reduce hiring time.

5. **Ensure Real-Time Application Tracking and Alerts**
   Notify job seekers of application status changes (viewed, shortlisted, rejected) via in-app notifications within **3 seconds**.

6. **Guarantee System Availability and Uptime**
   Maintain minimum **99.5% system uptime** through cloud-based deployment, ensuring users can reliably access services, even under peak usage.

7. **Simplify User Onboarding**
   Allow new users to register and complete profile setup—including resume upload, role selection, and preferences—in under **10 minutes**, with validation and clear error messages.
8. **Promote User Data Security and Privacy Compliance**
   Enforce encryption of all sensitive data at rest and in transit, comply with GDPR-like principles, and establish access controls to prevent unauthorized data manipulation or breaches.

9. **Apply the Pareto Principle for Feature Prioritization**
   Focus on building the top **20% of features** (job posting, profile management, filtering, applying, chat) that deliver **80% of system value**, ensuring time-bound delivery within available resources.

10. **Design Tests for Independency and Repeatability**
    All test cases are designed to be independent, reusable, and data-agnostic, ensuring robustness of test coverage and easier debugging.

11. **Mitigate Risks and Account for Dependencies**
 Account for risks such as 3rd-party service failure (email, cloud), connectivity issues, and screen incompatibilities. Dependencies between modules (e.g., job test ↔ job post) will be managed through interface contracts and clear sequencing in development.

Constraints Considered

- Limited to a 4-month academic timeline with 4 core developers.
- Restricted budget and reliance on open-source libraries.
- Technology stack fixed to MERN (MongoDB, Express.js, React.js, Node.js).

1.4 Scope

The scope of the **GradHire Job Portal** project encompasses the design, development, testing, and deployment of a comprehensive web-based recruitment platform that connects job seekers with employers. The primary goal is to streamline and enhance the recruitment process by providing advanced tools and a user-centric experience for both stakeholder groups.

Clear and Specific Description

This project will focus on building a modern, scalable job portal that supports job searching, application processing, and recruitment management. The platform will include core functionalities such as:

- **Automated Resume Filtering** to help employers efficiently shortlist candidates.
- **Personalized Job Recommendations** to assist users in finding relevant opportunities based on their profiles.
- **Comprehensive Company Profiles** to help job seekers evaluate potential employers.
- **Secure Authentication and User Management** for both employers and applicants.
- **Admin Dashboard** for monitoring and managing platform activity.

Defined Deliverables

- Fully functional web application accessible to job seekers and employers.
- Resume filtering engine with keyword and skills-based ranking.
- Job recommendation engine using predefined algorithms or AI/ML (if feasible within constraints).
- Employer and job seeker dashboards.
- Company profile management module.
- Scalable backend and responsive frontend.
- System documentation and user manuals.

System Boundaries

- **In-Scope:**
  - Job posting, search, and application modules
  - Resume parsing and filtering
  - User authentication and profiles
  - Employer dashboards and candidate tracking
  - Admin control panel for moderation
- **Out-of-Scope:**

    - Payroll or in-depth HR management tools
    - Third-party job aggregator integration
    - Offline recruitment processes
    - On-premise deployment or mobile-native apps (initial phase)

Delimitations to Avoid Scope Creep

- Only essential features for MVP (Minimum Viable Product) will be developed in the first release.
- Integration with third-party services (e.g., LinkedIn, ATS systems) is considered future work.
- Mobile apps and multilingual support are excluded from the initial scope.

Stakeholders

- **Job Seekers** – Users looking for employment opportunities.
- **Employers/Recruiters** – Users managing job listings and hiring.
- **Platform Administrators** – Responsible for system oversight and moderation.
- **Development Team** – In charge of designing, developing, and testing the system.
- **Investors/Project Sponsors** – Overseeing resource allocation and progress milestones.

Alignment with Objectives

The scope is directly aligned with the system objectives by focusing on enhancing job-matching accuracy, streamlining application review, and enabling scalable recruitment operations.

Priority Features

1. User authentication and profile creation
2. Job posting and search functionality
3. Resume filtering mechanism
4. Job recommendation engine
5. Company profile management
6. Admin tools for managing listings and users

Constraints

- **Time:** Initial development must be completed within a 4-month timeline.
- **Technology:** The solution must use modern web technologies (e.g., React, Node.js, PostgreSQL).

- **Resources:** Limited to a small development team with fixed hardware and software tools.

Milestones

1. **Requirement Analysis & Design** – Week 1–3
2. **Backend & Database Setup** – Week 4–6
3. **Frontend Development (MVP Features)** – Week 7–10
4. **Integration & Testing** – Week 11–13
5. **User Acceptance Testing (UAT)** – Week 14
6. **Deployment & Documentation** – Week 15–16

1.5 System Components

Client Web App (Job Seeker Web Portal)

1. **User Registration & Authentication**
   - Email/password registration and login
   - OTP verification for secure access
   - Forgot password & reset feature
2. **Profile Management**
   - Create/update personal and professional profile
   - Upload resume and documents
   - Set job preferences and criteria
3. **Job Search & Application**
   - Search/filter job listings
   - View job details
   - Apply for jobs directly
   - Track application status
4. **Notifications**
   - Receive updates about job matches, application status, and company messages

- Email and in-app notifications

Client Mobile App (Job Seeker Android/iOS App)

1. **User Registration & Login**
    - Mobile-friendly signup and login with verification
    - Social login (if applicable)
2. **Profile Setup**
    - Easy-to-use interface to input educational and work details
    - Upload profile picture, resume, etc.
3. **Job Discovery & Application**
    - Location-based job suggestions
    - Swipe or tap to apply functionality
    - Save favorite jobs
4. **Application Tracker**
    - View job application history
    - Real-time application updates
5. **Push Notifications**
    - Alerts for new job postings, interview calls, or profile views

Admin Web App (System Management Panel)

1. **User Management**
    - Manage job seekers and employer accounts
    - Verify new company sign-ups
    - Block or flag suspicious users
2. **Job Post Moderation**
    - Approve or reject job posts
    - Monitor posted content for quality and compliance

3. **Analytics & Reporting**
    - Dashboard with metrics (user activity, job post stats)
    - Generate reports for system performance and user engagement
4. **Content Management**
    - Update FAQs, About Us, Terms and Privacy content
    - Publish announcements or system updates
5. **Feedback & Support**
    - View and respond to user complaints and feedback
    - Issue resolution tracking

Module 1: User Registration & Authentication

List of Features:

- Account creation for job seekers and recruiters using email and password
- OTP/email verification for secure account activation
- Login/logout functionality with session/token management
- “Forgot Password” and secure password reset via email
- Encrypted password storage
- Role-based login (Job Seeker, Employer, Admin)
- CAPTCHA integration to prevent bots
- Validation and error handling for input fields

Module 2: Profile Management

List of Features:

- Create and update personal and professional details

- Upload and manage resume and documents
- Add educational background, experience, and skills
- Profile picture upload
- Set and update job preferences (e.g., location, type, industry)
- Profile visibility settings (public/private)

Module 3: Job Search & Application (Client Web/Mobile)

List of Features:

- Search jobs using filters: category, location, experience, salary range
- View detailed job descriptions
- Apply directly from the job listing page
- Save jobs to favorites/bookmarks
- Track applications submitted
- Receive recommended jobs based on profile data
- Report spam or fake job listings

Module 4: Company Profile & Job Posting (Employer Web App)

List of Features:

- Company registration and login
- Create/update company profile and branding information
- Post job vacancies with detailed descriptions
- Set criteria for applications (skills, experience, education)
- View and manage posted jobs
- Shortlist candidates based on resume filters
- Schedule interviews and send communication to candidates

Module 5: Admin Panel (Admin Web App)

List of Features:

- View and manage all user accounts (job seekers and employers)
- Verify or reject employer/company registrations
- Approve, reject, or flag job posts
- View usage statistics (e.g., number of active users, job posts)
- Generate performance and activity reports
- Content management (FAQs, terms, privacy policy)
- Handle user complaints and feedback
- Manage system-wide notifications or announcements

Module 6: Notification & Communication System

List of Features:

- In-app notifications for job updates and application statuses
- Email notifications for account activity, interviews, and recommendations
- Push notifications (mobile app) for real-time alerts
- Notification center for users to view history
- Admin broadcast messages to all or selected users

Module 7: Resume Filtering

List of Features:

- Automatically rank and filter candidate resumes based on job criteria
- Keyword and skill-based search in resumes
- Highlight matched skills/keywords for employers
- Suggest improvements to job seeker profiles
- Match job seeker profiles with relevant job openings

Module 8: Job Recommendation System

List of Features:

- Analyze user profile and activity to suggest relevant jobs
- Regular updates of job suggestions based on user interactions
- Option to personalize recommendation criteria
- Display recommended jobs on dashboard and email

Module 10: Feedback & Support System

List of Features:

- Allow users to submit queries or complaints
- Ticketing system for issue tracking
- Feedback forms for platform improvements
- Admin response management interface
- Rating and feedback after job application/interview experience

Related System Analysis/Literature Review

Briefly provide an analysis of the related system which may help you to specify the contribution of the proposed project. This includes the characteristics of the existing/similar systems related to your proposed project. Don't use more than 4 sentences for explaining a single system/application. Highlight its characteristics including the features lacking in the existing systems that may be present in your system.

Application Name	**Weakness**	**Proposed Project Solution**
LinkedIn	Job recommendations are generic and not tailored for fresh graduates. The platform is more networking-focused than job-focused.	GradHire will use intelligent resume parsing and preference-based filters to provide highly personalized job suggestions for fresh graduates.
Indeed	Lacks efficient resume filtering and does not provide smart matching tools for employers.	GradHire introduces an automated resume filtering engine and criteria-based shortlisting to help employers quickly identify ideal candidates.
Rozee.pk	Outdated user interface and absence of advanced features like AI-driven recommendations and real-time application tracking.	GradHire will offer a modern UI/UX, real-time tracking dashboard, and AI-driven recommendation systems to enhance the user experience.

Table 1 Related System Analysis with proposed project solution

1.6 Vision Statement

The following keyword template works well for crafting a product vision statement:

For fresh graduates and employers
Who need a simplified, efficient, and targeted way to connect for entry-level job opportunities
The GradHire
Is a job portal platform
That provides personalized job recommendations, resume filtering, and an intuitive application and hiring experience
Unlike conventional job portals that offer generic listings and rely heavily on manual screening
Our product delivers a focused, modern solution that streamlines recruitment and job searching specifically for early-career professionals and companies looking to hire them.

1.7 System Limitations and Constraints

Limitations *(External factors beyond control)*

- **Third-party service dependency:** The project may rely on third-party services (e.g., email verification, hosting, payment gateways), which can affect system availability or performance if those services face downtime.

- **Internet connectivity:** Since the platform is web and mobile-based, its usability depends on users having stable internet access.

- **Device compatibility issues:** User experience may vary slightly across different devices, screen sizes, and operating systems despite best efforts for responsive design.

- **Legal and regulatory compliance:** Adhering to evolving data privacy laws (e.g., GDPR) and employment regulations may impose certain restrictions.

- **Market adoption:** User adoption may take time due to competition from established platforms and resistance to switching from existing systems.

Constraints *(Self-imposed or internal controllable factors)*

- **Project scope:** The initial scope is limited to entry-level job seekers and small-to-medium employers, excluding internships, freelance, or international recruitment.

- **Development timeline:** The project must be completed within a fixed academic or organizational deadline, limiting the number of features in the initial release.

- **Resource availability:** The team has limited manpower, budget, and technical resources, affecting the complexity of the system.

- **Technology stack:** The project will use a defined set of technologies (e.g., specific frontend/backend frameworks) which limits flexibility for integrating certain tools.

- **Feature prioritization:** Only core functionalities will be implemented in the first version; some advanced features (like analytics dashboards or chat integration) may be deferred to future updates.

1.8 Tools and Technologies

	Tools	Version	Rationale
Tools, Libraries, And Technologies	1)Figma**,** 2)Postman, 3)Git, 4)VS Code, 5)Docker	Latest	1) Design tool for creating user interfaces and prototypes for the platform. 2) Tool for efficiently testing and debugging APIs. 3) Version control system for tracking changes and collaboration. 4) Lightweight IDE with robust debugging and extension support. 5) Ensures consistent environments through containerization.
	Libraries	**Version**	**Rationale**
	1) Express.js 2) Reactjs, 3) Mongoose	Latest	1) A flexible Node.js web application framework that provides a robust set of features for web and mobile applications, simplifying server-side development. 2) React.js is ideal for building interactive user interfaces quickly with reusable components, enhancing both developer productivity and application performance. 3) An ODM (Object Data Modeling) library for MongoDB and Node.js, providing a straightforward way to model application data.

Table 2 Tools and Technologies

1.9 Project Deliverables

List down the project deliverables. This include planning document, SRS, Design, Prototypes, project, project report. (these documents will be added as appendices of FYP report)

Project Planning Document

- Includes project scope, objectives, milestones, timeline, and team roles.

Software Requirements Specification (SRS)

- Detailed functional and non-functional requirements of the system, use cases, and system constraints.

System Design Document

- Architecture design, database schema, system components/modules, and data flow diagrams.

Prototypes

- Wireframes and low/high-fidelity prototypes of the web and mobile interfaces showing user interactions.

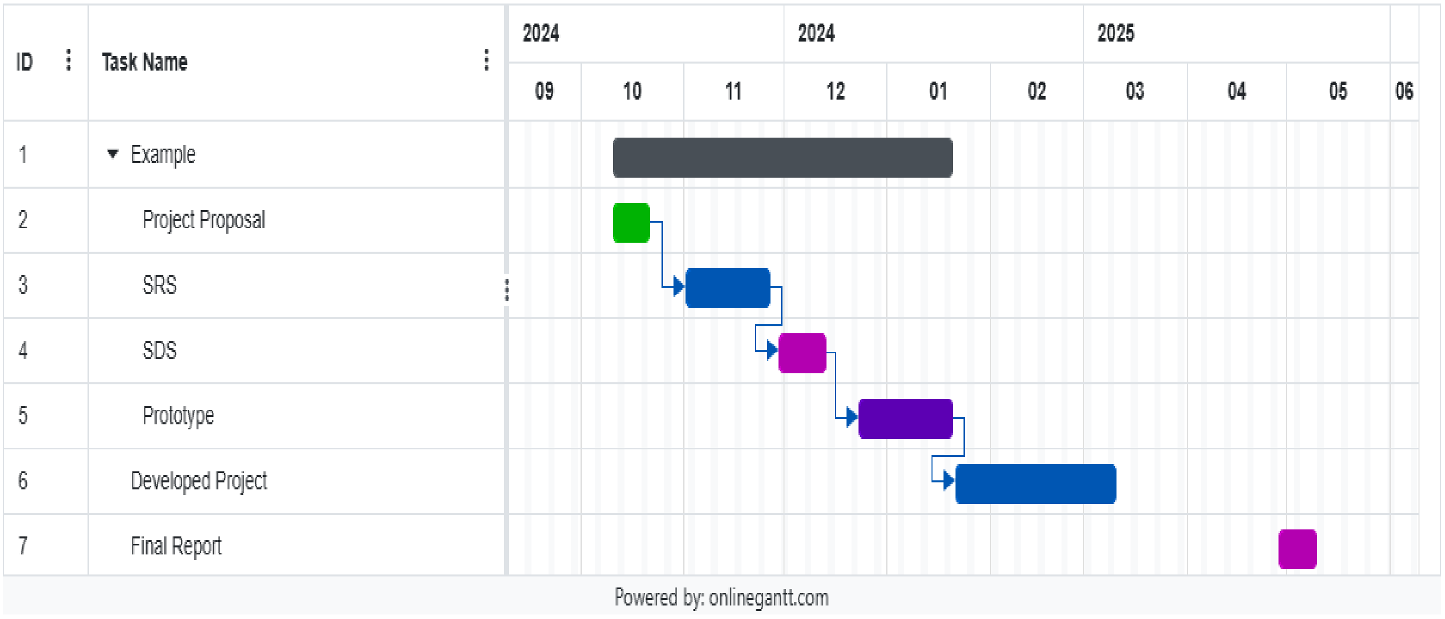
Final Project (Working System)

- Fully developed web and mobile applications with core functionalities implemented as per scope.

Project Report

- Comprehensive documentation covering all phases: introduction, literature review, methodology, implementation, testing, limitations, and conclusion.

1.10 Project Planning



Summary

The chapter introduces **GradHire**, a platform designed to improve recruitment efficiency by offering personalized job recommendations, automated resume filtering, and enhanced communication. It addresses challenges like long hiring timelines, high costs, and poor job matching, proposing a solution using AI and automation to streamline the process. Key features include job search, resume parsing, employer dashboards, and notifications, with the goal of reducing time-to-hire and costs.

Importance:

This chapter sets the foundation for **GradHire**, aligning the project's objectives with its solution. By automating processes and leveraging intelligent algorithms, GradHire aims to enhance the recruitment experience for both employers and job seekers, ensuring efficiency, cost reduction, and increased satisfaction.

Chapter 2 Requirements Analysis

2.Analysis

2.1 User classes and characteristics

User Class	User Characteristics
Job Seekers	Individuals looking for job opportunities. Varying levels of technical skills, need easy navigation, resume submission, job search functionality, and real-time application tracking.
Company	Company representatives posting job openings. Require tools for posting jobs, reviewing applications, and setting job criteria. Some may need advanced features for applicant tracking and management.

Table 3 User classes

2.2 Requirement Identifying Technique

For this project, **interviews** were used as the primary technique to identify and gather requirements. Through structured interviews with key stakeholders—such as job seekers, recruiters, and company representatives—detailed insights were gathered regarding user needs, system expectations, and the required functionalities.

- **Interviews with Job Seekers**: These helped in identifying the essential features for job searching, applying, and profile management, along with usability expectations.
- **Interviews with Recruiters/Companies**: These focused on the needs for job posting, applicant tracking, and filtering criteria, which were critical to shaping the system's recruitment functionalities.

This interview technique was instrumental in deriving clear and specific functional requirements for the system.

2.3 Functional Requirements

Identifier	FR-1
Title	**User Signup**
Requirement	A user shall be able to sign up to the system. During sign-up, the user must provide a valid email, username, and password. After completing registration, the user will be able to log in to the application.
Source	Requirement Engineering, captured from user needs.
Rationale	Provides initial access for users to use the platform.
Business Rule (if required)	Email and username must be unique, and passwords must meet security guidelines.
Dependencies	Valid email address, unique username.
Priority	High

Table 4 Functional Requirement 1

Identifier	FR-2
Title	**User Login**
Requirement	A user shall log in to the system by entering a valid email and password.
Source	Security and access control requirements.
Rationale	Allows registered users to access their accounts.
Business Rule (if required)	Only registered users can access the system. Invalid credentials will be rejected.
Dependencies	User accounts must exist.
Priority	High

Table 5 Functional Requirement 2

Identifier	FR-3
Title	**Add Profile**
Requirement	A job seeker shall add their profile, including personal, educational, and professional details, which act as a resume
Source	User requirements and application workflow.
Rationale	Enables job seekers to share their qualifications with recruiters.
Business Rule (if required)	All fields must be completed for a profile to be saved.
Dependencies	Users must be logged in.
Priority	High

Table 6 Functional Requirement 3

Identifier	FR-4
Title	**View Status**
Requirement	A job seeker shall view their application status, including pending reviews, submitted applications, and responses. A recruiter shall view the status of posted jobs and the number of applications per job.
Source	User requirements and application workflow.
Rationale	Helps users track their application progress and assists recruiters in managing job postings and applications.
Business Rule (if required)	Status is only visible to the logged-in user (job seeker or recruiter).
Dependencies	User account and application data.
Priority	Medium

Table 7 Functional Requirement 4

Identifier	FR-5
Title	**Switch Between Company and Job Seeker Mode**
Requirement	A job seeker shall be able to switch between company mode and job seeker mode as required. In company mode, they can register or manage their company, post jobs, and handle applicants. In job seeker mode, they can search for jobs, apply for them, and manage their applications.
Source	User requirements to facilitate dual roles (both job seekers and company owners).
Rationale	Allows users who act as both a job seeker and company owner to manage both roles within a single account.
Business Rule (if required)	Users must complete required fields for each mode (e.g., company details in company mode, job seeker profile in job seeker mode).
Dependencies	Valid user account, mode-switching functionality.
Priority	High

Table 8 Functional Requirement 5

Identifier	FR-6
Title	**Search Jobs**
Requirement	A job seeker shall search for jobs available on the platform.
Source	User requirements.
Rationale	Provides users with a method to find job opportunities.
Business Rule (if required)	Users must be logged in to view job details.
Dependencies	Job postings data.
Priority	High

Table 9 Functional Requirement 6

Identifier	FR-7
Title	**Apply Filters**
Requirement	A job seeker shall be able to apply various filters such as job type, salary range, and skills to refine job search results.
Source	User feedback on search functionality.
Rationale	Enhances search relevance for job seekers
Business Rule (if required)	Available filters depend on the attributes of job postings.
Dependencies	Job postings data, filter functionality.
Priority	Medium

Table 10 Functional Requirement 7

Identifier	FR-8
Title	**Take Test**
Requirement	A job seeker shall take a test if a company requires it as part of the application process.
Source	Recruiter feedback and applicant process needs.
Rationale	Assesses candidate qualifications against job requirements.
Business Rule (if required)	Tests must be passed for further application consideration.
Dependencies	Test content, scoring system.
Priority	Medium

Table 11 Functional Requirement 8

Identifier	FR-9
Title	**Compare Companies**
Requirement	A job seeker shall compare multiple companies based on various criteria like ratings and reviews.
Source	User needs for informed job selection.
Rationale	Assists job seekers in choosing the right company.
Business Rule (if required)	Only companies with profiles are available for comparison.
Dependencies	Company profiles and reviews.
Priority	Low

Table 12 Functional Requirement 9

Identifier	FR-10
Title	**Review Profile Before Applying**
Requirement	A job seeker shall review their profile before applying to any company to ensure all required information, such as personal details, professional experience, and qualifications, is up-to-date and complete.
Source	User requirements for accurate job application submissions.
Rationale	Ensures job seekers submit accurate and up-to-date information to companies, improving their chances of being shortlisted.
Business Rule (if required)	A profile must be fully completed (mandatory fields filled) before submitting applications. The system will prompt users to review and update incomplete fields.
Dependencies	Profile data, job application submission process.
Priority	Medium

Table 13 Functional Requirement 10

Identifier	FR-11
Title	**Update Profile**
Requirement	A job seeker shall update their profile details as needed, including new skills or experiences.
Source	User feedback.
Rationale	Ensures up-to-date information for recruiters.
Business Rule (if required)	Changes must be saved before exiting the profile editor.
Dependencies	User profile data.
Priority	High

Table 14 Functional Requirement 11

Identifier	FR-12
Title	**Give Company Reviews**
Requirement	A job seeker shall be able to rate and review companies they've interacted with on the platform.
Source	User feedback and company transparency.
Rationale	Allows job seekers to provide feedback, improving company accountability.
Business Rule (if required)	Only registered users can leave reviews.
Dependencies	Company profiles, review system.
Priority	Medium

Table 15 Functional Requirement 12

Identifier	FR-13
Title	**Filter and Discard Irrelevant Resumes/CVs**
Requirement	The system shall filter submitted resumes/CVs and automatically discard irrelevant ones based on criteria specified by the recruiter, such as skills, experience, or qualifications.
Source	Recruiter requirements for streamlining the hiring process.
Rationale	Helps recruiters by reducing the time spent reviewing irrelevant applications.
Business Rule (if required)	The filtering criteria must be defined by the recruiter when posting the job, and only resumes/CVs that do not meet these criteria will be discarded.
Dependencies	Applicant profiles, recruiter-specified criteria, resume filtering algorithm.
Priority	Medium

Table 16 Functional Requirement 13

Identifier	FR-14
Title	**Notify Job Seeker of Resume/CV Discard Reason**
Requirement	The system shall notify the job seeker with the reason for discarding their resume/CV if it does not meet the criteria for a particular job.
Source	User feedback and recruiter requirements for transparency.
Rationale	Ensures job seekers are informed about why their application was rejected, improving user experience and helping them refine their profile for future opportunities.
Business Rule (if required)	The notification must include the specific reason(s) the resume/CV was discarded (e.g., insufficient experience or missing required skills).
Dependencies	Applicant profiles, notification system, filtering criteria.
Priority	Low

Table 17 Functional Requirement 14

Identifier	FR-15
Title	**Register Company**
Requirement	A recruiter shall be able to register their company and provide necessary details.
Source	Business requirement for recruiter access.
Rationale	Enables companies to manage job postings and applications.
Business Rule (if required)	Company name must be unique.
Dependencies	Recruiter account, company database.
Priority	High

Table 18 Functional Requirement 15

Identifier	FR-16
Title	**Create Company Portfolio**
Requirement	The recruiter shall be able to create and manage a portfolio for their registered company, which includes company details such as name, description, industry, location, and other relevant information. This portfolio will be visible to job seekers when they view job postings or search for companies.
Source	Recruiter requirements for company representation.
Rationale	Provides a professional overview of the company, allowing job seekers to learn about the company before applying, thus improving the company's credibility and attracting qualified candidates.
Business Rule (if required)	The company portfolio must include mandatory fields like company name, description, and location, and recruiters can update the portfolio at any time.
Dependencies	Recruiter account, company registration, company portfolio database.
Priority	High

Table 19 Functional Requirement 16

Identifier	FR-17
Title	**Create Job**
Requirement	A recruiter shall create a job posting with specific criteria for applicants.
Source	Business requirement for recruitment process.
Rationale	Allows companies to advertise job opportunities.
Business Rule (if required)	Job postings must include job title, description, and qualifications.
Dependencies	Company account, job postings database.
Priority	High

Table 20 Functional Requirement 17

Identifier	FR-18
Title	**Set Criteria for Job Post**
Requirement	The recruiter shall be able to set specific criteria for a job post, including required qualifications, skills, experience, and other conditions that candidates must meet to apply for the position.
Source	Recruiter requirements for streamlining the recruitment process.
Rationale	Ensures only candidates who meet the job's specific requirements are considered, reducing the number of irrelevant applications.
Business Rule (if required)	Criteria such as minimum experience, required qualifications, or certifications must be clearly defined by the recruiter. The system will use these criteria to filter applicants.
Dependencies	Job post creation system, applicant filtering mechanism, recruiter-defined criteria.
Priority	High

Table 21 Functional Requirement 18

Identifier	FR-19
Title	**Create Test for Job Applicants**
Requirement	The recruiter shall be able to create a test for job applicants as part of the application process. The test may include questions related to the job's required skills, qualifications, or industry knowledge.
Source	Recruiter requirements to assess candidate competency.
Rationale	Helps recruiters evaluate candidates' suitability for the role by testing their knowledge and skills before further consideration.
Business Rule (if required)	The recruiter must be able to define the test's content, format (e.g., multiple choice, essay), and duration. Applicants must complete the test to proceed in the application process.
Dependencies	Test creation system, job applicant profiles, scoring system.
Priority	Low

Table 22 Functional Requirement 19

Identifier	FR-20
Title	**View Job Post Details on Dashboard**
Requirement	The recruiter shall be able to view the details of all the jobs they have posted on the dashboard. This includes the job title, description, number of applicants, status (e.g., active, closed), and any other relevant job information.
Source	Recruiter requirements for job management.
Rationale	Allows recruiters to manage and track the performance of their job posts, helping them monitor the hiring process and make necessary adjustments.
Business Rule (if required)	The system must display a list of all job posts created by the recruiter.
Dependencies	Job posting system, recruiter dashboard, job management tools.
Priority	High

Table 23 Functional Requirement 20

Identifier	FR-21
Title	**View Applicant Details for Each Job Post**
Requirement	The recruiter shall be able to view the details of all applicants who have applied for each job post. This includes the applicant’s name, contact information, resume/CV, qualifications, and any other relevant details submitted during the application process.
Source	Recruiter requirements for evaluating applicants.
Rationale	Allows recruiters to efficiently assess the qualifications of each applicant and decide on the next steps in the hiring process.
Business Rule (if required)	Only applicants who have completed their application for the job post will appear in the recruiter’s dashboard. The details shown must be accurate and up-to-date.
Dependencies	Applicant profiles, job post system, recruiter dashboard.
Priority	High

Table 24 Functional Requirement 21

Identifier	FR-22
Title	**Filter Applicants**
Requirement	A recruiter shall filter applicants based on criteria such as experience, skills, and education.
Source	Recruiter feedback.
Rationale	Improves the efficiency of candidate evaluation.
Business Rule (if required)	Filters are based on the job posting requirements.
Dependencies	Applicant profiles, filter functionality.
Priority	Medium

Table 25 Functional Requirement 22

Identifier	FR-23
Title	**Compare Applications**
Requirement	A recruiter shall compare multiple applicants for a job based on their qualifications and test scores.
Source	Recruiter requirements.
Rationale	Helps in selecting the best-suited candidate for the role.
Business Rule (if required)	Comparison is available only if there are multiple applicants.
Dependencies	Applicant profiles, comparison tools.
Priority	Medium

Table 26 Functional Requirement 23

Identifier	FR-24
Title	**Apply Tags to Applicants**
Requirement	A recruiter shall apply tags to applicant profiles for categorization.
Source	Recruiter feedback for application tracking.
Rationale	Enhances organization of applicants.
Business Rule (if required)	Tags are customizable by recruiters.
Dependencies	Applicant profiles, tagging functionality.
Priority	Low

Table 27 Functional Requirement 24

Identifier	FR-25
Title	**Bookmark Applicants**
Requirement	A recruiter shall bookmark applicant profiles to save them for future reference.
Source	Recruiter feedback.
Rationale	Allows recruiters to save potential candidates.
Business Rule (if required)	Bookmarked profiles are accessible from the recruiter’s dashboard.
Dependencies	Applicant profiles, bookmarking feature.
Priority	Low

Table 28 Functional Requirement 25

Identifier	FR-26
Title	**Create Mail Template**
Requirement	A recruiter shall create and use email templates to contact multiple applicants at once.
Source	Recruiter feedback.
Rationale	Saves time in applicant communication.
Business Rule (if required)	Templates must be customizable.
Dependencies	Email system, recruiter dashboard.
Priority	Medium

Table 29 Functional Requirement 26

Identifier	FR-27
Title	**Chat with Applicants**
Requirement	A recruiter shall chat with applicants who have applied for a job, if needed.
Source	User feedback and business requirements.
Rationale	Enhances communication between recruiter and applicant.
Business Rule (if required)	Only accessible to applicants who have applied to the recruiter’s jobs.
Dependencies	Applicant profiles, messaging system.
Priority	Medium

Table 30 Functional Requirement 27

Identifier	FR-28
Title	**Apply for Jobs**
Requirement	A job seeker shall be able to apply for available job postings on the platform.
Source	User requirements and application workflow.
Rationale	Enables users to directly apply for job opportunities they are interested in.
Business Rule (if required)	Job seekers must be logged in.
Dependencies	Job postings data, user profile data.
Priority	High

Table 31 Functional Requirement 28

2.4 Non-Functional Requirements

2.4.1 Reliability

The system must ensure a **99.9% uptime** to provide uninterrupted access for all users.

Requirements about how often the software fails. The measurement is often expressed in MTBF (mean time between failures)., and Mean time to Recover Be sure to specify the consequences of software failure, how to protect from failure, a strategy for error detection, and a strategy for correction.

Example:
Incremental Backup will be automatically saved on every Saturday at 9pm Data Integrity will ensure that 95% .

2.4.2 Usability

Ease of Learning: Users should be able to navigate and perform key tasks (e.g., posting or applying for jobs) with minimal training, within **15 minutes**.

Task Efficiency: Job seekers should apply for jobs in **3-4 clicks**, while recruiters should post a job in **5 steps**.

Error Handling: Real-time validation and clear error messages should help users avoid and easily correct mistakes.

Mobile-Friendly: The platform should be responsive, working seamlessly across desktop, tablet, and mobile devices.

2.4.3 Performance

Response Time: Key tasks (e.g., job search, posting) should complete in **5-7 seconds**. **Scalability**: Support **500+ concurrent users** without performance degradation.
Resource Utilization: Maintain resource usage under **70%** with **100 concurrent users**. **Job Application**:

Submissions should process within **5 seconds**.

Data Retrieval: Recruiters should access job details in under **5 seconds**.

2.4.4 Security

The system must protect sensitive data from unauthorized access, ensuring it requires a **high level of effort and skill** to breach.

Data breaches should take **no less than 72 hours** of sustained, skilled attack to compromise.

All user data must remain **confidential** and protected against both internal and external threats.

Any detected intrusion attempts must trigger **immediate alerts** to system administrators.

2.5 External Interface Requirements

2.5.1 User Interfaces Requirements

User Interfaces Requirements

Describe the logical characteristics of each user interface that the system needs. Some possible items to include are:

GUI Standards:
The user interface shall follow modern web and mobile design principles, adhering to established style guides for consistency in fonts, icons, button labels, and color schemes.

Visual Design:

- **Fonts**: Standard, legible fonts like **Sans-serif** with consistent sizing and spacing.
- **Icons and Buttons**: Icons should be intuitive and consistent across the platform, with buttons clearly labeled for common actions (e.g., “Apply,” “Post Job”).
- **Color Scheme**: Use a neutral color palette with accessible contrast for visually impaired users, adhering to **WCAG 2.1 Level AA** standard

Navigation:

- **Standard Buttons**: Every screen shall include standard navigation buttons such as **Home, Companies**, **Jobs**, and **Profile**.

- **Menu Layout**: Use a fixed header for primary navigation (e.g., Job Listings, My Profile, Dashboard) to maintain accessibility across screens.

- **Shortcut Keys**: Provide keyboard navigation shortcuts for common actions like form submission and navigating between fields.

Screen Layout:

- **Resolution Constraints**: The system must support a range of screen sizes, from **1024x768** on desktop to responsive mobile layouts. All elements should adjust to different screen sizes without distortion.

Message Display:

- **Conventions**: Use clear, informative error messages in case of invalid inputs (e.g., "Please enter a valid email"). Success and status messages should be non-intrusive but noticeable.

Accessibility:

Visually Impaired Users: The platform must include screen reader compatibility, high contrast mode, and alternative text for all images to accommodate users with visual impairments.

2.5.2 Software interfaces

Database Interface (MongoDB):
MongoDB (Version 4.x or above) will be used to store and manage data like user profiles and job listings, interfaced via Mongoose. Data will be stored in JSON-like format with TLS/SSL encryption and authentication controls.

Email Service Provider (SMTP):
The platform will send email notifications (e.g., verification, password resets) using SMTP. Integration will be handled via Nodemailer, ensuring secure data transmission.

2.5.3 Hardware interfaces

Supported Devices:

1. **Desktop/Laptop Computers:** Accessible via web browsers (Chrome, Firefox, etc.).
2. **Mobile Devices:** Optimized for smartphones and tablets (responsive design).

Data and Control Interactions:

3. No specialized hardware control.
4. Basic input/output through keyboard, mouse, touchscreen.

Communication Protocols:

5. No direct hardware communication; uses standard web-based protocols (HTTP/HTTPS).

2.5.4 Communications interfaces

Email Protocol: SMTP for system notifications (e.g., status updates, password resets).
Web Browsers: Supports Chrome, Firefox, Safari, and Edge using HTTPS for secure communication.
Network Protocols: TCP/IP and HTTP/HTTPS for web-based interactions.
External APIs: Uses RESTful APIs for integrating external services like job listings and authentication.

Summary

This chapter outlined the core processes and deliverables of the requirements analysis phase for the GradHire system. It began by identifying different user classes and their characteristics, followed by the use of detailed use case techniques to gather and validate functional requirements. Each functional requirement was precisely defined with identifiers, rationale, dependencies, and priority levels. Non-functional requirements—including reliability, usability, performance, and security—were documented to ensure the system's quality and compliance. External interface needs for user experience, software, hardware, and communication were also addressed. Overall, this chapter reinforced the importance of a well-structured requirements engineering process as the foundation for designing a reliable, scalable, and user-centric recruitment platform.

Chapter 3

Design and Architecture

3. System Design

3.1 Product Perspective

The software, **GradHire**, is a web-based client-server application designed to facilitate the recruitment process for job seekers and recruiters. It operates as an intermediary platform where job seekers can find relevant job opportunities and recruiters can efficiently manage candidates.

- **Dependencies and Interactions with Other Systems:**
  - **Third-party Authentication**: The system integrates with Google and Facebook for user sign-up and login, simplifying account creation.
  - **Database Management**: Uses MongoDB for storing user and company profiles, job postings, applications, and other critical data.
  - **Email Services**: Relies on email APIs for verification, password recovery, and communication between recruiters and applicants.
  - **Frontend-Backend Communication**: Utilizes RESTful APIs for data exchange between the frontend (React) and backend (Express.js).
- **Key Interactions:**
  - Users interact with the platform through a responsive web interface.
  - Recruiters use features like filtering candidates, posting jobs, and setting criteria for hiring.
  - Job seekers utilize search, filters, and comparison tools for finding jobs and evaluating companies.

3.2 Design Constraints:

- **Performance Requirements**:
  - Must handle at least 500 concurrent users with an average response time of 300ms.
  - Real-time notifications and chat must not exceed 1-second latency.
- **Security Constraints**:
  - Implements strong encryption (TLS/SSL) for all data in transit.
  - Adheres to data privacy regulations like GDPR to protect user information.
- **Scalability**:
  - Designed to scale horizontally to accommodate increased traffic during peak hiring seasons.
- **Cross-platform Compatibility**:
  - Fully responsive design to support various devices (mobile, tablet, desktop).
- **Compliance Requirements**:

Requires users to provide accurate data and prohibits fake profiles through rigorous verification methods.

3.3 Design considerations

Assumptions and Dependencies

Assumptions:

a. Users will have a stable internet connection to access the platform.
b. Recruiters will provide accurate job descriptions and criteria.
c. The MongoDB database will remain operational without major downtimes.

Dependencies:

d. Reliant on third-party authentication (Google/Facebook) and email APIs.
e. Continuous availability of the MongoDB database.
f. External libraries like Material UI for enhanced development efficiency.

Limitations:

- **Functionality**:
  - Limited support for regional languages in the initial version.
  - Advanced analytics features for recruiters are planned for future releases.
- **Performance**:
  - Initial deployment may experience slower response times under extreme loads (>1000 concurrent users).
- **Compatibility**:
  - Requires modern browsers; may not support legacy browsers like Internet Explorer

Risks:

- **Technical Risks**:
  - Dependency on third-party APIs like Google and Facebook might cause disruptions if services are unavailable.
    - **Mitigation**: Implement fallback authentication methods.
  - Risk of data breaches due to sensitive user data storage.
    - **Mitigation**: Use encryption and secure access controls.
- **Operational Risks**:
- Recruiters may misuse the platform for spam or fake job postings.
- **Mitigation**: Implement a reporting and moderation system.
- **Schedule Risks**:
  - Delays in development due to team members' lack of expertise in certain technologies.
    - **Mitigation**: Allocate time for training and prioritize tasks.

3.4 Design Models

3.5 Design Class Diagram (DCD)

1. Class Diagram

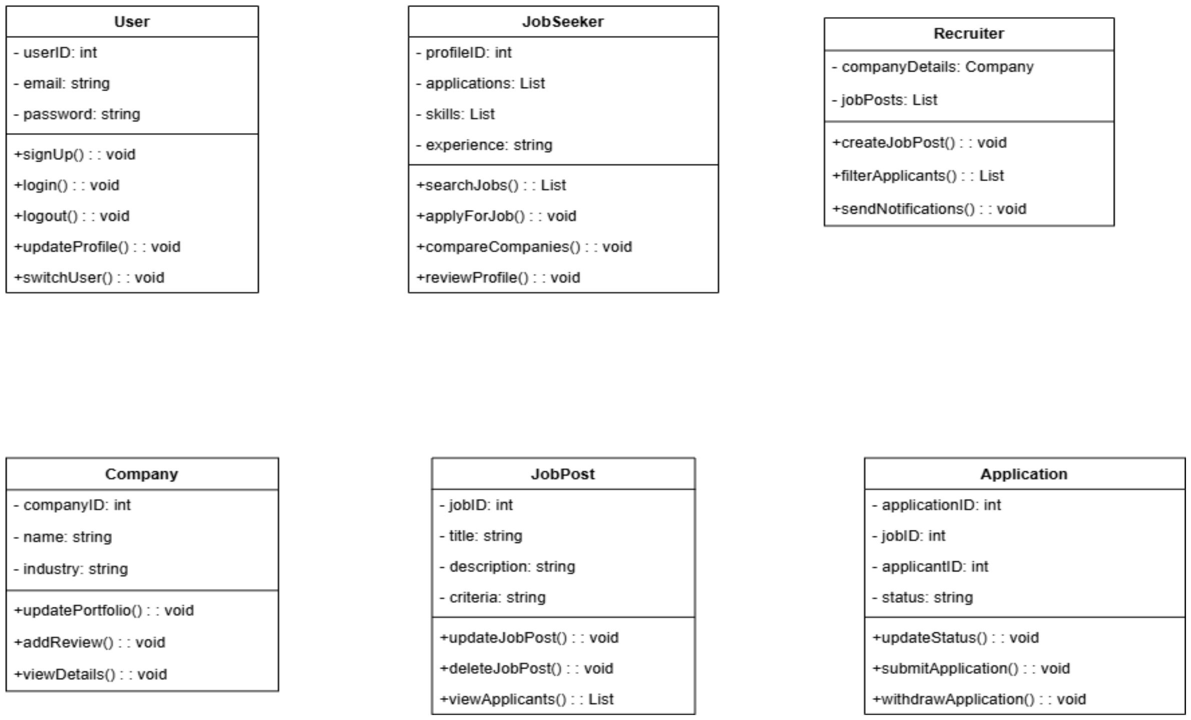


Figure 1 Class Diagram

3.6 Interaction Diagram (Either sequence or collaboration)

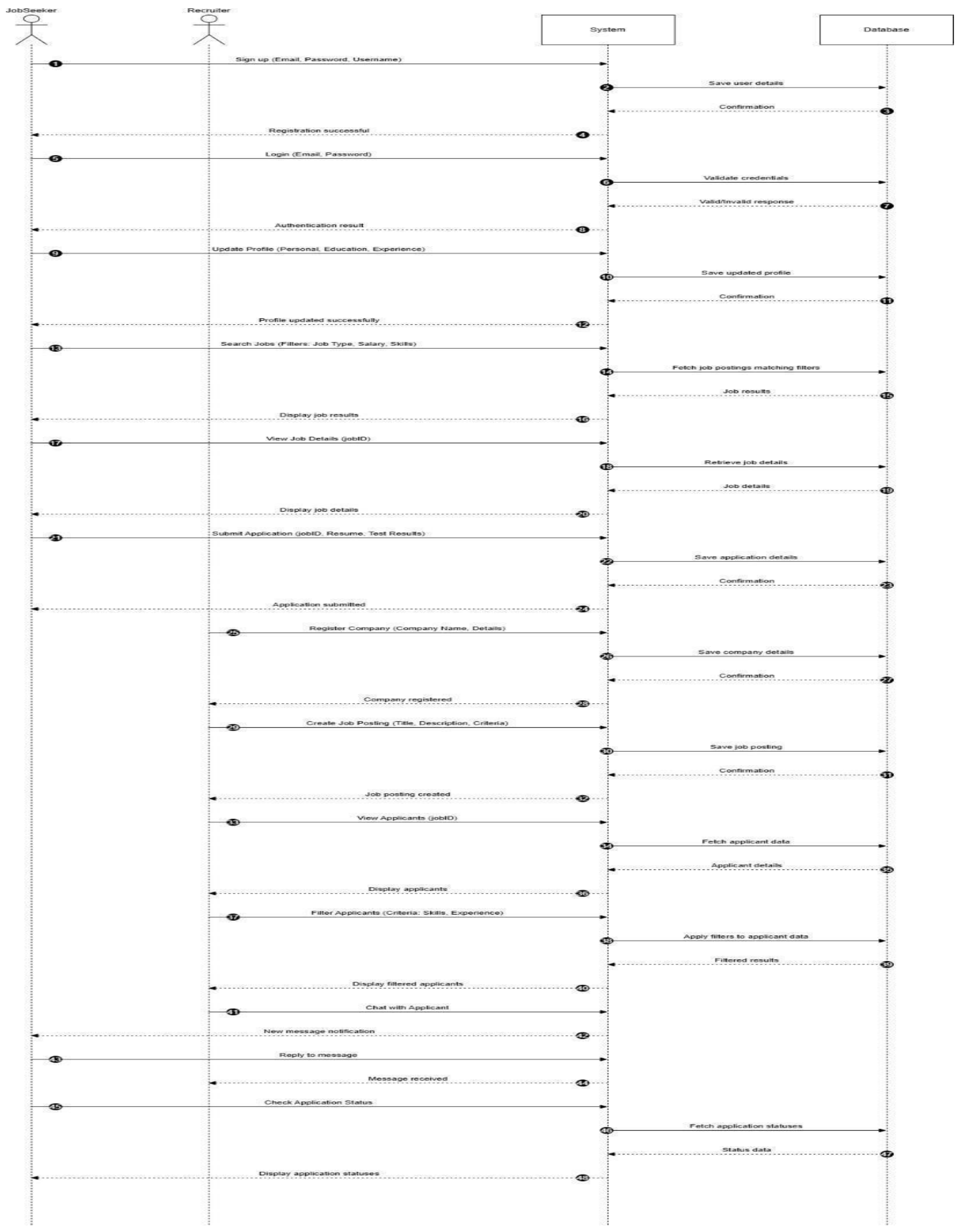

JobSeeker
Recruiter
System
Database
1 Sign up (Email, Password, Username)
2 Save user details
3 Confirmation
4 Registration successful
5 Login (Email, Password)
6 Validate credentials
7 Valid/Invalid response
8 Authentication result
9 Update Profile (Personal, Education, Experience)
10 Save updated profile
11 Confirmation
12 Profile updated successfully
13 Search Jobs (Filters: Job Type, Salary, Skills)
14 Fetch job postings matching filters
15 Job results
16 Display job results
17 View Job Details (jobID)
18 Retrieve job details
19 Job details
20 Display job details
21 Submit Application (jobID, Resume, Test Results)
22 Save application details
23 Confirmation
24 Application submitted
25 Register Company (Company Name, Details)
26 Save company details
27 Confirmation
28 Company registered
29 Create Job Posting (Title, Description, Criteria)
30 Save job posting
31 Confirmation
32 Job posting created
33 View Applicants (jobID)
34 Fetch applicant data
35 Applicant details
36 Display applicants
37 Filter Applicants (Criteria: Skills, Experience)
38 Apply filters to applicant data
39 Filtered results
40 Display filtered applicants
41 Chat with Applicant
42 New message notification
43 Reply to message
44 Message received
45 Check Application Status
46 Fetch application statuses
47 Status data
48 Display application statuses

Figure 2 Sequence Diagram

3.7 State Transition Diagram (for the projects which include event handling and backend processes)

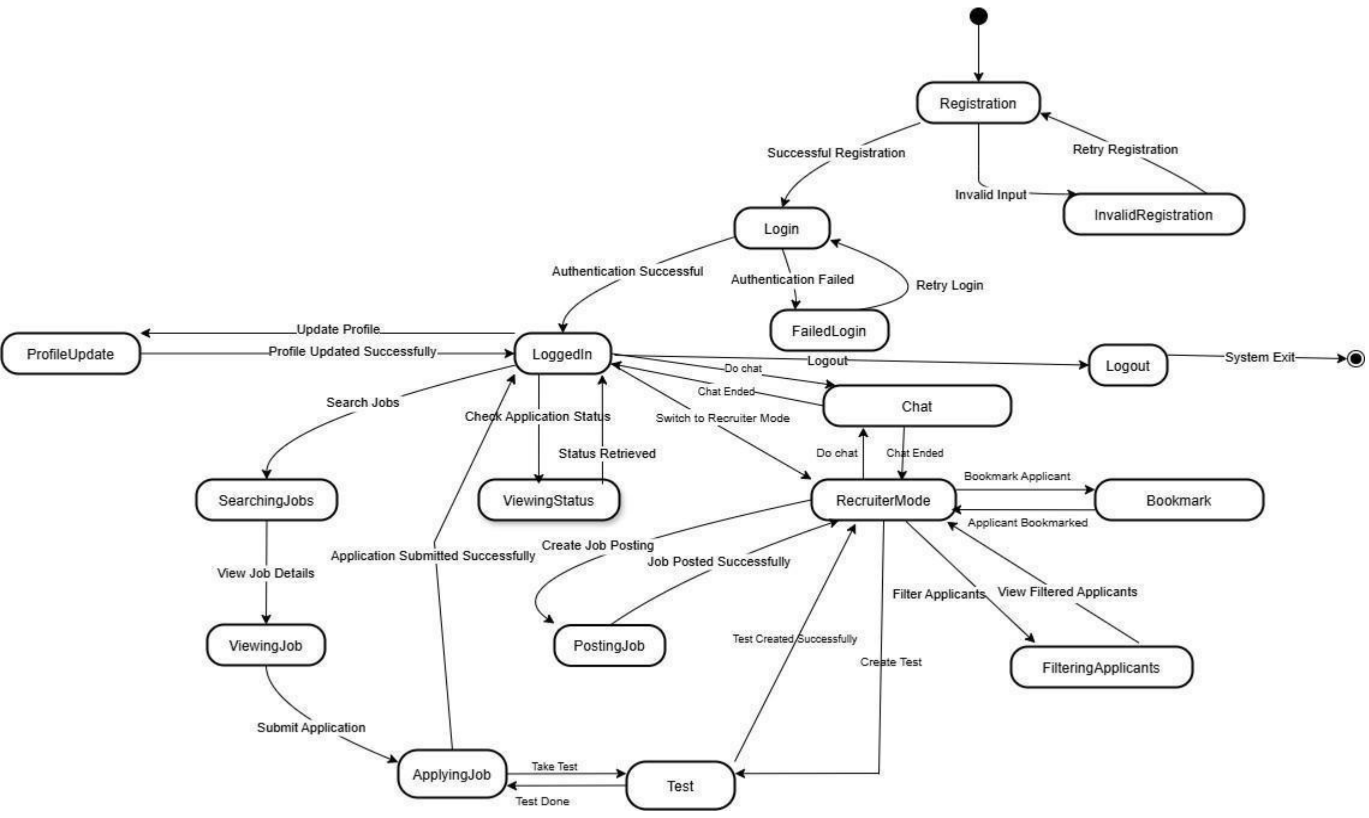


Figure 3 State Transition Diagram

3.8 Architectural Design

UML Class Relationship Diagram

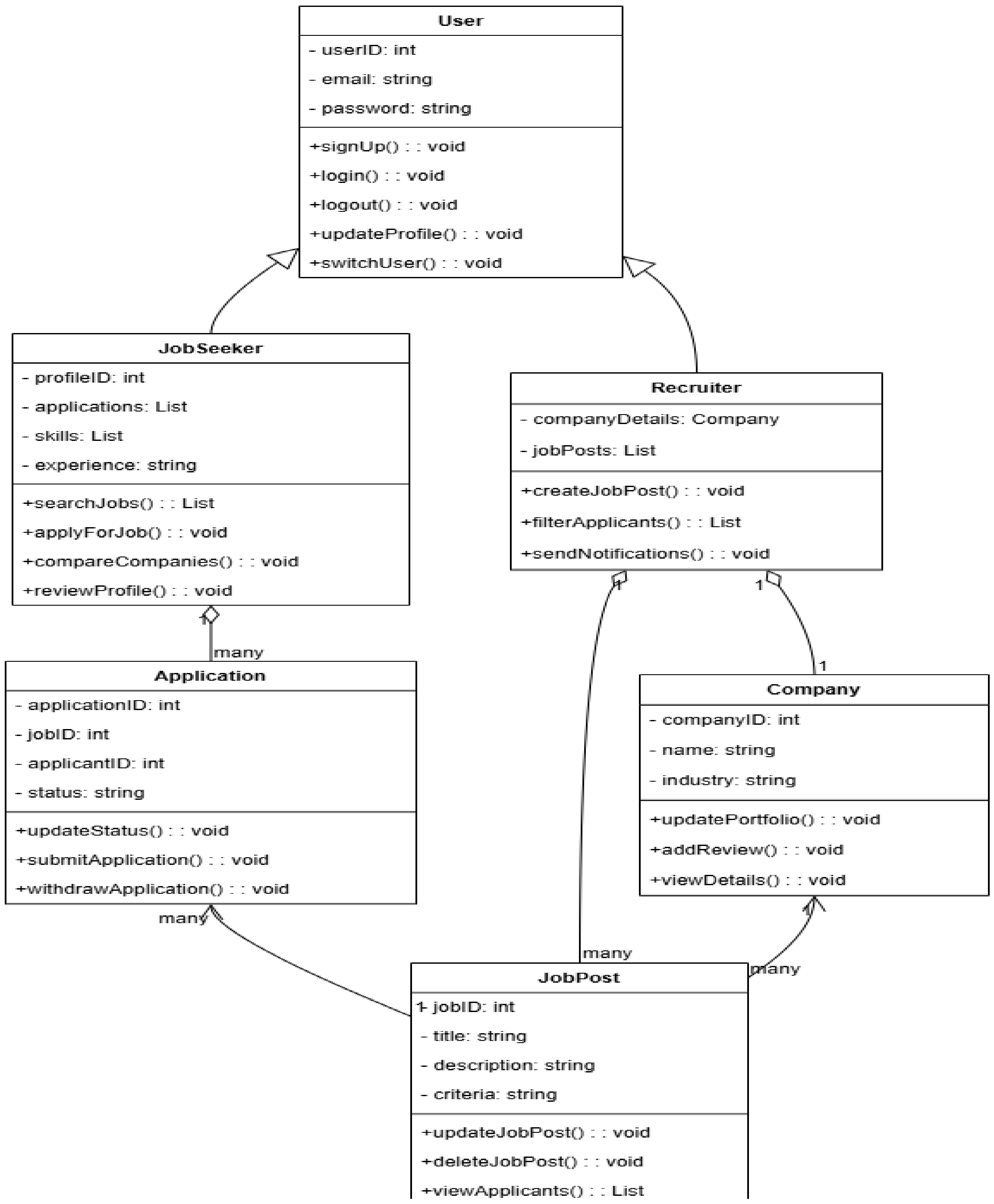


Figure 4 UML Class Relationship Diagram

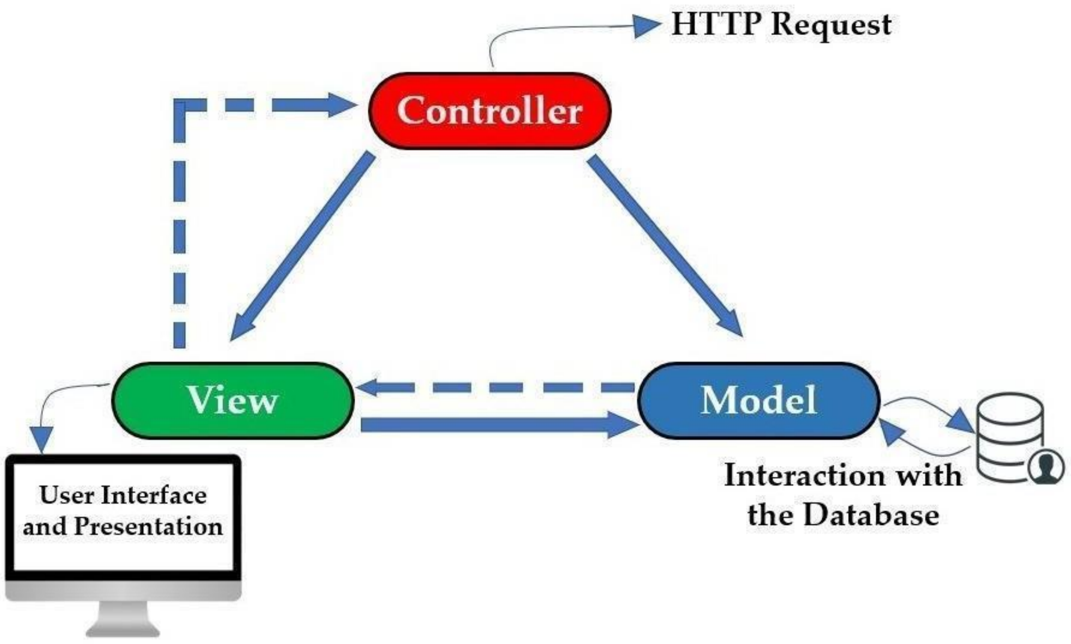

HTTP Request
Controller
View
Model
User Interface and Presentation
Interaction with the Database

Figure 5 MVC Architecture

3.9 MVC Architecture Mapping

1. Model (Data Layer)

The **Model** represents the data structure and business logic directly tied to the database. It defines how data is organized, relationships between entities, and provides interfaces for CRUD operations.

Components in the Model Layer:

- **User Model**:
    - Represents users in the system.
    - Attributes: name, email, password, photo, favouriteJobs, companies, etc.
    - Relationships:
        - Linked to Application and Review via references.

- **Company Model**:
    - Represents companies registered in the system.
    - Attributes: name, address, photo, followers, reviews, jobs.
    - Relationships:
        - Linked to Job and Review.

- **Job Model**:
    - Represents job postings.
    - Attributes: title, description, company, noOfClicks, noOfApplicants.
    - Relationships
    - Linked to Company and Application.

2. **Application Model**:
   - Represents job applications submitted by users.
   - Attributes: user, job, status, bookmarked.
   - Relationships:
     - Linked to User and Job.

3. **Review Model**:
   - Represents reviews submitted by users about companies.
   - Attributes: user, company, rating, content, upvotes, downvotes.
   - Relationships:
     - Linked to User and Company.

4. View (Presentation Layer)

The **View** is responsible for presenting the data to the users. While the provided code does not include a specific frontend, the API outputs JSON responses, making it suitable for consumption by frontend frameworks like React, Vue.js, or Angular.

Components in the View Layer:

- **API Responses**:
  - Controllers return responses in JSON format for integration with a frontend.
  - Example:
    - GET jobs returns a list of jobs with associated company details.
    - POST signup returns a token and user details upon successful registration.
- **Frontend Consumption**:
  - The data provided by the APIs is meant to be consumed and rendered by a client-side framework.

5. Controller (Logic Layer)

The **Controller** is the intermediary between the Model and the View. It processes user requests, applies business logic, and interacts with the Model to retrieve or modify data.

Components in the Controller Layer:

- **Application Controller (applicationController.js)**:
  - Handles operations related to job applications:
    - getAllApplication: Retrieves all applications.
    - createApplication: Creates a new application.
    - updateApplication: Updates an existing application.
    - deleteApplication: Deletes an application.
    - acceptApplication and rejectApplication: Updates application statuses.
- **Auth Controller (authController.js)**:
  - Manages authentication and user session:
    - signup: Registers a new user.
    - login: Logs in a user and issues a token.
    - forgotPassword and resetPassword: Handles password recovery.
- **Company Controller (companyController.js)**:
  - Manages company-related operations:
    - getAllCompany: Retrieves all companies.
    - createCompany: Creates a new company.
    - updateCompany: Updates company details.
    - deleteCompany: Deletes a company.
    - addFollower: Allows a user to follow a company.
- **Job Controller (jobController.js)**:
  - Manages job postings:
    - getAllJob: Fetches all jobs.
    - createJob: Adds a new job.
    - updateJob and deleteJob: Modifies or removes jobs.
    - addClick: Increments view count for a job.

- **Review Controller (reviewController.js)**:
    - Handles reviews for companies:
        - getAllReview: Fetches all reviews.
        - createReview: Adds a new review.
        - upvote and downvote: Handles user interactions with reviews.
- **User Controller (userController.js)**:
    - Handles user-specific operations:
        - getAllUser: Retrieves all users.
        - updateUser: Updates user information.
        - addFavourite and removeFavourite: Manages favorite jobs for a user.

6. Utilities and Middleware

These components support the MVC layers and ensure smooth operation.

Components Supporting MVC:

- **Middleware**:
    - catchAsync: Wraps asynchronous functions to catch errors.
    - appError: Standardizes error messages and codes.
    - Authentication Middleware: Validates tokens and restricts access to secure endpoints.
- **Factory Functions (handlerFactory.js)**:
    - Provides reusable CRUD functions to avoid redundancy in controllers:
        - getAll, getOne, createOne, updateOne, and deleteOne.
- **Email Utilities (emailHtml.js, email.js)**:

    - Sends email notifications for signup, password recovery, etc.

How Components Work Together

2. **User Signup**:
    a. **Controller**: authController.signup validates input, creates a new user, and issues a JWT.
    b. **Model**: User is updated with the new record.
    c. **View**: API responds with a token and user details.
3. **Job Application**:
    a. **Controller**: applicationController.createApplication validates the request and links the application to a user and job.
    b. **Model**: Application stores the relationship between the User and Job.
    c. **View**: API responds with the application details.
4. **Review Submission**:
    a. **Controller**: reviewController.createReview ensures the user hasn't already reviewed the company, then creates a new review.
    b. **Model**: Review stores the user's rating and feedback, linked to the company.
    c. **View**: API responds with the review details.
5. **Job Management**:
    a. **Controller**: jobController.createJob adds a new job linked to a company.
    b. **Model**: Job stores the job details, including references to the company.
    c. **View**: API responds with the newly created job.

3.10 Data Design

Database Design:

The information domain of the system is transformed into data structures through the use of **Mongoose schemas** in a **NoSQL database (MongoDB)**. Each major entity from the domain (e.g., Users, Jobs, Applications) is represented as a **document** within a collection, while relationships between these entities are maintained using **references (ObjectId)**.

Storage, Processing, and Organization of Data Entities

1. Storage

- **Documents and Collections:**
  Each entity (e.g., **User, Job, Application,Company,Review**) is stored as a JSON-like document in its respective collection.
  - Example:
    - **User Collection**: Stores user profiles for both job seekers and companies.
    - **Job Collection**: Stores job listings and their details.
    - **Application Collection**: Stores application data including status and applicant details.
    - **Company Collection**:To store all the information related to a company, including its details, job postings, and reviews.
    - **Review Collection**:To manage and store reviews submitted by job seekers about different companies.
- **Attributes:**
  Entities include attributes such as **name, email, skills**, which are stored as key-value pairs. Arrays and nested objects handle multi-valued or hierarchical data.
- **Relationships:**
  - **Many-to-One**: An Application belongs to one Job.
  - **Many-to-Many**: A User can apply to many jobs and a job can have many applicants.
  - **One-to-Many**: A single Job can have many Applications.

2. Processing

- **Mongoose Middleware:**
  Pre-hooks like **Pre , Post** and **Methods** ensure data is processed consistently. For example:
  - Passwords are hashed before saving.
  - Related data (e.g., user or job details) is automatically populated when fetching applications.
- **Query Optimization:**
  Use of Mongoose's query methods (e.g., **.populate()**) allows for efficient data retrieval and processing.

3. Organization

- **Hierarchical and Relational Structure:**
  The system organizes data hierarchically:
  - Jobs belong to companies.
  - Applications link users to jobs.
  - Reviews are tied to users and companies.
- **Embedded Data:**
  Attributes like skills in Job or questions in Job.test are stored as arrays or nested objects for easy access and scalability.

Data Model:

Collection-based schema:

- **User Collection**: Stores user profiles for both job seekers and companies.
- **Job Collection**: Stores job listings and their details.
- **Application Collection**: Stores application data including status and applicant details.

3.11 Data Dictionary

 Alphabetical List of System Entities or Major Data with Types and Descriptions:

User Collection:

Terminology	**Description**
_id	Unique identifier for the user (generated automatically by MongoDB). (Data Type: ObjectId)
name	Full name of the user. (Data Type: String)
email	Unique email address (used for login). (Data Type: String)
password	Encrypted password for authentication. (Data Type: String)
photo	URL or path to the user's profile picture. (Data Type: String)
role	Role of the user (`job-seeker` or `company-representative`). (Data Type: String)
favouriteJobs	References to jobs the user has marked as favorite. (Data Type: Array of ObjectId)
createdAt	Timestamp of account creation. (Data Type: Date)
applications	References to applications made by the user. (Data Type: Array of ObjectId)

Table 32 User Collection

Job Collection:

Terminology	Description
_id	Unique identifier for the job. (Data Type: ObjectId)
title	Title of the job position (e.g., Software Engineer). (Data Type: String)
description	Detailed description of the job. (Data Type: String)
location	Location of the job (e.g., Remote, Onsite, City). (Data Type: String)
salaryRange	Salary range for the job posting. (Data Type: String)
skills	Required or preferred skills for the job. (Data Type: Array of String)
company	Reference to the company posting the job. (Data Type: ObjectId)
datePosted	Timestamp when the job was posted. (Data Type: Date)
applications	References to applications submitted for this job. (Data Type: Array of ObjectId)

Table 33 Job Collection

Application Collection:

Terminology	Description
_id	Unique identifier for the application. (Data Type: ObjectId)
job	Reference to the associated job. (Data Type: ObjectId)
user	Reference to the user who submitted the application. (Data Type: ObjectId)
status	Status of the application (\`Pending\`, \`Accepted\`, \`Rejected\`). (Data Type: String)
testResult	Test score, if applicable. (Data Type: Number)
createdAt	Timestamp of application submission. (Data Type: Date)

Table 34 Application Collection

Company Collection:

Terminology	Description
_id	Unique identifier for the company. (Data Type: ObjectId)
name	Name of the company. (Data Type: String)
description	Description of the company. (Data Type: String)
industry	Industry type of the company (e.g., IT, Finance). (Data Type: String)
jobs	References to jobs posted by the company. (Data Type: Array of ObjectId)
reviews	References to reviews about the company. (Data Type: Array of ObjectId)

Table 35 Company Collection

Review Collection:

Terminology	**Description**
_id	Unique identifier for the review. (Data Type: ObjectId)
user	Reference to the user who submitted the review. (Data Type: ObjectId)
company	Reference to the company being reviewed. (Data Type: ObjectId)
rating	Rating given by the user (e.g., 1 to 5 stars). (Data Type: Number)
title	Title or summary of the review. (Data Type: String)
description	Detailed description of the review. (Data Type: String)
upvotes	References to users who upvoted the review. (Data Type: Array of ObjectId)
downvotes	References to users who downvoted the review. (Data Type: Array of ObjectId)
createdAt	Timestamp of when the review was submitted. (Data Type: Date)

Table 36 Review Collection

Structured Approach: Functions and Function Parameters:

User Controller:

1. **getAllUser**
   - Parameters: None
   - Description: Fetches all users from the database.
   - Parameter Details: N/A
2. **getSingleUser**
   - Parameters: **id (string)**
   - Description: Fetches a specific user by ID.
   - Parameter Details:
     - **id**: The unique identifier of the user.
3. **createUser**
   - Parameters: **data (object)**
   - Description: Adds a new user to the database.
   - Parameter Details:
     - **data**: Object containing user details (e.g., name, email).

4. **updateUser**
   - Parameters: **id (string), data (object)**
   - Description: Updates a user's details.
   - Parameter Details:
     - **id**: The unique identifier of the user.
     - **data**: Object containing updated fields.

5. **deleteUser**
   - Parameters: **id (string)**
   - Description: Deletes a user by ID.
   - Parameter Details:
     - **id**: The unique identifier of the user.

Job Controller:

1. **getAllJob**
   - Parameters: None
   - Description: Fetches all jobs from the database and populates associated company fields.
   - Parameter Details: N/A
2. **getSingleJob**
   - Parameters**: id (string)**
   - Description: Fetches a specific job by ID.
   - Parameter Details:
     - **id**: The unique identifier of the job.
3. **createJob**
   - Parameters**: data (object)**
   - Description: Adds a new job to the database.
   - Parameter Details:
     - **data**: Object containing job details (e.g., title, company).
4. **updateJob**
   - Parameters: **id (string), data (object)**
   - Description: Updates a job by ID.
   - Parameter Details:
     - **id**: The unique identifier of the job.
     - **data**: Object containing updated fields.
5. **deleteJob**
   - Parameters: **id (string)**
   - Description: Deletes a job by ID.
   - Parameter Details:
     - **id**: The unique identifier of the job.

Application Controller:

1. **getAllApplication**
   - Parameters: None
   - Description: Fetches all applications from the database and populates associated job and user fields.
   - Parameter Details: N/A
2. **getSingleApplication**
   - Parameters: **id (string)**
   - Description: Fetches a specific application by ID.
   - Parameter Details:
     - **id**: The unique identifier of the application (string).
3. **createApplication**
   - Parameters: **data (object)**
   - Description: Creates a new application in the database.
   - Parameter Details:
     - **data**: Object containing application details (e.g., user, job).
4. **updateApplication**
   - Parameters: **id (string), data (object)**
   - Description: Updates an application by ID.
   - Parameter Details:
     - **id**: The unique identifier of the application (string).
     - **data**: Object containing updated fields.
5. **deleteApplication**
   - Parameters: **id (string)**
   - Description: Deletes an application from the database.
   - Parameter Details:
     - **id**: The unique identifier of the application.

Authentication Controller:

1. **signup**
   - Parameters: **name (string), email (string), password (string), passwordConfirm (string), photo (file)**
   - Description: Registers a new user and generates a token.
   - Parameter Details:
     - **name**: User's full name.
     - **email**: User's email.
     - **password**: Secure password.
     - **passwordConfirm**: Password confirmation.
     - **photo**: Optional user profile picture.
2. **login**
   - Parameters: **email (string), password (string)**
   - Description: Logs in a user and sends a token.
   - Parameter Details:
     - **email**: User's email.
     - **password**: User's password.
3. **forgotPassword**
   - Parameters: **email (string)**
   - Description: Sends a password reset token to the user's email.
   - Parameter Details:
     - **email**: User's email address.
4. **resetPassword**
   - Parameters: **token (string), password (string), passwordConfirm (string)**
   - Description: Resets the user's password.
   - Parameter Details:
     - **token**: The reset token.
     - **password**: New password.
     - **passwordConfirm**: Confirmed new password.

Company Controller:

1. **getAllCompany**
   - Parameters: None
   - Description: Fetches all companies from the database.
   - Parameter Details: N/A
2. **createCompany**
   - Parameters: **data (object)**
   - Description: Adds a new company to the database.
   - Parameter Details:
     - **data**: Object containing company details.
3. **updateCompany**
   - Parameters: **id (string), data (object)**
   - Description: Updates a company's details.
   - Parameter Details:
     - **id**: The unique identifier of the company.
     - **data**: Object containing updated fields.

4. **deleteCompany**
   - Parameters: **id (string)**
   - Description: Deletes a company by ID.
   - Parameter Details:
     - **id**: The unique identifier of the company.

Review Controller:

5. **getAllReview**
   - Parameters: None
   - Description: Fetches all reviews from the database.
   - Parameter Details: N/A

6. **getSingleReview**
   - Parameters: **reviewId (String)**
   - Description: Retrieves a specific review by its unique ID.
   - Parameter Details:
     - **reviewId**: The ID used to locate the review in the database

7. **createReview**
   - Parameters: **companyId (String) ,userId (String), rating (Number),content (String)**
   - Description: Creates a new review in the database.
   - Parameter Details:
     - **companyId**: Identifies the company being reviewed.
     - **userId**: Identifies the user creating the review.
     - **rating**: A numeric score representing the quality of the company.
     - **content**: Textual feedback about the company.

8. **updateReview**
   - Parameters:**reviewId (String), data (Object)**
   - Description: Updates the details of a specific review.
   - Parameter Details:
     - **reviewId**: Used to identify the review to update.
     - **data**: Contains the updated values for the review fields.

9. **deleteReview**
   - Parameters:**reviewId (String)**
   - Description: Deletes a review from the database
   - Parameter Details:
     - **reviewId**: Identifies the review to be removed.

10. **upvote**
    - Parameters:**reviewId (String), userId (String)**
    - Description: Adds an upvote to the review and removes any existing downvote by the same user.
    - Parameter Details:
      - **reviewId**:Specifies the review to upvote.

- **userId**:Identifies the user performing the action.

11. **downvote**
    - Parameters:**reviewId (String), userId (String)**
    - Description: Adds a downvote to the review and removes any existing upvote by the same user.
    - Parameter Details:
        - **reviewId**:Specifies the review to downvote.
        - **userId**:Identifies the user performing the action.

 Object-Oriented (OO) Approach: Objects, Attributes, Methods, & Method Parameters:

1. User Object

Attributes:

- **name (String)**: Full name of the user.
- **email (String)**: Email address for login and communication.
- **photo (String)**: Profile picture URL or file path.
- **password (String)**: Encrypted user password.
- **favouriteJobs (Array of Job IDs)**: List of jobs the user has marked as favorite.
- **companies (Array of Company IDs)**: List of companies the user is associated with.

Methods:

1. **addFavourite(jobId)**
    - Parameters:
        - **jobId (String)**: The ID of the job to add to favorites.
    - Description: Adds a job to the user's list of favorite jobs.
2. **removeFavourite(jobId)**
    - Parameters:
        - **jobId (String)**: The ID of the job to remove from favorites.
    - Description: Removes a job from the user's list of favorite jobs.
3. **getFavouriteJobs()**
    - Parameters: None
    - Description: Retrieves all jobs marked as favorites by the user.
4. **verifyEmail()**
    - Parameters: None
    - Description: Marks the user's email as verified.
5. **updateUser(data)**
    - Parameters:
        - **data (Object)**: Fields to update (e.g., address, phone, etc.).
    - Description: Updates the user's details with the provided data.

2. Company Object

Attributes:

- **name (String)**: Name of the company.
- **address (String)**: Address of the company.
- **photo (String)**: Company logo or profile picture.
- **followers (Array of User IDs)**: List of users following the company.
- **reviews (Array of Review IDs)**: List of reviews related to the company.
- **jobs (Array of Job IDs)**: List of jobs posted by the company.

Methods:

1. **addFollower(userId)**
   - Parameters:
     - **userId (String)**: The ID of the user following the company.
   - Description: Adds a user to the company's followers list.
2. **getFollowing(userId)**
   - Parameters:
     - **userId (String)**: The ID of the user.
   - Description: Retrieves companies the user is following.
3. **getCompanyReviews(companyId)**
   - Parameters:
     - companyId (String): The ID of the company.
   - Description: Fetches reviews associated with the company.

3. Job Object

Attributes:

- **title (String)**: Title of the job position.
- **description (String):** Detailed description of the job.
- **company (Company ID):** The company offering the job.
- **noOfClicks (Number):** The number of times the job has been viewed.
- **noOfApplicants (Number):** The number of users who applied for the job.

Methods:

1. **addClick()**
   - Parameters: None
   - Description: Increments the number of clicks (views) for the job.
2. **addApplyClick()**
   - Parameters: None
   - Description: Increments the number of applications for the job.

4. Application Object

Attributes:

- **user (User ID)**: The user who submitted the application.
- **job (Job ID)**: The job applied for.
- **status (String):** The status of the application (e.g., Pending, Accepted, Rejected).
- **bookmarked (Boolean):** Whether the application is bookmarked by the user.

Methods:

1. **bookMarkApplication()**
    - Parameters:
        - **applicationId (String):** The ID of the application to bookmark.
    - Description: Toggles the bookmark status for an application.
2. **acceptApplication()**
    - Parameters:
        - **applicationId (String):** The ID of the application to accept.
    - Description: Changes the status of the application to "Accepted".
3. **rejectApplication()**
    - Parameters:
        - **applicationId (String):** The ID of the application to reject.
    - Description: Changes the status of the application to "Rejected".

5. Review Object

Attributes:

- **user (User ID):** The user who wrote the review.
- **company (Company ID):** The company being reviewed.
- **rating (Number):** The rating given by the user (e.g., 1-5).
- **content (String):** The content of the review.
- **upvotes (Array of User IDs):** List of users who upvoted the review.
- **downvotes (Array of User IDs):** List of users who downvoted the review.

Methods:

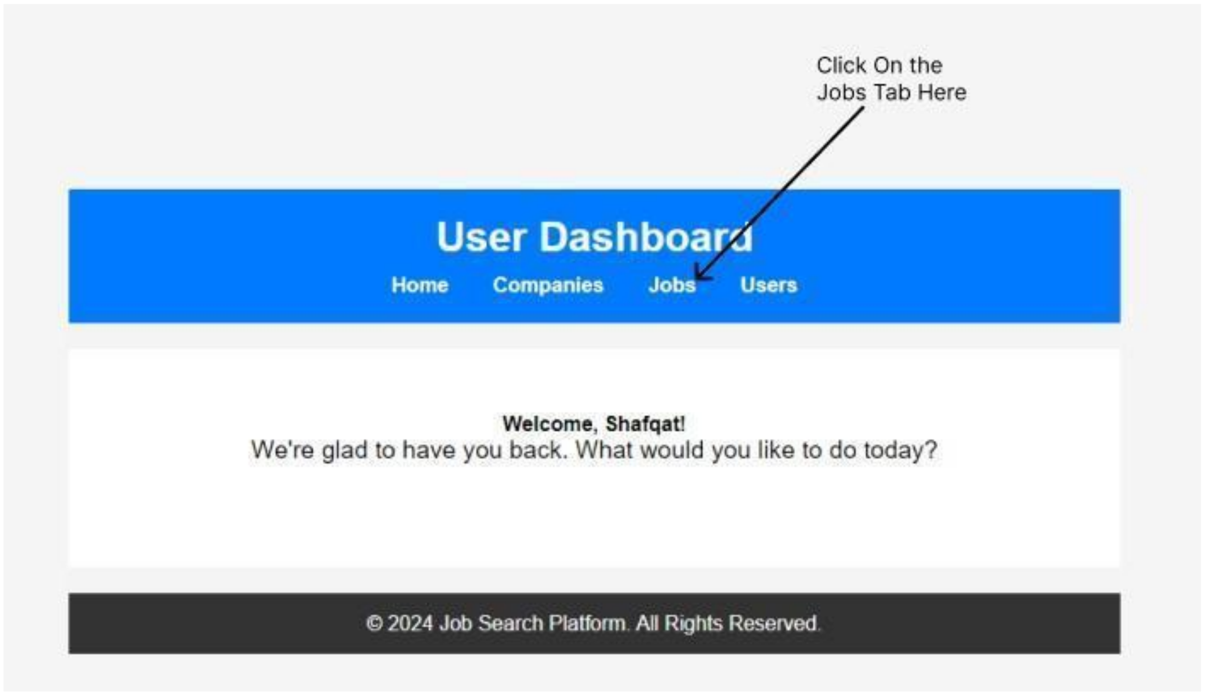
1. **upvote(userId)**
    - Parameters:
        - **userId (String):** The ID of the user upvoting the review.
    - Description: Adds an upvote to the review and removes any existing downvote by the same user.
2. **downvote(userId)**
    - Parameters:
        - **userId (String):** The ID of the user downvoting the review.
    - Description: Adds a downvote to the review and removes any existing upvote by the same user.

3.12 User Interface Design

Describe overview of the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user. [See the User Interface Design in Appendix -]

Screen Images

Job Search Feature:



Welcome, Shafqat!
We're glad to have you back. What would you like to do today?

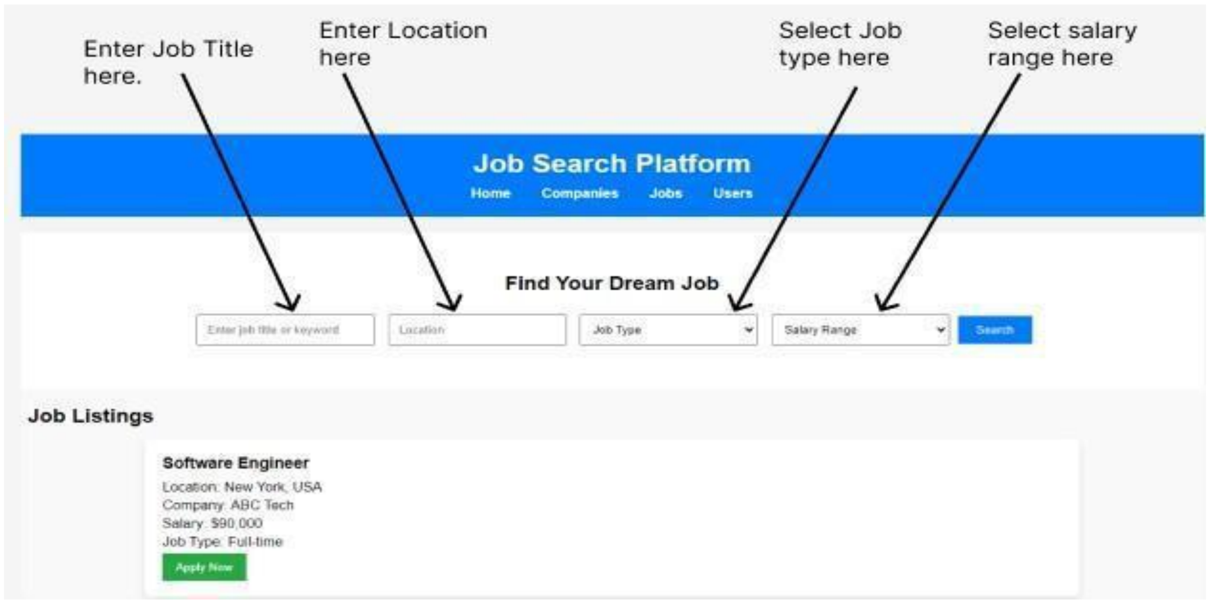
© 2024 Job Search Platform. All Rights Reserved.

Figure 6 Job Search Feature:

3 User is on the dashboard, logged in and viewing navigation options.

4 User clicks the Jobs tab in the navigation menu as shown by the arrow in the above image.

5 The system redirects to the job search page with a search bar and filters.



Software Engineer
Location: New York, USA
Company: ABC Tech
Salary: $90,000
Job Type: Full-time

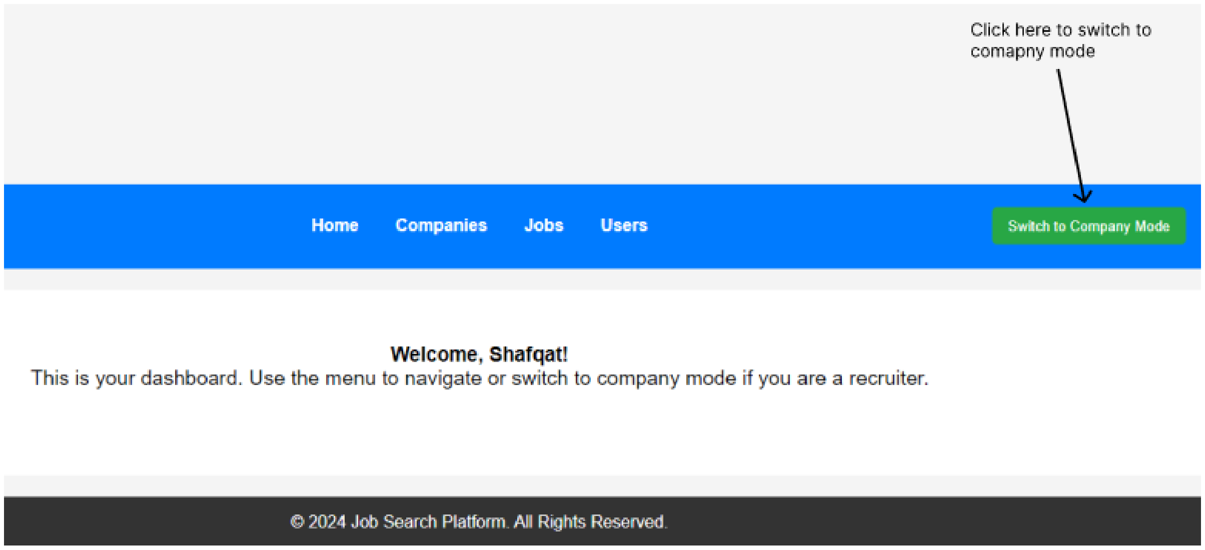
Figure 7 Job Search Feature 1

6 User enters a job title or keyword, and optionally selects location, salary range, or job type.

7 User clicks the Search button.

8 The system displays job listings matching the search criteria.

Switch To Company Mode Feature:



Welcome, Shafqat!
This is your dashboard. Use the menu to navigate or switch to company mode if you are a recruiter.

© 2024 Job Search Platform. All Rights Reserved.

Figure 8 Switch To Company Mode

9 User is on the main dashboard, managing job applications and profiles.

10 "Switch to Company Mode" button is located in the top-right corner.

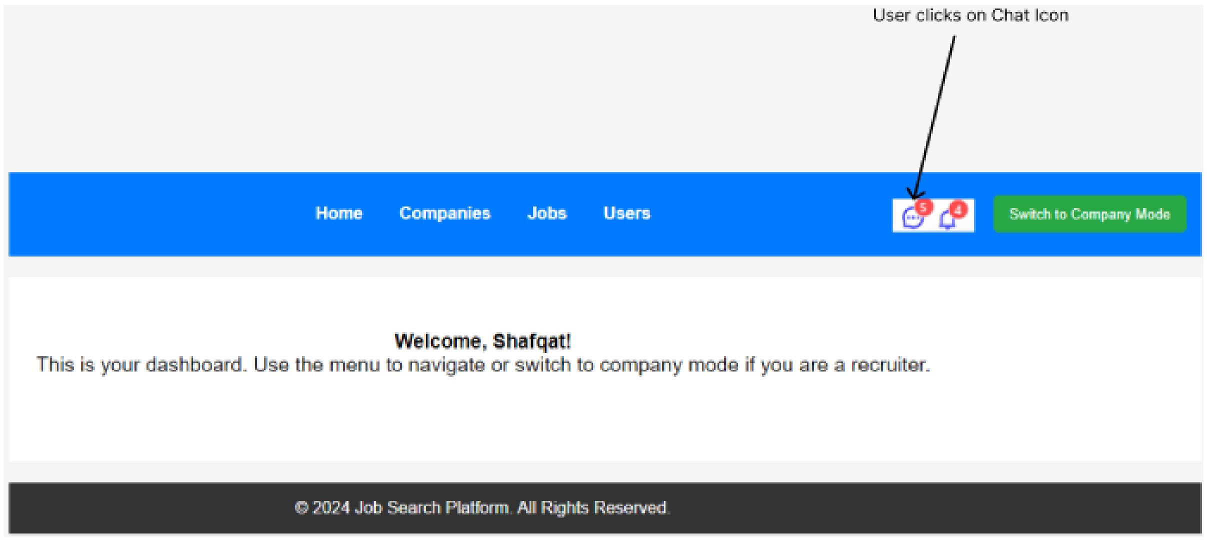
11 User clicks the button to switch from Job Seeker Mode to Company Mode.

12 Dashboard updates to display company-related options: job posting, applicant management, etc.

13 A visible indicator confirms the user is now in Company Mode.

14 Users can switch back to Job Seeker Mode by clicking the corresponding button.

Chat Feature:



Welcome, Shafqat!
This is your dashboard. Use the menu to navigate or switch to company mode if you are a recruiter.

© 2024 Job Search Platform. All Rights Reserved.

Figure 9 Chat Feature

15 Users are on the dashboard, viewing their profile or job applications.

16 "Chat" icon is located in the top-right corner of the screen.

17 User clicks the "Chat" icon, and a chat window opens.

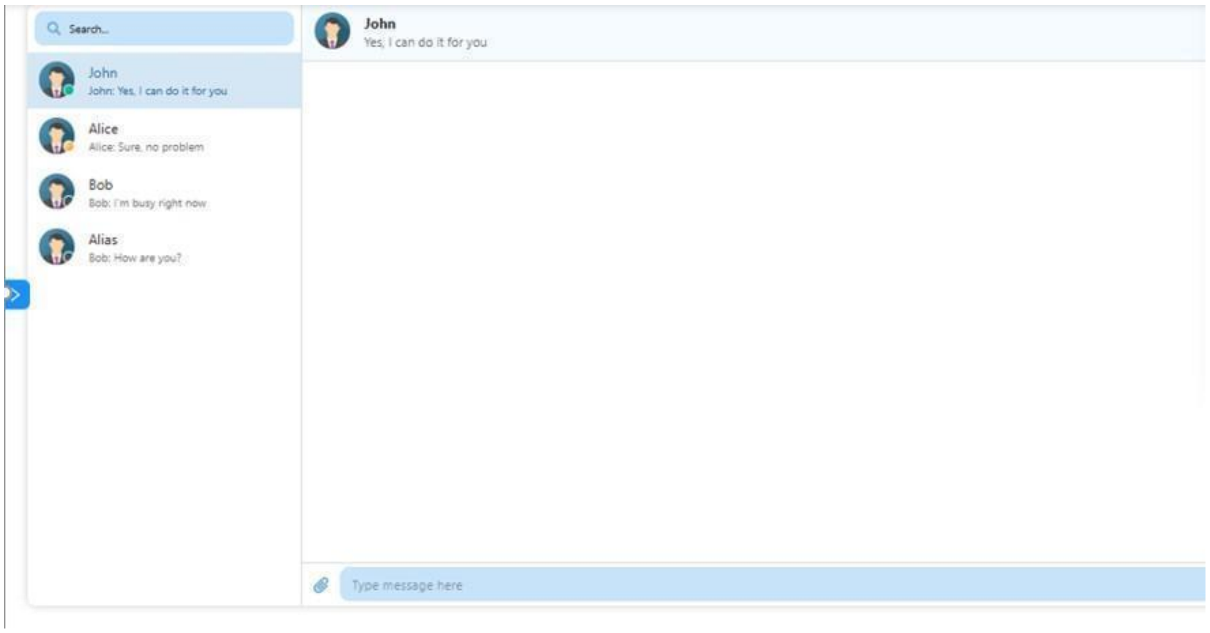


Figure 10 Chat Feature 1

18 In the chat window, there is a search bar where the user can search for a specific user to chat with.

19 Users type the name of the person they want to chat with, and relevant results appear.

20 User clicks on the desired user, and the conversation window opens, displaying the chat history

3.13 Screen Objects and Actions

Screen Objects and Actions: Job Search Use Case

1. User Dashboard

- **Objects**: Navigation bar (tabs like "Jobs"), welcome text, footer.
- **Action**: User clicks the "Jobs" tab, and the system redirects to the Job Search page.

2. Job Search Page

- **Objects**: Search inputs (Job Title, Location, Job Type, Salary Range), "Search" button, job listings, "Apply Now" button.
- **Actions**:
    1. User fills in search criteria and clicks "Search" to filter jobs, with results dynamically displayed.
    2. User clicks "Apply Now" on a job, triggering the system to handle the application submission or redirect.

Switch to Company Mode Use Case

Screen Objects:

- **Navigation Bar**: Tabs like "Home," "Companies," "Jobs," and "Users."
- **Button**: "Switch to Company Mode."

Actions:

- **Click "Switch to Company Mode"**: User clicks the button, and the system switches to the company dashboard for recruiter functionalities like managing job postings and applications.

3.14 Behavioural Model

Interaction Diagram (Either sequence or collaboration)

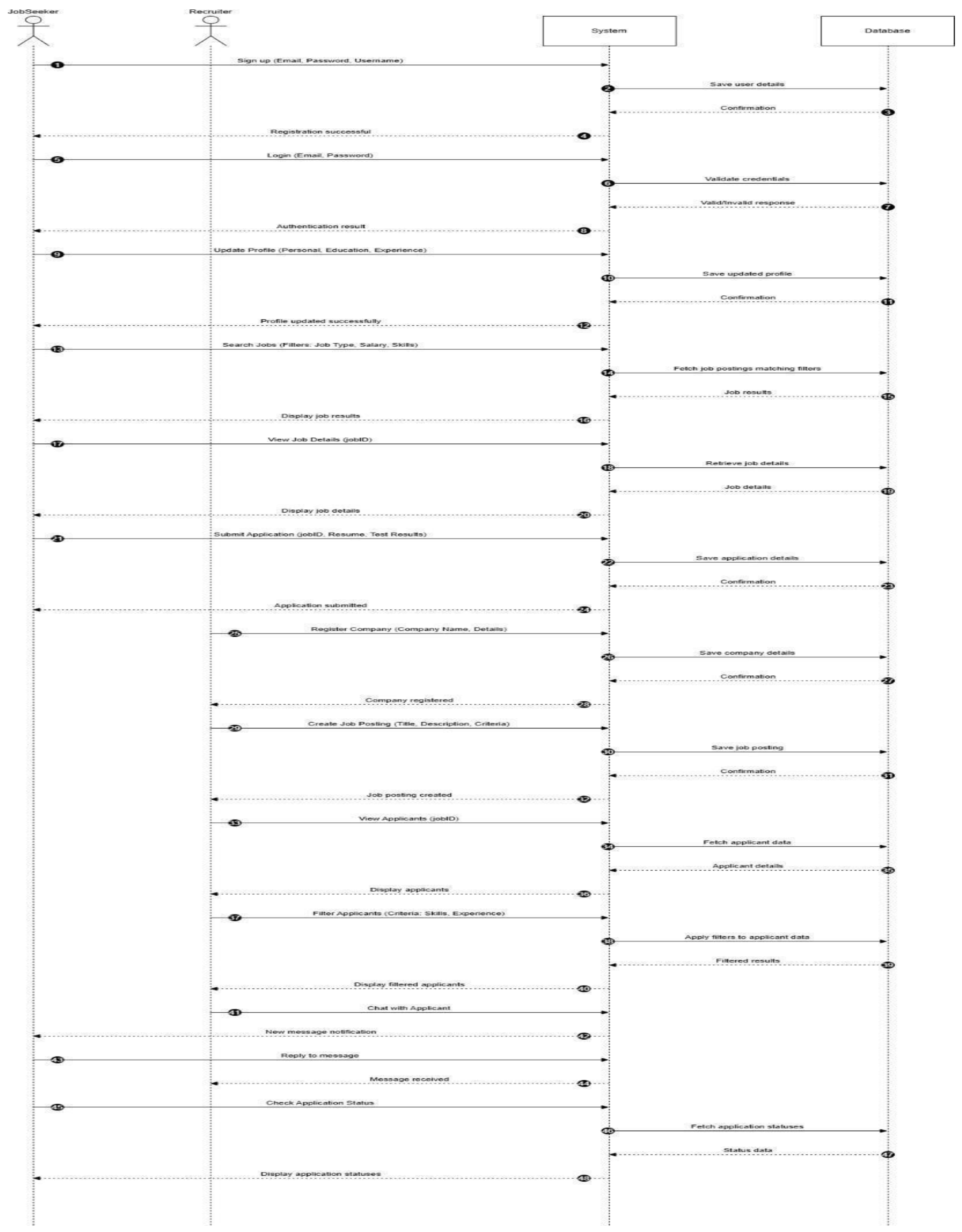


Figure 11 Sequence Diagram

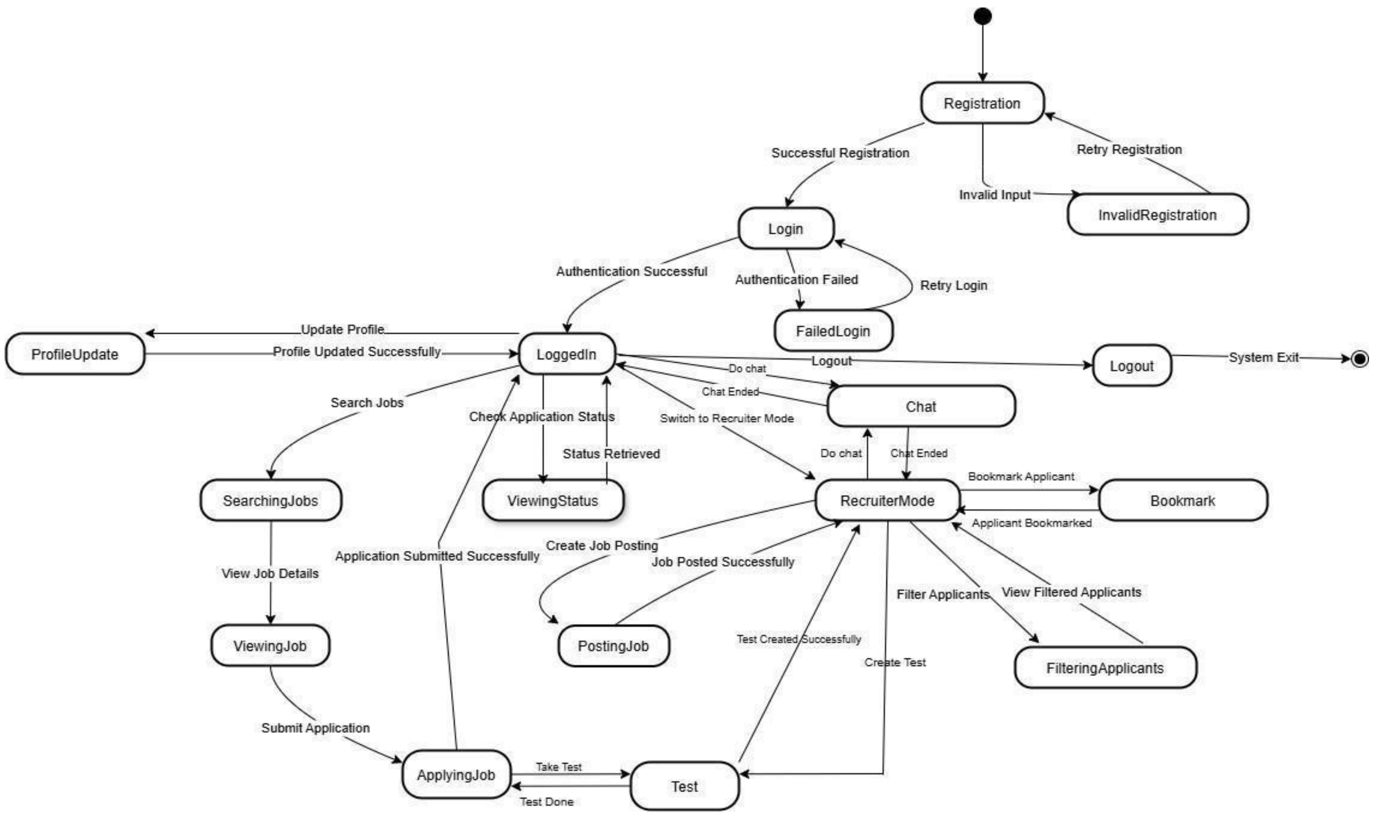


Figure 12 State Transition Diagram

3.15 Design Decisions

Design Decision

Aspect	**Decision**	**Reason**
Design Pattern	Procedural	Simpler structure and straightforward implementation.
Communication Protocols	RESTful	Statelessness, ease of integration, and uniformity.
Database Normalization	Referencing & Embedding	Ensures data integrity and avoids redundancy.
Security Measures	JSON Web Tokens	Secure, scalable, and stateless.
Database Type	NoSQL	Handles unstructured data and provides high scalability.
Frontend-Backend Format	JSON	Lightweight, easy to parse, and widely supported.
Error Handling	Centralized Middleware	Consistent error messages and easier debugging.
Algorithmic Approach	Pagination and Filtering	Improves performance and user experience.

3.16 Summary

This chapter summarizes the **key design principles** and decisions taken to build a modular, scalable, and secure system using the **MVC architecture**. The **Model** encapsulates data and relationships, the **Controller** manages business logic, and the **View** outputs integration-ready JSON for frontend use.

Key refinements include centralized error handling, reusable factory functions for CRUD operations, and dynamic data enrichment using populate. RESTful APIs with JWT-based authentication and a normalized NoSQL database ensure security, data integrity, and efficient performance.

These decisions align with the project’s objectives by enabling robust functionality for user management, job applications, and reviews while ensuring scalability and maintainability for future enhancements.

Chapter 4

Implementation

4. Introduction

4.1 Algorithm

CompanyComparisonView Pseudocode

Input:

- **GET Request: request.user: Authenticated user (must be logged in).**
- **POST Request (JSON Body):**
  - **first_company_id: ID of the first company selected.**
  - **second_company_id: ID of the second company selected.**

Output:

- **GET Request: Company comparison data (graphical comparison, details).**
- **POST Request: Success or Error message (if companies are not selected)**

Function: GET(request)

```
- Check if the user is authenticated
- If not authenticated → redirect to login page
- Retrieve all companies from the database
- If no companies found → return error message
- Display comparison options (e.g., dropdowns to select companies)
- Return list of companies to frontend for user selection
```

Function: POST(request)

```
- Extract `first_company_id` and `second_company_id` from request body
- If either `first_company_id` or `second_company_id` is missing → return error message
- Retrieve the selected companies from the database by their IDs
- If any company not found → return error message
- Get key details of the selected companies (e.g., name, photo, rating, location)
- Generate comparison graph based on selected parameters (e.g., reviews, ratings, location)
- Return success message with comparison graph data
```

Pseudocode for CV Filtering Logic (UC13)

Input:

GET Request: request.user: Authenticated user (must be a recruiter).

POST Request (JSON Body):

job_id: ID of the posted job.

cv_data: List of CVs (job applications) received for the job.

Output:

GET Request: List of filtered CVs that meet the job criteria.

POST Request: Success or Error message (if no CVs meet criteria or system failure occurs).

Function: POST(request)

```
- Extract `job_id` and `cv_data` from request body
- If `job_id` or `cv_data` is missing → return error message
- Retrieve job details using `job_id`
- Extract job criteria (e.g., required skills, experience, qualifications) from job details
- Filter out irrelevant CVs based on job criteria:
    - For each CV in `cv_data`:
        - Extract skills and experience from the CV
        - Compare extracted information with the job's required skills and experience
        - If CV does not meet the criteria:
            - Mark it as irrelevant
        - If CV meets the criteria:
            - Add it to the list of relevant CVs
- If no relevant CVs are found → return message "No CVs match the job criteria"
- If relevant CVs are found → return success message with filtered CVs data
```

4.2 External APIs/SDKs

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

Table 5- 1 Details of APIs used in the project

Name of API and version	**Description of API**	**Purpose of usage**	**List down the API endpoint/function/class in which it is used**
JWT (jsonwebtoken v9.0.0)	Token-based authentication	To create and verify login/authentication tokens	`createJWT()`, `verifyJWT()`

4.3 Code Repository

Git Repository Link:
The repository for the group project can be accessed using the following link:
Grad Hire Repository

i. Metrics of the Git Repository: The following metrics will be monitored to ensure effective use of the Git repository:

Commits: 22 commits across frontend and backend files.

Branches: 3 active branches; 2 merged to main.

Contributors:

- Muhib Arshad
- Shafqat Abbas
- Muhammad Bilal
- Faiz Muhammad

Code Reviews: The number of reviews conducted on pull requests to ensure code quality and compliance with project standards.

4.4 Summary

This chapter outlines the implementation architecture of GradHire, presenting core algorithmic logic through pseudocode and documenting the third-party APIs that enhanced system functionality. It also demonstrates the development workflow managed through a Git-based version control system. The implementation aligns with the objectives of the project by ensuring maintainability, modularity, responsiveness, and collaborative efficiency throughout the development process.

Chapter 5

Testing and Evaluation

5. Introduction

5.1 Unit Testing (UT)

Unit Testing Details for UC1: User Signup

1. **Use Case Identifier:** UC1

2. **Test ID: UT-UC1-01**
Validate Email Format
   - **Attribute:** Email
   - **Value:** invalidemail.com
   - **Expected Result:** System shows “Invalid Email Address” message.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass

3. **Test ID: UT-UC1-02**
Validate Password Strength
   - **Attribute:** Password
   - **Value:** 123
   - **Expected Result:** System shows “Password is weak” message.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass

4. **Test ID: UT-UC1-03**
Validate Password Match
   - **Attribute:** Password
   - **Value:** abc123 and xyz456
   - **Expected Result:** System shows “Passwords don’t match” message.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass

5. **Test ID: UT-UC1-04**
Successful Registration
   - **Attribute:** Name, Email, Password
   - **Value:** John Doe, john@example.com, StrongPass123
   - **Expected Result:** User is registered, and the system updates the database.
   - **Actual Result:** User registered successfully.
   - **Result:** Pass

Unit Testing Details for UC2: User Login

- **Use Case Identifier:** UC2

1. **Test ID: UT-UC2-01**
   Validate Email Field
   - **Attribute:** Email
   - **Value:** invalidemail.com
   - **Expected Result:** System shows “Invalid Email Address” message.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass
2. **Test ID: UT-UC2-02**
   Validate Password Field
   - **Attribute:** Password
   - **Value:** abc123
   - **Expected Result:** System shows “Password is required” or “Invalid Password” message if empty or incorrect.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass
3. **Test ID: UT-UC2-03**
   Successful Login
   - **Attribute:** Email, Password
   - **Value:** john@example.com, StrongPass123
   - **Expected Result:** User is logged in, redirected to dashboard.
   - **Actual Result:** User logged in successfully.
   - **Result:** Pass
4. **Test ID: UT-UC2-04**
   Handle Incorrect Credentials
   - **Attribute:** Email, Password
   - **Value:** wrongemail@example.com, abc123
   - **Expected Result:** System shows “Incorrect email or password” message.
   - **Actual Result:** Correct message displayed.
   - **Result:** Pass

Unit Testing Details for UC3: Add Applicant Profile

1. **Use Case Identifier:** UC3

2. **Test ID: UT-UC3-01**
   Validate Required Fields

   - **Attribute:** Name, Email, Skills
   - **Value:** Empty fields for required details.
   - **Expected Result:** System prompts the user to fill in the required fields.
   - **Actual Result:** Correct prompt displayed.
   - **Result:** Pass

3. **Test ID: UT-UC3-02**
   Validate Skills Field

   - **Attribute:** Skills
   - **Value:** Python, JavaScript

- **Expected Result:** System accepts the skills and saves them successfully.
- **Actual Result:** Skills saved successfully.
- **Result:** Pass

4. **Test ID: UT-UC3-03**
Validate Work Experience Field

- **Attribute:** Work Experience
- **Value:** 3 years
- **Expected Result:** System accepts and saves the work experience details successfully.
- **Actual Result:** Work experience saved successfully.
- **Result:** Pass

5. **Test ID: UT-UC3-04**
Successful Profile Addition

- **Attribute:** Name, Email, Skills, Work Experience
- **Value:** John Doe, john@example.com, Python, 3 years
- **Expected Result:** Profile saved successfully, and the system updates the database.
- **Actual Result:** Profile saved successfully.
- **Result:** Pass

Unit Testing Details for UC15: Register Company

Unit Testing Details:

1. **Test ID: UT-UC15-01**
Validate Form Submission

- **Attribute:** Form Data
- **Value:** Name: Tech Solutions Inc., Email: hr@techsolutions.com
- **Expected Result:** Form submission is processed without errors.
- **Actual Result:** Form processed successfully.
- **Result:** Pass

2. **Test ID: UT-UC15-02**
Validate Email Format

- **Attribute:** Email
- **Value:** hr.techsolutions (invalid format)
- **Expected Result:** The system displays an error message: “Invalid email format.”
- **Actual Result:** Correct error message displayed.
- **Result:** Pass

3. **Test ID: UT-UC15-03**
Validate Database Entry

- **Attribute:** Company Record
- **Value:** Name: Tech Solutions Inc., Email: hr@techsolutions.com
- **Expected Result:** Company data is stored in the database successfully.
- **Actual Result:** Data stored successfully.

- **Result:** Pass

Unit Testing Details for UC16: Create Company Portfolio

Unit Testing Details:

1. **Test ID: UT-UC16-01**
Validate Form Submission
    - **Attribute:** Form Data
    - **Value:** Mission: "Empowering Innovation", Vision: "To be the leader in tech", Job Offerings: "Software Engineer, Data Scientist"
    - **Expected Result:** Form submission is processed without errors.
    - **Actual Result:** Form processed successfully.
    - **Result:** Pass
2. **Test ID: UT-UC16-02**
Validate Missing Required Fields
    - **Attribute:** Form Data
    - **Value:** Mission: Empty, Vision: "To be the leader in tech"
    - **Expected Result:** The system displays an error message: “Mission is required.”
    - **Actual Result:** Correct error message displayed.
    - **Result:** Pass
3. **Test ID: UT-UC16-03**
Validate Database Entry
    - **Attribute:** Portfolio Record
    - **Value:** Mission: "Empowering Innovation", Vision: "To be the leader in tech", Job Offerings: "Software Engineer, Data Scientist"
    - **Expected Result:** Portfolio data is saved in the database successfully.
    - **Actual Result:** Data saved successfully.
    - **Result:** Pass
4. **Test ID: UT-UC16-04**
Handle Duplicate Portfolio Submission
    - **Attribute:** Duplicate Portfolio Data
    - **Value:** Mission: "Empowering Innovation", Vision: "To be the leader in tech"
    - **Expected Result:** The system displays an error message: “A portfolio already exists for this company.”
    - **Actual Result:** Correct error message displayed.
    - **Result:** Pass

Unit Testing Details for UC17: Create Job

Unit Testing Details:

1. **Test ID: UT-UC17-01**
Validate Form Submission
   - **Attribute:** Form Data
   - **Value:** Title: Software Engineer, Description: "Develop and maintain software", Location: Remote, Salary: $70,000–$90,000
   - **Expected Result:** Form submission is processed without errors.
   - **Actual Result:** Form processed successfully.
   - **Result:** Pass

2. **Test ID: UT-UC17-02**
Validate Missing Fields in Form Submission
   - **Attribute:** Form Data
   - **Value:** Title: Empty, Description: "Develop and maintain software", Location: Remote
   - **Expected Result:** The system displays an error message: “Job title is required.”
   - **Actual Result:** Correct error message displayed.
   - **Result:** Pass

3. **Test ID: UT-UC17-03**
Validate Database Entry for Job Posting
   - **Attribute:** Job Record
   - **Value:** Title: Software Engineer, Description: "Develop and maintain software", Location: Remote, Salary: $70,000–$90,000
   - **Expected Result:** Job data is stored in the database successfully.
   - **Actual Result:** Data stored successfully.
   - **Result:** Pass

4. **Test ID: UT-UC17-04**
Handle Duplicate Job Posting
   - **Attribute:** Duplicate Job Data
   - **Value:** Title: Software Engineer, Location: Remote
   - **Expected Result:** The system displays an error message: “A job with this title already exists.”
   - **Actual Result:** Correct error message displayed.
   - **Result:** Pass

Unit Testing Details for UC19: Create Job Test

Unit Testing Details:

1. **Test ID: UT-UC19-01**
Validate Form Submission for MCQs Only
   - **Attribute:** Test Data
   - **Value:** Questions: "What is Python? [Options: A, B, C, D]"

- **Expected Result:** Form submission is processed without errors, and the MCQs are saved.
- **Actual Result:** Form processed successfully.
- **Result:** Pass

2. **Test ID: UT-UC19-02**

Handle Invalid Question Type

- **Attribute:** Question Type
- **Value:** Open-ended question: "Explain Python features."
- **Expected Result:** The system displays an error message: “Only MCQs are supported for this test.”
- **Actual Result:** Correct error message displayed.
- **Result:** Pass

3. **Test ID: UT-UC19-03**

Validate Time Limit Range

- **Attribute:** Time Limit
- **Value:** -5 minutes (negative value)
- **Expected Result:** The system displays an error message: “Invalid time limit. Please set a valid time between 5 and 180 minutes.”
- **Actual Result:** Correct error message displayed.
- **Result:** Pass

4. **Test ID: UT-UC19-04**

Validate Duplicate Test Titles

- **Attribute:** Test Data
- **Value:** Test Title: "Technical Skills Test" (already exists for the job)
- **Expected Result:** The system displays an error message: “A test with this title already exists for this job.”
- **Actual Result:** Correct error message displayed.
- **Result:** Pass

5. **Test ID: UT-UC19-05**

Validate Database Entry for Test Creation

- **Attribute:** Test Data
- **Value:** Test Title: "Technical Skills Test", Questions: "What is Python? [Options: A, B, C, D]", Time Limit: 60 minutes
- **Expected Result:** Test data is stored in the database successfully.
- **Actual Result:** Data stored successfully.
- **Result:** Pass

Unit Testing Details for UC22: Filter Applicants

Unit Testing Details:

1. **Test ID: UT-UC22-01**
   Validate Filtering Logic
   - **Attribute:** Applicant Data and Filter Criteria
   - **Value:** Filter Criteria: Experience: 3+ years, Skills: Python; Applicant: Experience: 4 years, Skills: Python
   - **Expected Result:** The system identifies the applicant as matching the criteria.
   - **Actual Result:** Applicant identified successfully.
   - **Result:** Pass
2. **Test ID: UT-UC22-02**
   Handle No Matching Applicants
   - **Attribute:** Applicant Data and Filter Criteria
   - **Value:** Filter Criteria: Experience: 10+ years, Skills: Java; Applicant: No matching applicants
   - **Expected Result:** The system identifies no matching applicants and returns a "No matching applicants" result.
   - **Actual Result:** Correct result returned.
   - **Result:** Pass
3. **Test ID: UT-UC22-03**
   Validate Combination of Multiple Criteria
   - **Attribute:** Applicant Data and Filter Criteria
   - **Value:** Filter Criteria: Experience: 5+ years, Skills: Python, React; Applicant: Experience: 6 years, Skills: Python, React
   - **Expected Result:** The system identifies the applicant as matching all criteria.
   - **Actual Result:** Applicant identified successfully.
   - **Result:** Pass
4. **Test ID: UT-UC22-04**
   Simulate Filtering Error
   - **Attribute:** Filter Logic Execution
   - **Value:** Simulated error in filter execution (e.g., null pointer exception)
   - **Expected Result:** The system logs the error and displays a message: "Unable to filter applicants. Please try again later."
   - **Actual Result:** Correct error logged and message displayed.
   - **Result:** Pass
5. **Test ID: UT-UC22-05**
   Validate Database Query for Filtering
   - **Attribute:** Filter Query Execution
   - **Value:** Query: SELECT * FROM applicants WHERE experience >= 3 AND skills LIKE '%Python%'
   - **Expected Result:** The system executes the query successfully and retrieves matching applicants.

- **Actual Result:** Query executed successfully and data retrieved.
- **Result:** Pass

Unit Testing Details for UC28: Apply for Jobs

Unit Testing Details:

1. **Test ID: UT-UC28-01**
Validate Application Submission Logic

   - **Attribute:** Job and Profile Data
   - **Value:** Job: Software Engineer; Profile: Complete (Name, Skills, Experience)
   - **Expected Result:** The system processes and saves the application without errors.
   - **Actual Result:** Application processed and saved successfully.
   - **Result:** Pass

2. **Test ID: UT-UC28-02**
Handle Incomplete Profile Validation

   - **Attribute:** Profile Status
   - **Value:** Incomplete Profile (Missing Experience Section)
   - **Expected Result:** The system prevents the application submission and prompts the user to complete their profile.
   - **Actual Result:** Correct prompt displayed.
   - **Result:** Pass

3. **Test ID: UT-UC28-03**
Simulate Network Issues During Submission

   - **Attribute:** Network Status
   - **Value:** Disconnected during submission
   - **Expected Result:** The system displays an error message: “Network error. Please check your connection and try again.”
   - **Actual Result:** Correct error message displayed.
   - **Result:** Pass

4. **Test ID: UT-UC28-04**
Validate Confirmation Notification Logic

   - **Attribute:** Notification Content
   - **Value:** "Your application for the Software Engineer position has been submitted successfully."
   - **Expected Result:** The system generates and sends the notification to the job seeker.
   - **Actual Result:** Notification generated and sent successfully.
   - **Result:** Pass

5. **Test ID: UT-UC28-05**
Validate Database Entry for Application

   - **Attribute:** Application Record
   - **Value:** Job: Software Engineer, Applicant: John Doe, Status: Submitted
   - **Expected Result:** The application is saved in the database successfully.

- **Actual Result:** Application record saved successfully.
- **Result:** Pass

5.2 Functional Testing (FT)

Functional Testing 1: User Sign Up

Objective:
To validate that the system correctly registers new users and handles alternate flows.

1. **Testcase ID:** FT-UC1-01
Title: Sign Up with Valid Details
Description: Validate that the system registers new users with valid details.
Preconditions:

- The system must be running.
- The "Sign Up" page must be accessible.

Test Steps:

1. Navigate to the "Sign Up" page.
2. Enter valid details: Name: John Doe, Email: john@example.com, Password: StrongPass123.
3. Click the "Submit" button.

Expected Results:

- The user is registered successfully.
- The user's data is stored in the database.

Actual Result: User registered successfully, and database updated.
Status: Pass
Remarks: None

2. **Testcase ID:** FT-UC1-02
Title: Sign Up with Weak Password
Description: Validate that the system rejects a weak password during user registration.
Preconditions:

- The system must be running.
- The "Sign Up" page must be accessible.

Test Steps:

1. Navigate to the "Sign Up" page.
2. Enter valid details except for a weak password: Password: 123.
3. Click the "Submit" button.

Expected Results:

- The system shows a message: “Password is weak.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure the password validation criteria are robust.

3. **Testcase ID:** FT-UC1-03
Title: Sign Up with Invalid Email
Description: Validate that the system rejects an invalid email address during user registration.
Preconditions:

- The system must be running.
- The "Sign Up" page must be accessible.

Test Steps:

1. Navigate to the "Sign Up" page.
2. Enter valid details except for an invalid email: Email: invalidemail.com.
3. Click the "Submit" button.

Expected Results:

- The system shows a message: “Invalid Email Address.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: None

4. **Testcase ID:** FT-UC1-04
Title: Sign Up with Mismatched Passwords
Description: Validate that the system rejects mismatched passwords during user registration.
Preconditions:

- The system must be running.
- The "Sign Up" page must be accessible.

Test Steps:

1. Navigate to the "Sign Up" page.
2. Enter valid details except for mismatched passwords: Password: abc123, Confirm Password: xyz456.
3. Click the "Submit" button.

Expected Results:

- The system shows a message: “Passwords don’t match.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: None

Functional Testing 2: User Login

Objective:
To validate that users can log into the system using valid credentials and handle alternate flows effectively.

1. **Testcase ID:** FT-UC2-01
Title: Login with Valid Credentials
Description: Validate that the system allows users to log in with valid credentials.
Preconditions:

- The user must be registered in the system with valid credentials.
- The "Login" page must be accessible.

Test Steps:

1. Navigate to the "Login" page.
2. Enter valid credentials: Email: john@example.com, Password: StrongPass123.
3. Click the "Login" button.

Expected Results:

- The user is logged in successfully.
- The user is redirected to the dashboard.

Actual Result: User logged in successfully and redirected to the dashboard.
Status: Pass
Remarks: Ensure session tokens are correctly created after login.

2. **Testcase ID:** FT-UC2-02
Title: Login with Incorrect Credentials
Description: Validate that the system rejects incorrect credentials during login.
Preconditions:

- The user must be registered in the system with valid credentials.
- The "Login" page must be accessible.

Test Steps:

1. Navigate to the "Login" page.
2. Enter invalid credentials: Email: wrongemail@example.com, Password: abc123.
3. Click the "Login" button.

Expected Results:

- The system displays an error message: “Incorrect email or password.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC2-03
Title: Login with Empty Fields
Description: Validate that the system prompts users to fill in the required fields when they attempt to log in with empty email and password fields.
Preconditions:

- The "Login" page must be accessible.

Test Steps:

1. Navigate to the "Login" page.
2. Leave both the Email and Password fields empty.
3. Click the "Login" button.

Expected Results:

- The system displays an error message: "Email and password are required."

Actual Result: Correct message displayed.
Status: Pass
Remarks: None

4. **Testcase ID:** FT-UC2-04
Title: Login with System Error
Description: Validate that the system handles backend issues gracefully during the login process.
Preconditions:

- Simulate a database connection issue.

Test Steps:

1. Navigate to the "Login" page.
2. Enter valid credentials: Email: john@example.com, Password: StrongPass123.
3. Simulate a database connection failure and click the "Login" button.

Expected Results:

- The system displays an error message: "Unable to process request. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logging is implemented for debugging.

Functional Testing 3: Add Applicant Profile

Objective:
To validate that the system allows users to add their profile and handles alternate flows effectively.

1. **Testcase ID:** FT-UC3-01
Title: Add Profile with All Details
Description: Validate that the system saves the user profile successfully when all required fields are provided.
Preconditions:

   - The user must be logged in.
   - The "Add Profile" page must be accessible.

Test Steps:

1. Navigate to the "Add Profile" page.
2. Fill in all required fields:
   1. Name: John Doe
   2. Email: john@example.com
   3. Skills: Python
   4. Work Experience: 3 years
3. Click the "Save" button.

Expected Results:

- The profile is saved successfully.
- The system updates the database with the user's profile data.

Actual Result: Profile saved successfully, and the database updated.
Status: Pass
Remarks: Ensure data validation checks are performed before saving.

2. **Testcase ID:** FT-UC3-02
Title: Add Profile with Missing Required Fields
Description: Validate that the system prompts the user to fill in the required fields when they are missing.
Preconditions:

   - The user must be logged in.
   - The "Add Profile" page must be accessible.

Test Steps:

1. Navigate to the "Add Profile" page.
2. Leave required fields (e.g., Name and Skills) empty.
3. Click the "Save" button.

Expected Results:

- The system displays a message prompting the user to complete the required fields.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure error messages are clear and concise.

3. **Testcase ID:** FT-UC3-03
Title: Add Profile with Invalid Data
Description: Validate that the system rejects invalid data, such as negative values for work experience.
Preconditions:

- The user must be logged in.
- The "Add Profile" page must be accessible.

Test Steps:

1. Navigate to the "Add Profile" page.
2. Enter invalid data:
    1. Work Experience: -3 years (negative value)
3. Click the "Save" button.

Expected Results:

- The system displays an error message: “Invalid work experience.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure validation logic prevents the saving of invalid data.

4. **Testcase ID:** FT-UC3-04
Title: System Failure During Profile Saving
Description: Validate that the system handles failures gracefully during the profile saving process.
Preconditions:

- The user must be logged in.
- Simulate a database connection issue.

Test Steps:

1. Navigate to the "Add Profile" page.
2. Fill in all required fields:
    1. Name: John Doe
    2. Email: john@example.com
    3. Skills: Python

        4. Work Experience: 3 years

    3. Click the "Save" button.

    4. Simulate a database connection issue.

Expected Results:

- The system displays an error message: “Unable to save profile. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure the error is logged for debugging purposes.

Functional Testing 4: Search Jobs

Objective:
To validate that the system allows users to search for jobs using specific filters and handles alternate flows effectively.

1. **Testcase ID:** FT-UC6-01
Title: Search with Valid Filters
Description: Validate that the system displays job postings matching the applied filters.
Preconditions:

- The user must be logged in.
- The system should allow the user to navigate to the job search functionality.

Test Steps:

    1. Navigate to the "Jobs" section.

    2. Apply valid filters:

        1. Job Type: Full-Time

        2. Salary Range: $50,000–$70,000

    3. Click the "Search" button.

Expected Results:

- The system displays job postings that match the applied filters.

Actual Result: Relevant job postings displayed.
Status: Pass
Remarks: Ensure filter combinations are applied correctly for search queries.

2. **Testcase ID:** FT-UC6-02
Title: Search with No Matching Jobs
Description: Validate that the system handles cases where no job postings match the applied filters.
Preconditions:

- ○ The user must be logged in.
- ○ The system should allow the user to navigate to the job search functionality.

Test Steps:

1. Navigate to the "Jobs" section.
2. Apply filters that do not match any job postings:
    1. Job Type: Part-Time
    2. Salary Range: $100,000–$120,000
3. Click the "Search" button.

Expected Results:

- ○ The system displays a message: “No jobs found.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure the system informs users clearly when no results are found.

3. **Testcase ID:** FT-UC6-03
Title: Simulate System Error While Searching
Description: Validate that the system handles backend errors gracefully during job searches.
Preconditions:

- ○ The user must be logged in.
- ○ The system should allow job searches to proceed.

Test Steps:

1. Navigate to the "Jobs" section.
2. Apply any filter:
    1. Job Type: Remote
3. Simulate a database connection issue.
4. Click the "Search" button.

Expected Results:

- ○ The system displays an error message: “Unable to fetch job search results. Please try again later.”

Actual Result: Correct error message displayed.

Status: Pass
Remarks: Ensure error handling mechanisms are robust, and errors are logged for debugging.

Functional Testing 5: Take Test

Objective:
To validate that the system allows job seekers to take a test, handles incomplete tests, and stores the test results correctly.

1. **Testcase ID:** FT-UC8-01
Title: Take Test and Submit Within Time
Description: Validate that the system allows job seekers to complete and submit a test within the allotted time.
Preconditions:

   - The job seeker must be logged in.
   - The test invitation must be available on the user's dashboard.

Test Steps:

   1. Log in to the system as a job seeker.
   2. Open the test invitation from the dashboard.
   3. Complete all questions within the allotted time.
   4. Click the "Submit" button.

Expected Results:

   - The system stores the test results successfully.

Actual Result: Test results stored successfully.
Status: Pass
Remarks: Ensure the system validates the timer and stores results in the database.

2. **Testcase ID:** FT-UC8-02
Title: Incomplete Test
Description: Validate that the system handles incomplete tests and sends reminder notifications to the user.
Preconditions:

   - The job seeker must be logged in.
   - The test invitation must be available on the user's dashboard.

Test Steps:

   1. Log in to the system as a job seeker.
   2. Open the test invitation from the dashboard.
   3. Leave the test incomplete and exit without submitting.

Expected Results:

   - The system sends a reminder notification to the user.

Actual Result: Reminder notification sent successfully.

Status: Pass
Remarks: Verify that reminders are logged and sent at the appropriate time.

3. **Testcase ID:** FT-UC8-03
Title: Simulate System Error During Submission
Description: Validate that the system displays an appropriate error message if a system error occurs during test submission.
Preconditions:

- ○ The job seeker must be logged in.
- ○ Simulate a network timeout or database error.

Test Steps:

1. Log in to the system as a job seeker.
2. Open the test invitation from the dashboard.
3. Complete all questions and click the "Submit" button.
4. Simulate a system error (e.g., network timeout or database connection failure).

Expected Results:

- ○ The system displays an error message: "Unable to submit the test. Please try again."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are generated for debugging.

4. **Testcase ID:** FT-UC8-04
Title: Test Notification Handling
Description: Validate that the system displays the test notification on the user's dashboard.
Preconditions:

- ○ The job seeker must be logged in.
- ○ A test invitation must be generated by the system.

Test Steps:

1. Log in to the system as a job seeker.
2. Check the dashboard for a new test notification.

Expected Results:

- ○ The system displays the test notification in the user's dashboard.

Actual Result: Test notification displayed successfully.
Status: Pass
Remarks: Verify that notifications are delivered in real-time.

Functional Testing 6: Update Profile

Objective:
To validate that the system allows job seekers to update their profile and handles alternate flows and exceptions appropriately.

1. **Testcase ID:** FT-UC11-01
Title: View Current Profile
Description: Validate that the system displays the user's current profile for editing.
Preconditions:

- The user must be logged in.
- A profile must already exist in the system.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Profile" section.
3. Click the "Edit" button to view the current profile.

Expected Results:

- The system displays the current profile details for editing.

Actual Result: Profile displayed successfully.
Status: Pass
Remarks: Ensure profile data loads correctly without delays.

2. **Testcase ID:** FT-UC11-02
Title: Update Profile with Valid Data
Description: Validate that the system saves the updated profile when all fields are correctly filled.
Preconditions:

- The user must be logged in.
- A profile must already exist in the system.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Profile" section.
3. Click the "Edit" button and update the following fields:
    1. Skills: Python, React
    2. Work Experience: 5 years
4. Click the "Save" button.

Expected Results:

- The system saves the updated profile successfully.

Actual Result: Profile updated and saved successfully.
Status: Pass
Remarks: Verify database is updated with the new profile information.

3. **Testcase ID:** FT-UC11-03
Title: Redirect to Login for Unauthenticated Users
Description: Validate that the system redirects logged-out users to the login page when they attempt to update their profile.
Preconditions:

- The user must be logged out.

Test Steps:

1. Attempt to access the "Profile" section without logging in.
2. Click the "Edit" button to update the profile.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

4. **Testcase ID:** FT-UC11-04
Title: Handle Invalid Data During Update
Description: Validate that the system rejects invalid profile data during an update and provides appropriate error messages.
Preconditions:

- The user must be logged in.
- A profile must already exist in the system.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Profile" section.
3. Click the "Edit" button and enter the following invalid data:
    1. Email: invalidemail
    2. Work Experience: -3 years
4. Click the "Save" button.

Expected Results:

- The system displays an error message: “Invalid data. Please correct the highlighted fields.”

Actual Result: Correct error message displayed.

Status: Pass
Remarks: Ensure validation logic for profile fields is robust.

5. **Testcase ID:** FT-UC11-05
Title: Simulate System Error While Updating Profile
Description: Validate that the system handles backend errors gracefully during the profile update process.
Preconditions:

- The user must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Profile" section.
3. Click the "Edit" button and update any field (e.g., Skills: Python, React).
4. Simulate a database connection timeout and click the "Save" button.

Expected Results:

- The system displays an error message: “Unable to update profile. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure errors are logged for debugging purposes.

Functional Testing 7: Company Reviews

Objective:
To validate that the system allows users to provide reviews and feedback for companies, handles invalid ratings appropriately, and stores the reviews correctly.

1. **Testcase ID:** FT-UC12-01
Title: Submit Review with Valid Data
Description: Validate that the system saves a review when valid data is provided.
Preconditions:

- The user must be logged in.
- The company’s profile must be accessible.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company’s profile.
3. Enter the following review details:
    1. Rating: 4 stars
    2. Feedback: "Great company with excellent work culture."

4. Click the "Submit Review" button.

Expected Results:

- The system stores the review and displays it under the company’s profile.

Actual Result: Review submitted and displayed successfully.
Status: Pass
Remarks: Verify that reviews are stored in the database and displayed in real-time.

2. **Testcase ID:** FT-UC12-02
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to submit a review.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the company’s profile without logging in.
2. Click the "Submit Review" button.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC12-03
Title: Handle Missing Review Details
Description: Validate that the system prompts the user to fill in feedback when it is missing.
Preconditions:

- The user must be logged in.

Test Steps:

1. Log in to the system as a user.

2. Navigate to the company’s profile.
3. Enter the following review details:
    1. Rating: 5 stars
    2. Feedback: Empty
4. Click the "Submit Review" button.

Expected Results:

- The system displays a message prompting the user to provide feedback.

Actual Result: Correct message displayed.

Status: Pass
Remarks: Ensure feedback is a mandatory field.

4. **Testcase ID:** FT-UC12-04
Title: Handle Invalid Rating (Greater than 5)
Description: Validate that the system rejects ratings greater than 5.
Preconditions:

- The user must be logged in.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company's profile.
3. Enter the following review details:
    1. Rating: 6 stars
    2. Feedback: "Good Company."
4. Click the "Submit Review" button.

Expected Results:

- The system displays an error message: "Rating must be between 1 and 5."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: None

5. **Testcase ID:** FT-UC12-05
Title: Handle Invalid Rating (Less than 0)
Description: Validate that the system rejects ratings less than 0.
Preconditions:

- The user must be logged in.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company's profile.
3. Enter the following review details:
    1. Rating: -1 stars
    2. Feedback: "Bad company."
4. Click the "Submit Review" button.

Expected Results:

- The system displays an error message: "Rating must be between 1 and 5."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: None

6. **Testcase ID:** FT-UC12-06
Title: Handle Invalid Rating (0)
Description: Validate that the system rejects a rating of 0.
Preconditions:

- The user must be logged in.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company's profile.
3. Enter the following review details:
    1. Rating: 0 stars
    2. Feedback: "Neutral review."
4. Click the "Submit Review" button.

Expected Results:

- The system displays an error message: "Rating must be between 1 and 5."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: None

7. **Testcase ID:** FT-UC12-07
Title: Simulate System Error During Review Submission
Description: Validate that the system handles errors during review submission gracefully.
Preconditions:

- The user must be logged in.
- Simulate a network timeout or database connection issue.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company's profile.
3. Enter the following review details:
    1. Rating: 4 stars
    2. Feedback: "Great company."

4. Simulate a network or database error and click the "Submit Review" button.

Expected Results:

- The system displays an error message: "Unable to submit review. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Verify error logging for debugging purposes.

8. **Testcase ID:** FT-UC12-08
Title: Validate Display of Submitted Review
Description: Validate that the system displays submitted reviews accurately under the company's profile.
Preconditions:

- A review must already be submitted and stored in the database.

Test Steps:

1. Log in to the system as a user.
2. Navigate to the company's profile.
3. View the "Reviews" section to verify the review display.

Expected Results:

- The system displays the submitted review accurately.

Actual Result: Review displayed successfully.
Status: Pass
Remarks: Ensure reviews appear in chronological order.

Functional Testing 8: Discard Irrelevant CVs

Objective:
To validate that the system filters CVs based on the recruiter's specified criteria and handles exceptions and alternate flows appropriately.

1. **Testcase ID:** FT-UC13-01
Title: Filter CVs Based on Job Criteria
Description: Validate that the system discards CVs that do not meet the recruiter's job criteria.
Preconditions:

- A job posting must exist with specified criteria.
- CVs must be submitted for the job posting.

Test Steps:

1. Log in as a recruiter.
2. View the list of CVs submitted for a job.
3. Set job criteria:

        1. Required Experience: 3+ years
        2. Required Skills: Python, React
4. Run the CV filtering process.

Expected Results:

- ○ The system discards the CV if it does not meet the experience criteria (e.g., CV Experience: 2 years, Skills: Python).

Actual Result: CV discarded successfully.
Status: Pass
Remarks: Ensure filtering logic accurately checks criteria against CV data.

2. **Testcase ID:** FT-UC13-02
Title: Retain Relevant CVs
Description: Validate that the system retains CVs that meet all job criteria.
Preconditions:

- ○ A job posting must exist with specified criteria.
- ○ Relevant CVs must be submitted for the job posting.

Test Steps:

1. Log in as a recruiter.
2. View the list of CVs submitted for a job.
3. Set job criteria:
    1. Required Experience: 3+ years
    2. Required Skills: Python, React
4. Run the CV filtering process.

Expected Results:

- ○ The system retains CVs that meet all the criteria (e.g., CV Experience: 4 years, Skills: Python, React).

Actual Result: CV retained successfully.
Status: Pass
Remarks: Verify retained CVs are displayed to the recruiter.

3. **Testcase ID:** FT-UC13-03
Title: Handle No Applications
Description: Validate that the system notifies the recruiter if no CVs are submitted for a job posting.
Preconditions:

- ○ A job posting must exist with no CVs submitted.

Test Steps:

1. Log in as a recruiter.
2. View the list of CVs for the job posting.

Expected Results:

- ○ The system notifies the recruiter: "No CVs received for this job posting."

Actual Result: Correct notification displayed.
Status: Pass
Remarks: Ensure clear notifications for empty CV lists.

4. **Testcase ID:** FT-UC13-04
Title: Simulate System Error During Filtering
Description: Validate that the system handles errors during CV filtering gracefully.
Preconditions:

- ○ A job posting must exist with CVs submitted.
- ○ Simulate a database connection issue.

Test Steps:

1. Log in as a recruiter.
2. View the list of CVs for the job posting.
3. Simulate a system error during the filtering process.

Expected Results:

- ○ The system displays an error message: "Unable to filter CVs. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure errors are logged for debugging.

5. **Testcase ID:** FT-UC13-05
Title: Filter CVs with Missing Data
Description: Validate that the system discards CVs with incomplete or missing data and notifies the recruiter.
Preconditions:

- ○ A job posting must exist with submitted CVs containing incomplete data.

Test Steps:

1. Log in as a recruiter.

2. View the list of CVs for the job posting.

3. Run the CV filtering process with the following data:

1. CV missing the Experience field.

Expected Results:

- ○ The system discards incomplete CVs and notifies the recruiter of the missing data.

Actual Result: CV discarded successfully, and notification displayed.
Status: Pass
Remarks: Ensure recruiters are informed about missing fields in rejected CVs.

6. **Testcase ID:** FT-UC13-06
Title: Handle Invalid Criteria
Description: Validate that the system prompts the recruiter to provide valid criteria when invalid data is entered.
Preconditions:

- ○ A job posting must exist.

Test Steps:

1. Log in as a recruiter.
2. Set invalid job criteria:
    1. Negative years of experience (-3 years).
    2. Mismatched skills (e.g., nonexistent technologies).
3. Attempt to filter CVs with invalid criteria.

Expected Results:

- ○ The system prompts the recruiter to provide valid criteria.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure validation logic for criteria fields is robust.

Functional Testing 9: Notify Applicant

Objective:
To validate that the system sends notifications to applicants regarding their application status and handles invalid emails and system errors appropriately.

1. **Testcase ID:** FT-UC14-01
Title: Send Notification for Accepted Application
Description: Validate that the system sends a notification when an application is accepted.
Preconditions:

- ○ The recruiter must mark the application as "Accepted."
- ○ The applicant's email address must be valid.

Test Steps:

1. Log in as a recruiter.
2. Change the application status to "Accepted."
3. Trigger the notification process.

Expected Results:

- The system sends a notification to the applicant: “Your application has been accepted.”

Actual Result: Notification sent successfully.
Status: Pass
Remarks: Ensure the email delivery is logged.

2. **Testcase ID:** FT-UC14-02
Title: Send Notification for Rejected Application
Description: Validate that the system sends a notification when an application is rejected.
Preconditions:

- The recruiter must mark the application as "Rejected."
- The applicant's email address must be valid.

Test Steps:

1. Log in as a recruiter.
2. Change the application status to "Rejected."
3. Trigger the notification process.

Expected Results:

- The system sends a notification to the applicant: “Your application has been rejected.”

Actual Result: Notification sent successfully.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC14-03
Title: Handle Invalid Email Address
Description: Validate that the system handles invalid email addresses appropriately during notification sending.
Preconditions:

- The applicant’s email address must be invalid (e.g., invalidemail.com).

Test Steps:

1. Log in as a recruiter.
2. Change the application status to "Accepted" or "Rejected."
3. Trigger the notification process.

Expected Results:

- The system logs the failure to notify and flags the invalid email.

Actual Result: Correct log entry created and email flagged.
Status: Pass
Remarks: Ensure flagged emails are displayed in error logs.

4. **Testcase ID:** FT-UC14-04
Title: Simulate System Error While Sending Notification
Description: Validate that the system handles system errors gracefully during notification sending.
Preconditions:

- Simulate a network timeout or system failure.

Test Steps:

1. Log in as a recruiter.
2. Change the application status to "Accepted" or "Rejected."
3. Simulate a network timeout and trigger the notification process.

Expected Results:

- The system logs the error: “Unable to send notification. Please retry.”

Actual Result: Correct error log created.
Status: Pass
Remarks: Ensure error logs are clear and retried notifications are handled.

5. **Testcase ID:** FT-UC14-05
Title: Verify Notification Content
Description: Validate that the notification content is accurate and personalized for the applicant.
Preconditions:

- The application status must be "Accepted."

Test Steps:

1. Log in as a recruiter.
2. Change the application status to "Accepted."
3. Trigger the notification process.

Expected Results:

- The notification content is accurate and personalized, e.g., “Congratulations! Your application for the Software Developer position has been accepted.”

Actual Result: Notification content verified successfully.
Status: Pass
Remarks: Ensure message templates are properly formatted and customizable.

6. **Testcase ID:** FT-UC14-06
Title: Send Notification to Multiple Applicants
Description: Validate that the system sends notifications to multiple applicants based on their respective statuses.

Preconditions:

- Multiple applicants must exist for a job posting.
- The recruiter must assign different statuses to the applicants (e.g., Accepted, Rejected).

Test Steps:

1. Log in as a recruiter.
2. Change the application statuses for multiple applicants (e.g., Accepted: 3, Rejected: 2).
3. Trigger the batch notification process.

Expected Results:

- Notifications are sent to all applicants with their respective statuses.

Actual Result: Notifications sent successfully to all applicants.
Status: Pass
Remarks: Ensure batch processing handles large volumes of notifications without delays.

Functional Testing 10: Register Company

Objective:
To validate that the system allows recruiters to register their company, handles alternate flows for unauthenticated users, and manages exceptions effectively.

1. **Testcase ID:** FT-UC15-01
Title: Register Company with Valid Details
Description: Validate that the system registers a company when all required fields are provided correctly.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Register Company" section.
3. Enter the following details:
    1. Name: Tech Solutions Inc.
    2. Address: 123 Main St
    3. Email: hr@techsolutions.com
4. Click the "Submit" button.

Expected Results:

- The system registers the company successfully, and the recruiter can start posting jobs.

Actual Result: Company registered successfully.
Status: Pass
Remarks: Ensure company details are saved in the database.

2. **Testcase ID:** FT-UC15-02
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to register a company.
Preconditions:

- ○ The user must not be logged in.

Test Steps:

1. Attempt to access the "Register Company" section without logging in.
2. Fill in any company details.
3. Click the "Submit" button.

Expected Results:

- ○ The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC15-03
Title: Handle Missing Required Fields in Registration
Description: Validate that the system prompts the recruiter to fill in required fields if any are missing.
Preconditions:

- ○ The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Register Company" section.
3. Leave one or more required fields blank (e.g., Name or Email).
4. Click the "Submit" button.

Expected Results:

- ○ The system displays a message prompting the recruiter to fill in the missing fields.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure all required fields are marked and validated correctly.

4. **Testcase ID:** FT-UC15-04
Title: Simulate System Error While Registering Company
Description: Validate that the system handles backend errors gracefully during the company registration process.

Preconditions:

- The recruiter must be logged in.
- Simulate a database connection issue.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Register Company" section.
3. Enter all required company details.
4. Simulate a database connection timeout and click the "Submit" button.

Expected Results:

- The system displays an error message: “Unable to register company. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

Functional Testing 11: Create Company Portfolio

Objective:
To validate that the system allows recruiters to create and submit a company portfolio, handles incomplete portfolios appropriately, and manages exceptions effectively.

1. **Testcase ID:** FT-UC16-01
Title: Create Portfolio with Valid Details
Description: Validate that the system saves the portfolio when all required fields are filled with valid data.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Portfolio" section.
3. Enter the following details:
    1. Mission: "Empowering Innovation"
    2. Vision: "To be the leader in tech"
    3. Job Offerings: "Software Engineer, Data Scientist"
4. Click the "Save" button.

Expected Results:

- The system saves the portfolio and makes it visible to job seekers.

Actual Result: Portfolio created and displayed successfully.
Status: Pass
Remarks: Verify that all portfolio details are stored in the database.

2. **Testcase ID:** FT-UC16-02
Title: Handle Incomplete Portfolio Submission
Description: Validate that the system prompts the recruiter to fill in missing fields when submitting an incomplete portfolio.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Portfolio" section.
3. Enter incomplete data:
    1. Mission: Empty
    2. Vision: "To be the leader in tech"
    3. Job Offerings: "Software Engineer"
4. Click the "Save" button.

Expected Results:

- The system displays a message prompting the recruiter to fill in the missing mission statement.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure required fields are validated properly.

3. **Testcase ID:** FT-UC16-03
Title: Simulate System Error While Saving Portfolio
Description: Validate that the system handles backend errors gracefully during the portfolio saving process.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Portfolio" section.
3. Enter valid portfolio details:

        1. Mission: "Empowering Innovation"
        2. Vision: "To be the leader in tech"
        3. Job Offerings: "Software Engineer, Data Scientist"
    4. Simulate a database connection timeout and click the "Save" button.

Expected Results:

- ○ The system displays an error message: "Unable to save portfolio. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure errors are logged for debugging purposes.

4. **Testcase ID:** FT-UC16-04
Title: Validate Display of Saved Portfolio
Description: Validate that the system displays the saved portfolio accurately to job seekers.
Preconditions:

- ○ The recruiter must have successfully saved a portfolio.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "View Portfolio" section.
3. Verify the displayed portfolio details:
    1. Mission: "Empowering Innovation"
    2. Vision: "To be the leader in tech"
    3. Job Offerings: "Software Engineer, Data Scientist"

Expected Results:

- ○ The saved portfolio is visible to job seekers.

Actual Result: Portfolio displayed successfully.
Status: Pass
Remarks: Ensure all portfolio fields are displayed correctly without truncation.

5. **Testcase ID:** FT-UC16-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to create a portfolio.
Preconditions:

- ○ The user must not be logged in.

Test Steps:

1. Attempt to access the "Create Portfolio" section without logging in.
2. Fill in any portfolio details.
3. Click the "Save" button.

Expected Results:

- The system redirects the recruiter to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

Functional Testing 12: Create Job

Objective:
To validate that the system allows recruiters to create job postings, handles alternate flows for unregistered companies, and manages exceptions during job posting.

1. **Testcase ID:** FT-UC17-01
Title: Create Job with Valid Details
Description: Validate that the system allows recruiters to successfully create a job posting when all required fields are provided.
Preconditions:

- The recruiter must be logged in.
- The recruiter must have registered a company.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Enter the following job details:
   1. Title: Software Engineer
   2. Description: "Develop and maintain software"
   3. Location: Remote
   4. Salary: $70,000–$90,000
4. Click the "Post Job" button.

Expected Results:

- The system posts the job successfully and makes it visible to job seekers.

Actual Result: Job posted successfully.
Status: Pass

Remarks: Ensure job postings are stored in the database and visible on the job board.

2. **Testcase ID:** FT-UC17-02
Title: Prompt for Unregistered Company
Description: Validate that the system prompts recruiters to register a company if they attempt to create a job without a registered company.
Preconditions:

  - The recruiter must be logged in.
  - The recruiter must not have registered a company.

Test Steps:

  1. Log in to the system as a recruiter.
  2. Navigate to the "Create Job" section.
  3. Attempt to post a job without a registered company.

Expected Results:

  - The system displays a prompt: "Please register your company before posting a job."

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC17-03
Title: Handle Missing Required Fields in Job Details
Description: Validate that the system prompts the recruiter to fill in missing required fields when submitting incomplete job details.
Preconditions:

  - The recruiter must be logged in.
  - The recruiter must have registered a company.

Test Steps:

  1. Log in to the system as a recruiter.
  2. Navigate to the "Create Job" section.
  3. Enter incomplete job details:
     1. Title: Empty
     2. Description: "Develop and maintain software"
     3. Location: Remote
  4. Click the "Post Job" button.

Expected Results:

  - The system displays a prompt: "Please fill in the missing title field."

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure all required fields are validated properly.

4. **Testcase ID:** FT-UC17-04
Title: Simulate System Error While Posting Job
Description: Validate that the system handles backend errors gracefully during job posting.
Preconditions:

- The recruiter must be logged in.
- The recruiter must have registered a company.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Enter valid job details:
    1. Title: Software Engineer
    2. Description: "Develop and maintain software"
    3. Location: Remote
    4. Salary: $70,000–$90,000
4. Simulate a database connection issue and click the "Post Job" button.

Expected Results:

- The system displays an error message: “Unable to post the job. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure errors are logged for debugging purposes.

5. **Testcase ID:** FT-UC17-05
Title: Verify Display of Posted Job
Description: Validate that the system displays the posted job accurately to job seekers.
Preconditions:

- The recruiter must have successfully posted a job.

Test Steps:

1. Log in to the system as a recruiter.

2. Navigate to the "View Jobs" section.
3. Verify the displayed job details:
    1. Title: Software Engineer
    2. Location: Remote
    3. Salary: $70,000–$90,000

Expected Results:

- The posted job is visible to job seekers with accurate details.

Actual Result: Job displayed successfully.
Status: Pass
Remarks: Ensure job postings appear correctly without missing fields.

6. **Testcase ID:** FT-UC17-06
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to create a job.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Create Job" section without logging in.
2. Fill in any job details.
3. Click the "Post Job" button.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

Functional Testing 13: Set Job Criteria

Objective:
To validate that the system allows recruiters to set criteria for job postings, ensures only relevant applications are accepted, and handles alternate flows and exceptions effectively.

1. **Testcase ID:** FT-UC18-01
Title: Set Valid Criteria for Job Posting
Description: Validate that the system saves job criteria and ensures only relevant applications are accepted.
Preconditions:

- The recruiter must be logged in.

- The recruiter must have registered a company.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Enter job criteria:
    1. Required Qualifications: Bachelor’s Degree in Computer Science
    2. Required Skills: Python, React
    3. Experience: 3+ years
4. Click the "Save Criteria" button.

Expected Results:

- The system saves the job post with the specified criteria, and only relevant applications are accepted.

Actual Result: Job criteria saved successfully, and irrelevant applications are filtered.
Status: Pass
Remarks: Ensure criteria logic is applied consistently during filtering.

2. **Testcase ID:** FT-UC18-02
Title: Set No Criteria for Job Posting
Description: Validate that the system allows job postings with no specific criteria and accepts all applications.
Preconditions:

- The recruiter must be logged in.
- The recruiter must have registered a company.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Leave the job criteria fields empty.
4. Click the "Save Criteria" button.

Expected Results:

- The system saves the job post and allows all applicants to apply.

Actual Result: Job post saved successfully, and all applications accepted.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC18-03
Title: Validate Criteria Logic for Filtering Applications
Description: Validate that the system filters applications correctly based on job criteria.
Preconditions:

- The recruiter must be logged in.
- The recruiter must have set criteria for a job post.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section for a job post with criteria:
    1. Criteria: Bachelor's Degree, Skills: Python
3. View an applicant's details:
    1. Applicant: Diploma, Skills: Python
4. Verify if the application is filtered.

Expected Results:

- The system discards the applicant as they do not meet the qualifications.

Actual Result: Applicant discarded successfully.
Status: Pass
Remarks: Ensure filtering criteria match the job post requirements accurately.

4. **Testcase ID:** FT-UC18-04
Title: Handle Missing Required Fields in Criteria
Description: Validate that the system prompts recruiters to complete required fields when saving criteria.
Preconditions:

- The recruiter must be logged in.
- The recruiter must be creating or editing a job post.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Leave the "Required Skills" field empty.
4. Click the "Save Criteria" button.

Expected Results:

- The system displays a message prompting the recruiter to complete all required fields.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure all required fields are validated before saving criteria.

5. **Testcase ID:** FT-UC18-05
Title: Simulate System Error While Setting Criteria
Description: Validate that the system handles backend errors gracefully during criteria saving.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Job" section.
3. Enter valid job criteria:
    1. Required Qualifications: Bachelor's Degree in Computer Science
    2. Required Skills: Python, React
    3. Experience: 3+ years
4. Simulate a database connection issue and click the "Save Criteria" button.

Expected Results:

- The system displays an error message: "Unable to save criteria. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

Functional Testing 14: Create Job Test

Objective:
To validate that the system allows recruiters to create tests for job applicants with valid time limits, supports MCQs only, handles alternate flows for missing job postings, and manages exceptions effectively.

1. **Testcase ID:** FT-UC19-01
Title: Create Test with Valid MCQs and Time
Description: Validate that the system allows recruiters to create a test with valid MCQs and a time limit.
Preconditions:

- The recruiter must be logged in.
- A job posting must exist.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Test" section.

3. Enter the following details:
    1. Test Title: "Technical Skills Test"
    2. Questions: "What is Python? [Options: A, B, C, D]"
    3. Time Limit: 60 minutes
4. Click the "Save Test" button.

Expected Results:

- ○ The system saves the test successfully and displays a popup: “Test created successfully.”

Actual Result: Test created successfully and popup displayed.
Status: Pass
Remarks: Verify that all test details are stored in the database.

2. **Testcase ID:** FT-UC19-02
Title: Handle Invalid Time Limit
Description: Validate that the system rejects invalid time limits for test creation.
Preconditions:

- ○ The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Test" section.
3. Enter the following test details:
    1. Test Title: "Technical Skills Test"
    2. Questions: "What is Python? [Options: A, B, C, D]"
    3. Time Limit: -10 minutes
4. Click the "Save Test" button.

Expected Results:

- ○ The system displays an error message: “Invalid time limit. Please set a valid time between 5 and 180 minutes.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure time limit validation is implemented properly.

3. **Testcase ID:** FT-UC19-03
Title: Validate Test Supports Only MCQs
Description: Validate that the system only supports MCQs for test creation.
Preconditions:

- ○ The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Test" section.
3. Enter the following test details:
    1. Test Title: "Technical Skills Test"
    2. Questions: Open-ended question: "Describe Python features."
    3. Time Limit: 60 minutes
4. Click the "Save Test" button.

Expected Results:

- The system displays an error message: “Only MCQs are supported for this test.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: None

4. **Testcase ID:** FT-UC19-04
Title: Prompt for Missing Job Posting
Description: Validate that the system prompts recruiters to create a job before creating a test if no job exists.
Preconditions:

- The recruiter must be logged in.
- No job posting must exist.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Test" section.
3. Enter test details without creating a job post.

Expected Results:

- The system prompts the recruiter to create a job before creating a test.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure recruiters cannot bypass this prompt.

5. **Testcase ID:** FT-UC19-05
Title: Handle Missing Required Fields in Test Creation
Description: Validate that the system prompts recruiters to fill in missing fields when creating a test.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.

2. Navigate to the "Create Test" section.
3. Enter the following test details:
    1. Test Title: Empty
    2. Questions: "What is Python? [Options: A, B, C, D]"
    3. Time Limit: 60 minutes
4. Click the "Save Test" button.

Expected Results:

- The system displays a message prompting the recruiter to fill in the missing title field.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: None

6. **Testcase ID:** FT-UC19-06
Title: Simulate System Error While Saving Test
Description: Validate that the system handles backend errors gracefully during test creation.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Create Test" section.
3. Enter the following test details:
    1. Test Title: "Technical Skills Test"
    2. Questions: "What is Python? [Options: A, B, C, D]"
    3. Time Limit: 60 minutes
4. Simulate a database connection timeout and click the "Save Test" button.

Expected Results:

- The system displays an error message: “Unable to save test. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure errors are logged for debugging purposes.

Functional Testing 15: View Job Details

Objective:
To validate that the system allows recruiters to view details of job postings they have created, handles scenarios where no jobs are created, and manages exceptions effectively.

1. **Testcase ID:** FT-UC20-01
Title: View Job Details with Existing Jobs
Description: Validate that the system displays a list of jobs created by the recruiter along with their details.
Preconditions:

- ○ The recruiter must be logged in.
- ○ The recruiter must have created at least one job post.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "My Jobs" section.
3. View the list of jobs.

Expected Results:

- ○ The system displays a list of jobs with their respective details (title, description, location, etc.).

Actual Result: Job list and details displayed successfully.
Status: Pass
Remarks: Ensure all job details are fetched from the database accurately.

2. **Testcase ID:** FT-UC20-02
Title: Handle No Jobs Created
Description: Validate that the system displays an appropriate message when no jobs have been created by the recruiter.
Preconditions:

- ○ The recruiter must be logged in.
- ○ No job posts must exist.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "My Jobs" section.

Expected Results:

- ○ The system displays a message: “No jobs have been created yet.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure clarity and visibility of the message for empty job lists.

3. **Testcase ID:** FT-UC20-03
Title: Verify Job Details Display
Description: Validate that the system displays accurate details for a selected job.
Preconditions:

- ○ The recruiter must be logged in.
- ○ The recruiter must have created at least one job post.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "My Jobs" section.
3. Select a job to view its details.

Expected Results:

- The system displays accurate job details, such as:
  - Title: "Software Engineer"
  - Location: Remote
  - Salary: $80,000–$100,000
  - Description: "Develop and maintain software"

Actual Result: Job details displayed accurately.
Status: Pass
Remarks: Verify that all job details are displayed in the correct format.

4. **Testcase ID:** FT-UC20-04
Title: Simulate System Error While Fetching Job Details
Description: Validate that the system handles errors gracefully during job detail fetching.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "My Jobs" section.
3. Simulate a database connection timeout while fetching job details.

Expected Results:

- The system displays an error message: “Unable to fetch job details. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC20-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to view job details.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "My Jobs" section without logging in.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: Ensure unauthenticated users cannot bypass this restriction.

Functional Testing 16: View Applicant Details

Objective:
To validate that the system allows recruiters to view details of job applicants, handles scenarios with no applicants, and manages exceptions effectively.

1. **Testcase ID:** FT-UC21-01
Title: View Applicant Details with Existing Applicants
Description: Validate that the system displays a list of applicants and allows recruiters to view individual applicant details.
Preconditions:

- The recruiter must be logged in.
- At least one applicant must have applied for a job posting.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section for a specific job.
3. View the list of applicants.
4. Select an applicant to view their details.

Expected Results:

- The system displays a list of applicants with their respective details.
- Individual applicant details can be viewed.

Actual Result: Applicant list and details displayed successfully.
Status: Pass
Remarks: Ensure applicant data is retrieved accurately from the database.

2. **Testcase ID:** FT-UC21-02
Title: Handle No Applicants Found
Description: Validate that the system displays an appropriate message when no applicants have applied for the job.
Preconditions:

- The recruiter must be logged in.
- No applicants must have applied for the job posting.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section for a specific job.

Expected Results:

- ○ The system displays a message: “No applicants found.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure the message is clear and visible.

3. **Testcase ID:** FT-UC21-03
Title: Verify Applicant Details Display
Description: Validate that the system displays complete and accurate details for a selected applicant.
Preconditions:

- ○ The recruiter must be logged in.
- ○ At least one applicant must have applied for a job posting.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section for a specific job.
3. Select an applicant to view their details.

Expected Results:

- ○ The system displays accurate and complete details for the selected applicant, such as:
  - ■ Name: John Doe
  - ■ Skills: Python, JavaScript
  - ■ Experience: 3 years
  - ■ Education: Bachelor’s in Computer Science

Actual Result: Applicant details displayed successfully.
Status: Pass
Remarks: Verify that all fields are displayed correctly without truncation.

4. **Testcase ID:** FT-UC21-04
Title: Simulate System Error While Fetching Applicant Details
Description: Validate that the system handles errors gracefully when fetching applicant details.
Preconditions:

- ○ The recruiter must be logged in.
- ○ Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.

2. Navigate to the "Applications" section for a specific job.
3. Simulate a database connection issue while retrieving applicant details.

Expected Results:

- ○ The system displays an error message: “Unable to fetch applicant details. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC21-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to view applicant details.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Applications" section without logging in.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: Ensure proper security measures are in place to restrict unauthorized access.

Functional Testing 17: Filter Applicants

Objective:
To validate that the system allows recruiters to filter applicants based on specific criteria, handles cases with no matching applicants, and manages exceptions effectively.

1. **Testcase ID:** FT-UC22-01
Title: Filter Applicants Based on Valid Criteria
Description: Validate that the system displays applicants who meet the specified filter criteria.
Preconditions:

- The recruiter must be logged in.
- Applicants must exist in the system.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Apply the following filter criteria:
    1. Experience: 3+ years
    2. Skills: Python
4. Click the "Apply Filter" button.

Expected Results:

- ○ The system displays a list of applicants matching the criteria.

Actual Result: Matching applicants displayed successfully.
Status: Pass
Remarks: Ensure filtering logic is applied correctly.

2. **Testcase ID:** FT-UC22-02
Title: Handle No Matching Applicants
Description: Validate that the system handles scenarios where no applicants match the filter criteria.
Preconditions:

- ○ The recruiter must be logged in.
- ○ No applicants should match the criteria.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Apply the following filter criteria:
    1. Experience: 10+ years
    2. Skills: Java
4. Click the "Apply Filter" button.

Expected Results:

- ○ The system displays a message: “No matching applicants.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure message visibility and clarity for empty results.

3. **Testcase ID:** FT-UC22-03
Title: Validate Multiple Filter Criteria
Description: Validate that the system displays applicants meeting all selected filter criteria.
Preconditions:

- ○ The recruiter must be logged in.
- ○ Applicants must exist with varying qualifications.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Apply the following filter criteria:
    1. Experience: 5+ years

        2. Skills: Python, React
    4. Click the "Apply Filter" button.

Expected Results:

- The system displays applicants meeting all selected criteria.

Actual Result: Relevant applicants displayed successfully.
Status: Pass
Remarks: Verify combined filtering logic for multiple criteria.

4. **Testcase ID:** FT-UC22-04
Title: Simulate System Error While Filtering Applicants
Description: Validate that the system handles backend errors gracefully during applicant filtering.

Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

3. Log in to the system as a recruiter.

4. Navigate to the "Applications" section.
5. Apply valid filter criteria.
6. Simulate a database connection timeout and click the "Apply Filter" button.

Expected Results:

- The system displays an error message: “Unable to filter applicants. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC22-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when attempting to filter applicants.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Applications" section without logging in.

2. Apply any filter criteria.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: Ensure proper security measures are in place to restrict unauthorized access.

Functional Testing 18: Compare Applications

Objective:
To validate that the system allows recruiters to compare applicants based on job-relevant criteria, handles scenarios with insufficient applicants, and manages exceptions during the comparison process.

1. **Testcase ID:** FT-UC23-01
Title: Compare Applicants with Valid Data
Description: Validate that the system allows recruiters to compare two or more applicants based on their experience, skills, and qualifications.
Preconditions:

- The recruiter must be logged in.
- At least two applicants must have applied for a job posting.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select at least two applicants for comparison:
    1. Applicant 1: John Doe (Skills: Python, Experience: 3 years)
    2. Applicant 2: Jane Smith (Skills: React, Experience: 5 years)
4. Click the "Compare Applicants" button.

Expected Results:

- The system displays a comparison of the applicants based on their experience, skills, and qualifications.

Actual Result: Comparison displayed successfully.
Status: Pass
Remarks: Ensure comparison data is accurate and well-formatted.

2. **Testcase ID:** FT-UC23-02
Title: Handle Insufficient Applicants for Comparison
Description: Validate that the system handles scenarios where fewer than two applicants are selected for comparison.
Preconditions:

- The recruiter must be logged in.

- At least one applicant must exist in the system.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select only one applicant (e.g., John Doe) for comparison.
4. Click the "Compare Applicants" button.

Expected Results:

- The system displays a message: “At least two applicants are required for comparison.”

Actual Result: Correct message displayed.
Status: Pass
Remarks: Ensure the system prevents comparison with insufficient applicants.

3. **Testcase ID:** FT-UC23-03
Title: Verify Comparison Criteria Display
Description: Validate that the system displays a detailed comparison of applicants, showing their qualifications side by side.
Preconditions:

- The recruiter must be logged in.
- At least two applicants must have applied for a job posting.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select two applicants for comparison:
    1. Applicant 1: John Doe (Experience: 3 years, Skills: Python)
    2. Applicant 2: Jane Smith (Experience: 5 years, Skills: React)
4. Click the "Compare Applicants" button.

Expected Results:

- The system displays a comparison table showing the applicants’ details side by side.

Actual Result: Comparison table displayed successfully.
Status: Pass
Remarks: Ensure all relevant fields are included in the comparison table.

4. **Testcase ID:** FT-UC23-04
Title: Simulate System Error During Comparison
Description: Validate that the system handles backend errors gracefully during the comparison process.

Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select two applicants for comparison.
4. Simulate a database connection issue and click the "Compare Applicants" button.

Expected Results:

- The system displays an error message: “Unable to compare applicants. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC23-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when they attempt to compare applicants.
Preconditions:
- The user must not be logged in.

Test Steps:

1. Attempt to access the "Applications" section without logging in.
2. Try to select and compare applicants.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: Ensure proper security measures are in place to prevent unauthorized access.

Functional Testing 19: Apply Tags to Applicants

Objective:
To validate that the system allows recruiters to apply tags to applicants for easier categorization, handles scenarios where no applicants are selected, and manages exceptions effectively.

1. **Testcase ID:** FT-UC24-01
Title: Apply Valid Tags to Selected Applicants
Description: Validate that the system allows recruiters to apply valid tags to selected applicants.
Preconditions:

- The recruiter must be logged in.
- At least one applicant must exist in the system.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select one or more applicants (e.g., John Doe, Jane Smith).
4. Assign tags such as "High Priority" or "Potential Hire".
5. Click the "Apply Tags" button.

Expected Results:

- The system applies the tags to the selected applicants and displays a confirmation message: "Tags applied successfully."

Actual Result: Tags applied successfully.
Status: Pass
Remarks: Ensure tags are stored in the database and displayed with the applicant details.

2. **Testcase ID:** FT-UC24-02
Title: Handle No Applicants Selected for Tagging
Description: Validate that the system prompts recruiters to select applicants before applying tags.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Do not select any applicants.
4. Attempt to apply tags by clicking the "Apply Tags" button.

Expected Results:

- The system displays a prompt: "Please select at least one applicant to apply tags."

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: None

3. **Testcase ID:** FT-UC24-03
Title: Validate Tags Display with Applicant Details
Description: Validate that applied tags are displayed alongside applicant details.
Preconditions:

- The recruiter must have applied tags to applicants.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. View the details of an applicant (e.g., John Doe).

Expected Results:

- The tags assigned to the applicant (e.g., "High Priority") are displayed accurately.

Actual Result: Tags displayed successfully.
Status: Pass
Remarks: Ensure tags are visible and correctly formatted in the UI.

4. **Testcase ID:** FT-UC24-04
Title: Simulate System Error While Applying Tags
Description: Validate that the system handles backend errors gracefully when applying tags.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Applications" section.
3. Select one or more applicants.
4. Simulate a database connection issue and click the "Apply Tags" button.

Expected Results:

- The system displays an error message: "Unable to apply tags. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC24-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when attempting to apply tags.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Applications" section without logging in.
2. Try to select applicants and apply tags.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: None

Functional Testing 20: Create Mail Template

Objective:
To validate that the system allows recruiters to create mail templates for mass communication, handles cases with no templates created, and manages exceptions during template creation or saving.

1. **Testcase ID:** FT-UC26-01
Title: Create Mail Template with Valid Data
Description: Validate that the system allows recruiters to create and save a mail template with valid data.
Preconditions:

 ○ The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Mail Templates" section.
3. Enter the following template details:
   1. Template Name: "Interview Invitation"
   2. Subject: "Your Interview Schedule"
   3. Body: "Dear [Applicant Name], Your interview is scheduled for [Date]."
4. Click the "Save Template" button.

Expected Results:

 ○ The system saves the mail template successfully, and it is visible in the "Mail Templates" tab.

Actual Result: Template saved successfully and displayed in the "Mail Templates" tab.
Status: Pass
Remarks: Ensure placeholders like [Applicant Name] are handled correctly during usage.

2. **Testcase ID:** FT-UC26-02
Title: Handle Missing Template Fields
Description: Validate that the system prompts recruiters to complete missing fields in the mail template form.
Preconditions:

 ○ The recruiter must be logged in.

Test Steps:

4. Log in to the system as a recruiter.

 1. Navigate to the "Mail Templates" section.
 2. Enter the following template details:
    1. Template Name: "Interview Invitation"

    2. Subject: Empty
    3. Body: "Dear [Applicant Name], Your interview is scheduled for [Date]."
3. Click the "Save Template" button.

Expected Results:

- The system prompts the recruiter to complete the missing Subject field.

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure validation covers all required fields.

3. **Testcase ID:** FT-UC26-03
Title: Verify Multiple Template Creation
Description: Validate that the system allows recruiters to create and save multiple mail templates.
Preconditions:

- The recruiter must be logged in.

Test Steps:

1. Log in to the system as a recruiter.
2. Navigate to the "Mail Templates" section.
3. Create and save two templates:
    1. Template 1: "Interview Invitation"
    2. Template 2: "Rejection Notice"
4. Verify that both templates are displayed in the "Mail Templates" tab.

Expected Results:

- The system allows the recruiter to create and save multiple templates, and all are visible in the "Mail Templates" tab.

Actual Result: Multiple templates created and displayed successfully.
Status: Pass
Remarks: Ensure templates are saved independently and displayed correctly.

4. **Testcase ID:** FT-UC26-04
Title: Simulate System Error While Creating Template
Description: Validate that the system handles backend errors gracefully during mail template creation.
Preconditions:

- The recruiter must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a recruiter.

2. Navigate to the "Mail Templates" section.
3. Enter valid template details.
4. Simulate a database connection issue and click the "Save Template" button.

Expected Results:

- The system displays an error message: “Unable to save template. Please try again later.”

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5. **Testcase ID:** FT-UC26-05
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when attempting to create mail templates.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Mail Templates" section without logging in.
2. Try to create and save a mail template.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.
Status: Pass
Remarks: Ensure proper security measures are in place to restrict unauthorized access.

Functional Testing 21: Apply for Jobs

Objective:
To validate that the system allows job seekers to successfully apply for job postings, handles alternate flows for unauthenticated users, and manages exceptions during application submission.

1. **Testcase ID:** FT-UC28-01
Title: Apply for a Job with a Complete Profile
Description: Validate that job seekers with a complete profile can successfully apply for job postings.
Preconditions:

- The job seeker must be logged in.
- The profile must be complete.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Jobs" section and select a job (e.g., Software Engineer).

3. Click the "Apply" button.

Expected Results:

- The system saves the application successfully and sends a confirmation notification to the job seeker.

Actual Result: Application submitted successfully, and confirmation received.
Status: Pass
Remarks: Ensure application data is stored in the database accurately.

2. **Testcase ID:** FT-UC28-02
Title: Handle Incomplete Profile
Description: Validate that the system prevents job seekers with incomplete profiles from applying for jobs and prompts them to complete their profiles.
Preconditions:

- The job seeker must be logged in.
- The profile must be incomplete.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Jobs" section and select a job.
3. Click the "Apply" button with an incomplete profile (e.g., missing skills section).

Expected Results:

- The system prevents the application and displays a prompt: "Please complete your profile to apply for this job."

Actual Result: Correct prompt displayed.
Status: Pass
Remarks: Ensure that the system enforces profile completeness validation.

3. **Testcase ID:** FT-UC28-03
Title: Redirect Unauthenticated User to Login Page
Description: Validate that the system redirects unauthenticated users to the login page when attempting to apply for jobs.
Preconditions:

- The user must not be logged in.

Test Steps:

1. Attempt to access the "Jobs" section without logging in.
2. Select a job and click the "Apply" button.

Expected Results:

- The system redirects the user to the login page.

Actual Result: User redirected to login page successfully.

Status: Pass
Remarks: Ensure proper access control for job application functionality.

4. **Testcase ID:** FT-UC28-04
Title: Verify Confirmation Notification
Description: Validate that the system sends a confirmation notification to the job seeker after a successful application.
Preconditions:

- The job seeker must be logged in.
- The application must be submitted successfully.

Test Steps:

1. Log in to the system as a job seeker.
2. Apply for a job with a complete profile.

Expected Results:

- The system sends a confirmation notification: "Your application for the Software Engineer position has been submitted successfully."

Actual Result: Confirmation notification sent and displayed successfully.
Status: Pass
Remarks: Ensure notifications are delivered promptly and are personalized.

5. **Testcase ID:** FT-UC28-05
Title: Simulate System Error While Saving Application
Description: Validate that the system handles backend errors gracefully during job application submission.
Preconditions:

- The job seeker must be logged in.
- Simulate a database connection timeout.

Test Steps:

1. Log in to the system as a job seeker.
2. Navigate to the "Jobs" section and select a job.
3. Simulate a database connection issue and click the "Apply" button.

Expected Results:

- The system displays an error message: "Unable to save your application. Please try again later."

Actual Result: Correct error message displayed.
Status: Pass
Remarks: Ensure error logs are created for debugging purposes.

5.3 Integration Testing (IT)

Integration Testing 1: User Sign Up and Login Flow

Testing Objective: To validate that the User Sign Up and Login functionalities work together seamlessly and ensure proper data flow and control between the authentication and user management modules.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Integration Test 1: Valid Sign Up and Login	Email: john@example.com, Password: StrongPass123	The system registers the user successfully and allows login with the provided credentials.	User registered and logged in successfully.	Pass
2	Integration Test 2: Login with Unregistered Email	Email: unknown@example.com, Password: Any	The system displays an error message: “Invalid email or password.”	Correct error message displayed.	Pass
3	Integration Test 3: Sign Up with Existing Email	Email: john@example.com, Password: StrongPass456	The system displays an error message: “Email already registered.”	Correct error message displayed.	Pass

Table 37 Integration Testing 1

Integration Testing 2: Job Search and Application Flow

Testing Objective:

To validate that the Job Search and Application functionalities work together, ensuring smooth data flow between the job search, job listing, and application modules.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Integration Test 1: Search and Apply for a Job	Job Type: Full-Time, Salary: $70,000–$90,000	The system displays matching job listings and allows the user to successfully apply for one.	Job search and application succeeded.	Pass
2	Integration Test 2: Apply Without Profile Completion	Job: Software Engineer	The system prevents application submission and prompts the user to complete their profile.	Correct prompt displayed.	Pass
3	Integration Test 3: Search Without Matching Jobs	Job Type: Part-Time, Salary: $100,000+	The system displays a message: “No jobs found.”	Correct message displayed.	Pass

Table 38 Integration Testing 2

Integration Testing 3: Recruiter Workflow (Company Registration to Job Posting)

Testing Objective:
To validate that the workflow for a recruiter registering their company, creating a portfolio, and posting jobs works seamlessly, ensuring proper integration between the Company Registration, Portfolio Management, and Job Posting modules.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Integration Test 1: Register Company and Create Job	Company Name: Tech Solutions Inc.; Job Title: Software Engineer	The system successfully registers the company and allows the recruiter to create a job posting.	Company registered and job created.	Pass
2	Integration Test 2: Create Portfolio Without Registration	Portfolio Data: Mission, Vision, Job Offerings	The system prevents portfolio creation and prompts the recruiter to register a company first.	Correct prompt displayed.	Pass
3	Integration Test 3: Create Job Without Portfolio	Job Title: Software Engineer	The system allows job creation even without a portfolio, but notifies the recruiter about its absence.	Job created, notification displayed.	Pass

Table 39 Integration Testing 3

Integration Testing 4: Applicant Workflow (Apply Tags, Compare, and Notify)

Testing Objective:
To validate that the system allows recruiters to tag applicants, compare them, and notify them about application statuses, ensuring smooth integration between the Applicant Management, Comparison, and Notification modules.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Integration Test 1: Tag and Compare Applicants	Applicant 1: John Doe; Applicant 2: Jane Smith	The system applies tags successfully and displays a comparison of the tagged applicants.	Tags applied, comparison displayed.	Pass
2	Integration Test 2: Notify After Comparison	Applicant Status: Accepted	The system sends a notification to the selected applicant: “Your application has been accepted.”	Notification sent successfully.	Pass
3	Integration Test 3: Handle No Applicants for Comparison	Applicant List: Empty	The system displays a message: “No applicants available for comparison.”	Correct message displayed.	Pass
4	Integration Test 4: Apply Tags Without Selection	Tags: "High Priority"	The system prevents tagging and prompts the recruiter to select applicants.	Correct prompt displayed.	Pass

5	Integration Test 5: System Error During Notification	Simulated system error during notification sending	The system displays an error message: “Unable to send notification. Please try again later.”	Correct error message displayed.	Pass

Table 40 Integration Testing 4

Summary

This chapter outlined the comprehensive testing approach adopted for the GradHire system, focusing on **unit, functional, and integration testing**. Each test was designed to ensure that individual modules perform as expected, system functionalities align with user requirements, and interactions between modules are seamless. Test cases were documented using a standardized format, covering objectives, inputs, steps, and outcomes. The chapter highlighted adherence to key testing principles such as **test independence, meaningful assertions, and data management**. While automated testing was not implemented due to system limitations, manual tests were effectively conducted for all critical features. This chapter plays a vital role in validating that the system aligns with its goals of enhancing recruitment efficiency and user satisfaction.

Chapter 6

System Conversion

6.Introduction

6.1 Conversion Method

GradHire adopts a **Phased Conversion** approach. The system is being implemented in successive modules — starting with authentication, job search, and company registration, followed by resume filtering, testing, chat integration, and notification services. This method provides a manageable transition while allowing thorough testing and user feedback at each stage, minimizing risks and ensuring system stability.

6.2 Deployment

The deployment steps for GradHire include:

1. **Build Preparation**: Ensure all frontend and backend components are built using npm build and the Node.js server is tested locally.
2. **Dockerization**:
   - Write Dockerfile for frontend and backend.
   - Create a docker-compose.yml to manage multi-container setup (MongoDB, frontend, backend).
3. **Database Setup**:
   - Use MongoDB Atlas for production DB or run a Docker Mongo container for testing.
   - Seed initial job, user, and company data.
4. **Environment Configuration**:
   - Setup .env files with production API keys and secrets.
5. **Deployment**:
   - Deploy containers on a server using Docker.
   - Test access to routes and frontend via browser.
6. **Monitoring**:
   - Use logging (winston, morgan) and container health checks to validate service activity.

6.2.1 Data Conversion

Since *GradHire* is a newly developed system, there is no existing legacy data to migrate. However, sample data (such as user profiles, job posts, and recruiter entries) was validated and seeded into the MongoDB database using scripts. Data was cleaned and verified for consistency before loading. Backup procedures using mongodump and mongorestore were also defined to ensure recovery and portability across environments.

6.2.2 Training

To ensure ease of use, a **User Manual** is provided for the two core use cases:

Use Case 1: Job Seeker - Apply for a Job

1. Login to your account.
2. Complete your profile.
3. Use the search filter to find jobs.
4. Click "Apply" and receive confirmation.

Use Case 2: Recruiter - Post a Job and View Applications

1. Login and switch to Company Mode.
2. Register your company.
3. Click "Create Job" and submit job details.
4. Navigate to "Applications" to view candidates.

6.3 Post Deployment Testing

To verify correct system deployment:

6 **Run Functional Test Cases**: Validate all core workflows like login, apply, post jobs, etc.

7 **Integration Testing**: Ensure services like resume filtering and notification are working with backend routes.

8 **Load Testing**: Simulate multiple user logins and job applications to ensure scalability

9 **UI/UX Review**: Manually check responsiveness and accessibility of the application.

10 **Error Logging Review**: Monitor for runtime errors using logs.

6.4 Challenges

List down the challenges encountered & their solutions while deploying the system and putting it in the operational usage.

Container Dependency Conflicts
Managing dependencies across containers sometimes caused version mismatches or unexpected crashes during Docker deployment. This required careful version pinning and service restarts.

Network Configuration Issues
Setting up network bridges for communication between containers (e.g., backend and database) occasionally failed due to misconfigured Docker Compose files.

Missing Environment Variables
Several features relied on environment variables (e.g., API keys, JWT secrets) that were not properly loaded during deployment, leading to broken functionality until fixed.

Slow Build Times
Building images with heavy dependencies (e.g., Node modules, Python packages) consumed significant time and system resources.

Data Migration Errors: Minor issues with data formatting and schema alignment were encountered during database initialization and backup restoration.

6.5 Summary

The GradHire system was successfully deployed using a Docker-based environment for demonstration purposes. The deployment included database initialization, server configuration, and service containerization. Several technical challenges such as environment mismatches and Docker network configuration were encountered and resolved through iterative debugging and testing. The absence of real-time user feedback during conversion was mitigated using mock data and controlled test environments. Overall, the deployment process laid a strong foundation for future operational deployment in a live environment.

Chapter 7

Conclusion

7 Introduction

This chapter concludes the project, evaluate the project objectives & goals and highlights future work.

7.1 Evaluation

Make a list of the objectives specified in Chapter 1 and trace whether they have been implemented in your developed FYDP.

Objectives	**Status**
Enable Dual-Role User Functionality	Completed
Support Responsive Design across Devices	Completed
Accelerate Job Search and Filtering	Completed
Automate Resume Filtering and Matching	Completed
Ensure Real-Time Application Tracking and Alerts	Completed
Guarantee System Availability and Uptime	Partially Completed
Simplify User Onboarding	Completed
Promote User Data Security and Privacy Compliance	Partially Completed
Apply the Pareto Principle for Feature Prioritization	Completed
Design Tests for Independency and Repeatability	Completed
Mitigate Risks and Account for Dependencies	Completed

Table 41 Evaluation Matrix

7.2 Traceability Matrix

Requirement ID	Requirement Description	Design Specification	Scope	Code	Test ID
FR1	The system must allow users to sign up by providing a valid email, username, and password.	Component User Authentication for sign-up validation and secure data storage.	Provides access to the platform for new users, enabling account creation and secure storage of user credentials.	SignUP.js	UT-UC1-01
FR2	The system must allow users to log in using valid credentials (email and password).	Component User Authentication for login management and session handling.	Ensures secure user access to their accounts and session management for authenticated operations.	index.js	UT-UC2-01
FR3	The system must enable job seekers to create and update their profiles with personal, educational, and professional details	Component Profile Data Management.	Allows job seekers to share qualifications and update profiles for recruiters to review.	Profile.js	UT-UC3-01
FR4	The system must allow users to track the status of job applications (e.g., pending, reviewed, accepted).	Component Application Tracker.	Improves transparency by showing job seekers the progress of their applications.	DashBoard.js	
FR5	The system must allow users to switch between job seeker and recruiter modes.	Component Mode Switcher.	Simplifies dual-role functionality for users acting as both recruiters and job seekers.	user.js	
FR6	The system must enable job seekers to search for jobs using keywords and filters.	Component Job Search and Search Filter.	Helps job seekers discover job opportunities efficiently by	JobPage.js	FT-UC6-01

			refining search results.		
FR7	The system must apply filters (e.g., skills, salary range) to job searches for better relevance.	Component Search Filter.	Enhances search precision and user satisfaction by narrowing down results to relevant jobs.	JobPage.js	
FR8	The system must allow recruiters to create and evaluate tests for job applicants	Component Assessment Module.	Assists recruiters in assessing applicant skills through tests.	Quiz.js	FT-UC8-01
FR9	The system must provide a comparison tool for job seekers to evaluate different companies.	Component Company Comparison Tool.	Enables job seekers to make informed decisions by comparing employers based on ratings and reviews.	CompareCompanies.js	
FR10*	The system must allow job seekers to review and verify their profile details before applying for a job.	Component Profile Review.	Ensures applicants submit accurate and up-to-date profiles, increasing their chances of success.	Profile.js	
FR11	The system must allow job seekers to update their profiles with new skills or experiences	Component Profile Updater.	Keeps profiles current, improving relevance and accuracy for recruiters.	Profile.js	FT-UC11-01
FR12	The system must allow users to give ratings and reviews for companies.	Component Review System.	Enhances platform transparency by allowing job seekers to provide feedback about employers.	GiveReview.js	FT-UC12-01

FR13	The system must filter and discard irrelevant CVs based on recruiter-defined criteria.	Component Resume Filter with defined algorithms.	Saves recruiters time by eliminating unsuitable candidates based on predefined filters.	JobDescription.js	FT-UC13-01
FR14	The system must notify applicants about the status of their job applications	Component Notification Service.	Keeps job seekers informed about the progress or outcome of their applications.		FT-UC14-01
FR15	The system must allow recruiters to register their companies.	Component Company Registration.	Enables companies to establish an official presence on the platform.	RegisterCompany.js	FT-UC15-01
FR16	The system must enable recruiters to create and manage company portfolios.	Component Company Portfolio Manager.	Attracts job seekers by providing detailed company information.	RegisterCompany.js, EditCompany.js	FT-UC16-01
FR17	The system must enable recruiters to create job postings with specific criteria.	Component Job Posting Manager.	Allows recruiters to advertise job opportunities tailored to their hiring needs.	PostJob.js	FT-UC17-01
FR18	The system must allow recruiters to set specific job criteria for filtering applicants.	Component Criteria Filter.	Simplifies candidate evaluation by filtering applicants based on predefined criteria.	Viewapplicants.js	FT-UC18-01
FR19*	The system must provide a test creation feature for recruiters to evaluate candidates.	Component Test Creation Tool.	Allows recruiters to assess candidate skills and qualifications.	Quiz.js	FT-UC19-01

FR20	The system must allow recruiters to view details of all job postings on a dashboard.	Component Job Dashboard.	Provides a centralized view of job postings and their performance.	HomePage.js	FT-UC20-01
FR21	The system must enable recruiters to view detailed profiles of job applicants.	Component Applicant Profile Viewer.	Assists recruiters in reviewing applicant details for better hiring decisions.	Viewapplicants.js	FT-UC21-01
FR22	The system must allow recruiters to filter applicants based on specific criteria.	Component Applicant Filter.	Streamlines candidate selection by focusing on the most suitable applicants.	Viewapplicants.js	UT-UC22-01
FR23	The system must provide tools for recruiters to compare multiple applicants.	Component Applicant Comparison Tool.	Helps recruiters evaluate and compare candidates for informed decision-making.	Viewapplicants.js	FT-UC23-01
FR24	The system must enable recruiters to tag applicants for easier categorization.	Compone nt Applicant Tagging.	Enhances applicant organization and quick reference for recruiters.	Viewapplicants.js	FT-UC24-01
FR25	The system must allow recruiters to bookmark applicants for future reference.	Component Bookmark Manager.	Allows recruiters to save promising applicants for consideration later.	Viewapplicants.js	FT-UC2-01
FR26*	The system must allow recruiters to create reusable email templates for communication with applicants.	Component Email Template Creator.	Simplifies and standardizes communication with candidates.		FT-UC26-01
FR27	The system must provide a chat feature for recruiters and applicants to communicate directly.	Component Chat Module.	Facilitates real-time communication to address	Chat.js	FT-UC2-01

			queries and clarify requirements.		
FR28	The system must enable job seekers to apply for job postings.	Component Job Application Manager.	Simplifies the application process for job seekers while increasing engagement.	JobDescription.js	FT-UC28-01

Table 42 Traceability Matrix

7.3 Conclusion

The GradHire project successfully fulfills its core objectives by providing a comprehensive and intelligent recruitment platform for job seekers and employers. It delivers key features like resume filtering, job recommendation, profile management, and real-time application tracking. The implementation aligns with the Pareto Principle, ensuring that the most critical 20% of functionalities were prioritized to deliver 80% of the platform's value within the project's academic and resource constraints.
While the majority of planned modules were completed and verified through extensive manual and functional testing, several enhancements are proposed for future development. These include a built-in **Resume Builder Tool** to help applicants create professional CVs, **Multi-Language Support** for broader accessibility, **Calendar Sync for Interviews** to streamline scheduling with platforms like Google and Outlook, and **Voice-Enabled Search** to make navigation more intuitive and accessible. Overall, the system provides a scalable foundation that meets its primary goals of improving the hiring experience, reducing recruiter workload, and increasing application relevance for candidates.

7.4 Future Work

Resume Builder Tool: Provide job seekers with a built-in resume creation tool using customizable templates.

Multi-Language Support: Add support for additional languages to make the platform accessible to a wider user base.

Calendar Sync for Interviews: Enable integration with Google/Outlook calendars for automatic interview scheduling and reminders.

Voice-Enabled Search: Let users search for jobs or candidates using voice commands, improving accessibility and ease of use.

References

1. Book

[12] R. N. Bolles, *What Color Is Your Parachute? A Practical Manual for Job-Hunters and Career-Changers*, 2022 ed., New York, NY, USA: Ten Speed Press, 2022, pp. 1-352.

2. Article in a Journal

[8] Findem, "LinkedIn Recruiter Alternatives," *Findem*. [Online]. Available: https://www.findem.ai/knowledge-center/linkedin-recruiter-alternatives/. [Accessed: Jan. 2, 2025].

[9] HeroHunt, "Best LinkedIn Recruiter Alternatives," *HeroHunt*. [Online]. Available: https://www.herohunt.ai/blog/best-linkedin-recruiter-alternatives/. [Accessed: Dec. 29, 2024].

[10] GeeksforGeeks, "LinkedIn Alternatives for Job Seekers," *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/linkedin-alternatives-for-job-seekers/. [Accessed: Jan. 2, 2025].

3. World Wide Web

[1] LinkedIn, "LinkedIn Alternatives," *LinkedIn*. [Online]. Available: https://pk.linkedin.com/. [Accessed: Jan. 03, 2025].

[2] Indeed, "Job search platform," *Indeed*. [Online]. Available: https://pk.indeed.com/. [Accessed: Dec. 31, 2024].

[3] S. Journal, "LinkedIn Alternatives: Exploring different job search options," *Search Engine Journal*. [Online]. Available: https://www.searchenginejournal.com/linkedin-alternatives/297409/. [Accessed: Jan. 02, 2025].

[4] WaffleBytes, "Comparing job search platforms," *WaffleBytes*. [Online]. Available: https://wafflebytes.com/blog/linkedin-alternatives/. [Accessed: Jan. 01, 2025].

4. ChatGPT

[5] OpenAI, ChatGPT, "ChatGPT Response on GradHire System Requirements Traceability Matrix." Accessed on: Jan. 1, 2025. [Online]. Available: https://chat.openai.com/

[5] OpenAI, ChatGPT, "ChatGPT Response on MVC for GradHire." Accessed on: Jan. 1, 2025. [Online]. Available: https://chat.openai.com/

[5] OpenAI, ChatGPT, "ChatGPT Response on Use Case Descriptions for GradHire." Accessed on: Jan. 1, 2025. [Online]. Available: https://chat.openai.com/

Appendix A

Use case Description Template

Appendix-A Use Case Description (Fully Dressed Format)

Use Case #1: User Sign Up (UC1)

UC Identifier	*UC1*
Requirements Traceability	**FR1 – A user shall sign up to the system first.**
Purpose	**To allow a new user to sign up for the platform.**
Priority	**High**
Preconditions	**The user must have a valid email address, and the system should be functioning properly.**
Post conditions	**The user is registered successfully, and the database is updated.**
Actors	**Job Seeker, Recruiter**
Extends	**N/A**
Main Success Scenario	**1. User clicks "Sign Up".** **2. The system displays a registration form.** **3. User enters details (name, email, password).** **4. User will press the submit button.** **5. The system validates and registers the user.**
Alternate Flows	**1. If the password is weak, the system displays a "Password is weak" message.** **2. If passwords do not match, the system shows "Passwords don't match".** **3. If a user enters an invalid email then a pop up will be shown with a message "Invalid Email Address".**
Exceptions	**If the system fails to validate the data, an error message is shown.**
Includes	*N/A*

Table 43 Use Case 1

Use Case #2: User Login (UC2)

UC Identifier	*UC2*
Requirements Traceability	**FR2 – A user shall log into the system using valid credentials.**
Purpose	**To allow users to log into the platform using their credentials.**
Priority	**High**
Preconditions	**The user must be registered with a valid email and password.**
Post conditions	**The user is successfully logged in and redirected to their dashboard.**
Actors	**Job Seeker, Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The user enters their email and password.** **2. User will press the Log-In button.** **3. The system validates the credentials.** **4. If valid, the user is logged in.**
Alternate Flows	**If credentials are incorrect, the system displays "Incorrect email or password".**
Exceptions	**System failure or error while logging in.**
Includes	**Signup,**

Table 44 Use Case 2

Use Case #3: Add Applicant Profile (UC3)

UC Identifier	**UC3**
Requirements Traceability	**FR3 – A job seeker shall add their profile.**
Purpose	**To allow a user to add their profile to the application, which will act as their resume.**
Priority	**High**
Preconditions	**The user must be logged in.**
Post conditions	**The profile is saved and updated in the system.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The user clicks on "Add Profile".** **2. The system displays a profile form.** **3. The user fills in the details.** **4. User will press the "add profile" button.** **4. The system saves the profile.**
Alternate Flows	**If required fields are missing, the system prompts the user to fill them.**
Exceptions	**System failure while saving the profile**
Includes	**Login**

Table 45 Use Case 3

Use Case #4: View Status (UC4)

UC Identifier	**UC4**
Requirements Traceability	**FR4 – A user shall view their application status.**
Purpose	**To allow a user to view the status of their job applications, including pending reviews, submitted applications, and responses from recruiters. On the company side, the user (recruiter) can view the current status of posted jobs and the number of applications received for each job.**
Priority	**Medium**
Preconditions	**The user must be logged in.**
Post conditions	**The user is shown the status of job applications or posted jobs on the dashboard.**
Actors	**Job Seeker, Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The user opens their dashboard.** **2. The system shows the status of applications and responses.**
Alternate Flows	***N/A***
Exceptions	**System failure while fetching the status.**
Includes	**Login**

Table 46 Use Case 4

Use Case #5: Switch between Company and Job Seeker Mode (UC5)

UC Identifier	**UC5**
Requirements Traceability	**FR5 – A user shall be able to switch between company mode and job seeker mode.**
Purpose	**To allow users (who are both recruiters and job seekers) to switch between job seeker mode and company mode.**
Priority	**Medium**
Preconditions	**The user must be registered and logged in.**
Post conditions	**The system switches the mode, allowing the user to interact with the platform in the selected mode.**
Actors	**User (Job Seeker, Recruiter)**
Extends	**N/A**
Main Success Scenario	**1. The user logs in.** **2. The system presents the option to switch between modes.** **3. The user selects a mode, and the system updates the interface accordingly.**
Alternate Flows	***If the user is not allowed to switch (due to role restrictions), an error message is displayed.***
Exceptions	**System error while switching between modes.**
Includes	***N/A***

Table 47 Use Case 5

Use Case #6: Search Jobs (UC6)

UC Identifier	**UC6**
Requirements Traceability	**FR6 – A job seeker shall search for jobs using filters.**
Purpose	**To allow users to search for jobs using specific filters.**
Priority	**High**
Preconditions	**The user must be logged in.**
Post conditions	**The job search results are displayed according to filters.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The user navigates to the "S Jobs" section.** **2. The user applies filters (e.g., job type, salary).** **3. The system displays relevant job postings.**
Alternate Flows	***If no jobs match the search, the system shows a "No jobs found" message.***
Exceptions	**System error while searching for jobs.**
Includes	***N/A***

Table 48 Use Case 6

Use Case #7: Apply Filters (UC7)

UC Identifier	**UC7**
Requirements Traceability	**FR7 – A job seeker shall search for jobs using filters.**
Purpose	**To allow users to search for jobs using specific filters.**
Priority	**High**
Preconditions	**The user must be logged in.**
Post conditions	**The job search results are displayed according to filters.**
Actors	**Job Seeker**
Extends	**Search Jobs**
Main Success Scenario	**1. The user navigates to the "Jobs" section.** **2. The user applies filters (e.g., job type, salary).** **3. The system displays relevant job postings.**
Alternate Flows	***If no jobs match the search, the system shows a "No jobs found" message.***
Exceptions	**System error while searching for jobs.**
Includes	***N/A***

Table 49 Use Case 7

Use Case #8: Take Test (UC8)

UC Identifier	***UC8***
Requirements Traceability	**FR8 – A job seeker shall be able to take a test for further processing.**
Purpose	**To allow job seekers to take tests as part of the recruitment process.**
Priority	**Medium**
Preconditions	**The user must be logged in and applied for a job.**
Post conditions	**The test is completed and the result is stored.**
Actors	**Job Seeker, System**
Extends	**Apply For Jobs**
Main Success Scenario	**1. The user receives a notification for a test.** **2. The user opens and completes the test with the specified time..** **3. The system stores the results.**
Alternate Flows	**If the user does not complete the test, the system sends a reminder.**
Exceptions	**System failure during test submission.**
Includes	***N/A***

Table 50 Use Case 8

Use Case #9: Compare Companies (UC9)

UC Identifier	*UC9*
Requirements Traceability	**FR9 – A job seeker shall compare different companies.**
Purpose	**To allow job seekers to compare companies based on various parameters.**
Priority	**Medium**
Preconditions	**The user must be logged in.**
Post conditions	**A comparison graph of companies is displayed.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. User will open his timeline.** **2. User will go to the company page.** **3. User will open the company tab.** **4. User will see a graphical comparison of reviews of different companies**
Alternate Flows	**If the user is not logged in, they are redirected to the login page.**
Exceptions	**System error while generating the comparison graph.**
Includes	*N/A*

Table 51 Use Case 9

Use Case #10: Review Profile (UC10)

UC Identifier	***UC10***
Requirements Traceability	**FR10 – A job seeker shall review their profile before applying for a job.**
Purpose	**To allow job seekers to check and confirm their profile details before application submission.**
Priority	**High**
Preconditions	**The user must be logged in and have a completed profile.**
Post conditions	**The user can view their profile and make any necessary edits.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The user clicks "Review Profile" before applying for a job.** **2. The system displays the user's profile information.** **3. The user reviews the information and decides to apply or make edits.**
Alternate Flows	**If the profile is incomplete, the system prompts the user to fill in missing information.**
Exceptions	**System failure while retrieving profile data.**
Includes	**Apply For Jobs**

Table 52 Use Case 10

Use Case #11: Update Profile (UC11)

UC Identifier	***UC11***
Requirements Traceability	**FR11 – A job seeker shall update their profile/CV.**
Purpose	**To allow job seekers to update their profile (CV).**
Priority	**High**
Preconditions	**The user must be logged in and already have a profile..**
Post conditions	**The profile is updated and saved in the system.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. User will open his dashboard.** **2. The user opens the "Profile" tab.** **3. The system displays the current profile.** **4. The user will click the edit icon/button.** **5. The user updates their information and saves it.**
Alternate Flows	**If the user is not logged in, they are redirected to the login page.**
Exceptions	**System failure while updating the profile.**
Includes	***N/A***

Table 53 Use Case 11

Use Case #12: Company Reviews (UC12)

UC Identifier	*UC12*
Requirements Traceability	**FR12 –A user shall be able to give reviews for companies.**
Purpose	**To allow users (job seekers or employees) to provide reviews and feedback about companies.**
Priority	**Medium**
Preconditions	**The user must be logged in.**
Post conditions	**The company review is submitted and visible to other users.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The user navigates to the company's profile.** **2. The system provides an option to "Give Review".** **3. The user submits their review by rating the company and providing feedback.** **4. The system stores the review and displays it under the company's profile.**
Alternate Flows	**If the user is not logged in and attempts to submit a review, they are redirected to the login page.**
Exceptions	**System error during review submission.**
Includes	**Login**

Table 54 Use Case 12

Use Case #13: Discard Irrelevant CVs (UC13)

UC Identifier	***UC13***
Requirements Traceability	**FR25 – The system shall filter out irrelevant CVs during the recruitment Process.**
Purpose	**To automatically discard CVs that do not meet the job criteria set by the Recruiter.**
Priority	**High**
Preconditions	**The recruiter must have posted a job, and applicants must have applied.**
Post conditions	**Irrelevant CVs are filtered out, and the system retains only relevant Applications.**
Actors	**System, Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter posts a job and specifies the required criteria.** **2. The system filters incoming CVs based on these criteria.** **3. The system discards any CVs that do not meet the criteria.**
Alternate Flows	**If there are no applications, the system notifies the recruiter that no CVs Were received.**
Exceptions	**System failure during the filtering process.**
Includes	**N/A**

Table 55 Use Case 13

Use Case #14: Notify Applicant (UC14)

UC Identifier	***UC214***
Requirements Traceability	**FR14 – The system shall notify applicants about the status of their Application.**
Purpose	**To keep applicants informed about the outcome of their job applications.**
Priority	**Medium**
Preconditions	**The applicant must have submitted a job application.**
Post conditions	**The applicant receives a notification regarding the status of their Application.**
Actors	**System, Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The recruiter reviews the applications.** **2. The system determines the outcome of each application.** **3. The system sends a notification to each applicant with their application status (accepted/rejected).**
Alternate Flows	**If the applicant's email is invalid, the system logs the failure to notify.**
Exceptions	**System error while sending notifications.**
Includes	**N/A**

Table 56 Use Case 14

Use Case #15: Register Company (UC15)

UC Identifier	***UC15***
Requirements Traceability	**FR15 – A recruiter shall register their company.**
Purpose	**To allow recruiters to register their company on the platform.**
Priority	**High**
Preconditions	**The recruiter must be logged in.**
Post conditions	**The company is successfully registered, and the recruiter can start posting jobs.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter clicks "Register Company".** **2. The system displays a form for company details.** **3. The recruiter fills in the details and clicks the save button.** **4. The system registers the company.**
Alternate Flows	**If the user is not logged in, they are redirected to the login page.**
Exceptions	**System error while registering the company.**
Includes	**Login**

Table 57 Use Case 15

Use Case #16: Company Portfolio (UC16)

UC Identifier	***UC16***
Requirements Traceability	**FR16 – A recruiter shall create a company portfolio for job seekers to view.**
Purpose	**To allow recruiters to create a portfolio showcasing their company for potential job applicants.**
Priority	**Medium**
Preconditions	**The recruiter must be logged in and registered as a company.**
Post conditions	**The company portfolio is saved and visible to job seekers.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter navigates to the "Create Portfolio" section.** **2. The system presents a form to add company details (e.g., mission, vision, job offerings).** **3. The recruiter submits the portfolio, and the system saves it.**
Alternate Flows	**If the portfolio is incomplete, the system prompts the recruiter to fill in missing details.**
Exceptions	**System failure while saving the portfolio.**
Includes	***N/A***

Table 58 Use Case 16

Use Case #17: Create Job (UC17)

UC Identifier	*UC17*
Requirements Traceability	FR17 – A recruiter shall create a job post for their company.
Purpose	To allow recruiters to create job postings.
Priority	High
Preconditions	The recruiter must be logged in and have registered a company.
Post conditions	The job is successfully posted, and visible to job seekers.
Actors	Recruiter
Extends	N/A
Main Success Scenario	1. The recruiter switches to "Company Mode". 2. The system displays the "Create Job" form. 3. The recruiter fills in job details and clicks the “create” button. 4. The system posts the job.
Alternate Flows	If the recruiter has not registered the company, the system prompts them to do so.
Exceptions	System error while posting the job.
Includes	Register Company

Table 59 Use Case 17

Use Case #18: Set Criteria for Job Post (UC18)

UC Identifier	*UC18*
Requirements Traceability	**FR18 – A recruiter shall set criteria for job postings to filter applicants.**
Purpose	**To allow recruiters to set criteria (e.g., qualifications, skills) for job posts to filter relevant applicants.**
Priority	**High**
Preconditions	**The recruiter must be logged in and posting a job.**
Post conditions	**The job post criteria are set, and only relevant applications are accepted.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter navigates to the "Create Job" section.** **2. The recruiter sets the required qualifications and criteria for the job.** **3. The system saves the job post with the defined criteria, and filters incoming applications based on them.**
Alternate Flows	**If no criteria are set, the system allows all applicants to apply.**
Exceptions	**System failure while setting job criteria.**
Includes	*N/A*

Table 60 Use Case 18

Use Case #19: Create Job Test (UC19)

UC Identifier	*UC19*
Requirements Traceability	**FR19 – A recruiter shall create a test for job applicants.**
Purpose	**To allow recruiters to create tests as part of the application process.**
Priority	**High**
Preconditions	**The recruiter must be logged in, have created a job, and be in company mode.**
Post conditions	**A popup will be shown saying "test created successfully" .**
Actors	**Recruiter**
Extends	**Create Job**
Main Success Scenario	**1. The recruiter creates a job.** **2. The recruiter sets the test criteria.** **3. The recruiter creates the test questions and saves the test by clicking the save/create button.**
Alternate Flows	**If the job is not created, the system prompts the recruiter to create one.**
Exceptions	**System failure while saving the test.**
Includes	*N/A*

Table 61 Use Case 19

Use Case #20: View Job Details (UC20)

UC Identifier	***UC20***
Requirements Traceability	**FR20 – A recruiter shall view details of jobs they posted.**
Purpose	**To allow recruiters to view details of job postings they have created.**
Priority	**High**
Preconditions	**The recruiter must be logged in and in company mode.**
Post conditions	**Job details are displayed on the dashboard.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter logs in and switches to company mode.** **2. The recruiter will go to its dashboard** **3. The system displays a list of jobs.**
Alternate Flows	**If no jobs are created, the system shows an empty list.**
Exceptions	**System error while fetching job details.**
Includes	**N/A**

Table 62 Use Case 20

Use Case #21: View Applicant Details (UC21)

UC Identifier	***UC21***
Requirements Traceability	**FR21 – A recruiter shall view details of job applicants.**
Purpose	**To allow recruiters to view details of applicants who applied for their job.**
Priority	**Medium**
Preconditions	**The recruiter must be logged in, in company mode, and have posted a job.**
Post conditions	**Applicant details are displayed.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter navigates to the "Applications" tab. 2. The system displays a list of applicants for the job. 3. The recruiter clicks to view details of an applicant.**
Alternate Flows	**If no applicants applied, the system displays "No applicants found".**
Exceptions	**System error while fetching applicant details.**
Includes	**Create Job**

Table 63 Use Case 21

Use Case #22: Filter Applicants (UC22)

UC Identifier	***UC15***
Requirements Traceability	**FR15 – A recruiter shall filter applicants based on criteria (experience, skills).**
Purpose	**To allow recruiters to filter applicants based on specific criteria.**
Priority	**High**
Preconditions	**The recruiter must be logged in, in company mode, and have applicants.**
Post conditions	**Filtered applicants are displayed based on the criteria.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter navigates to "Applications".** **2. The recruiter selects filter criteria (e.g., experience, skills).** **3. The system filters the applicants accordingly.**
Alternate Flows	**If no applicants match the criteria, the system shows "No matching applicants".**
Exceptions	**System error while filtering applicants.**
Includes	**View Applications**

Table 64 Use Case 22

Use Case #23: Compare Applications (UC23)

UC Identifier	***UC23***
Requirements Traceability	**FR23 – A recruiter shall compare applicants based on the job description.**
Purpose	**To allow recruiters to compare applicants based on relevant job criteria.**
Priority	**Medium**
Preconditions	**The recruiter must be logged in, in company mode, and have applicants.**
Post conditions	**A comparison of applicants is displayed.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter selects applicants to compare.** **2. The system displays a comparison based on experience, skills, etc.**
Alternate Flows	**If fewer than two applicants are available, comparison is not possible.**
Exceptions	**System error during comparison.**
Includes	**View Applications**

Table 65 Use Case 23

Use Case #24: Apply Tags to Applicants (UC24)

UC Identifier	***UC24***
Requirements Traceability	**FR24–A recruiter shall apply tags to applicants for easier categorization.**
Purpose	**To allow recruiters to tag applicants based on various categories (e.g., priority).**
Priority	**Low**
Preconditions	**The recruiter must be logged in, in company mode, and have applicants.**
Post conditions	**Tags are successfully applied to the applicants.**
Actors	**Recruiter**
Extends	**View Applicants**
Main Success Scenario	**1. The recruiter selects applicants to tag.** **2. The recruiter assigns tags (e.g., "High Priority").** **3. The system applies the tags.**
Alternate Flows	**If no applicants are selected, the system shows "Select applicants to tag".**
Exceptions	**System error while applying tags.**
Includes	**N/A**

Table 66 Use Case 24

Use Case #25: Bookmark Applicants (UC25)

UC Identifier	***UC25***
Requirements Traceability	**FR25 – A recruiter shall bookmark applicants for future reference.**
Purpose	**To allow recruiters to bookmark applicants for future reference.**
Priority	**Low**
Preconditions	**The recruiter must be logged in, in company mode, and have applicants.**
Post conditions	**Applicants are successfully bookmarked.**
Actors	**Recruiter**
Extends	**View Applicants**
Main Success Scenario	**1. The recruiter selects applicants to bookmark.** **2. The system bookmarks the selected applicants.**
Alternate Flows	**If no applicants are selected, the system shows "Select applicants to bookmark".**
Exceptions	**System error while bookmarking applicants.**
Includes	**N/A**

Table 67 Use Case 25

Use Case #26: Create Mail Template (UC26)

UC Identifier	***UC26***
Requirements Traceability	**FR26 – A recruiter shall create mail templates to send to multiple Applicants.**
Purpose	**To allow recruiters to create mail templates for mass communication.**
Priority	**Low**
Preconditions	**The recruiter must be logged in and in company mode.**
Post conditions	**The mail template is saved and can be used for future emails.**
Actors	**Recruiter**
Extends	**N/A**
Main Success Scenario	**1. The recruiter navigates to the "Mail Templates" tab.** **2. The recruiter creates a template and saves it.**
Alternate Flows	**If no template is created, the system prompts the recruiter to create one.**
Exceptions	**System error while creating or saving the template.**
Includes	**N/A**

Table 68 Use Case 26

Use Case #27: Chat with Applicant (UC27)

UC Identifier	***UC27***
Requirements Traceability	**FR27– A recruiter shall chat with job applicants.**
Purpose	**To allow recruiters to chat with applicants for further communication.**
Priority	**Medium**
Preconditions	**The recruiter must be logged in, in company mode, and have applicants.**
Post conditions	**The recruiter can chat with selected applicants.**
Actors	**Recruiter, Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The recruiter selects an applicant to chat with.** **2. The system opens a chat window.** **3. The recruiter sends a message.**
Alternate Flows	**If the applicant is offline, the system displays "Waiting for applicant".**
Exceptions	**System error while sending or receiving messages.**
Includes	**Login**

Table 69 Use Case 27

Use Case #28: Apply For Jobs (UC28)

UC Identifier	*UC28*
Requirements Traceability	**FR28 – A job seeker shall be able to apply for a job posting.**
Purpose	**To allow a job seeker to submit an application for a job listing they are interested in.**
Priority	**High**
Preconditions	**- The job seeker must be logged in to the system.** **- The job seeker has a completed profile.**
Post conditions	**The job application is successfully submitted, and the job seeker receives a confirmation.**
Actors	**Job Seeker**
Extends	**N/A**
Main Success Scenario	**1. The job seeker navigates to the "Jobs" section and searches for jobs of interest.** **2. The job seeker selects a job and clicks on "Apply".** **3. The system displays an application confirmation dialog.** **4. The job seeker confirms and submits the application.** **5. The system saves the application and sends a confirmation notification.**
Alternate Flows	**If the job seeker is not logged in, the system redirects them to the login page.**
Exceptions	**System error while saving the application or network issues preventing submission.**
Includes	**N/A**

Table 70 Use Case 28

Appendix-B

Coding Standards

Appendix-B General Coding Standards & Guidelines

1. Follow a consistent variable naming convention throughout the code. E.g.
   - Snakecase (words are delimited by "_" like: variable_one)
   - Pascalcase (words are delimited by capital letters like: VariableOne)
   - Camelcase (words are delimited by capital letters except the initial word like: variableOne)
   - Hungarian Notation (describes the variable type or purpose at the start of the variable name like: arrDistributeGroup)
2. Use naming that visually describes scope like privateField, Const etc
3. Use read only/immutable when a field's value should not be changed after initialization
4. Use only get, for properties that should not be updated from outside
5. Name functions according to their functionalities.
6. Insert appropriate comments to make the code understandable to any reader. Additionally follow a consistent style to do so. E.g.
   ```
   /* the below function will be used for the addition of two variables*/ int
   Add(){
   //logic of the function
   }
   ```
   Avoid commenting on obvious things
7. Make use of indentation for indicating the start and end of the control structures along with a clear specification of where the code is between them.

8. Follow consistent naming convention for files and folders.
9. Follow proper structure for classes
10. Group code entities logically into projects/packages/modules/folders
    a. Separate logical layers of application into different modules/services/utilities etc.
    b. User separate files for each class, struct, interface, enum etc. Name of the file and the enclosing entity must be same. E.g., class Employee in Employee.cs/Employee.java
11. Define and use everything within the minimum scope possible
12. Use proper access modifier for all code entities if required
13. Code entities should have maximum cohesion and least coupling possible.

14. Follow DRY law.
    a. Do not repeat code.
    b. A piece of knowledge should exist only in one place within the codebase/application

   c. Reuse code as much as possible
   d. Always write short methods
   e. Single method should not have too many logic, long conditional flow or too many parameters

15. Strictly follow Single Responsibility Principle (SRP) when writing methods, classes, modules, projects, packages, or any other code entities.
16. Write classes and other code entities that are easy to extend without modification.
17. Handle exceptions
18. Log exception and other significant event details

Follow a consistent convention for logging all over the application

Appendix

C Prototype

Appendix-C Application Prototype

The following Figma link provides access to the interactive prototype of the GradHire application, showcasing key user interfaces and workflows for both job seekers and recruiters.

Prototype Link