EAST WEST UNIVERSITY

Course: CSE412 Software Engineering
Section: 1

Project Report

Project Title: Career Hive – Job Portal

Submitted To:

Nishat Tasnim Niloy
Lecture
Department of Computer Science & Engineering

Submitted By:

Student Name	Student ID	Contribution
Bushra Hoque	2022-1-60-154	33.33%
Sanjana Kazi Supti	2022-2-60-036	33.33%
Asif Ahmed Fahim	2022-2-60-084	33.33%

Date of Submission: 30-05-2025

Contents

Chapter 1: Introduction:.....1

1.1 Project Overview.....1

1.2 Purpose and Scope.....2

1.3 Intended Audience.....2

1.4 Stakeholders.....3

1.5 Technology Stack (Programming Language, frameworks,database).....3

1.5 Conclusion.....4

Chapter 2:Software Requirements Specification & Analysis.....5

2.1 Inception. 5

2.1.1 Identifying Stakeholders.....5

2.1.2 Stakeholder Needs & Analysis….....6

2.1.3 Asking the First Questions.....6

2.2 List of Requirements. 7

2.2.1 Functional Requirements (FRs)….....7

2.2.2 Non-Functional Requirements (NFRs).....7

2.2.3 Extra-ordinary Requirements (Wow Factors).....7

2.3 Requirements Elicitation.....7

2.3.1 Eliciting Requirements.….....8

2.3.2 Collaborative Requirements Gathering.....8

2.3.3 Quality Function Deployment.....9

2.3.4 Customer Requirements (CRs) List….....9

2.3.5 Engineering Requirements (TRs) List.....10

2.3.6 QFD Matrix (House of Quality).....10

2.4 Requirements Modeling 11

2.4.1 Use Case Scenario 11

2.4.2 Use Case Diagram.....12

2.4.3 Activity Diagrams 17

2.4.4 Prototyping (Wireframes & UI Sketches).....25

Chapter 3:Software Design…….....27

3.1 Architectural Design …….....……..27

3.1.1 Architectural Context Diagram 27

3.1.2 Top-Level Component….....30

3.1.3 Instantiation of Each Component with Component Elaboration.....32

3.2 Component-Level Design.....32

3.2.1 Elaboration of Design Components ….....32

3.2.2 Class Diagram ……..….....36

3.2.3 MongoDB…….....37

3.3 User Interface Design…….....38

3.3.1 Prototype .....…….38

3.3.2 Navigation Flow …….....…….38

Chapter 4: Implementation.....40

4.1 Code Structure Overview.....41

4.2 Github Repository URL.....42

Chapter 5 :Software Testing.....59

5.1 White Box Testing… …….....…….59

5.1.1 Unit Testing on Two Classes… 60

5.2 Black Box Testing……..….....65

5.2.1 Functional Testing on Two Core Feature.....66

5.3 Bug Detection and Solution.....67

5.3.1 Identification….....67

Chapter 6: Deployment:.....72

6.1 Deployment Platform…….72

6.2 Deployment Process.....73

6.3 Website URL…….....74

Chapter 7: Conclusion:.....77

7.1 Learnings…….....77

7.2 Limitation…….....77

7.3 Future Plan …….....78

**APPENDIX……
…...74**

Chapter 1: Introduction

This chapter is a part of the **Career Hive – Job Portal**, specifying the purpose of this document and its intended audience. It outlines the project overview, purpose, scope, and key stakeholders.

1.1 Project Overview

Career Hive is a job portal designed to revolutionize the recruitment process by efficiently connecting job seekers and recruiters. It offers AI-driven job recommendations, real-time interview scheduling, and an intelligent chatbot for career guidance. The platform aims to bridge the gap between employers and job seekers by optimizing hiring workflows with machine learning and automation.

Career Hive – A User Story:

Career Hive is a job portal designed to streamline job searching and hiring. The platform allows job seekers to register, create profiles, upload resumes, and receive AI-driven job recommendations based on their skills and experience. Using advanced search filters, they can browse job listings, apply with one click, and track applications in real time.

Recruiters can post job openings, manage applications, and filter candidates using AI-powered insights. The platform ranks the most suitable applicants, helping recruiters make quick and informed hiring decisions. Job seekers also benefit from getting resumes built in the same platform, while the built-in AI career chatbot offers interview tips and career guidance.

To enhance communication, Career Hive integrates real-time video and audio interviews, enabling recruiters to schedule and conduct interviews directly on the platform. Instant notifications keep users updated on job postings, application statuses, and interview invitations, ensuring they never miss an opportunity.

By combining AI-driven job matching, automated hiring processes, and real-time communication, Career Hive simplifies recruitment, helping job seekers find the right opportunities faster while empowering recruiters to make smarter hiring decisions efficiently.

1.2 Purpose and Scope

This document is the Software Requirements Specification (SRS) for Career Hive – Job Portal. It contains detailed functional, non-functional, and support requirements, establishing a requirements baseline for system development. The SRS serves as an official means of communication between stakeholders and the development team.

Scope of Career Hive:
Career Hive will provide:

- Job search by keyword.
- Job creation
- Profile creation with skills and resume upload
- Resume Building

1.3 Intended Audience

This SRS is intended for multiple audiences, including customers, project managers, designers, developers, and testers.

- **Job Seekers**: To ensure the system meets their needs for job search, recommendations, and application tracking.
- **Recruiters**: To verify that the platform enables efficient hiring, applicant tracking, and interview scheduling.
- **Project Managers**: To plan development milestones and ensure timely delivery.

- **Designers**: To create an intuitive UI/UX aligned with the specified requirements.
- **Developers**: To implement system functionalities based on documented specifications.
- **Testers**: To create test cases, ensuring that all requirements are met before deployment.

1.4 Stakeholders

Primary Stakeholders

- **Job Seekers** – Use the platform to search for jobs, apply, and receive AI-driven job recommendations.
- **Recruiters** – Post job listings, track applications, and conduct real-time interviews.
- **System Administrators** – Manage user data, platform security, and AI updates.

Secondary Stakeholders

- **Developers** – Implement system functionalities, maintain AI models, and ensure system scalability.

1.5 Technology Stack (Programming languages, frameworks, databases)

The **MERN (MongoDB, Express.js, React, Node.js) stack is used for the project**, all technologies will be within this ecosystem to ensure **consistency, maintainability, and ease of integration.**

- Frontend (React.js)

  TailwindCSS
  Redux for state management
  Shadcn

Radix UI

- Backend (Node.js + Express.js)

  Bcrypt.js
  Jwt (JSON web token)
  Express js
  Mongoose
  Multer

- Database (MongoDB + Mongoose)

- Deployment & DevOps

  CI/CD pipelines : GitHub actions
  Cloud Hosting: Cloudflare workers

- Testing & Development Tools

  Jest: for unit and API testing

1.5 Conclusion

This document serves as a comprehensive guide for all project stakeholders. It ensures that the system meets user expectations while providing a structured framework for design, development, and testing. The Career Hive SRS will evolve as the development progresses, incorporating feedback and refinements to deliver an optimized job portal experience.

Chapter 2: Software Requirements Specification & Analysis

The **Requirements Engineering Process** ensures that Career Hive – Job Portal meets stakeholder needs through a structured approach. It involves stakeholder analysis, requirement elicitation, and requirement specification to define functional, non-functional, and extraordinary (wow factor) requirements. Using the Quality Function Deployment (QFD) framework, customer needs are mapped to technical solutions, ensuring an optimized, scalable, and AI-driven job platform.

2.1 Inception

Inception is the beginning phase of requirements engineering. It defines how a software project gets started and what the scope and nature of the problem to be solved are. The goal of the inception phase is to identify the concurrence needs and conflict requirements among the stakeholders of a software project.

To establish the groundwork, we have worked with the following factors related to the inception phases:

- ➢ Identifying Stakeholders
- ➢ Recognizing multiple viewpoints
- ➢ Working towards collaboration
- ➢ Asking the First Questions

2.1.1 Identifying Stakeholders

The identified stakeholders are:

Primary Stakeholders:

- **Job Seekers:** Utilize Career Hive for job searches, AI-driven recommendations, and career guidance.

- **Recruiters:** Post job vacancies, track applicants, and conduct interviews.
- **System Administrators:** Ensure data security, platform stability, and AI model updates.

Secondary Stakeholders:

- **Developers:** Implement system functionalities, maintain AI algorithms, and ensure scalability.

2.1.2 Stakeholder Needs & Analysis

Stakeholder analysis is essential in defining system requirements. Career Hive involves multiple stakeholders, each with unique needs. Their inputs shape the system's functional and non-functional requirements to ensure an optimal job-seeking and hiring experience.

Requirement Elicitation Methods

To ensure comprehensive requirement gathering, the following methods were used:

- ➢ **Stakeholder Interviews** – Conducted structured meetings with job seekers and recruiters
- ➢ **Competitive Analysis** – Examined existing job portals to identify areas of improvement.

2.1.4 Asking the First Questions

We set our first set of context-free question**s** to focus on customers and stakeholders, overall project goals, and expected benefits. These questions helped us identify key stakeholders, assess the measurable benefits of the system, and explore possible alternatives. The next phase of questioning allowed us to gain a deeper understanding of the problem while enabling stakeholders to express their perspectives on the solution. Finally, the last set of questions evaluated the effectiveness of our communication strategy, ensuring that stakeholder inputs were properly incorporated into the requirement analysis.

2.2 List of Requirements

Functional Requirements (FRs)

- Secure user authentication and role-based access (Job Seeker, Recruiter).
- Recruiters can create companies with details.
- Recruiters can post job vacancies under the Company specification.
- Recruiters can see Job Seekers and approve.
- Recruiters can send an email.
- Recruiters manage job postings and applications.
- Job searching and filtering for users.
- Applicants can build a resume.

Non-Functional Requirements (NFRs)

- The system must be scalable with low latency.
- Encryption of personal data ensures data privacy and security.
- 97% uptime for uninterrupted job searches and recruitment.
- Maintainability: Automated deployments with GitHub Actions.

Extra-ordinary Requirements (Wow Factors)

- Real-time AI chatbot for career planning, interview preparation, and recommendations.
- Seamless WebRTC-powered interviews eliminate third-party dependencies.
- Resume Builder and formatting for professional resumes.

2.3 Requirements Elicitation

Requirements elicitation is the process of gathering information from stakeholders to understand what a system should do. It involves techniques like interviews, surveys, observations, and document analysis to identify user needs, expectations, and constraints. This step is crucial for defining clear and accurate requirements that guide the design and development of a successful software system.

2.3.1 Eliciting Requirements

The main task of this phase is to combine the elements of problem-solving, elaboration, negotiation, and specification. The collaborative working approach of the stakeholders is required to elicit the requirements. The following tasks were done to elicit requirements:

1. Collaborative Requirements Gathering
2. Quality Function Deployment
3. Usage Scenarios
4. Elicitation work products

2.3.2 Collaborative Requirements Gathering

INTERVIEW AND MARKET ANALYSIS RESULTS

To ensure Career Hive meets the needs of job seekers and recruiters, we interviewed two students, two career mentors, and one HR professional, alongside a market analysis of online job platforms.

Job Seekers Want: Students highlighted the struggles of finding relevant jobs and felt that existing platforms offer generic recommendations. Both agreed an AI-powered chatbot would make job searching easier, and one suggested it should provide resume feedback.

They also found resume formatting confusing and tracking applications difficult. Both emphasized the need for auto-generated resumes and email notifications for application updates. One student also requested video interview scheduling for smoother communication.

Mentors' Observations: Mentors noted that many job seekers lack career guidance, often applying for jobs without understanding industry trends. They recommended AI-driven career suggestions and resume evaluation tools to help candidates stand out.

Market Analysis: Market analysis showed platforms like LinkedIn and others indeed are shifting towards real-time tracking, reinforcing the need for these features in Career Hive.

Recruiters Need: The HR professional emphasized real-time communication through live chat or video calls and interview scheduling could save time in the hiring process.

2.3.3 Quality Function Deployment

Quality Function Deployment (QFD) is a technique that translates the needs of the customer into technical requirements for software. It concentrates on maximizing customer satisfaction from the Software engineering process. With respect to our project, the following requirements are identified by a QFD.

Customer Requirements (CRs) List

The following customer requirements (CRs) define the key expectations from the Career Hive platform:

1. Seamless job search and filtering options.
2. Job recommendations based on user profiles.
3. Resume upload, parsing.
4. User-friendly interface with intuitive navigation.
5. Secure authentication and data privacy.
6. Integrated video and audio interview system.

7. Employer dashboard to manage job postings and applicants.

Engineering Requirements (TRs) List

Each Customer Requirement (CR) is mapped to a corresponding Technical Requirement (TR) to ensure feasibility and proper implementation.

1. Resume building.
2. Secure authentication using JWT-based login and OAuth integration.
3. Scalable cloud architecture to handle high traffic.
4. Encrypted MongoDB storage with role-based access control.
5. Job search assistance.
6. Admin dashboard with advanced filtering and analytics for recruiters.

QFD Matrix (House of Quality)

Table 1: House of Quality

Customer Requirements (CRs)	**Engineering Requirements (TRs)**	**Relationship mapping**
Seamless job search & filtering options	Scalable search engine	Strong
Resume upload & AI-driven improvement	NLP-based AI resume analysis & optimization	Medium
User-friendly interface	User-friendly design & UX optimization	Medium
Secure authentication & data privacy	JWT-based login, OAuth, and role-based access control	Strong
Employer job management	Custom-built dashboard	Medium

2.4 Requirements Modeling:

The **Requirements Modeling** phase focuses on visually representing system functionalities, user interactions, and data flow to ensure clarity and structured development. This phase helps transform **stakeholder needs** and **elicited requirements** into **structured models** that guide system design and implementation.

2.4.1 Use Case Scenario

Table 2: Use case Scenario

Level 0	Level 1	Level 2
	Authentication & Management	Register/Sign up (Students & Recruiters)
		Sign in
		Sign out
	Job Search & Application	Search & Filter Jobs as a student
		Search employ as a recruiter
		Apply for Jobs
		Upload/Edit Resume

Career Hive	Resume & Profile Management	Manage Skills & Certifications
		Build Profile
	Employer & Job Management	Create/Edit Company Profile
		Post Job
		View & Shortlist Candidates
	Interview & Communication	Schedule Interviews
		Conduct Video & Audio Calls

Primary Actors: Job seekers, Recruiters.

Secondary Actors: Admin.

3.1 Use Case Diagram:

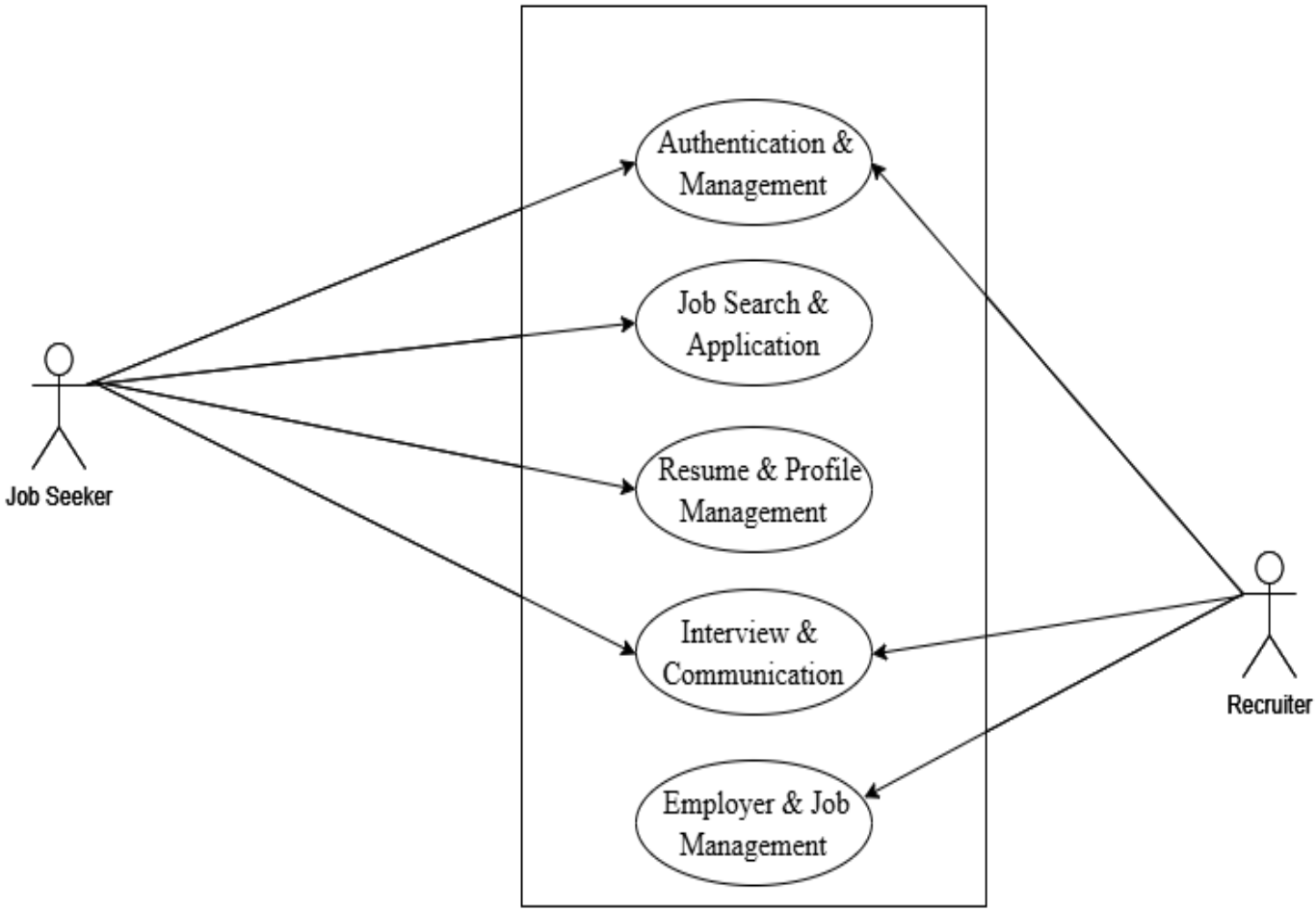


Figure 1: Level 1 Use Case Diagram for Career Hive

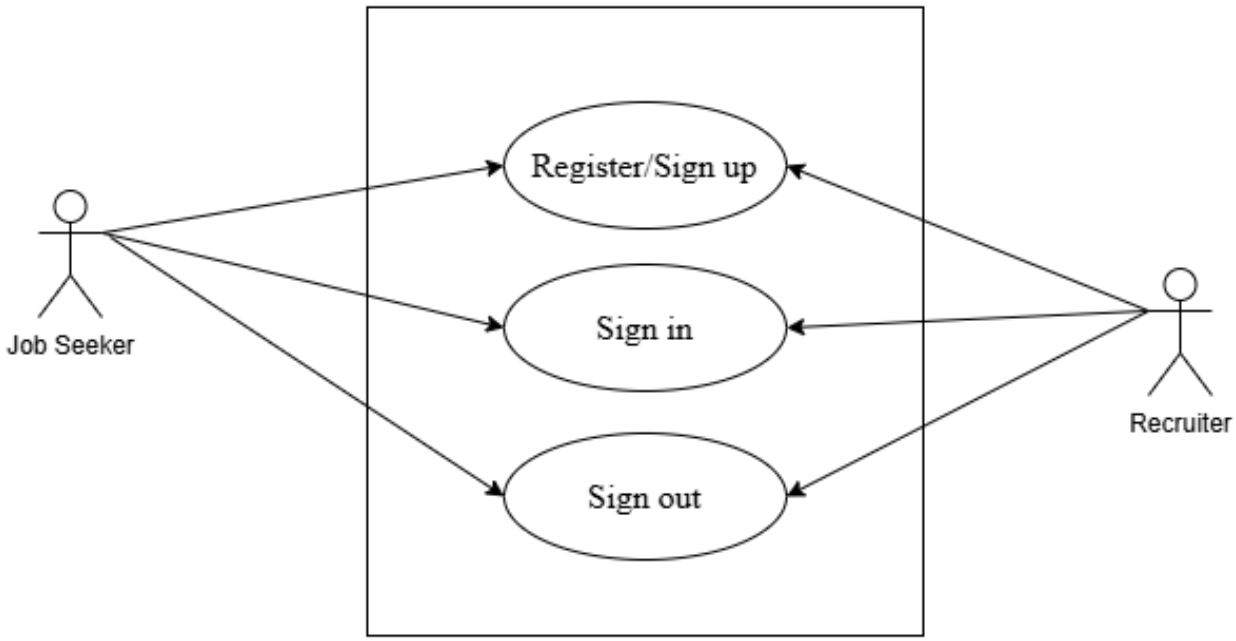


Figure 2: Use Case Diagram for Authentication

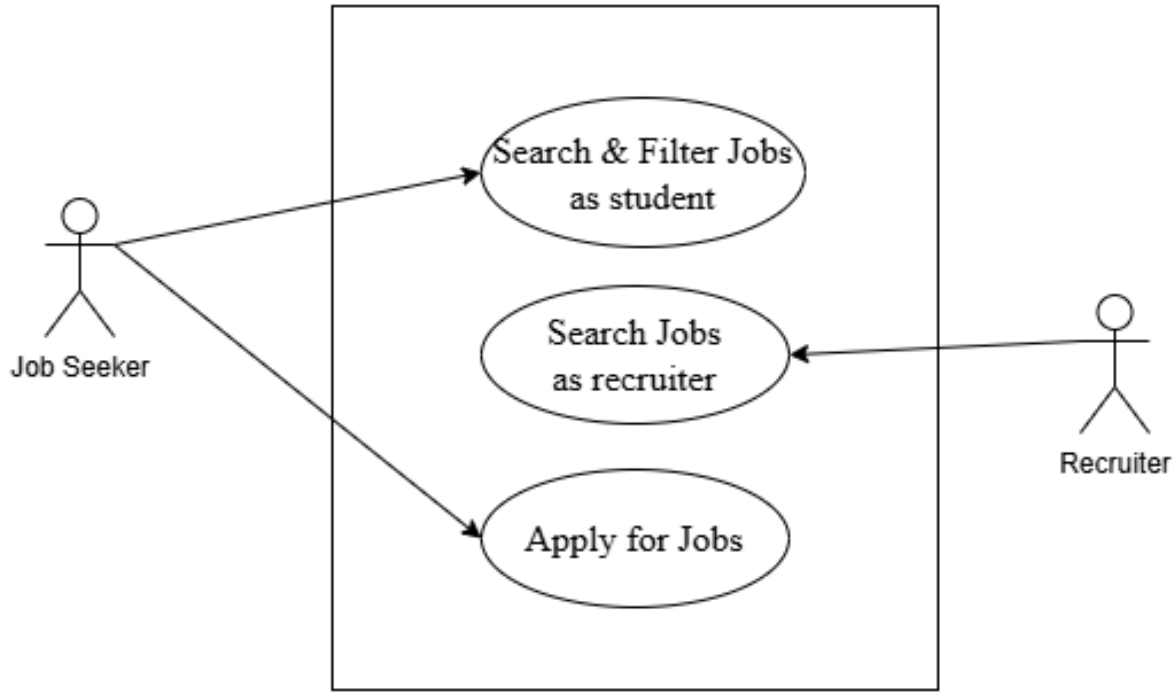


Figure 3: Use Case Diagram for Job Search & Application

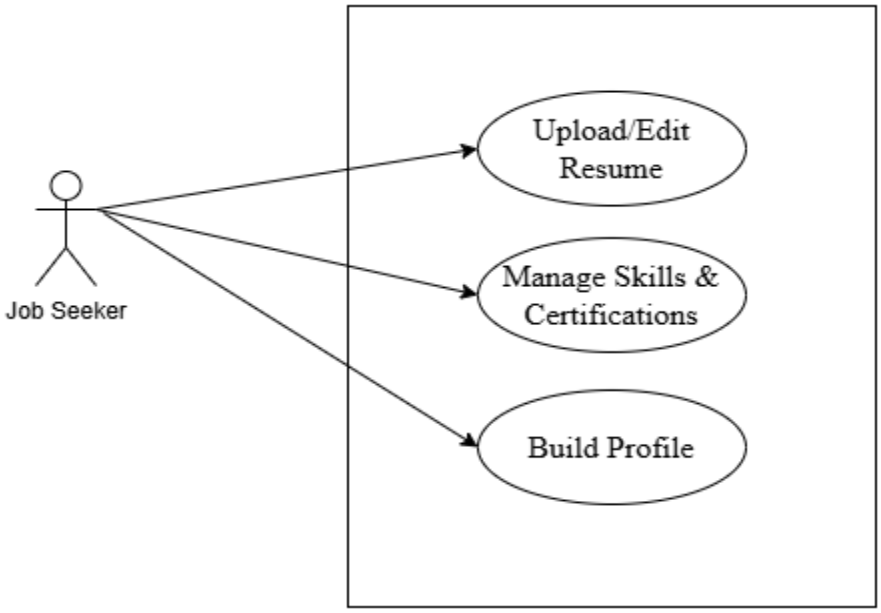


Figure 4: Use Case Diagram for Resume & Profile Management

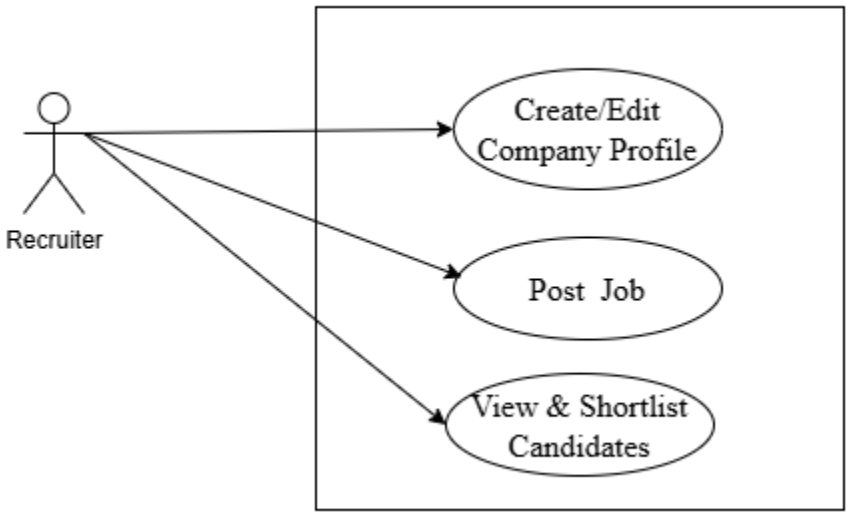


Figure 5: Use Case Diagram for Employer & Job Management

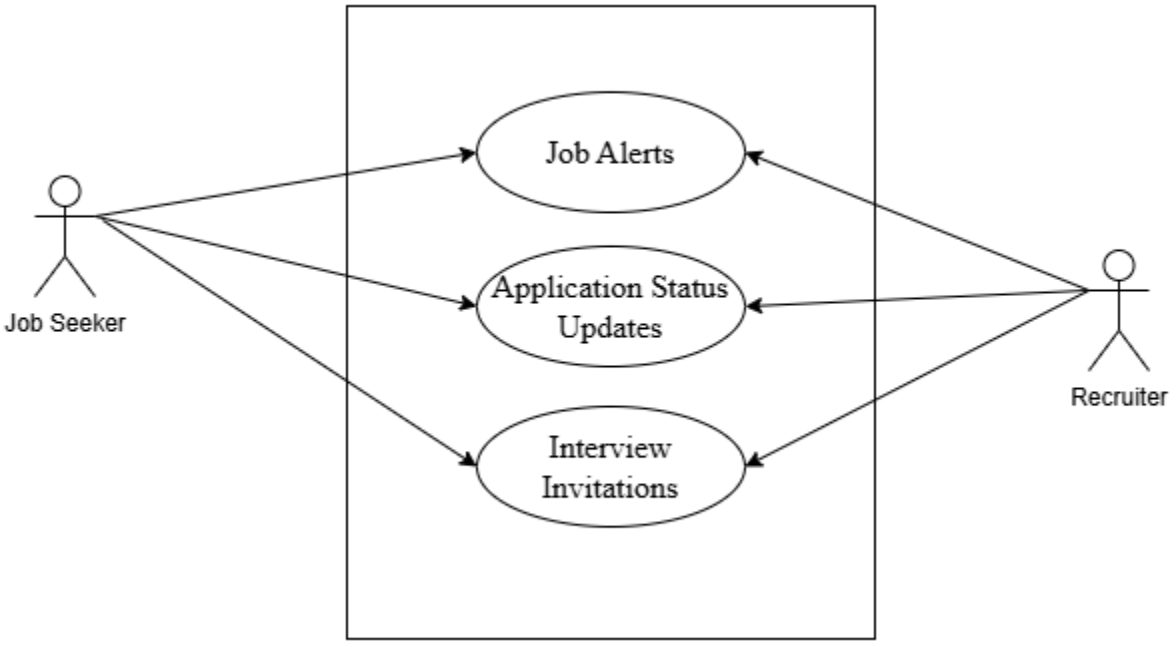


Figure 6: Use Case Diagram for Interview & Communication

3.2 Activity Diagram:

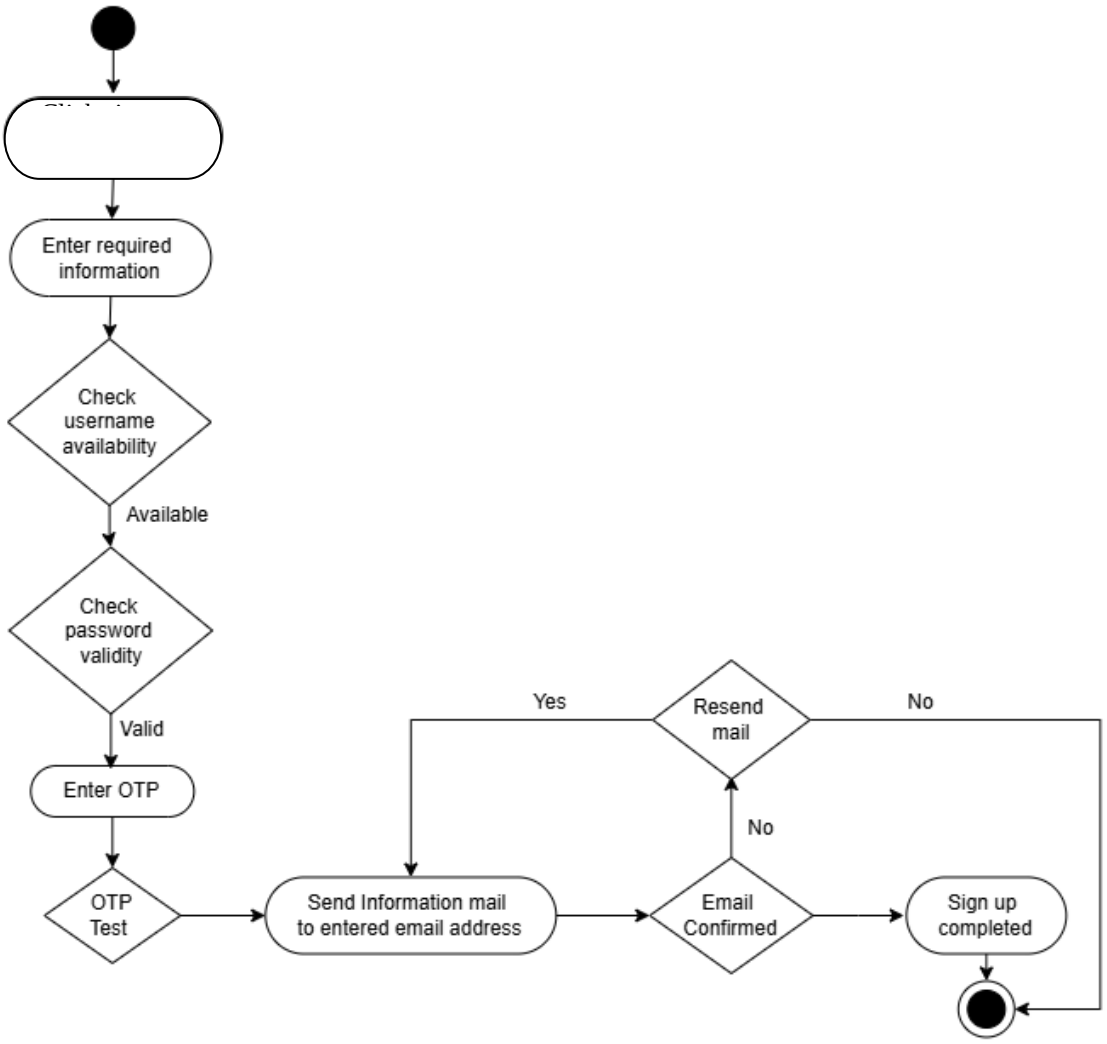


Figure 7: Activity Diagram for Sign up

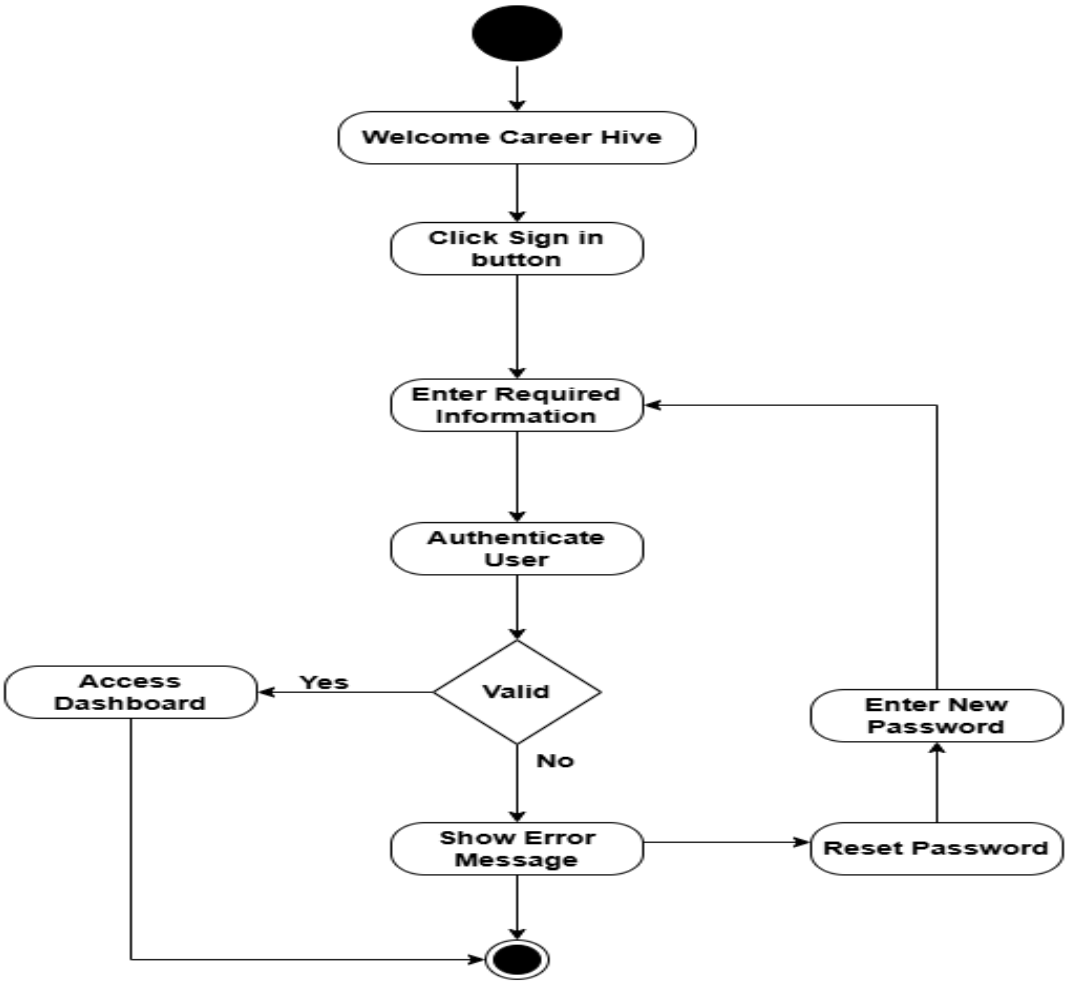


Figure 8: Activity Diagram for Sign in

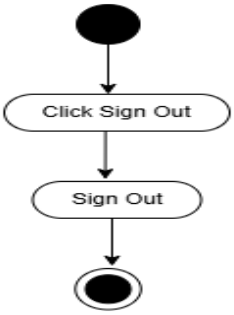


Figure 9: Activity Diagram for Sign out

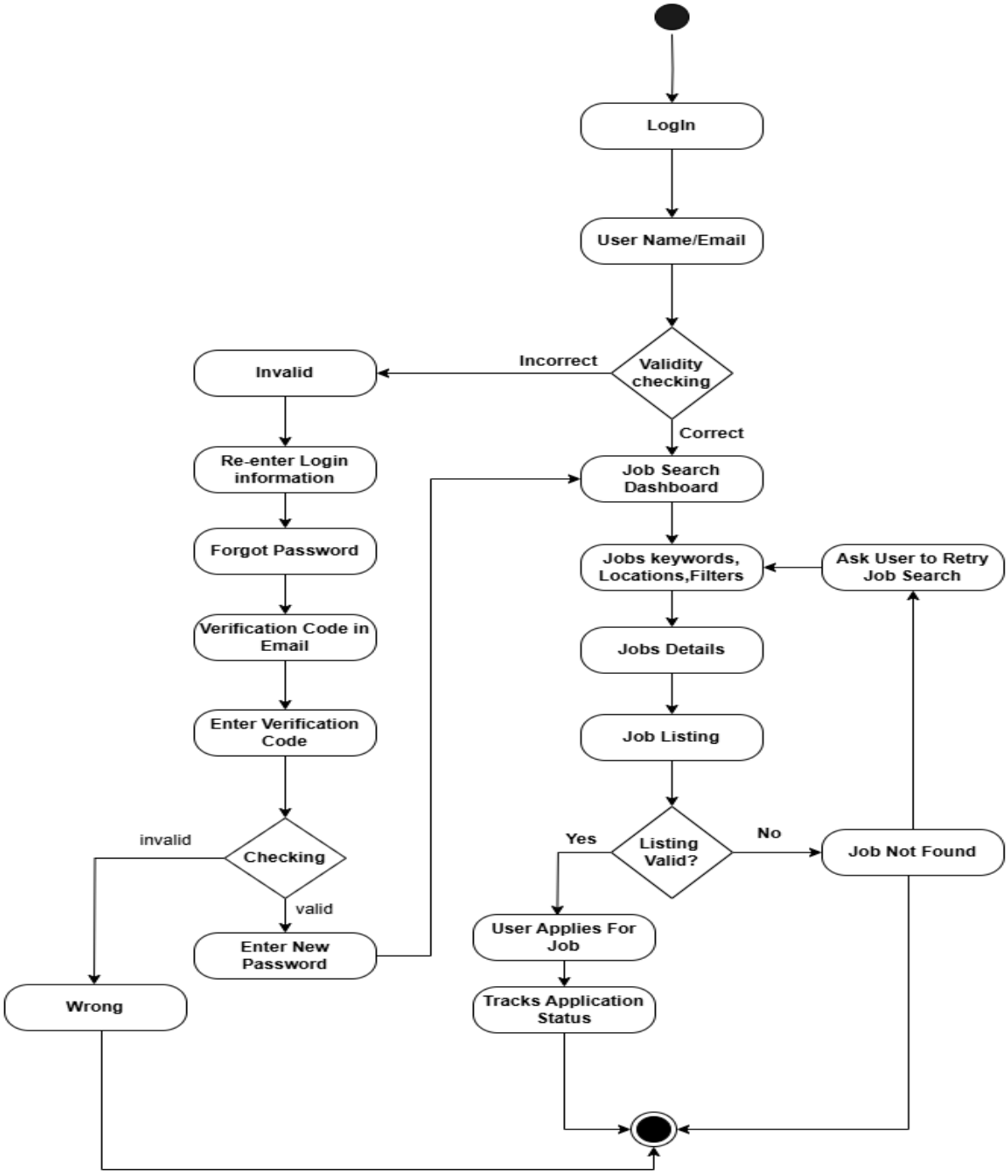


Figure 10: Activity Diagram for Search & Filter Jobs as a student

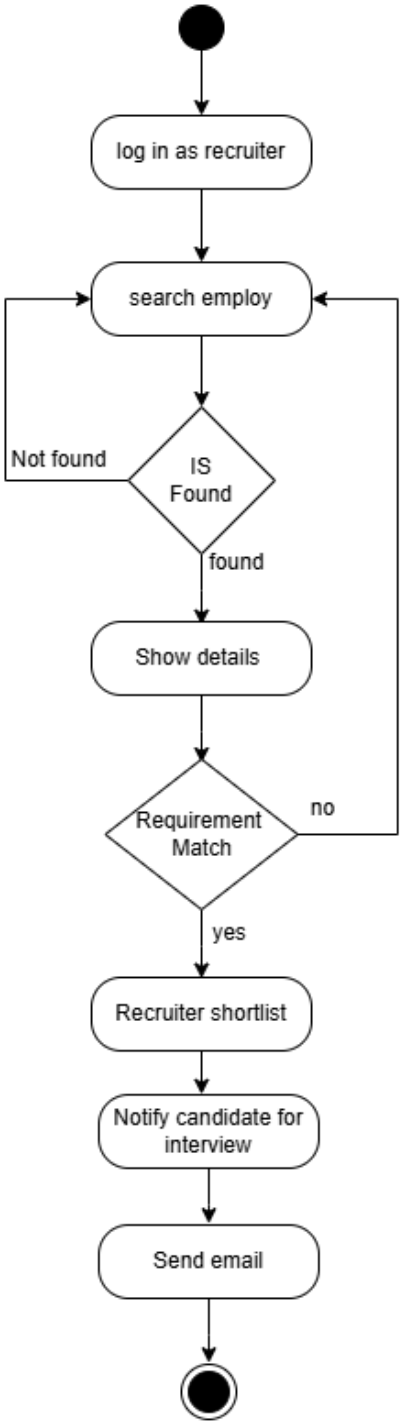


Figure 11: Activity Diagram for Search employ

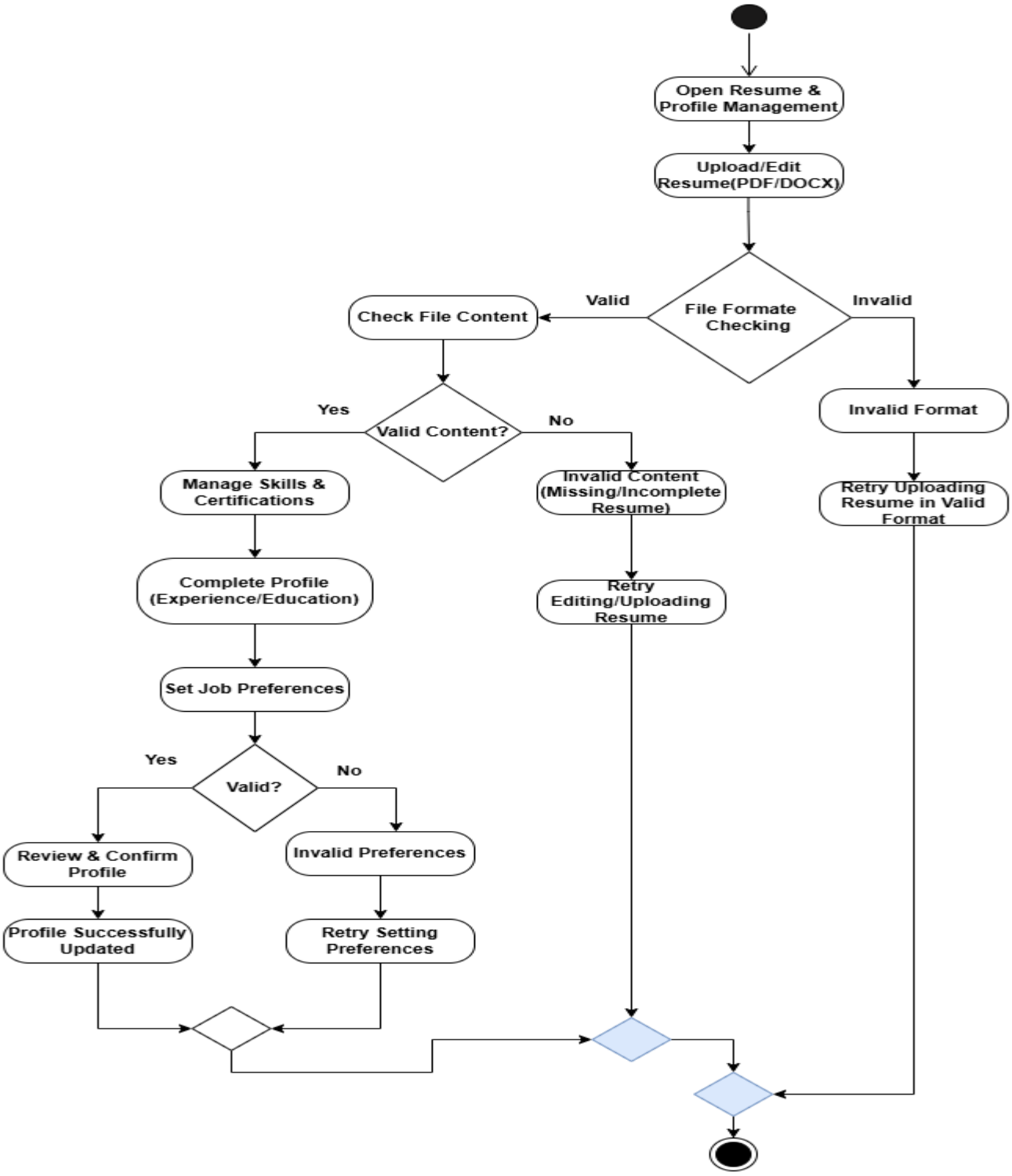


Figure 12: Activity Diagram for Resume & profile management

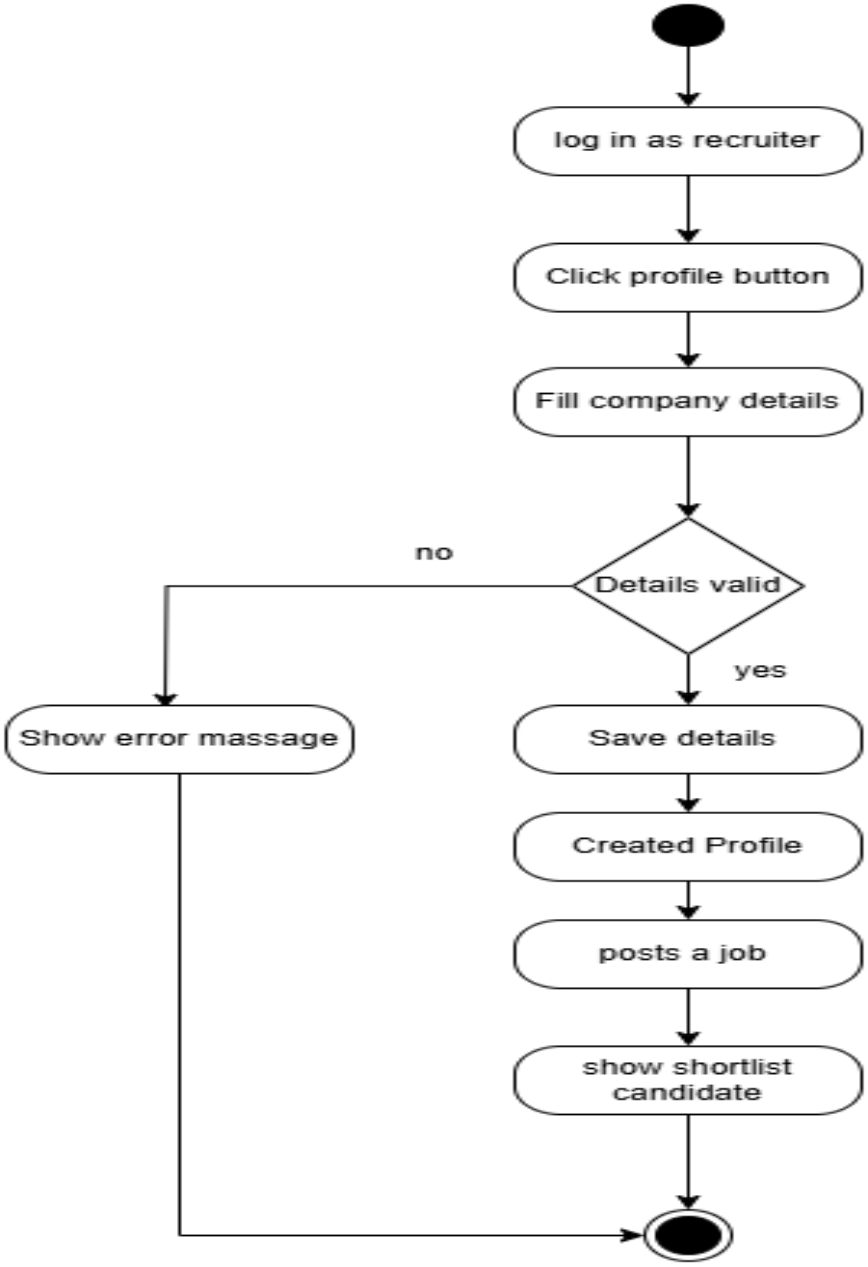


Figure 13: Activity Diagram for Employer & Job Management

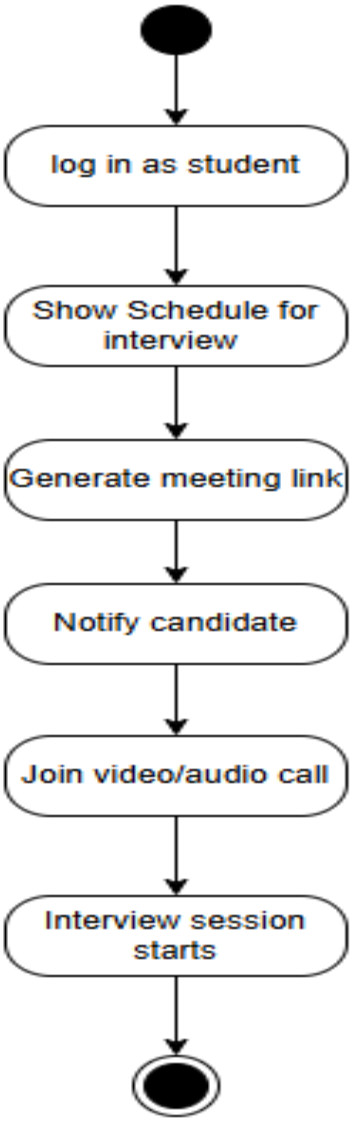


Figure 14: Activity Diagram for Interview & Communication

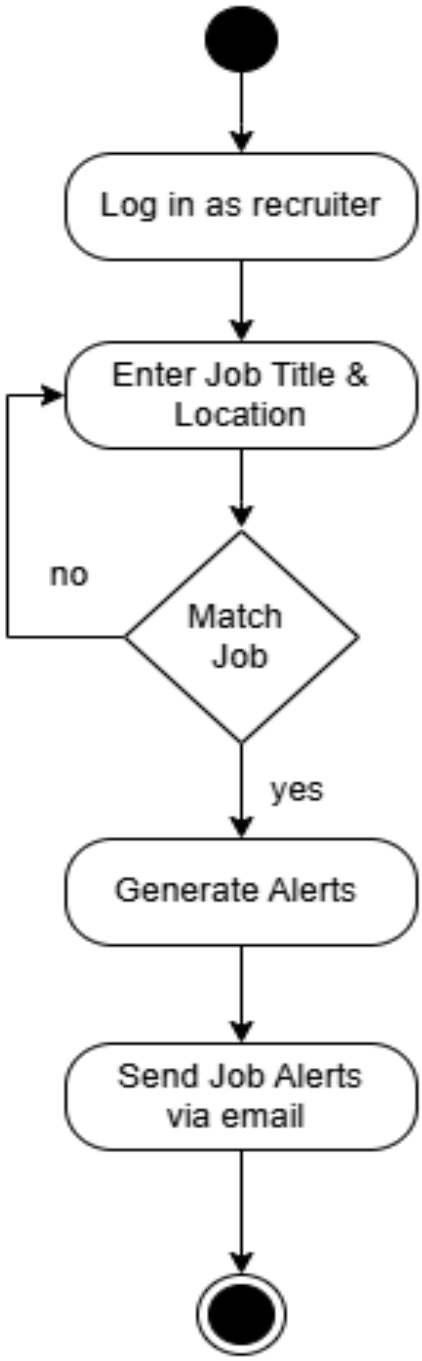


Figure 15: Activity Diagram for Job Alert

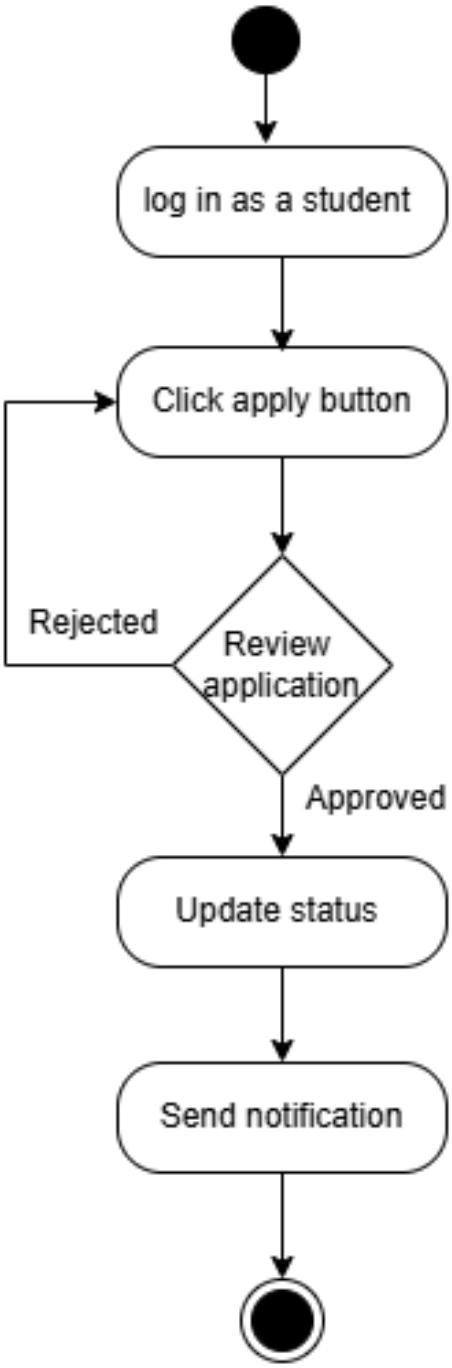
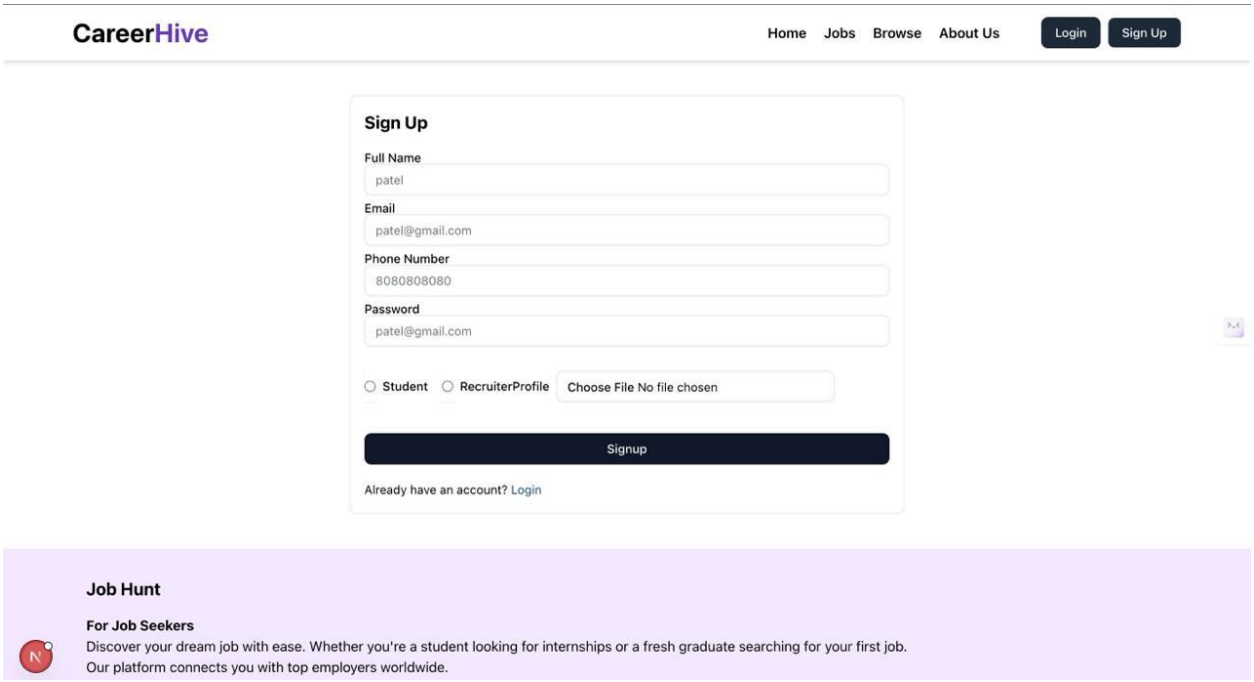


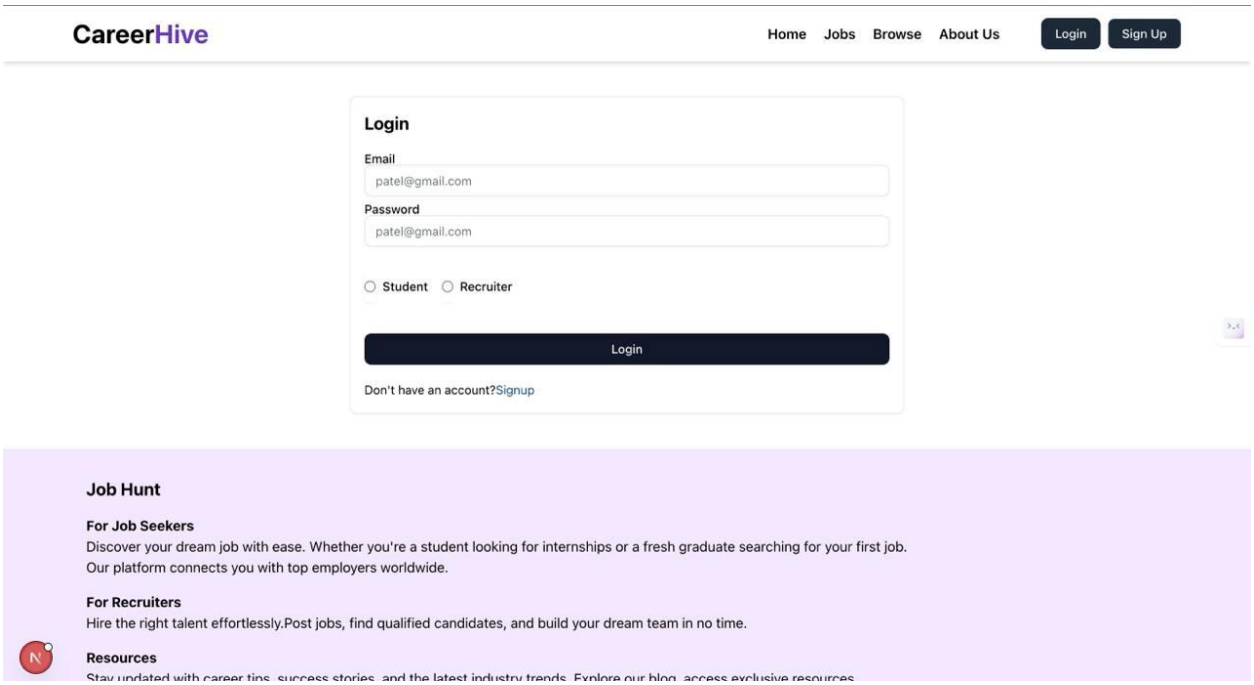
Figure 16: Activity Diagram for Application Status Updates

2.4.4 Prototyping (Wireframes & UI Sketches)

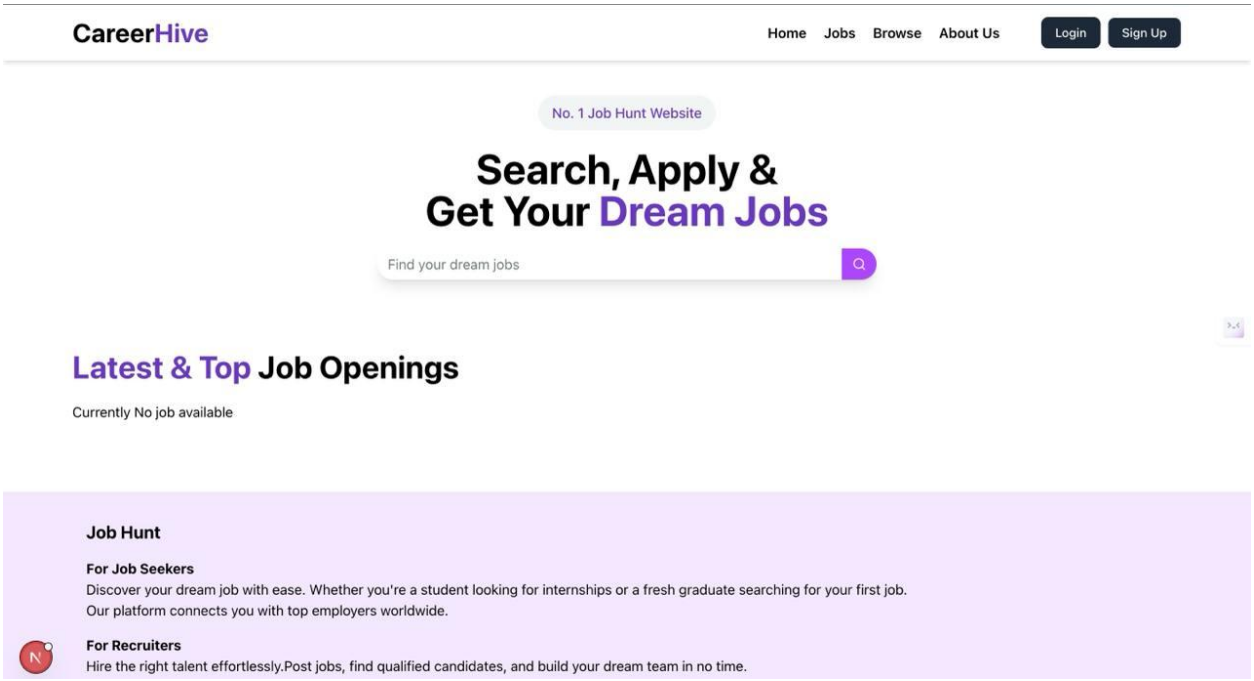
Sign up:



Log in:



Homepage:


CareerHive
Home
Jobs
Browse
About Us
Login
Sign Up
No. 1 Job Hunt Website
Search, Apply &
Get Your Dream Jobs
Find your dream jobs
Latest & Top Job Openings
Currently No job available
Job Hunt
For Job Seekers
Discover your dream job with ease. Whether you're a student looking for internships or a fresh graduate searching for your first job.
Our platform connects you with top employers worldwide.
For Recruiters
Hire the right talent effortlessly.Post jobs, find qualified candidates, and build your dream team in no time.

Chapter-3: Software Design

Software design is the process of defining the architecture, components, interfaces, and data for a software system to satisfy specified requirements. It serves as a blueprint for the development phase, guiding programmers in building efficient, scalable, and maintainable software solutions. Good software design balances user needs, technical constraints, and business goals, ensuring the final product is both functional and adaptable to future changes.

3.1 Architectural Design:

3.1.1 Architectural Context Diagram:

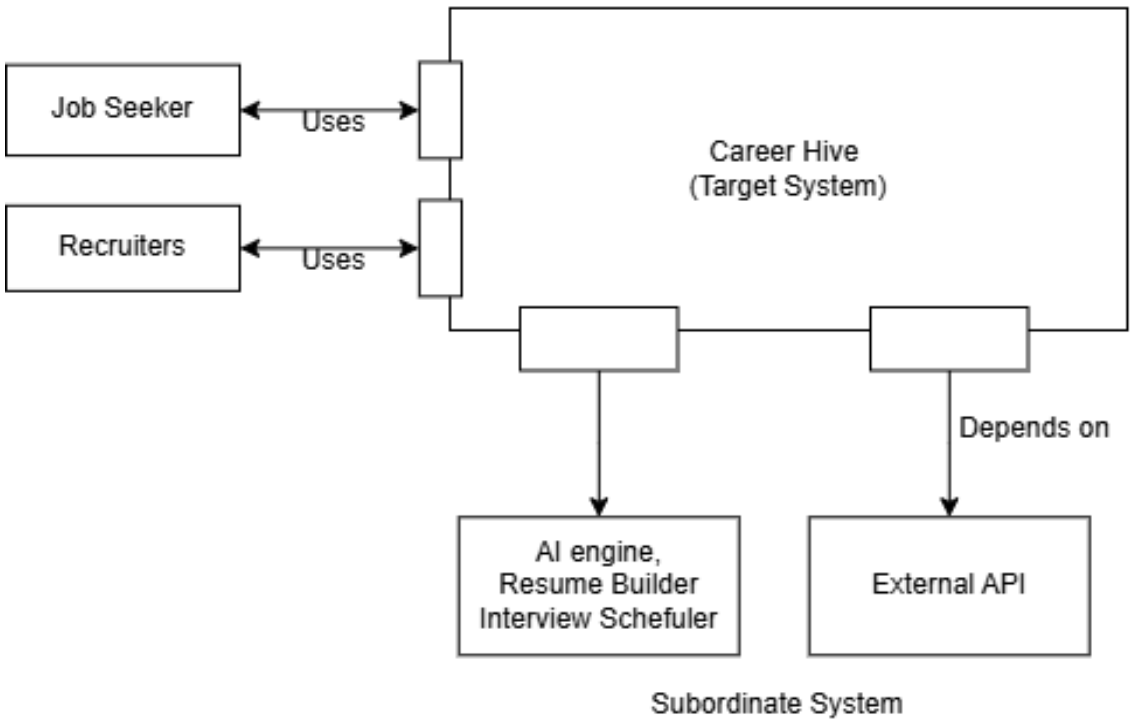


Figure 17: Level 0 for Career Hive

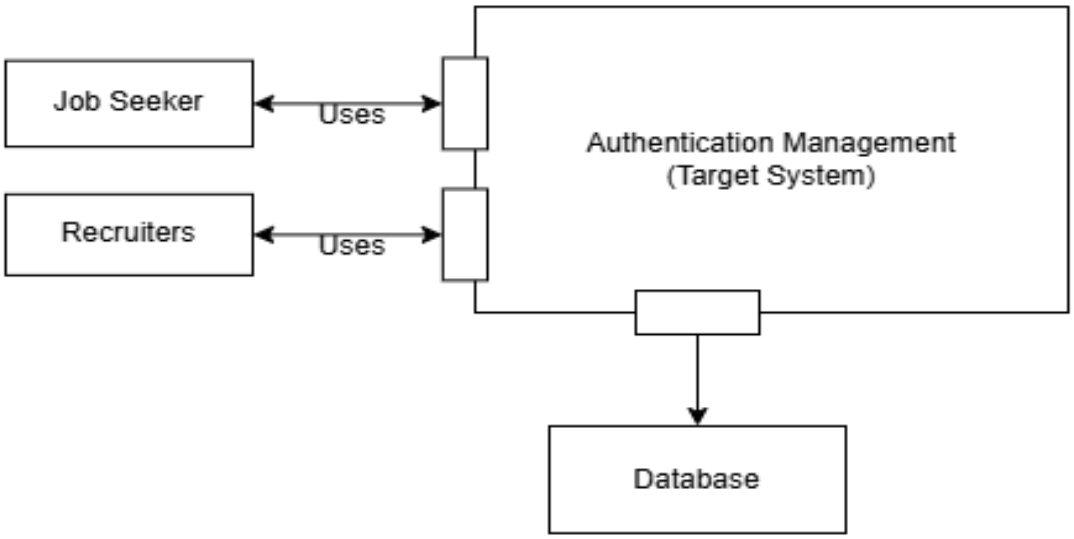


Figure 18: Level 1 for Authentication Management

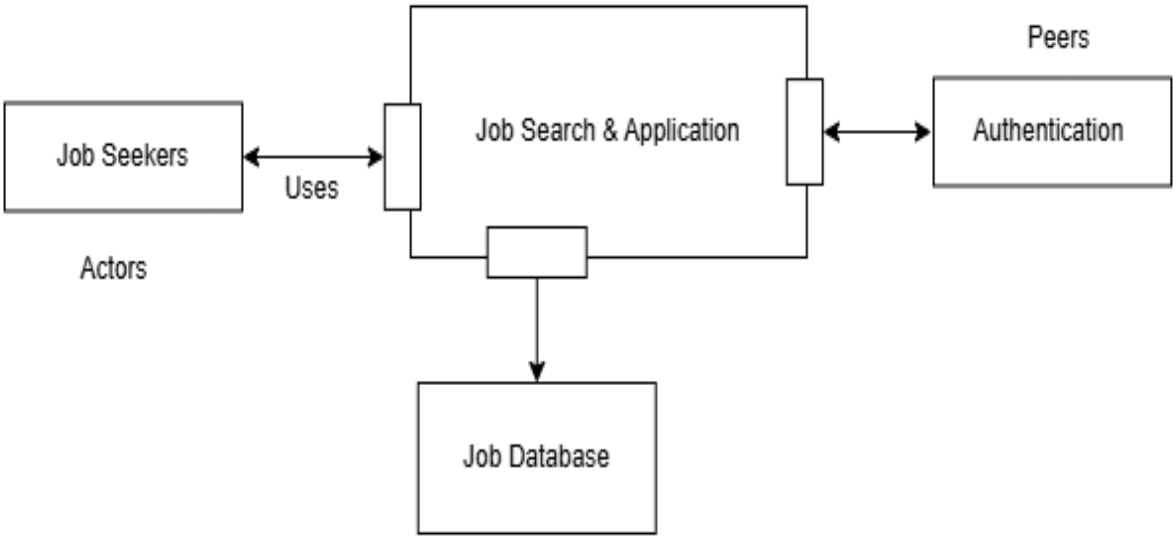


Figure 19: Level 1 for Job search and Application

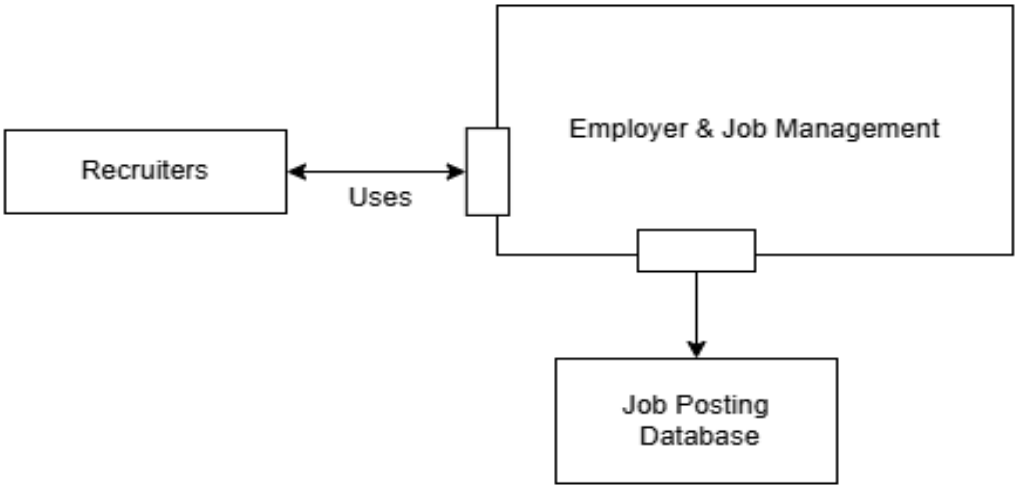


Figure 20: Level 1 for Employer & Job Management

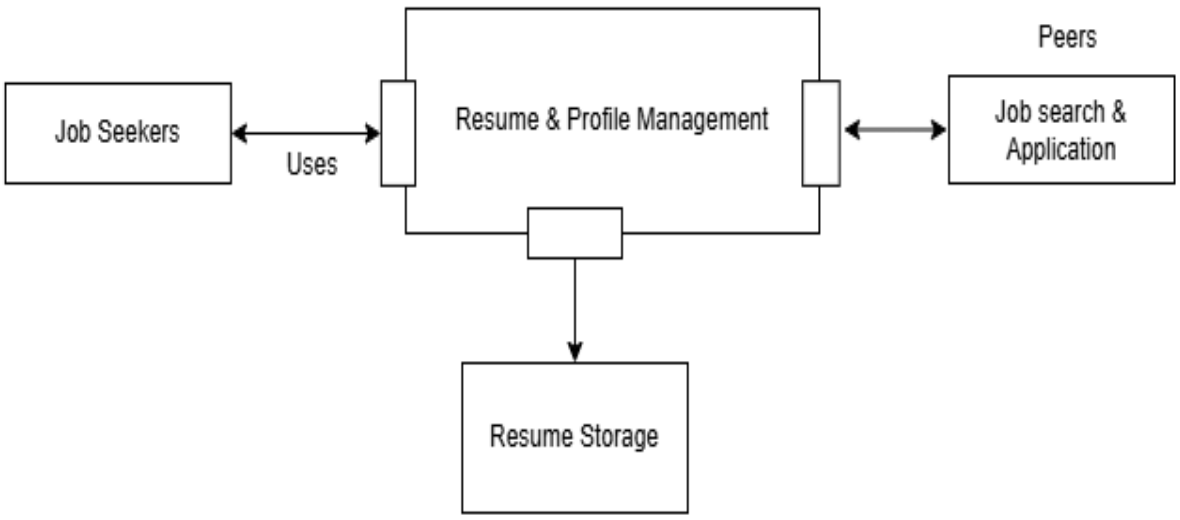


Figure 21: Level 1 for Resume & Profile Management

3.1.2 Top-Level Component Diagram:

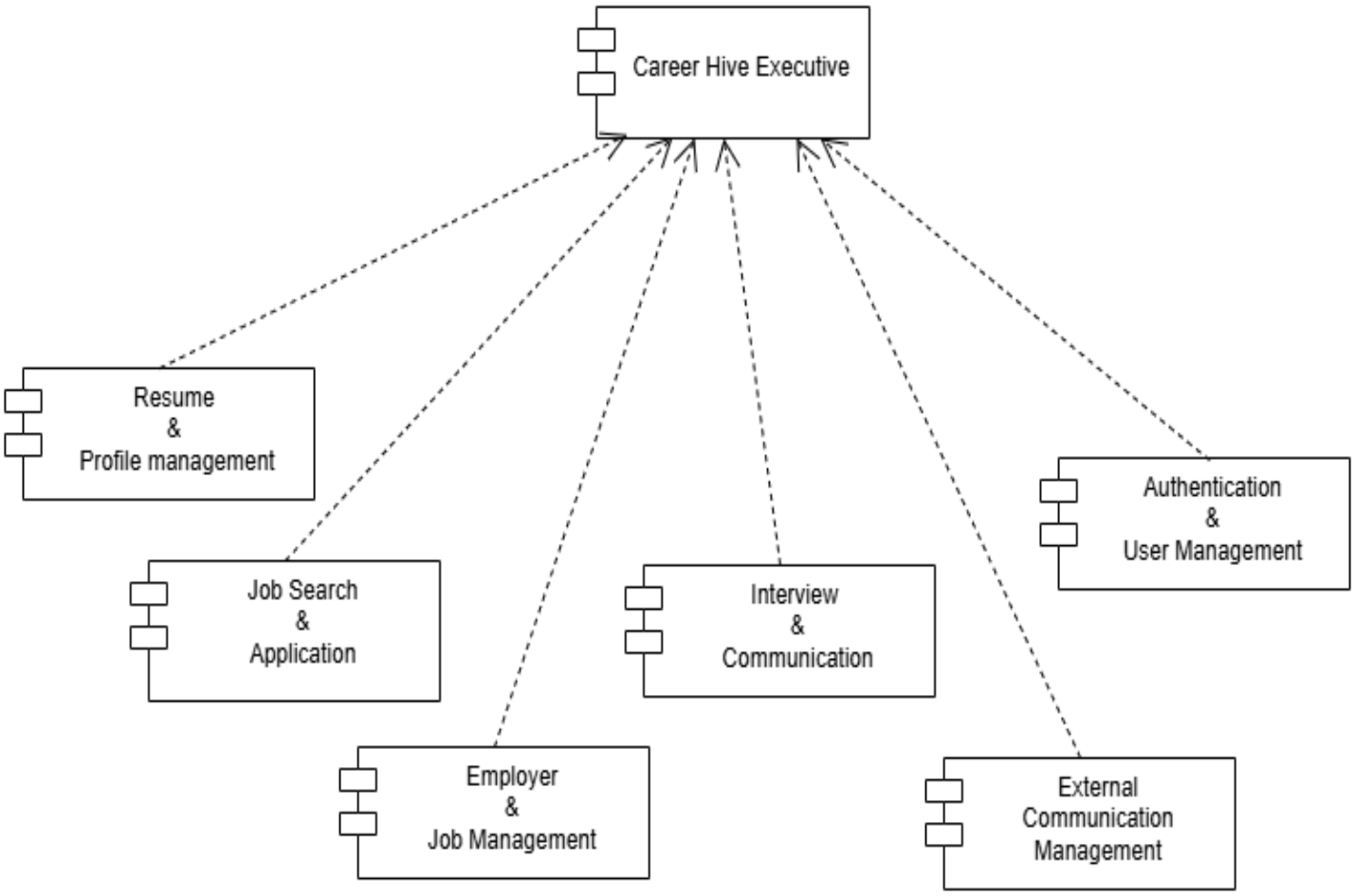


Figure 22: Top-level Component Diagram For Career Hive

3.1.3 Instantiation of Each Component :

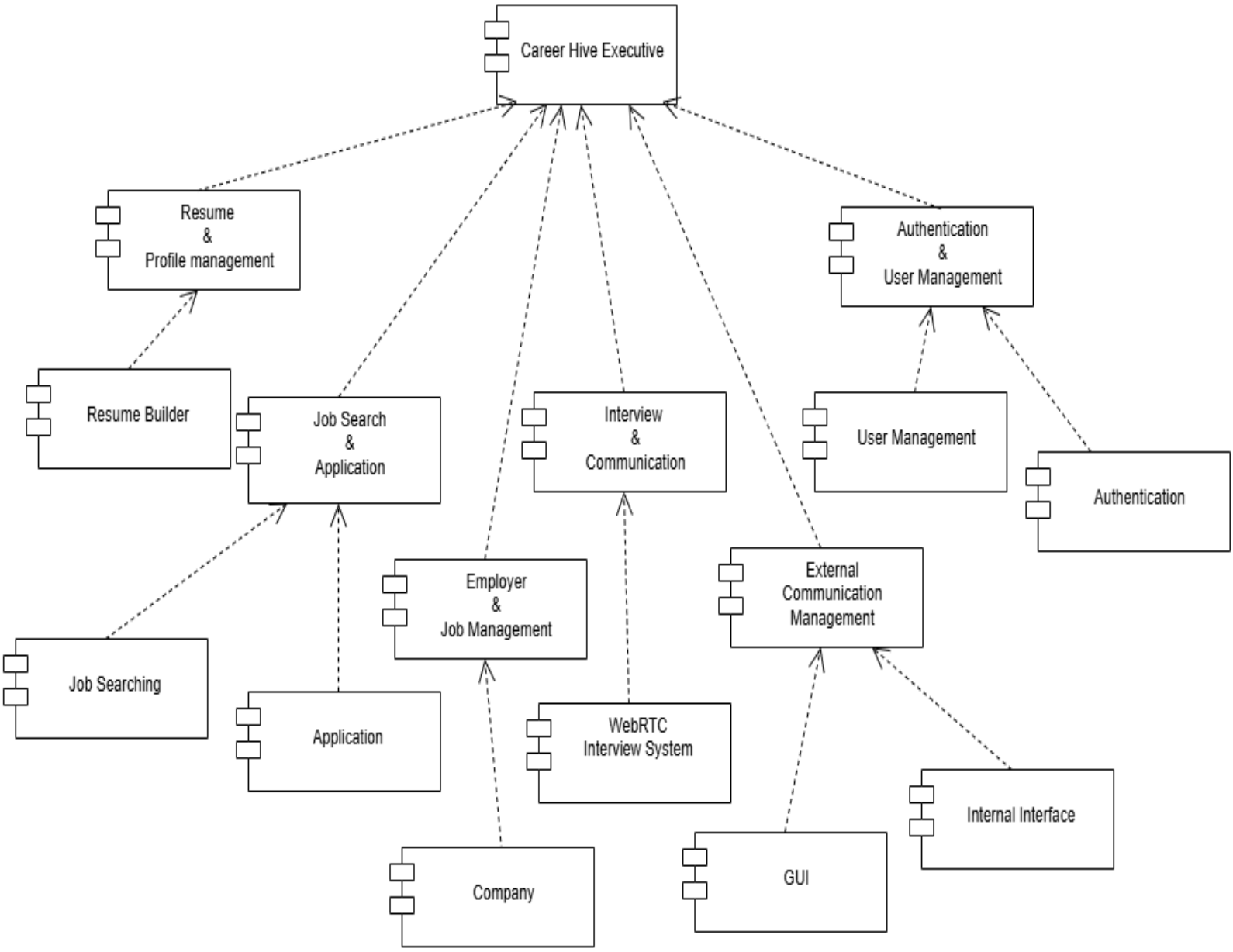


Figure 23: Instantiation of each Component Diagram For Career Hive

3.2 Component-Level Design:

3.2.1 Elaboration of Design Components:

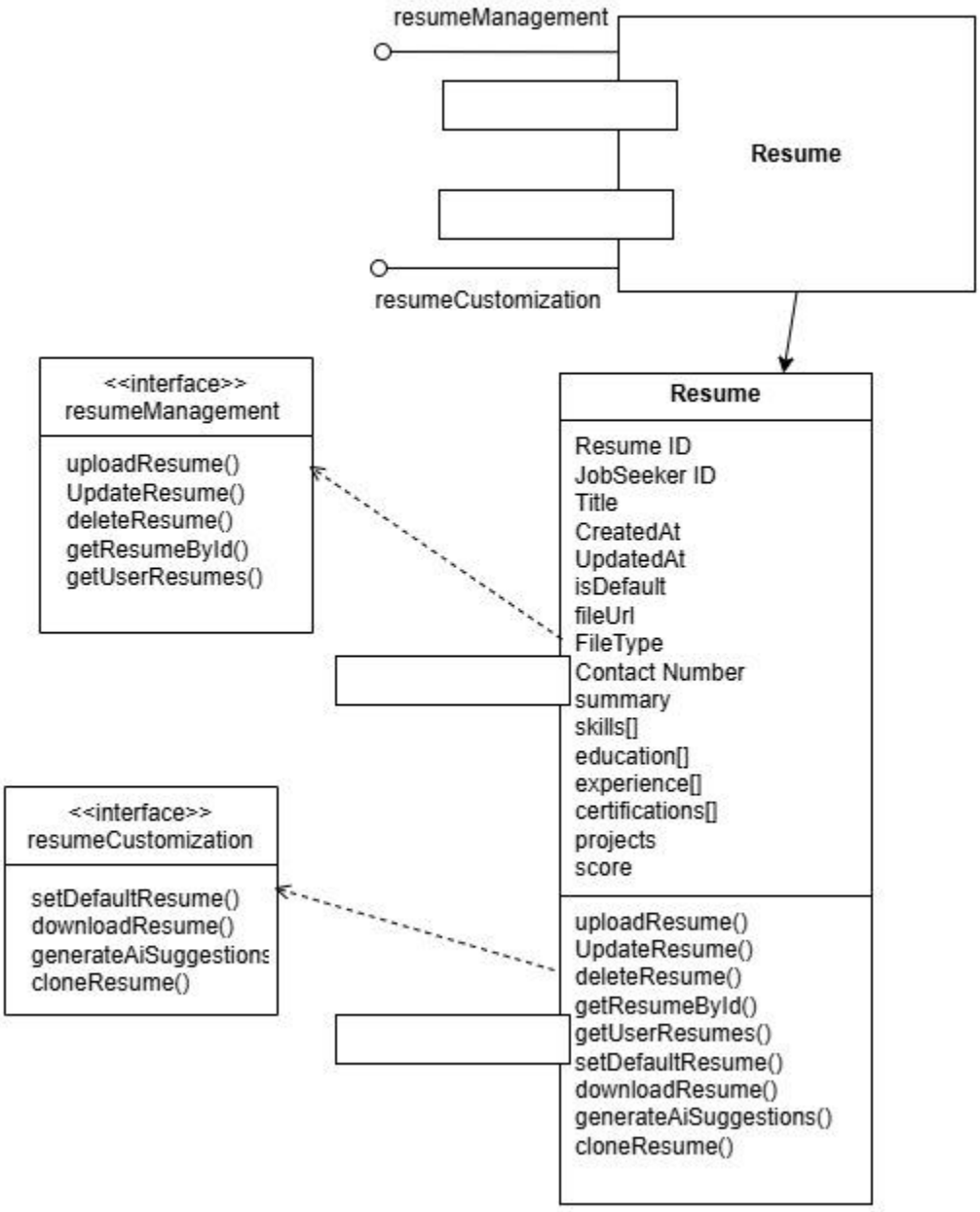


Figure 24: Elaboration Diagram of Resume For Career Hive

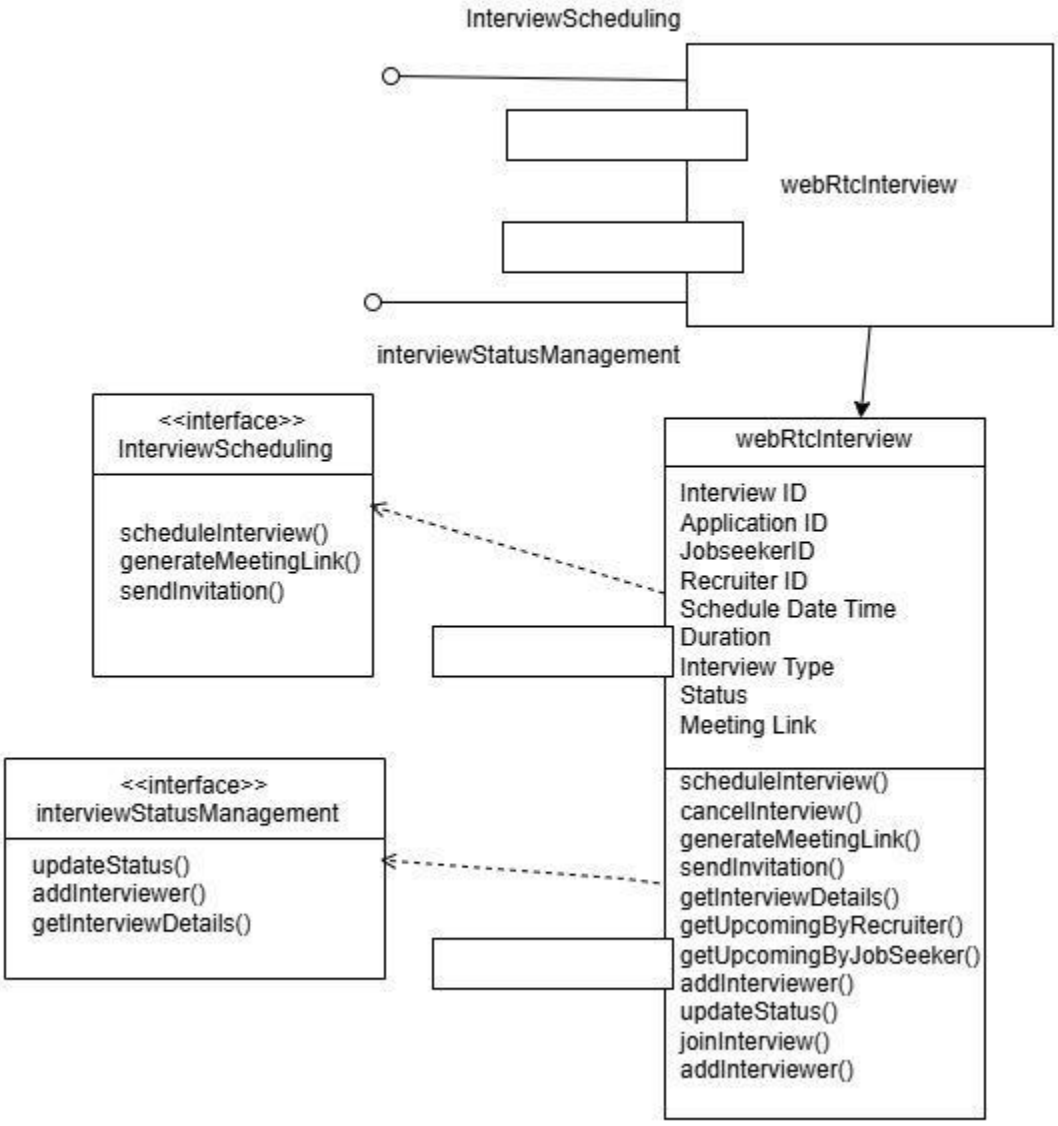


Figure 25: Elaboration Diagram of interview scheduling for Career Hive

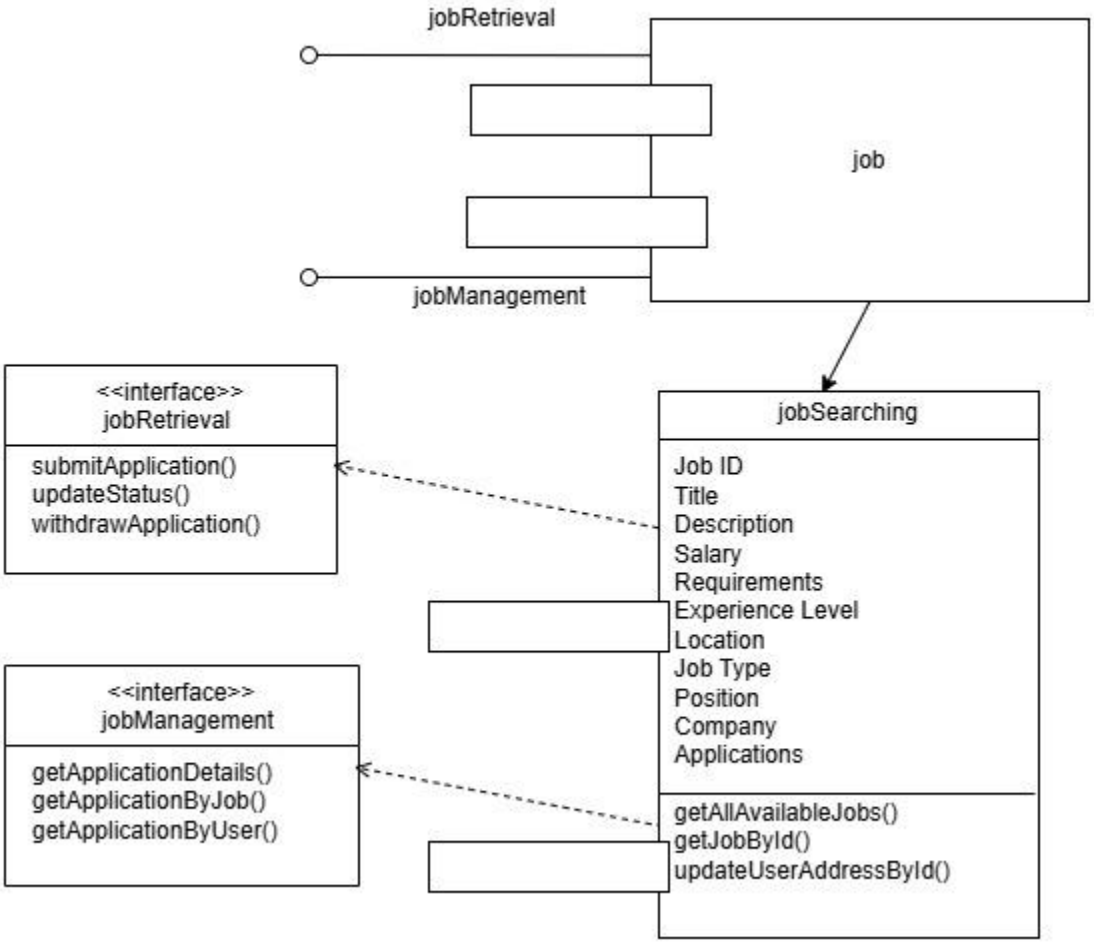


Figure 26: Elaboration Diagram of Job Searching for Career Hive

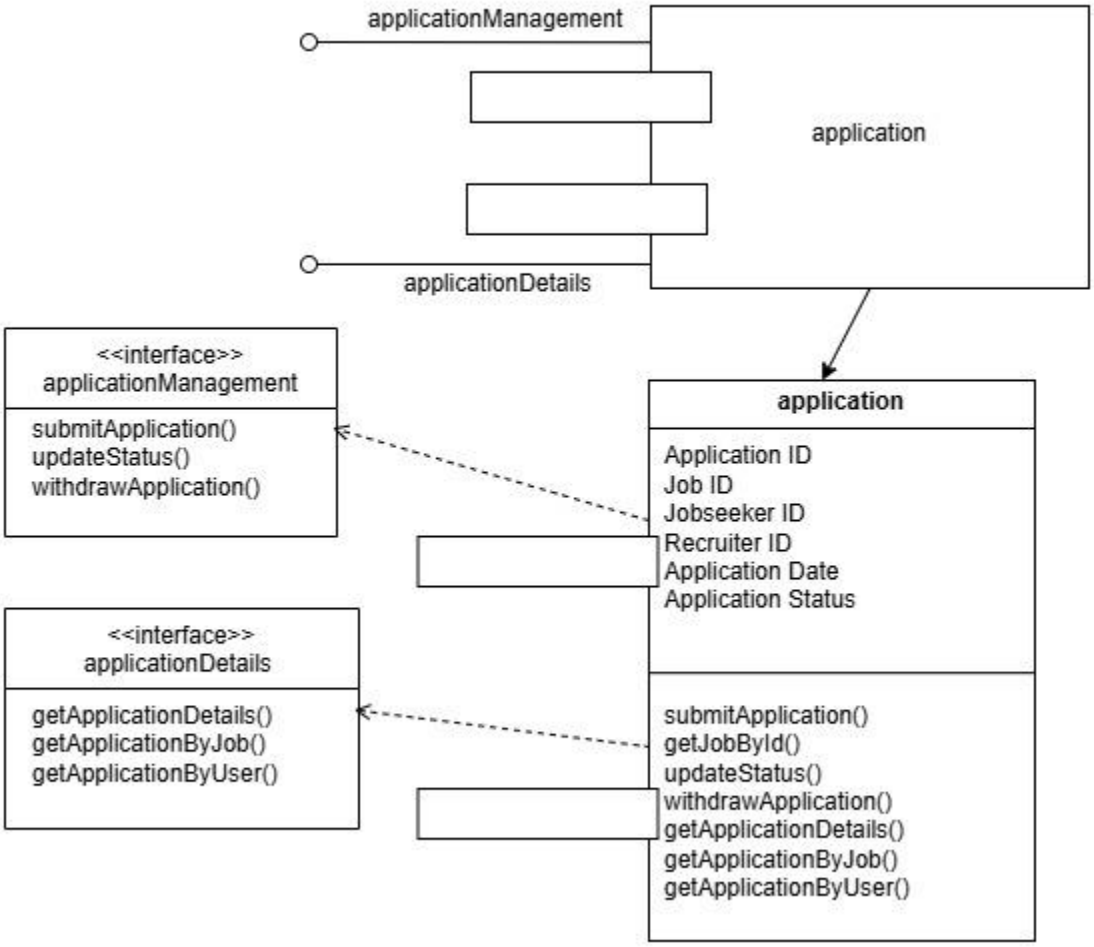


Figure 27: Elaboration Diagram of Application Management for Career Hive

3.2.2 Class Diagram:

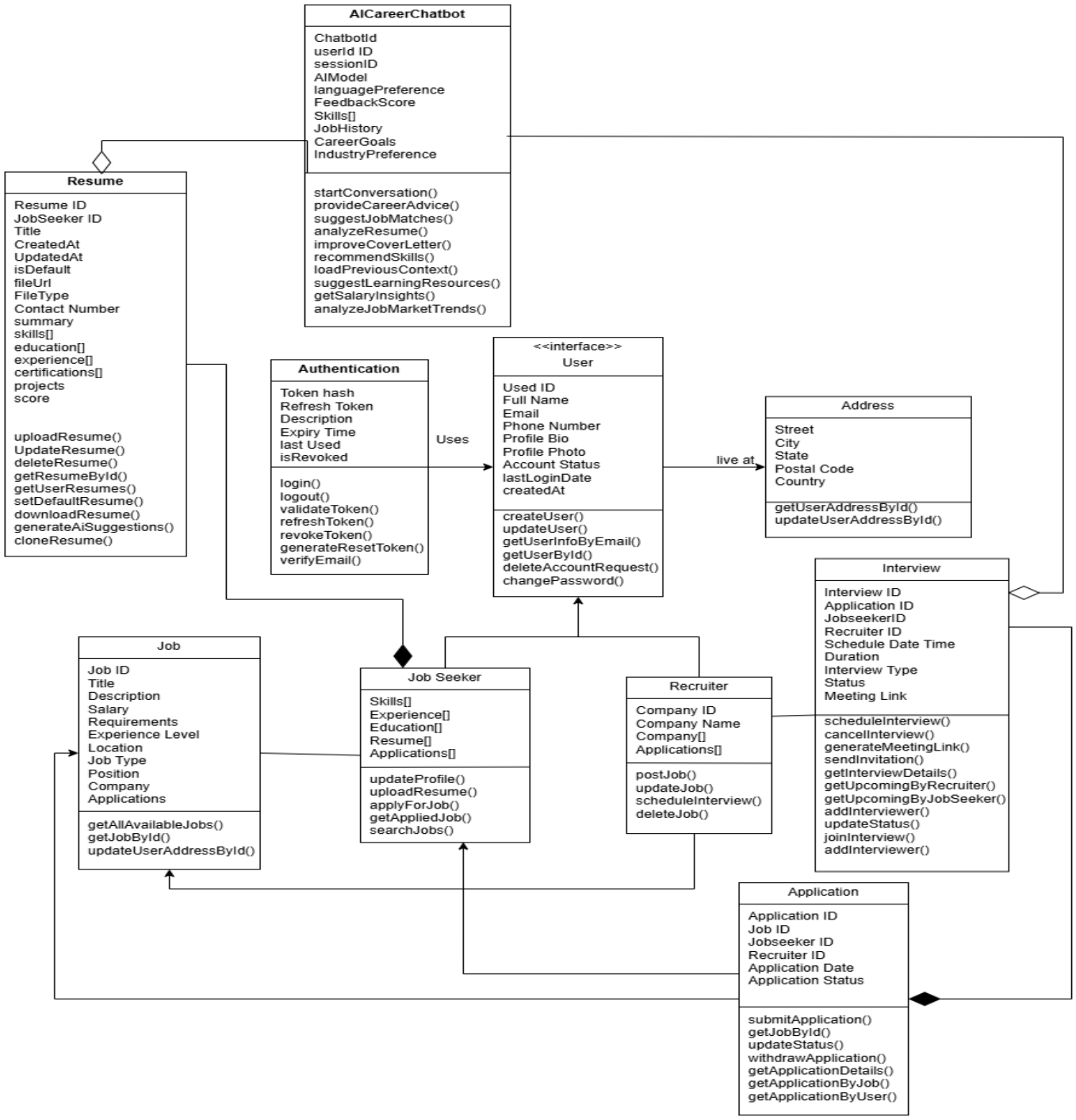


Figure 28: Class Diagram for Career Hive

3.2.3 MongoDB Properties:

Resume collection:

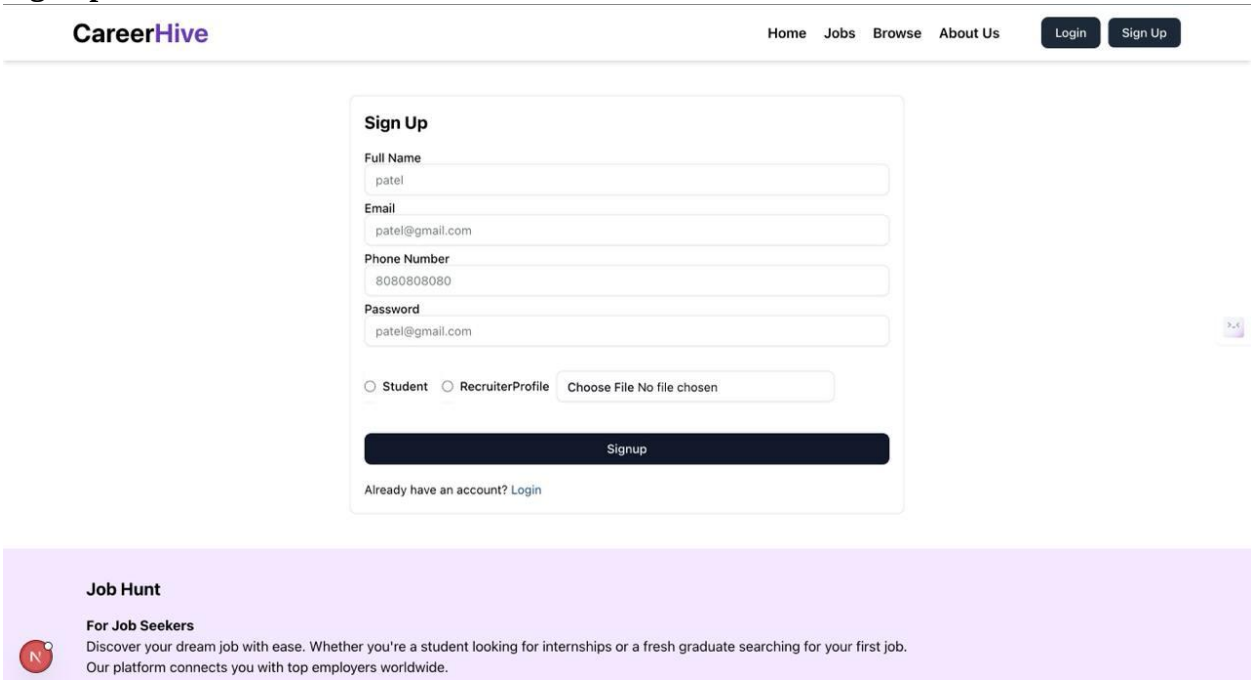
```
{
  "jobSeekerId": "ObjectId",
  "title": "string",
  "createdAt": "date",
  "updatedAt": "date",
  "isDefault": "boolean",
  "fileUrl": "string",
  "fileType": "string",
  "contactNumber": "string",
  "summary": "string",
  "skills": ["string"],
  "education": [
    {
      "degree": "string",
      "institution": "string",
      "startDate": "date",
      "endDate": "date",
      "description": "string"
    }
  ],
  "experience": [
    {
      "jobTitle": "string",
      "companyName": "string",
      "startDate": "date",
      "endDate": "date",
      "description": "string"
    }
  ],
  "certifications": [
    {
      "name": "string",
      "organization": "string",
      "date": "date"
    }
```

```
 ],
 "projects": [
  {
   "name": "string",
   “Description” : ["string"],
   "technologies": "string",
   "url": "string"
  }
 ],
 "score": "number"
}
```

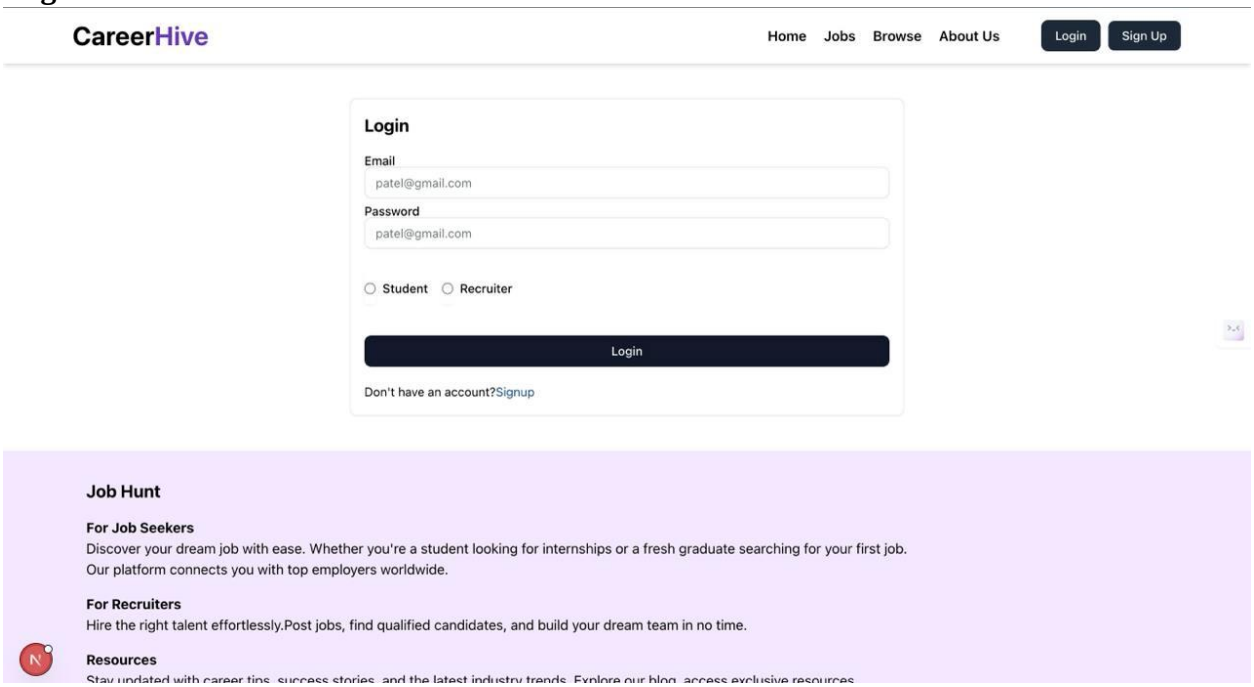
3.3 User Interface Design:

UI Wireframes/Mockups (Prototypes)**:**

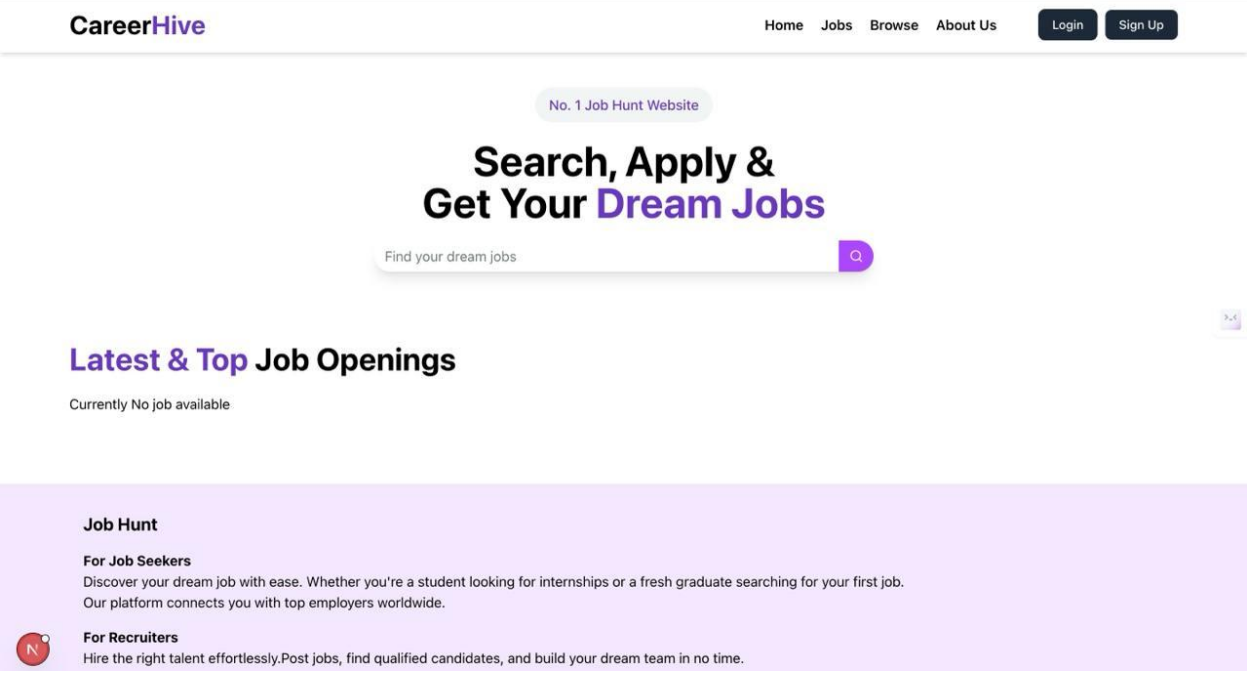
Sign up:



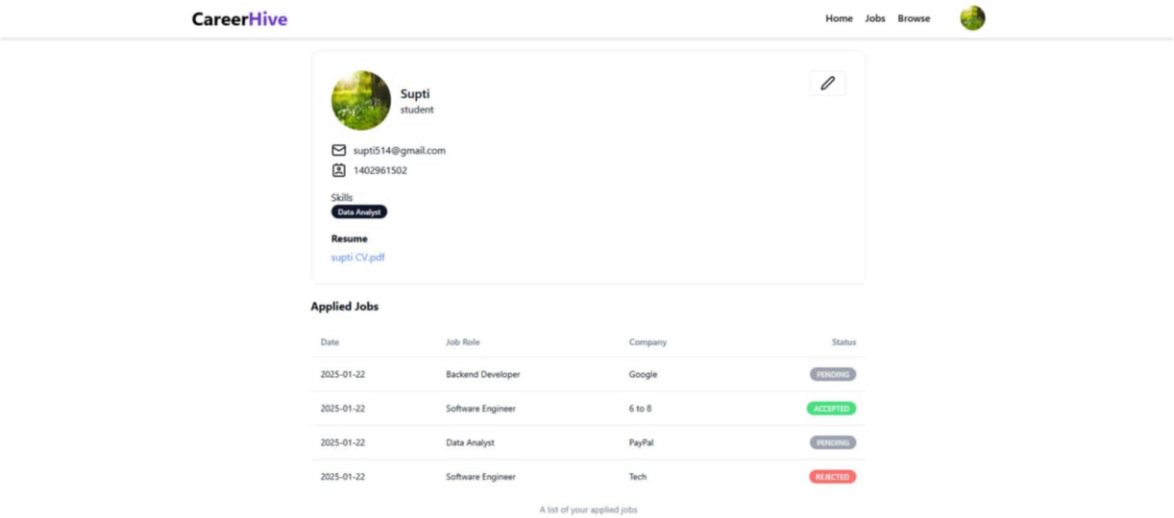
Log in:



Homepage:

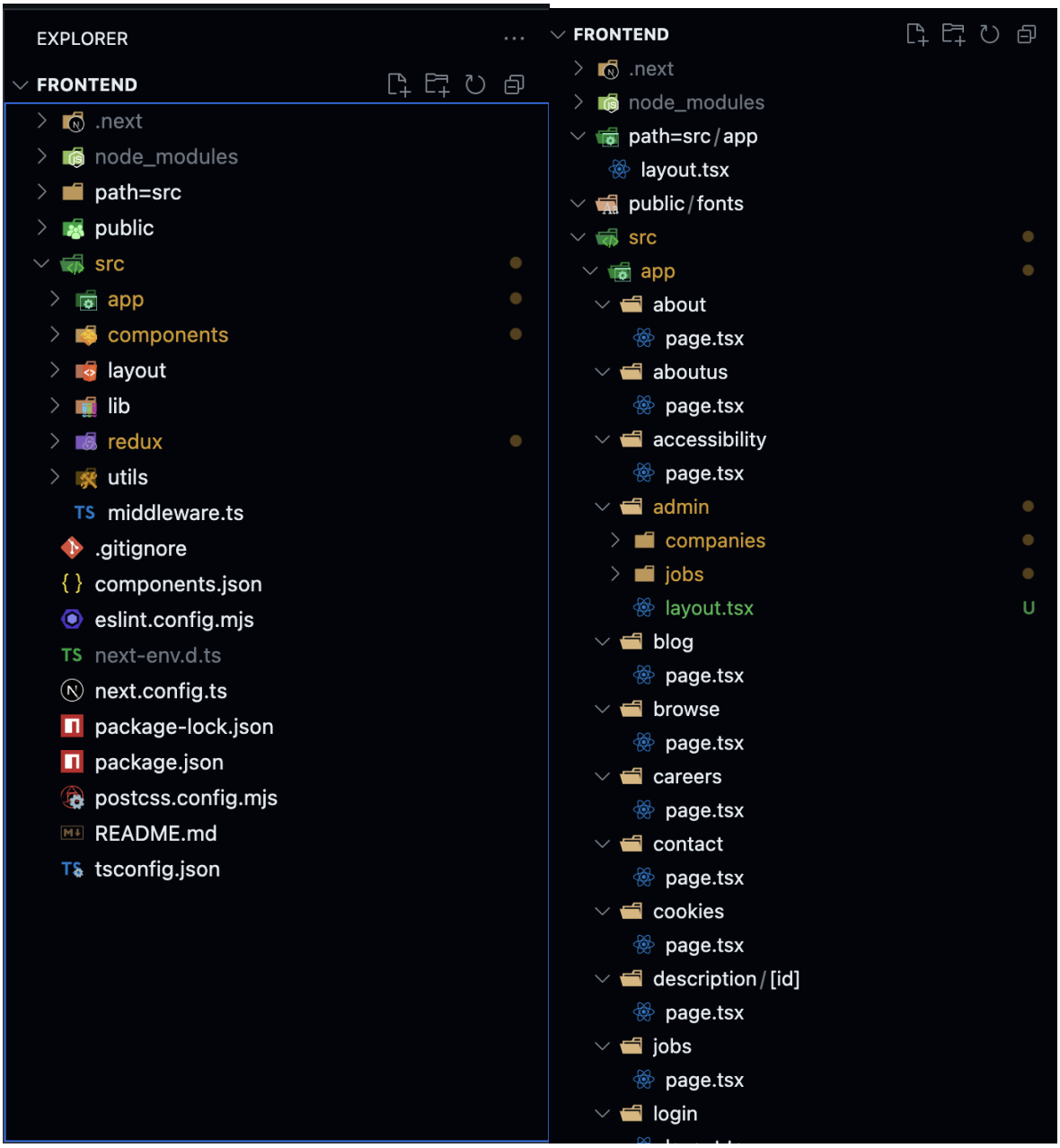


Job Confirmation:

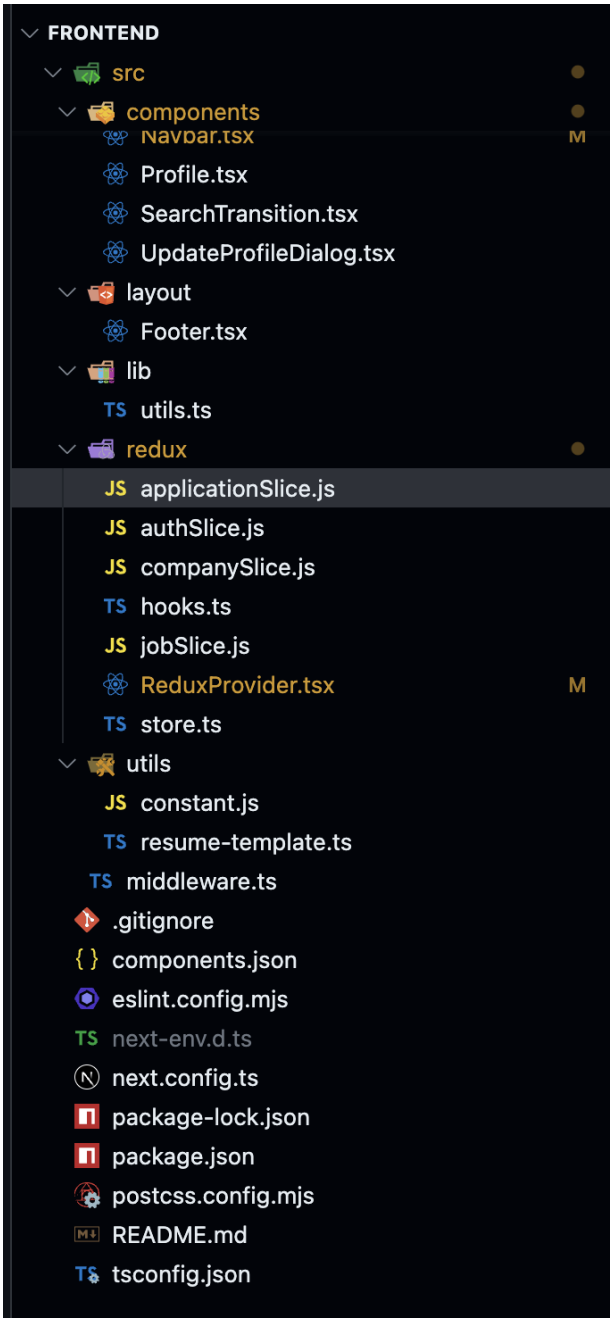


Chapter 4 . Implementation

Frontend:


EXPLORER
FRONTEND
.next
node_modules
path=src
public
src
app
components
layout
lib
redux
utils
middleware.ts
.gitignore
components.json
eslint.config.mjs
next-env.d.ts
next.config.ts
package-lock.json
package.json
postcss.config.mjs
README.md
tsconfig.json
FRONTEND
.next
node_modules
path=src/app
layout.tsx
public/fonts
src
app
about
page.tsx
aboutus
page.tsx
accessibility
page.tsx
admin
companies
jobs
layout.tsx
U
blog
page.tsx
browse
page.tsx
careers
page.tsx
contact
page.tsx
cookies
page.tsx
description/[id]
page.tsx
jobs
page.tsx
login


FRONTEND
src
app
login
layout.tsx
page.tsx
privacy
page.tsx
profile
page.tsx
resume-builder
page.tsx M
salary
page.tsx
signup
layout.tsx
page.tsx M
terms
globals.css
layout.tsx M
page.tsx M
components
admin
AdminJobs.tsx
AdminJobsTable.tsx
ProtectRoute.tsx M
applicants
Applicants.tsx
ApplicantsTable.tsx
company
Companies.tsx
CompaniesTable.tsx
CompanyCreate.tsx
FRONTEND
src
components
company
CompanySetup.tsx
hooks
useGetAllAdminJobs.tsx
useGetAllCompanies.tsx
useGetAllJobs.tsx M
useGetAppliedJobs.tsx
useGetCompanyById.tsx
useGetLatestJobs.tsx U
job
JobDescription.tsx
ui
AppliedJobTable.tsx
Browse.tsx M
CategoryCarousel.tsx
FilterCard.tsx
Footer.tsx
~/Documents/Me_and_Myself/Projects/job-portal
HomeFooter.tsx
Job.tsx
Jobs.tsx
LandingHome.tsx M
LatestJobCards.tsx
LatestJobs.tsx
Navbar.tsx M
Profile.tsx
SearchTransition.tsx
UpdateProfileDialog.tsx
layout
Footer.tsx


FRONTEND
src
components
Navbar.tsx
M
Profile.tsx
SearchTransition.tsx
UpdateProfileDialog.tsx
layout
Footer.tsx
lib
utils.ts
redux
applicationSlice.js
authSlice.js
companySlice.js
hooks.ts
jobSlice.js
ReduxProvider.tsx
M
store.ts
utils
constant.js
resume-template.ts
middleware.ts
.gitignore
components.json
eslint.config.mjs
next-env.d.ts
next.config.ts
package-lock.json
package.json
postcss.config.mjs
README.md
tsconfig.json

Files and implementations of FrontEnd:

Root Directory

- **package.json**
  Defines project metadata, dependencies, and scripts.

- **package-lock.json**
  Auto-generated lockfile ensuring consistent dependency versions.

- **next.config.ts**
  Configuration for customizing Next.js behavior.

- **tsconfig.json**
  TypeScript compiler settings and path aliases.

- **components.json**
  Likely a registry or metadata file for dynamic component handling.

- **README.md**
  Overview, setup instructions, and usage guidelines for the project.

- **postcss.config.mjs**
  Configures PostCSS for CSS transformation and plugins.

- **next-env.d.ts**
  Type definitions to support Next.js development in TypeScript.

- **.gitignore**
  Lists files and folders to be ignored by Git.

- **eslint.config.mjs**
  ESLint configuration for maintaining code quality.

public/

- **fonts/**
  Contains custom fonts used in the UI.

src/ - Application Source Code

src/app/ — Pages & Routing

- **layout.tsx**
  Root layout wrapper for the app (used in all routes).

- **page.tsx**
  Entry point or landing page of the application.

- **globals.css**
  Global stylesheet applied across all components.

Auth & User Flows

- **login/page.tsx**
  Login form and authentication logic.

- **signup/page.tsx**
  Registration form and user creation.

User Features

- **profile/page.tsx**
  Displays and manages user profile.

- **resume-builder/page.tsx**
  Resume generation tool/interface for users.

Jobs

- **jobs/page.tsx**
  Job listing or dashboard for users.

src/app/admin/ — Admin Interface

- **layout.tsx**
  Shared layout for all admin pages.

- **jobs/page.tsx**
  Admin dashboard for viewing all jobs.

- **jobs/edit/[id]/page.tsx**
  Edit specific job details by ID.

- **jobs/create/page.tsx**
  Create new job listings.

- **jobs/[id]/applicants/page.tsx**
  View list of applicants for a given job.

- **companies/page.tsx**
  Admin panel to manage companies.

- **companies/create/page.tsx**
  Form to create a new company.

- **companies/[id]/page.tsx**
  View or edit a specific company's details.

src/components/ — UI & Feature Components

General UI Sections

- **LandingHome.tsx, HeroSection.tsx, HomeFooter.tsx, Footer.tsx, Navbar.tsx**
  Components for home page layout and navigation.

Job Search & Listings

- **Browse.tsx, Jobs.tsx, Job.tsx, LatestJobs.tsx, LatestJobCards.tsx**
  Job cards, latest postings, and listings UI.

- **FilterCard.tsx, CategoryCarousel.tsx, SearchTransition.tsx**
  UI elements for job filtering and animated transitions.

User Profile

- **Profile.tsx, UpdateProfileDialog.tsx**
  Components for displaying and updating user info.

- **AppliedJobTable.tsx**
  Lists all jobs a user has applied to.

Admin Components

- **admin/***
  Job management tables, admin tools, and access protection.

Applicant Handling

- **applicants/***
  Tables and components to manage job applicants.

Company Features

- **company/***
  Company creation forms, listing views, and setup steps.

Job Details

- **job/***
  Job description and detail views.

src/hooks/ — Data Fetching & Custom Logic

- **Custom hooks for jobs, applicants, companies, etc.**
  Interact with APIs, handle fetching logic, and reuse stateful logic.

src/ui/ — Reusable UI Primitives

- **Buttons, tables, tabs, dialogs, popovers, carousels**
  Standardized UI elements used across the app for consistency and reusability.

src/layout/

- **Footer.tsx**
  Global footer used across page layouts.

src/redux/ — State Management with Redux

- **ReduxProvider.tsx**
  Provides Redux context to the entire app.
- **store.ts**
  Configures Redux store with middleware and reducers.
- **authSlice.js**
  Manages login, logout, and authentication state.
- **jobSlice.js**
  Handles state related to job listings.
- **applicationSlice.js**
  Tracks user applications and statuses.
- **companySlice.js**
  Stores company-related data and UI flags.
- **hooks.ts**
  Typed versions of useDispatch and useSelector.

src/utils/ — Helpers & Constants

- **resume-template.ts**
  Template layout and logic for building resumes.
- **constant.js**
  Shared constants like job categories, filters, or status codes.

src/lib/

- **utils.ts**
  Miscellaneous utility functions used across the project.

Key Features & Interactions

Authentication — User login, logout, and registration flows are handled via dedicated pages and Redux state.

Job System — Users can browse, search, and apply to jobs; admins can create, edit, and manage jobs and applicants.

Company Management — Admins and company users can create, update, and manage company data.

Resume Building — Users can generate resumes through a custom builder.

Admin Dashboard — Complete control panel for managing jobs, companies, and user applications.

State Management — Redux tracks jobs, companies, users, and applications efficiently.

Reusable UI — Built on modular components with shared design and utility functions.

User Flows

- **Authentication: Registration, login, profile management**
- **Job Search: Browse, filter, search, and apply to jobs**
- **Application Tracking: View applied jobs and application status**

Admin Capabilities

- **Job Management: Create, edit, and manage job postings**
- **Company Administration: Manage company profiles and settings**
- **Applicant Review: View and process job applications**

Technical Highlights

- **Modern Stack: Next.js 14+, TypeScript, Redux Toolkit**
- **Component Architecture: Reusable UI components with consistent design**
- **State Management: Centralized Redux store with type-safe hooks**

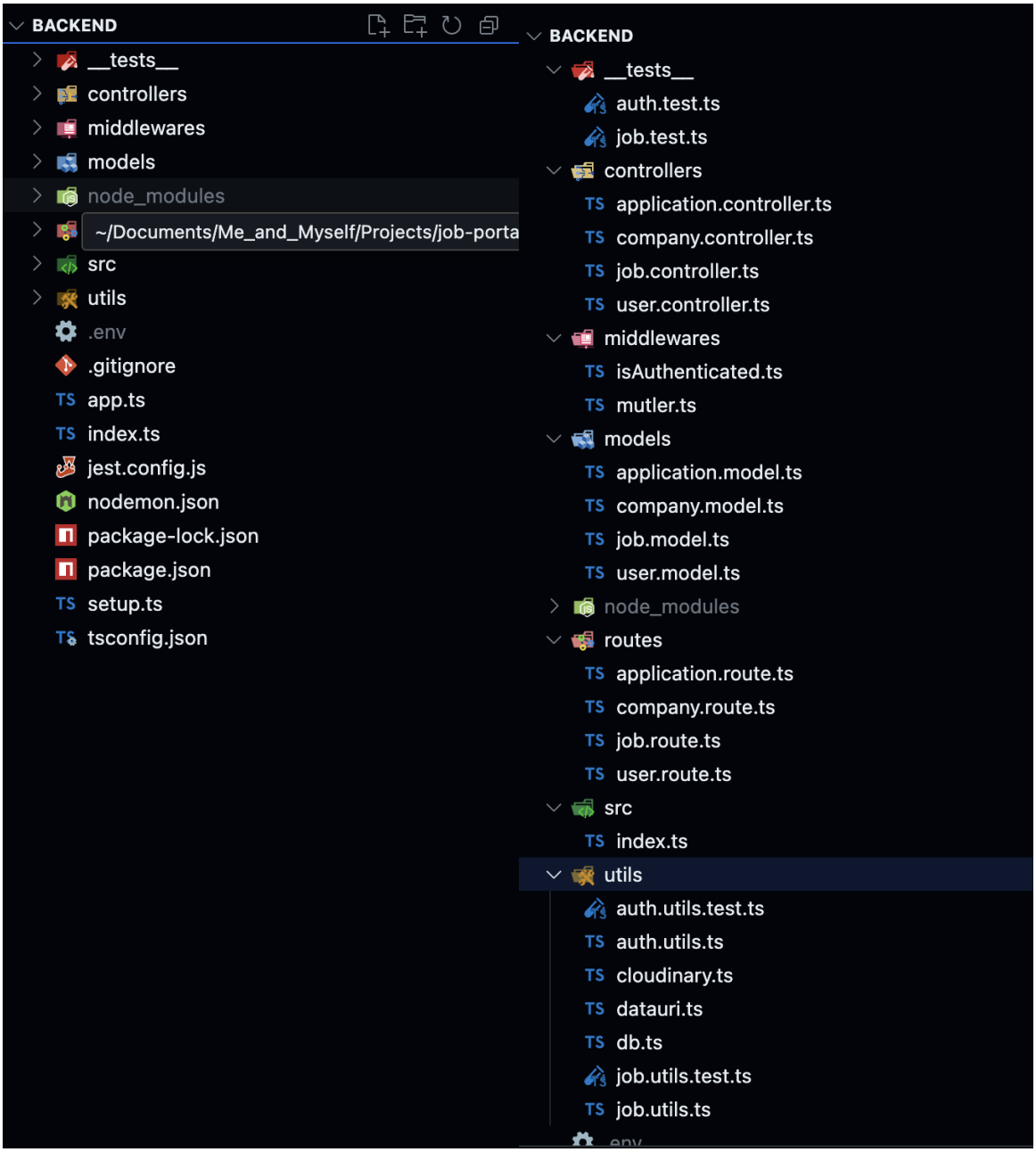
- **API Integration: Custom hooks for clean data fetching**
- **Responsive Design: Mobile-first UI components**

Development Workflow

The project uses modern development practices:

- **TypeScript for type safety**
- **ESLint for code quality**
- **PostCSS for advanced CSS processing**
- **Next.js App Router for optimal performance**
- **Component-based architecture for maintainability**

Backend Folder Structure, Files and Implementation:



Root Directory

- **index.ts**
  Entry point of the application. Initializes the Express server and connects to the MongoDB database.

- **app.ts**
  Sets up the Express application with middleware (JSON parsing, cookie handling, CORS) and registers all API routes. Acts as the central hub for handling incoming HTTP requests.

- **setup.ts**
  Configures an in-memory MongoDB server used during automated testing. Handles initialization and cleanup of the test environment.

- **jest.config.js**
  Configuration file for the Jest testing framework. Specifies environment settings, file patterns, and setup scripts.

- **package.json**
  Manages project dependencies and scripts for development, testing, and production builds. Covers both backend and frontend operations.

- **tsconfig.json**
  TypeScript configuration file. Defines compiler options, module resolution strategies, and file inclusion/exclusion rules.

- **nodemon.json**
  Nodemon configuration for automatic server restarts on file changes. Monitors TypeScript files while ignoring node_modules.

- **.gitignore**
  Lists files and directories to be excluded from version control.

src/

- **src/index.ts**
  Secondary entry point, potentially intended for modularization or future expansion.

controllers/

- **job.controller.ts**
  Contains logic for creating, retrieving, updating, and deleting job listings. Supports admin-specific and keyword-based job queries.

- **company.controller.ts**
  Manages operations for company accounts, including registration, updates, profile information, and logo uploads.

- **application.controller.ts**
  Handles job applications, including submission, listing applied jobs, viewing applicants, and updating application statuses.

- **user.controller.ts**
  Manages user registration, login, logout, and profile updates. Includes logic for authentication, password hashing, and profile photo handling.

middlewares/

- **isAuthenticated.ts**
  Middleware to verify user authentication via JWT stored in cookies. Protects private routes.

- **mutler.ts**
  Configures Multer for in-memory file uploads, typically used for uploading images such as profile photos or company logos.

models/

- **user.model.ts**
  Defines the schema for user accounts, covering both student and recruiter roles. Includes authentication details and company associations.

- **application.model.ts**
  Schema for job applications, linking users to job postings and tracking application statuses.

- **company.model.ts**
  Schema defining company profiles, including details such as logo and linked user account.

- **job.model.ts**
  Schema for job listings, including job requirements, associated company, and linked applications.

routes/

- **job.route.ts**
  Defines RESTful API endpoints for job-related operations (CRUD, admin routes). Secured via authentication middleware.

- **company.route.ts**
  Provides API endpoints for managing company data. Protected by authentication middleware.

- **application.route.ts**
  Implements endpoints for job application operations. Secured by authentication middleware.

- **user.route.ts**
  API endpoints for user management operations such as registration, login, logout, and profile editing.

utils/

- **job.utils.ts**
  Utility functions for job processing, including parsing and data normalization.

- **auth.utils.ts**
  Utility functions for authentication tasks such as token generation and validation.

- **auth.utils.test.ts**
  Unit tests for authentication utility functions.

- **job.utils.test.ts**
  Unit tests for job utility functions.

- **cloudinary.ts**
  Cloudinary configuration for image and file uploads.

- **datauri.ts**
  Utility to convert file buffers into Data URI format, used for file uploads.

- **db.ts**
  Contains logic for connecting to the MongoDB database.

tests_/

- **job.test.ts**
  Automated test cases for job-related API endpoints and controller logic.

- **auth.test.ts**
  Automated test cases for user authentication endpoints and logic.

Summary

This project implements a robust backend for a job portal application. It supports key features including user authentication, job posting and management, company profiles, and job applications. The codebase is cleanly organized following standard MVC practices, with a clear separation of concerns among controllers, models, routes, middlewares, and utilities. The project includes automated testing for core functionalities and is structured to support future scaling and maintenance.

Chapter 5. Software Testing

Software testing is the process of evaluating a software application to ensure it functions correctly, meets specified requirements, and is free from defects. It involves executing the software to identify bugs, verify performance, and ensure reliability. Testing can be manual or automated and includes various levels such as unit, integration, system, and acceptance testing. The main goal is to deliver a high-quality, error-free product that meets user expectations.

5.1 White Box Testing

White box testing involves examining the internal workings of the software components. In this phase, unit tests will be performed on at least two classes by creating test cases derived from their control flow graphs. This approach ensures that all independent execution paths are thoroughly tested for correctness and robustness.

For the Career Hive project, we have applied white box testing specifically on two key classes:Auth.utils.test.ts,job.utils.test.ts:. These classes handle critical functionalities—user registration and job searching respectively—and thorough testing helps guarantee the reliability and accuracy of these components.

5.1.1 Unit Testing on Two Classes

Auth.utils.test.ts:

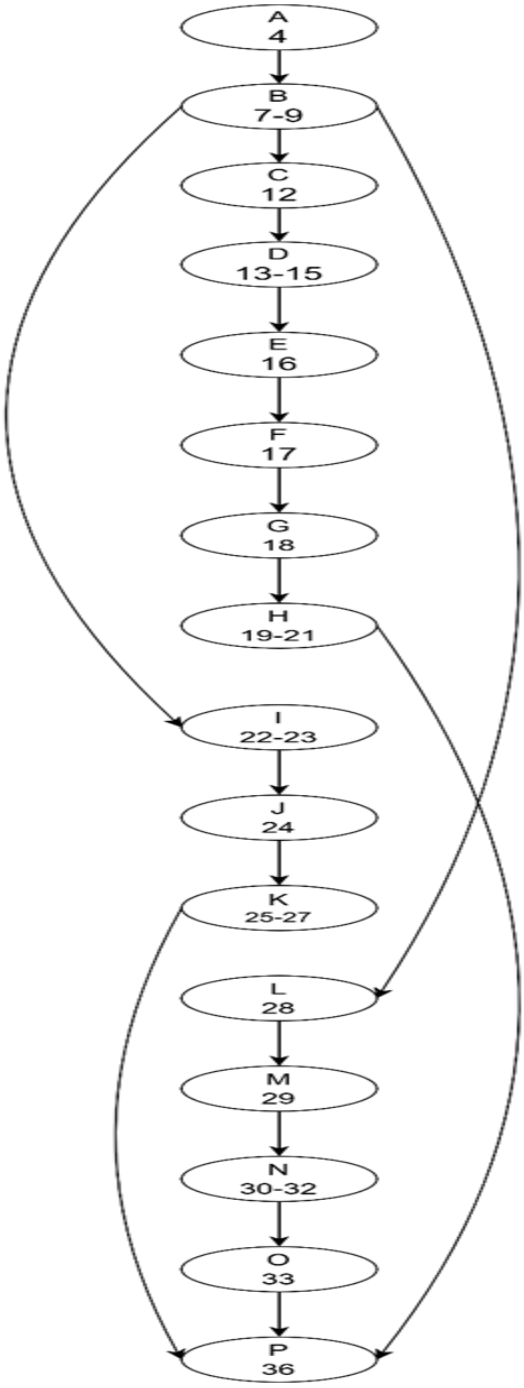


Figure 29: Control Flow Graph for Authentication

Cyclomatic Complexity Calculation:

Edge = 17 , Node =16 , P = 1

Cyclomatic Complexity = e-n+2p

=17-16+2

=3

Since the cyclomatic complexity of the graph is 3, there will be 3 independent paths in the graph, as shown below:

1: A – B – C – D – E – F – G – H – P

2: A – B – I – J – K – P

3: A – B – L – M – N – O – P

Test Case Design from the Independent Paths

Table 3: Test Case Design for Authentication

Test Case Id	Input Details	Expected Output	Independent Paths
TC1	User provides a valid userId and the SECRET_KEY is correctly set in the environment	A valid JWT token is generated. When decoded, it contains the correct userId, issued time (iat), and expiration time (exp)	A → B → C → D → E → F → G → H → P
TC2	User attempts to generate a token without setting the SECRET_KEY in the environment	The system throws an error indicating that SECRET_KEY is required to generate the token	A → B → I → J → K → P
TC3	User generates a token, then decodes it to check the expiration interval	The decoded token has an expiration time (exp) approximately 10 hours after the issued time (iat)	A → B → L → M → N → O → P

job.utils.test.ts:

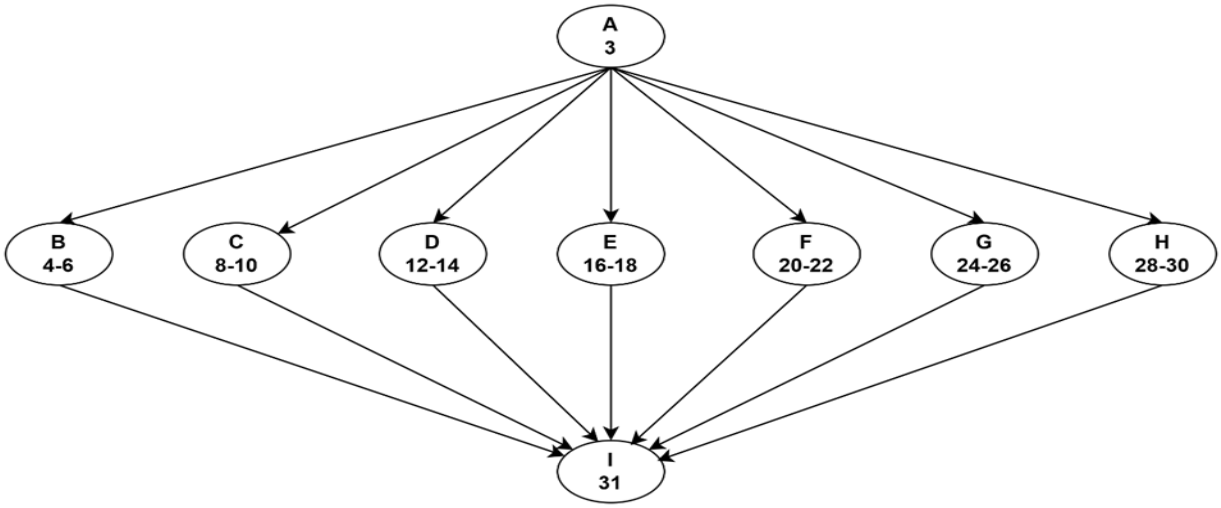


Figure 30: Control Flow Graph for Job Searching

Cyclomatic Complexity Calculation:

Edge = 14, Node =9 , P = 1

Cyclomatic Complexity = e-n+2p

=14-9+2P

=7

Since the cyclomatic complexity of the graph is 7, there will be 7 independent paths in the graph, as shown below:

1: A – B – I

2: A – C – I

3: A – D – I

4: A – E – I

5: A – F – I

6: A – G – I

7: A – H – I

Test Case Design from the Independent Paths

Table 4: Test Case Design Job Search

Test Case ID	Input Details	Expected Output	Independent Path
TC1	User leaves the input empty or doesn't enter anything	User enters only commas and spaces, e.g., “, , , , “	A → B → I
TC2	User enters an empty string	The system returns an empty list of requirements	A → C → I
TC3	User enters a single requirement like “JavaScript”	The system returns a list with one item: ["JavaScript"]	A → D → I
TC4	User enters multiple requirements separated by commas, e.g., “JavaScript, HTML, CSS”	The system returns ["JavaScript", "HTML", "CSS"]	A → E → I
TC5	User adds extra spaces around the skills, e.g., “ JavaScript , HTML “	The system trims the spaces and returns ["JavaScript", "HTML"]	A → F → I
TC6	User enters skills with extra commas or empty segments, e.g., “JavaScript,, ,HTML,, “	The system ignores empty items and returns ["JavaScript", "HTML"]	A → G → I
TC7	User enters only commas and spaces, e.g., “, , , , “	The system returns an empty list of requirements	A → H → I

5.2 Black Box Testing

Black box testing evaluates the system's external behavior by testing core features against expected outcomes. Functional tests will be designed for at least two critical features using boundary value analysis to verify the system's response to input limits and edge cases, ensuring reliable user-facing performance.

For the CarrerHive project, black box testing is performed on two essential features: auth.test.ts and Job.test.ts. These features are fundamental for user authentication and job searching , and testing their behavior under various input conditions helps ensure a smooth and error-free experience for end users.

Authentication feature:

The decision table for the feature is shown below:

Table 4: Decision Table for Authentication

		Rule 1	Rule2	Rule 3	Rule 4	Rule 5
Condition Sub	C1: User Exists in DB	|	|	True	True	False
	C2: Password Matches	|	|	True	False	|
	C3: Email Already Registered	False	True	|	|	|
	A1: Create User	Yes	No	No	No	No
	A2: Reject with Duplicate Email	No	Yes	No	No	No

Action Sub	A3: Login Success	No	No	Yes	No	No
	A4: Login Failure	No	No	No	Yes	Yes

Job Searching Feature :

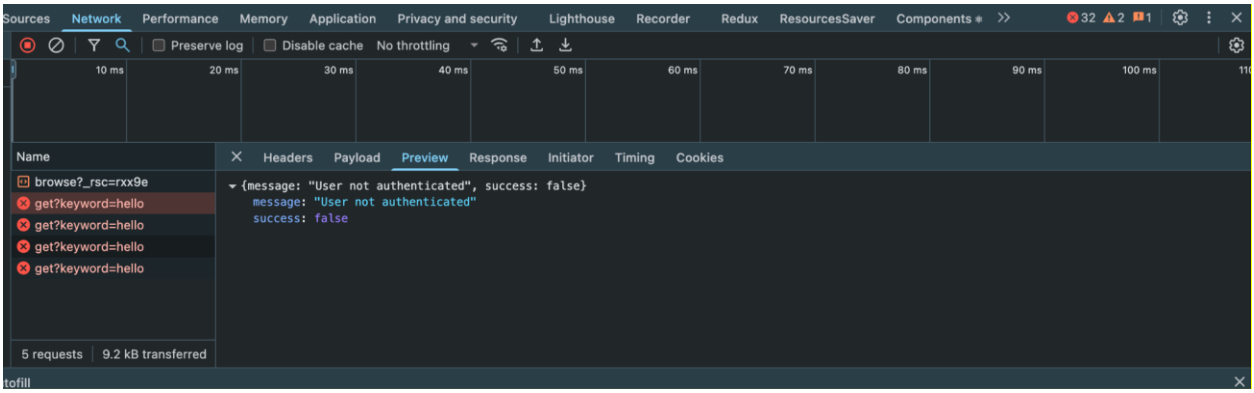
The decision table for the feature is shown below:

Table 5: Decision Table for Job Searching

		Rule 1	Rule2	Rule 3
Condition Sub	C1: Auth Token Present	Yes	Yes	Yes
	C2: Job ID Present	No	Yes	Yes
	C3: Job Exists in DB	|	Yes	No
Action Sub	A1: Return All Jobs	Yes	No	No
	A2: Return Job by ID	No	Yes	No

5.3 Bug Detection and Solution

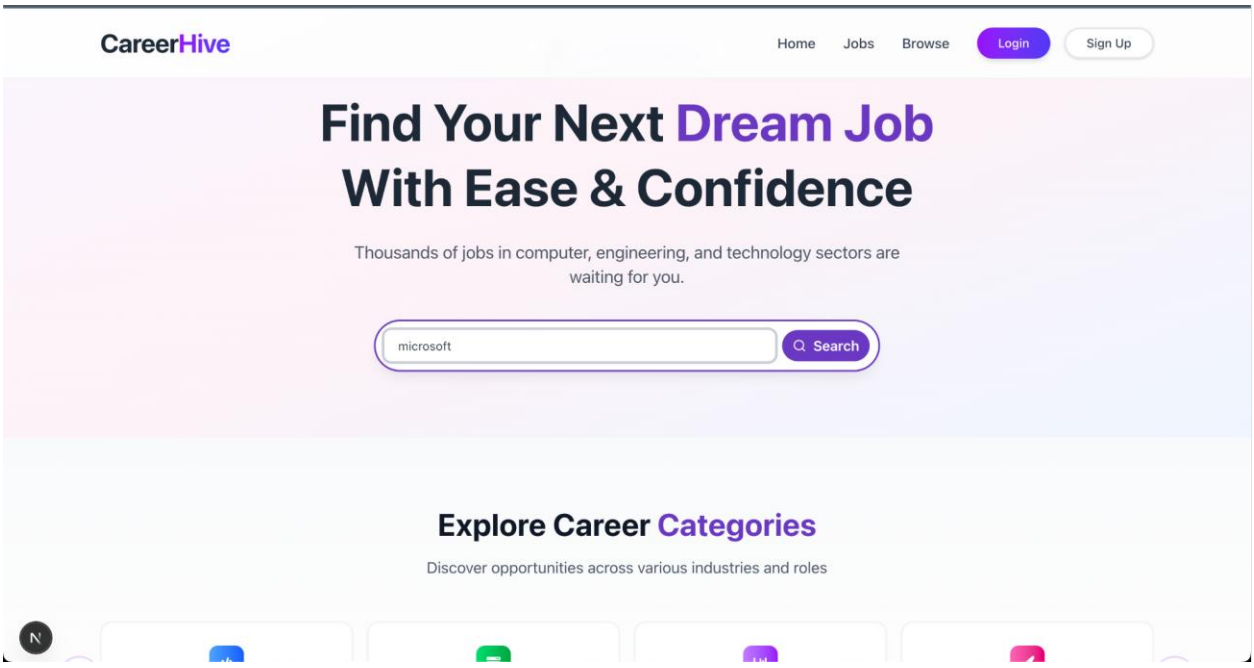
User not authenticated:

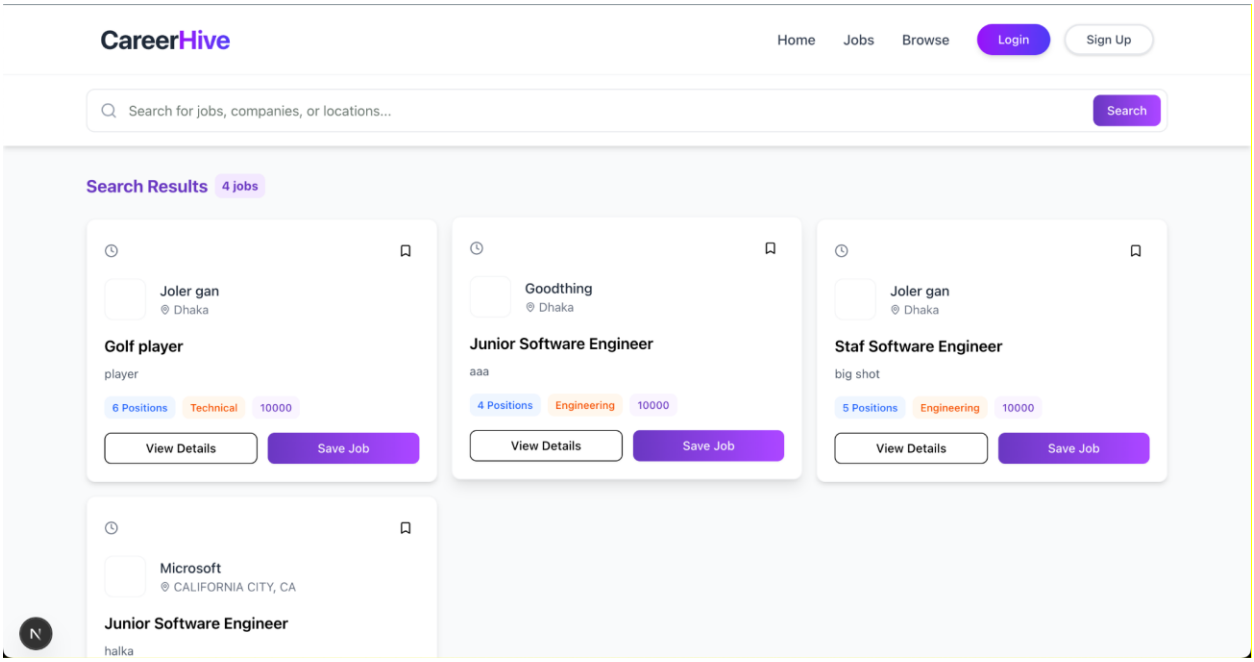


Solution:

Updated the logout implementation on the frontend to properly handle cookie clearance. Also reviewed and corrected the backend logic within the logout controller to ensure consistent user session termination.

Search query not performing:

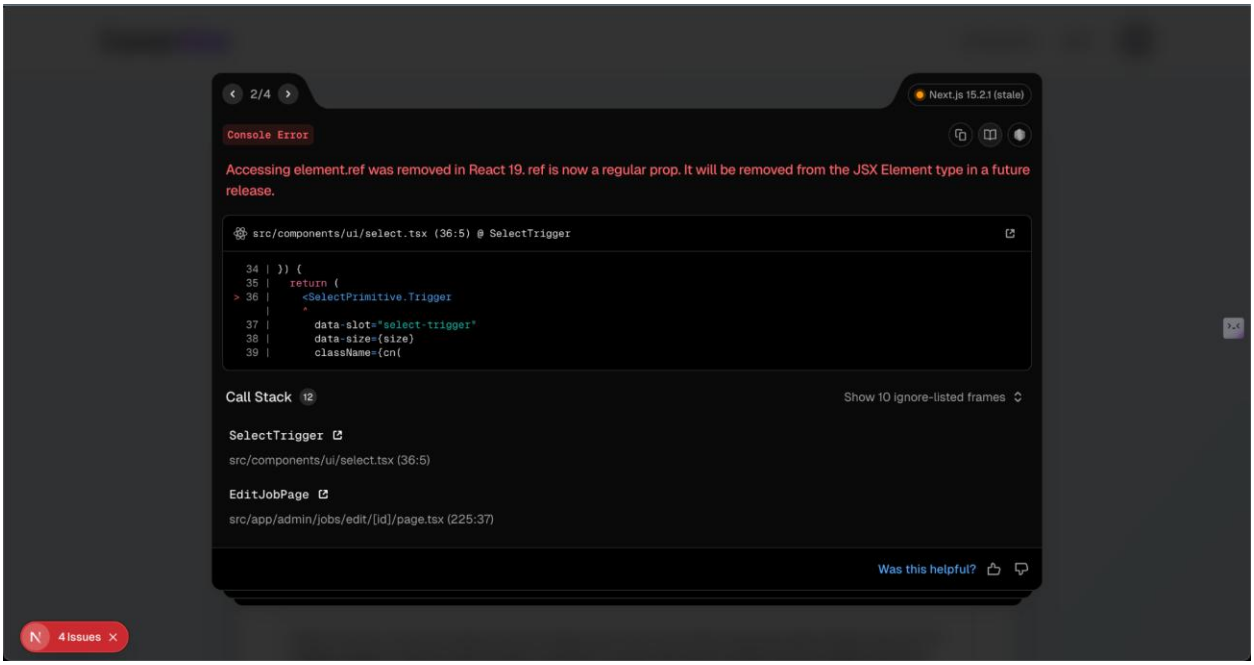




Solution:

Redesigned the GET request logic on the frontend to ensure proper request formatting and improved error handling. Additionally, restructured the backend controller to handle incoming requests more effectively and return consistent responses

Version mismatch type prop for using the latest React 19:



tsx Copy Edit

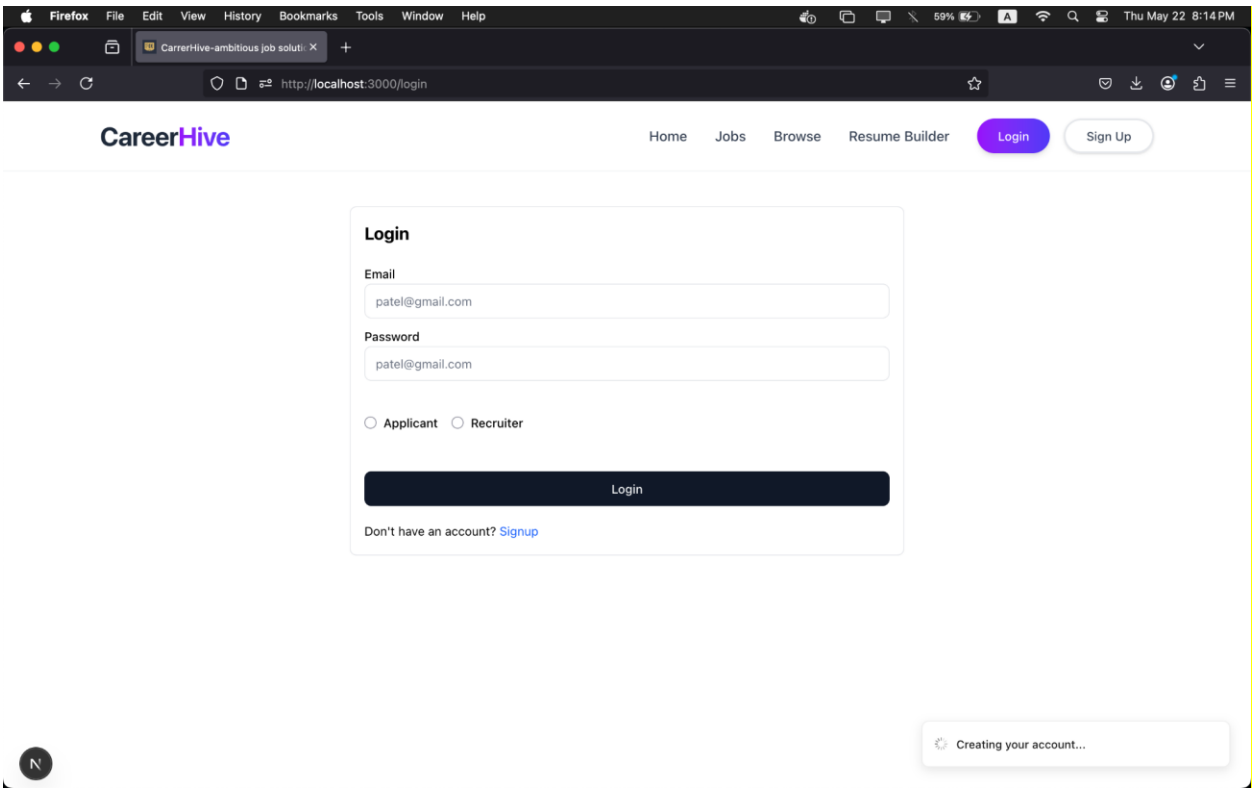
```tsx
const SelectTrigger = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Trigger>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Trigger> & {
    size?: "sm" | "default"
  }
>(({ className, size = "default", children, ...props }, ref) => {
  return (
    <SelectPrimitive.Trigger
      ref={ref} // 🔁 Forward the ref properly
      data-slot="select-trigger"
      data-size={size}
      className={cn(
        "border-input data-[placeholder]:text-muted-foreground ...",
        className
      )}
      {...props}
```

```
function SelectTrigger({
  className,
  size = "default",
  children,
  ...props
}: React.ComponentProps<typeof SelectPrimitive.Trigger> & {
  size?: "sm" | "default"
}) {
  return (
    <SelectPrimitive.Trigger
      data-slot="select-trigger"
      data-size={size}
      className={cn(
        "border-input data-[placeholder]:text-muted-foreground [&
        _svg:not([class*='text-'])]:text-muted-foreground
        focus-visible:border-ring focus-visible:ring-ring/50
        aria-invalid:ring-destructive/20
        dark:aria-invalid:ring-destructive/40
        aria-invalid:border-destructive dark:bg-input/30
        dark:hover:bg-input/50 flex w-fit items-center
        justify-between gap-2 rounded-md border bg-transparent px-3
        py-2 text-sm whitespace-nowrap shadow-xs transition-[color,
        box-shadow] outline-none focus-visible:ring-[3px]
        disabled:cursor-not-allowed disabled:opacity-50 data-
        [size=default]:h-9 data-[size=sm]:h-8 *:data-
        [slot=select-value]:line-clamp-1 *:data-[slot=select-value]
        :flex *:data-[slot=select-value]:items-center *:data-
        [slot=select-value]:gap-2 [& svg]:pointer-events-none [& svg]
```

Solution:

The warning indicates that element.ref access is deprecated in React 19+. Instead, ref should be passed and used as a regular prop. Fixing this requires updating third-party libraries like SelectPrimitive if they rely on the old pattern, which can be complex and may involve contributing to or forking the library. Applying this fix is non-trivial and may break compatibility without careful handling.

Creating an account is still showing on the login page:



Solution:

Reconfigured the toast notification. Change the notification loading to success.

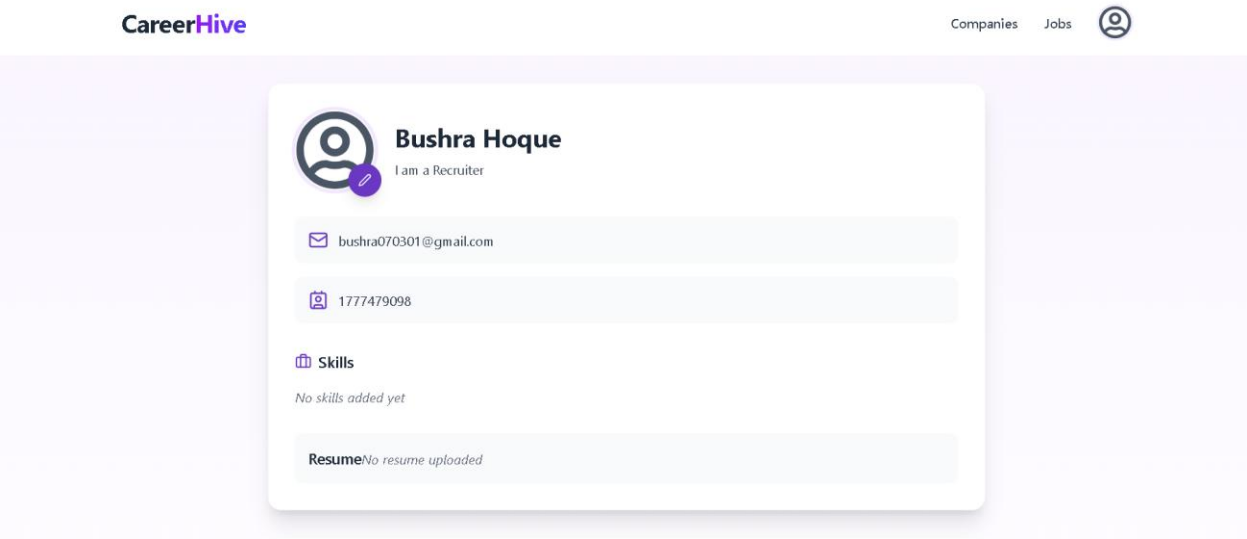
Due to integrating the latest versions of React 19 and Next.js,15 we encountered significant compatibility issues with existing libraries. As a result, we experienced numerous runtime issues that required extensive troubleshooting.

Chapter 6. Deployment

Due to time constraints, device issue and some unforeseen challenges, we were unable to deploy some part of the project live during this phase of development. However now we have added this part also in our report

Deployment Link

https://career-hive-web-seven.vercel.app/profile



Chapter 7. Conclusion

Learnings

- Developed a deep understanding of requirements engineering by engaging multiple stakeholders including job seekers, recruiters, and administrators.
- Gained experience mapping customer needs to technical specifications using tools like Quality Function Deployment (QFD) to ensure a user-centered design.
- Learned to model software requirements clearly through use case and activity diagrams, improving communication between design and development teams.
- Applied modern web technologies (MERN stack)
- Improved skills in system design, including architectural and component-level planning, leading to scalable and maintainable solutions.
- Practiced rigorous software testing, combining unit testing and functional testing to deliver a reliable product.

Limitations:

- Due to time constraints, the planned AI-powered job recommendation system and AI chatbot for career guidance were not developed.
- The real-time interview feature using WebRTC for audio and video calls pr could not be implemented.
- Integration with third-party job boards and professional networks remains unaddressed.
- The user interface, while prototyped, still requires usability testing and further refinement based on user feedback.

- Security implementations are basic and will require enhancement in future iterations to meet higher standards.

Future Plan:

- Implement AI-based job recommendations and a conversational chatbot to provide personalized career guidance.
- Develop and integrate the WebRTC-based real-time interview system to enable seamless audio and video communication.
- Expand the platform by integrating with popular job portals and professional networking APIs to increase job opportunities and candidate reach.
- Conduct usability studies to improve the user interface and overall experience.
- Strengthen security by adopting advanced authentication and data protection methods.
- Add features such as resume auto-building, skill gap analysis, and interview performance feedback to better support job seekers.

Appendix: **https://github.com/sanjana514/JOB-PORTAL**

THE END