*Chapter 1:*

# LITERATURE REVIEW – ONLINE JOB PORTAL

## 1. Purpose

### 1.1 Problem Statement

In today's digital world, job seekers struggle to find suitable job opportunities, and recruiters face difficulties reaching the right candidates. Traditional job hunting methods are time-consuming and inefficient. This project solves this problem by creating a centralized platform to connect job seekers with employers efficiently.

### 1.2 Goals/Objectives

• Enable job seekers to register, search, and apply for jobs.

• Allow recruiters to post jobs and review applicants.

• Provide an admin panel to manage users and job posts.

• Ensure a secure and responsive user experience.

### 1.3 Scope

**Included:**

• User registration and login

• Job posting and listing

• Job application process

• Admin dashboard to manage users and jobs

Recruiters can associate jobs with companies.

Jobs are organized into categories (like IT, Finance, etc.)

**Excluded:**

• Resume parsing or job recommendation AI

• Payment gateways

• Chat system

## 1.4 Existing System vs Proposed System

**Existing System:**

• Job seekers rely on newspapers or multiple job websites

• Recruiters manually filter resumes

• Poor user interface and no role separation

**Proposed System:**

• Central platform with clean UI

• Role-based access

• Structured, efficient, and scalable job management system

*Chapter 2*

# TECHNOLOGY OVERVIEW

## 2.1 Technology Stack

| Category | Technology / Tool | Purpose / Usage |
|---|---|---|
| Languages | JavaScript, HTML, CSS | Building frontend layout and adding interactivity |
| Frontend Framework | React.js | Designing dynamic and component-based user interface |
| Backend Framework | Node.js with Express.js | Handling server-side logic and API routing |
| Database | MySQL | Storing user, job, application, and company data |
| Tools | Visual Studio Code, Postman | VS Code: Writing code, Postman: Testing APIs |
| Version Control | Git and GitHub | Tracking changes and team collaboration |
| APIs | REST APIs (custom) | Communication between frontend and backend |
| Deployment | Vercel (Frontend), Render (Backend) | Hosting and running the project online |

**Table 1.1 Technology Stack**

## 2.2 Database

In this project, we use MySQL, a relational database management system, to store and manage all the application data. MySQL organizes data into tables like users, jobs, applications, companies, and categories, where each table contains rows and columns just like a spreadsheet.

The database helps the system:

Store information like user details, job postings, and applications.

Maintain relationships between tables using foreign keys (e.g., linking a job to a company or user).

Retrieve data quickly and securely when users perform actions like login, job search, or application submission.

## 2.3 Diagrams

### 2.3.1 System Architecture Diagram

This diagram 1.1 represents the communication between frontend, backend, and MySQL database.

**Frontend (React.js) ⟷ Backend (Node.js + Express.js) ⟷ MySQL Database**

The frontend sends HTTP requests to the backend APIs.

The backend processes requests and interacts with the database.

Clean 3-tier architecture (Presentation, Logic, Data)

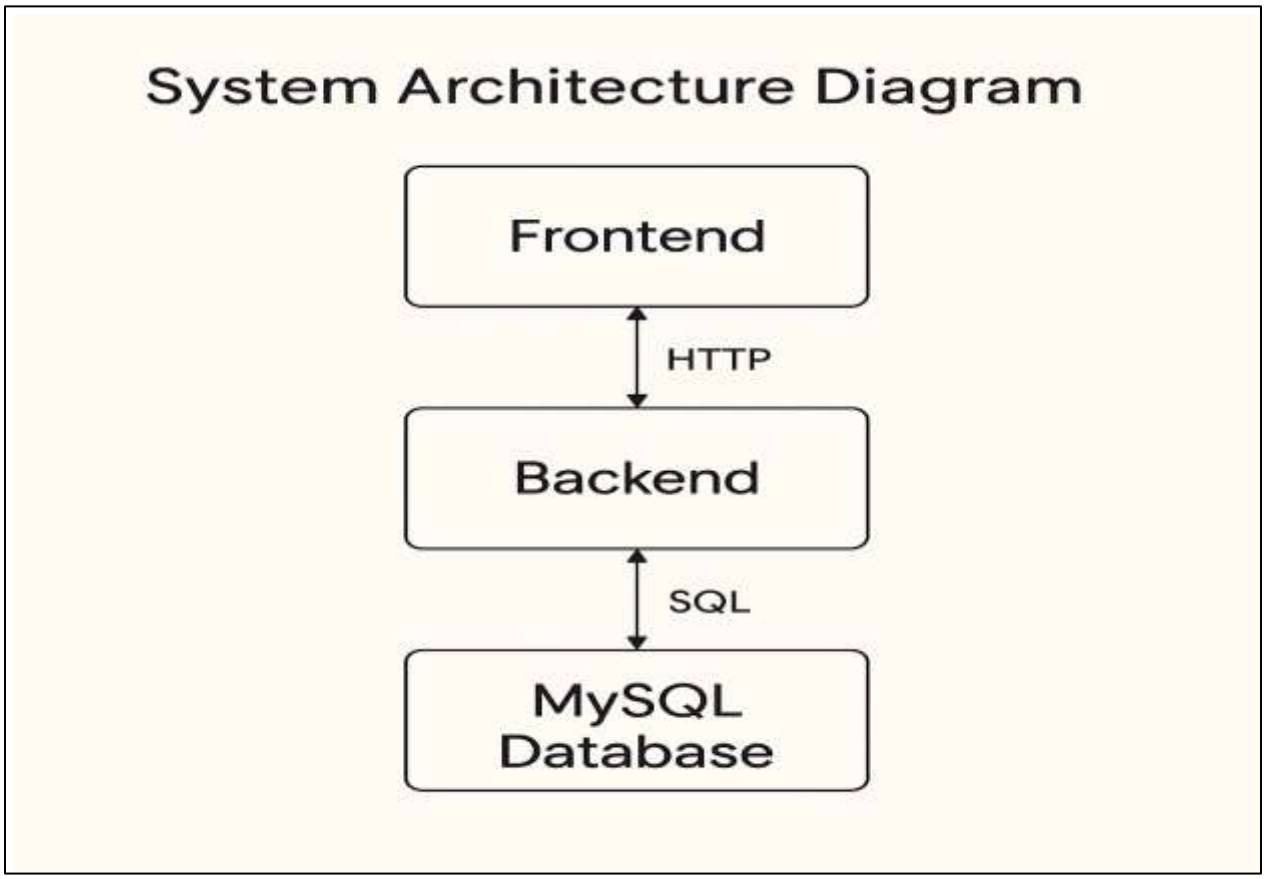


**Fig 1.1 System Architecture Diagram**

### 2.3.2 Entity-Relationship Diagram (ERD)

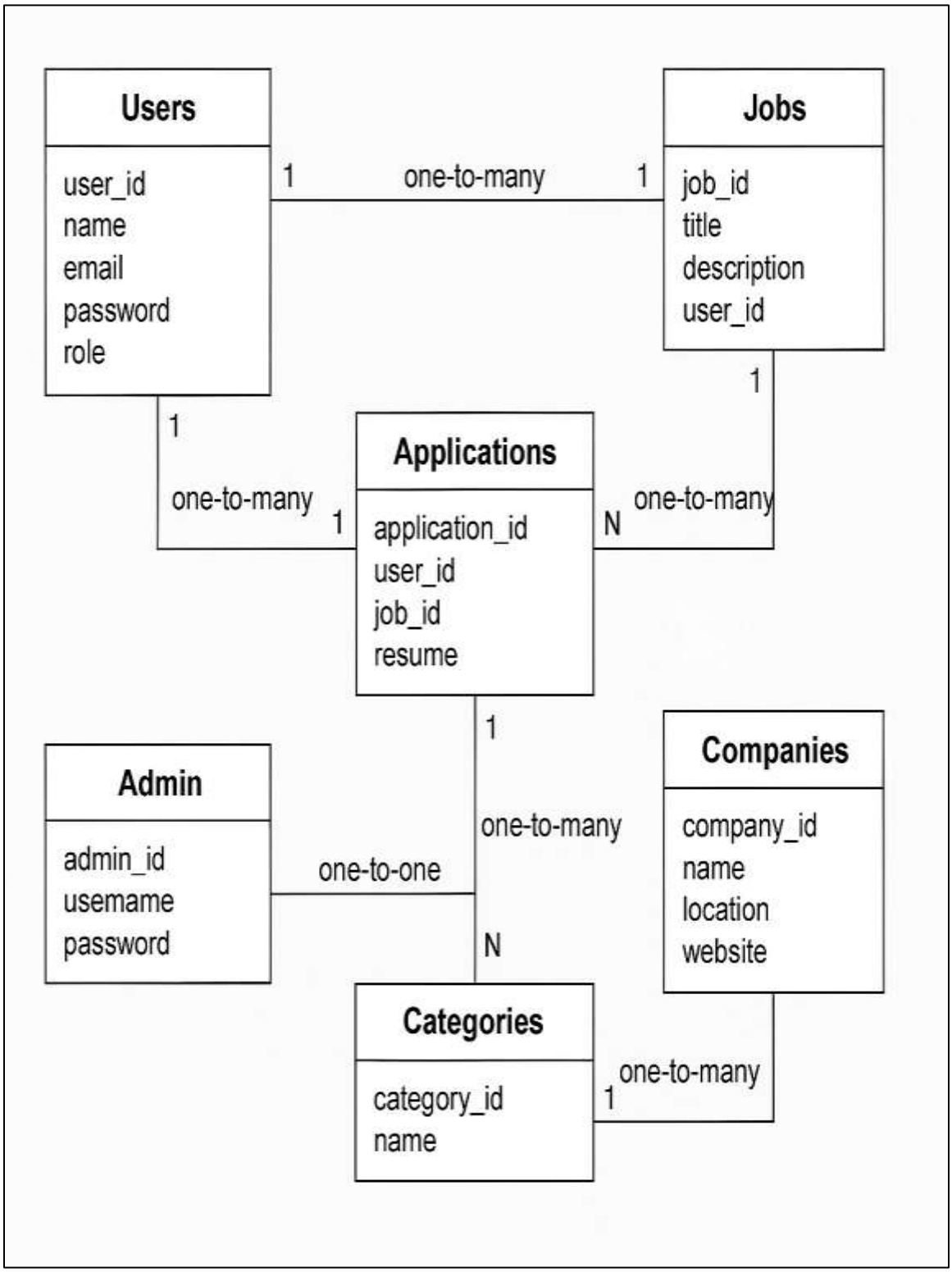


**Fig 1.2 ER-Diagram**

The Fig 1.2 ER-Diagram represents the structure of the Online Job Portal. The core entities include Users, Jobs, Applications, Categories, and Companies. Users can either be job seekers or recruiters. Recruiters post jobs under specific companies and categories. Applications link job seekers to job postings. Admin oversees the platform.

**Entities & Relationships:**

- One user can apply to many jobs (**1-to-many**: Users → Applications)
- One job can receive many applications (**1-to-many**: Jobs → Applications)
- One recruiter (user) can post multiple jobs (**1-to-many**: Users → Jobs)
- One category can contain many jobs (**1-to-many**: Categories → Jobs)
- One company can post many jobs (**1-to-many**: Companies → Jobs)

*Chapter 3*

# SYSTEM DESIGN

## 3.1 Modules

1. **Authentication Module**
    a. User login/registration
    b. Role-based access (job seeker, recruiter, admin)
2. **Job Seeker Module**
    a. Search and filter jobs
    b. Apply for jobs
    c. View application status
3. **Recruiter Module**
    a. Post and manage jobs
    b. View applications
4. **Admin Module**
    a. View/manage all users
    b. Manage job posts
    c. Monitor platform usage
5. **Job Module**
    a. Store job details: title, description, company, location, etc.
6. **Application Module**
    a. Submit resumes
    b. Track application records
7. **Company Module**
    a. Store company name, location, and website
    b. Linked with jobs (1 company → many jobs)

## 3.2 APIs

1. **Authentication APIs**

   POST /api/register – Register a user

   POST /api/login – User login

2. **User APIs**

   GET /api/users – Get list of users (admin)

   DELETE /api/users/:id – Delete user (admin)

3. **Job APIs**

   POST /api/jobs – Recruiter creates job

   GET /api/jobs – Get all job listings

   GET /api/jobs/:id – Get specific job details

   DELETE /api/jobs/:id – Delete job post

4. **Application APIs**

   POST /api/apply – Job seeker applies to job

   GET /api/applications/:userId – View job seeker's applications

   GET /api/job-applications/:jobId – Recruiter views applicants for a job

5. **Admin APIs**

   POST /api/admin-login – Admin authentication

   GET /api/admin/stats – View total users, jobs, etc.

6. **Company APIs**

   POST /api/companies – Add new company

   GET /api/companies – Get list of companies