Job Application

Project Plan: Job Portal Backend

Features: Enhanced Job Portal Backend Plan

1. User Management (Split Roles)

Employer Features

- Register as Employer (with company association)
- Login (JWT)
- View/update employer profile
- Manage company profile (if admin)

Job Seeker Features

- Register as Job Seeker
- Login (JWT)
- Upload resume (PDF/URL)
- Add skills/portfolio
- View/update profile

Shared Features

- Password reset
- Account deactivation

> 💡 deactivation need an additional process that will be worked on at the end of the project !

---

2. Company Management

- **Company Registration** (by first employer)

- **Company Profile**:
  - Name, description, logo
  - Location, website, industry
- **Team Management**:
  - Add/remove employers
  - Assign admin privileges (RH)
- **Verification System** (optional):
  - Verify company via email/document upload

> 💡 this check via email will be added if we have time for more security verification !

---

3. Job Management (Employer)

- **Job Posting**:
  - Title, description, requirements
  - Salary range, location (remote/onsite)
  - Job type (full-time, contract, etc.)
  - Associate with company
- **Job Lifecycle**:
  - Open/close applications
  - Extend deadline
- **Applicant Tracking**:
  - View applications per job
  - Change application status (Pending→Interview→Hired/Rejected)

> 💡 we may add closed or expired here for users (job-seekers

- Send status notifications

---

4. Job Application (Job Seeker)

- **Job Discovery**:
  - Browse all active jobs
  - Saved jobs/watchlist
- **Application System**:
  - Apply with resume/cover letter (url for CV/portfolio)
  - Track application status
  - Withdraw application
- **Alerts**:
  - New jobs matching skills
  - Application status changes

---

5. Search & Filtering

- **Job Search**:
  - By title, company, keywords
  - Location (city/remote)
  - Salary range
  - Experience level
- **Advanced Filters**:
  - Job type (full-time/part-time)
  - Posted date (last 24h, 7 days)
  - Company size/industry

Implementation Notes

1. **Role-Based Access Control**:
   - Middleware to check `user.role` and `employer.is_admin`

- Example: Only company admins can invite new employers

2. **File Handling**:
   - Resume storage (local filesystem)
3. **Data Validation**:
   - Prevent job seekers from posting jobs
   - Ensure applicants can't modify application status
4. **Security**:
   - Password hashing (BCrypt)
   - JWT expiration (short-lived tokens)
   - Rate limiting for auth endpoints

Suggested Development Order

1. **Phase 1**: Core User System
   - User registration/login (JWT)
   - Profile management split (employer vs seeker)
2. **Phase 2**: Company System
   - Company CRUD
   - Employer-company associations
3. **Phase 3**: Job Lifecycle
   - Job posting/management
   - Application workflow
4. **Phase 4**: Enhanced Features
   - Search/filters
   - Notifications
   - Analytics (jobs popularity, etc.)

Tech Stack

- **Backend**: Java (JDBC)

- **Database**: MySQL
- **Authentication**: JWT (JSON Web Tokens)
- **API Testing**: Postman

Updated Database Schema (Improved)

We'll use **5 tables** with proper relationships:

1. `users` (Base table for all users)
2. `employers` (Extends `users`, additional employer-specific fields)
3. `job_seekers` (Extends `users`, additional job seeker fields)
4. `companies` (Separate table for company details)
5. `jobs` (Posted jobs, linked to `employers` and `companies`)
6. `job_applications` (Tracks applications by `job_seekers`)

---

1. `users` (Base Table)

All users (employers & job seekers) share these fields.

Column	Type	Description
id	BIGINT (PK)	Unique user ID
email	VARCHAR(100) UNIQUE	User email (login)
password	VARCHAR(255)	Encrypted password
role	ENUM('EMPLOYER', 'JOB_SEEKER')	User role
created_at	TIMESTAMP	Account creation time

---

2. `employers` (Extends `users`)

Employers have additional details.

Column	Type	Description
user_id	BIGINT (PK, FK → users.id)	Links to `users` table
name	VARCHAR(100)	Employer's full name
company_id	BIGINT (FK → companies.id)	Employer's company

3. `job_seekers` (Extends `users`)

Job seekers have additional details (resume, skills, etc.).

Column	Type	Description
user_id	BIGINT (PK, FK → users.id)	Links to `users` table
name	VARCHAR(100)	Full name
resume_url	VARCHAR(255)	Resume file path/URL
skills	TEXT	Skills (JSON or comma-separated)

4. `companies` (Separate Table)

Stores company details (many employers can belong to one company).

Column	Type	Description
id	BIGINT (PK)	Company ID
name	VARCHAR(100)	Company name
description	TEXT	Company description
website	VARCHAR(255)	Company website

5. `jobs` (Posted Jobs)

Jobs posted by employers.

Column	Type	Description
id	BIGINT (PK)	Job ID
title	VARCHAR(100)	Job title
description	TEXT	Job details
location	VARCHAR(100)	Job location
salary	DECIMAL(10,2)	Salary range
posted_by	BIGINT (FK → employers.user_id)	Employer who posted
company_id	BIGINT (FK → companies.id)	Company offering job
created_at	TIMESTAMP	Job posting time

6. `job_applications` (Applications)

Tracks job applications by seekers.

Column	Type	Description
id	BIGINT (PK)	Application ID
job_id	BIGINT (FK → jobs.id)	Applied job
job_seeker_id	BIGINT (FK → job_seekers.user_id)	Job seeker who applied
status	ENUM('PENDING', 'ACCEPTED', 'REJECTED')	Application status
applied_at	TIMESTAMP	Application time

Entity-Relationship Diagram (ERD)

```
users (id, email, password, role)
 │
 ├── employers (user_id, name, company_id) → companies(id)
 │       └── jobs (posted_by)
 │
 ├── job_seekers (user_id, name, resume_url)
 │       └── job_applications (job_seeker_id)
 │
 └── companies (id, name, description)
         └── jobs (company_id)
```

API Endpoints

Auth

Method	Endpoint	Description
POST	/api/auth/register	Register (Employer or Job Seeker)
POST	/api/auth/login	Login (JWT Token)

Jobs (Employer)

Method	Endpoint	Description
POST	/api/jobs	Post a new job (Employer only)
GET	/api/jobs/my-jobs	Get all jobs posted by employer

PUT	/api/jobs/{jobId}	Update a job
DELETE	/api/jobs/{jobId}	Delete a job
GET	/api/jobs/{jobId}/applicants	View applicants for a job

Jobs (Job Seeker)

Method	Endpoint	Description
GET	/api/jobs	Browse all jobs
POST	/api/jobs/{jobId}/apply	Apply for a job
GET	/api/jobs/applied	View all applied jobs
DELETE	/api/jobs/{applicationId}/withdraw	Withdraw an application

Step-by-Step Implementation

1.Project Structure

```
job-portal-jdbc/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── config/          # Database config
│   │   │   ├── controllers/     # API endpoints (if REST)
│   │   │   ├── dao/             # Data Access Objects (JDBC)
│   │   │   ├── models/          # Entities (User, Job, etc.)
│   │   │   ├── services/        # Business logic
│   │   │   ├── utils/           # JWT, Validation
│   │   │   └── Main.java        # Entry point
│   │   └── resources/
│   │       └── application.properties  # DB config
│   └── test/                    # Unit tests
```

2. Database Setup

MySQL Schema

```
-- 1. Create 'users' table (base for all users)
CREATE TABLE users (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role ENUM('EMPLOYER', 'JOB_SEEKER') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Create 'companies' table
CREATE TABLE companies (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    website VARCHAR(255)
);

-- 3. Create 'employers' (extends users)
CREATE TABLE employers (
    user_id BIGINT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    company_id BIGINT,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (company_id) REFERENCES companies(id)
);

-- 4. Create 'job_seekers' (extends users)
CREATE TABLE job_seekers (
    user_id BIGINT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    resume_url VARCHAR(255),
    skills TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- 5. Create 'jobs' (posted jobs)
CREATE TABLE jobs (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
```

```
    title VARCHAR(100) NOT NULL,
    description TEXT,
    location VARCHAR(100),
    salary DECIMAL(10,2),
    posted_by BIGINT NOT NULL,
    company_id BIGINT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (posted_by) REFERENCES employers(user_id),
    FOREIGN KEY (company_id) REFERENCES companies(id)
);

-- 6. Create 'job_applications'
CREATE TABLE job_applications (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    job_id BIGINT NOT NULL,
    job_seeker_id BIGINT NOT NULL,
    status ENUM('PENDING', 'ACCEPTED', 'REJECTED') DEFAULT 'PENDIN
G',
    applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (job_id) REFERENCES jobs(id),
    FOREIGN KEY (job_seeker_id) REFERENCES job_seekers(user_id)
);
```

3. JDBC Database Connection

Create a `DatabaseConfig.java` to manage connections:

```
// src/main/java/config/DatabaseConfig.java
package com.stoonproduction.jobapplication.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String HOST = "127.0.0.1";
    private static final int PORT = 3306;
    private static final String DB_NAME = "jdbc_course_db";
```

```
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";

    private static Connection connection;

    public static Connection getConnection() {
        try {
            connection = DriverManager.getConnection(String.format("jdbc:my
sql://%s:%d/%s", HOST, PORT, DB_NAME), USERNAME, PASSWORD);
        } catch(SQLException se) {
            se.printStackTrace();
        }
        return connection;
    }
}
```

4. DAO (Data Access Object) Layer

Example: `UserDao.java` (for CRUD operations):

```
// src/main/java/dao/UserDao.java
public class UserDao {
    public void createUser(User user) throws SQLException {
        String sql = "INSERT INTO users (email, password, role) VALUES (?, ?,
?)";
        try (Connection conn = DatabaseConfig.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, user.getEmail());
            stmt.setString(2, user.getPassword());
            stmt.setString(3, user.getRole().toString());
            stmt.executeUpdate();
        }
    }

    public User getUserByEmail(String email) throws SQLException {
        String sql = "SELECT * FROM users WHERE email = ?";
        try (Connection conn = DatabaseConfig.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
```

```
            stmt.setString(1, email);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return new User(
                    rs.getLong("id"),
                    rs.getString("email"),
                    rs.getString("password"),
                    UserRole.valueOf(rs.getString("role"))
                );
            }
            return null;
        }
    }
}
```

5. JWT Authentication

Add `JwtUtil.java` for token handling:

```
// src/main/java/utils/JwtUtil.java
public class JwtUtil {
    private static final String SECRET_KEY = "your-secret-key";
    private static final long EXPIRATION_TIME = 86400000; // 24 hours

    public static String generateToken(User user) {
        return Jwts.builder()
            .setSubject(user.getEmail())
            .claim("role", user.getRole())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_
TIME))
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
            .compact();
    }

    public static boolean validateToken(String token) {
        try {
            Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token);
```

```
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

6. API Endpoints (Simple HTTP Server)

Use `HttpServer` (Java's built-in HTTP server) for REST APIs:

```
// src/main/java/Main.java
public class Main {
        private static final int PORT = 8080;
    public static void main(String[] args) throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(PORT),
0);

        // Auth Endpoints
        server.createContext("/api/auth/register", new AuthController::handleR
egister);
        server.createContext("/api/auth/login", new AuthController::handleLogi
n);

        // Job Endpoints
        server.createContext("/api/jobs", new JobController::handleJobs);

        server.start();
        System.out.println("Server running on port "+PORT);
    }
}
```

7. Test with Postman

- **Register** → `POST /api/auth/register`
- **Login** → `POST /api/auth/login`
- **Post Job** → `POST /api/jobs` (with JWT)

- **Apply for Job** → `POST /api/jobs/{jobId}/apply`
- **View Applicants** → `GET /api/jobs/{jobId}/applicants`

Postman Testing (Example)

1. **Register Employer**

```
POST /api/auth/register
{
   "name": "Google HR",
   "email": "hr@google.com",
   "password": "password123",
   "role": "EMPLOYER"
}
```

2. **Post a Job**

```
POST /api/jobs
Authorization: Bearer <JWT_TOKEN>
{
   "title": "Java Developer",
   "description": "Looking for a Spring Boot expert.",
   "company": "Google",
   "location": "Remote",
   "salary": 90000.00
}
```

3. **Job Seeker Applies**

```
POST /api/jobs/1/apply
Authorization: Bearer <JWT_TOKEN>
```
