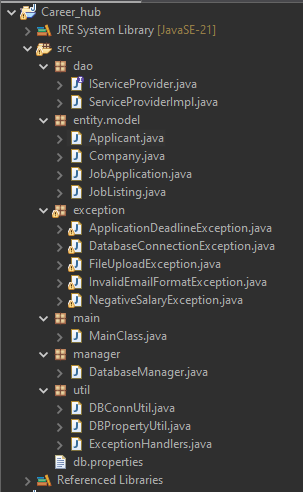
**Java Coding Challenge**

**Project Title: CareerHub, The Job Board**

**Problem Statement:**

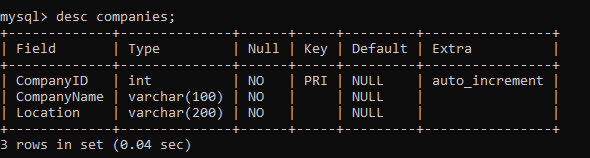
A Job Board scenario is a digital platform or system that facilitates the process of job searching and recruitment. In this scenario, various stakeholders, such as job seekers, companies, and recruiters, use the platform to post, search for, and apply to job opportunities.

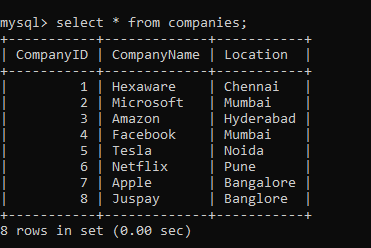
**Project Directory Structure:**



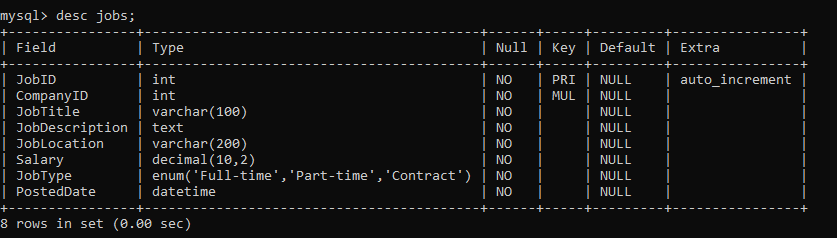
**Database schema:**

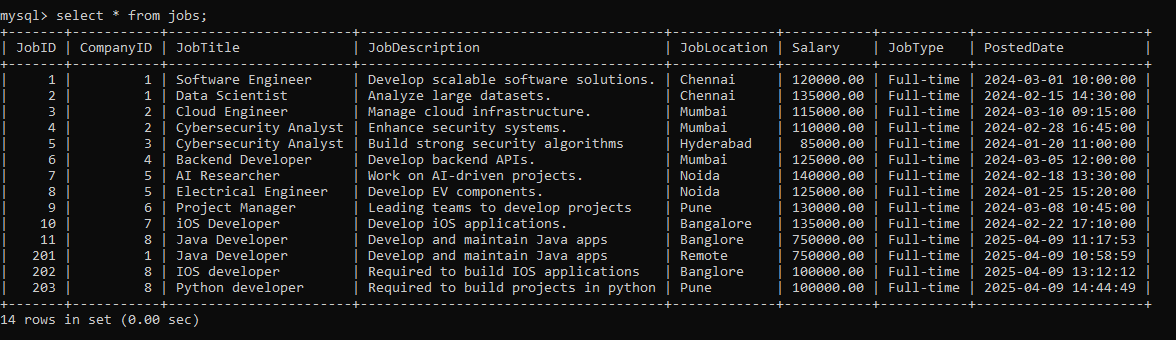
**Companies table:**

****

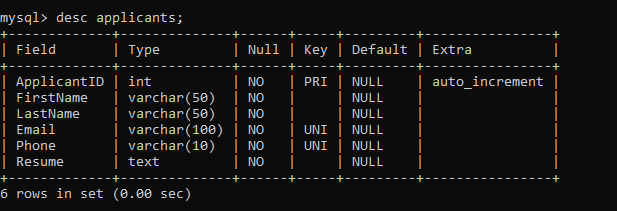
****

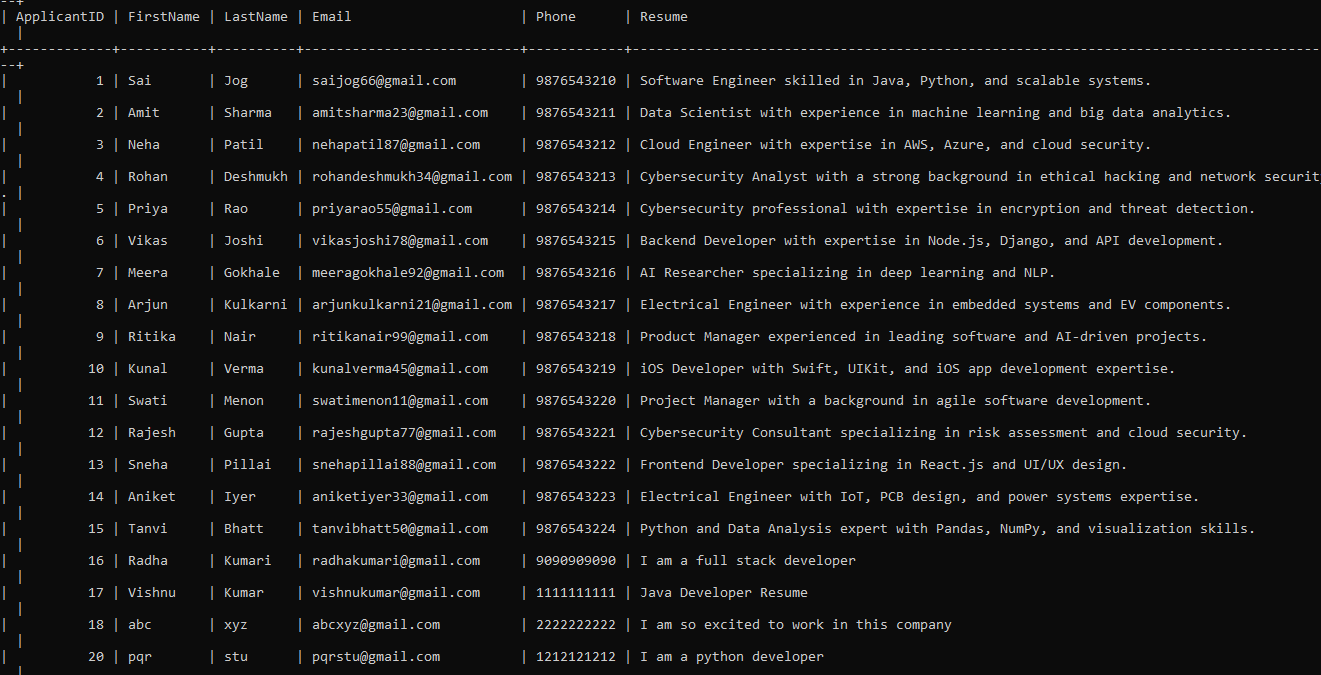
**Jobs table:**

****

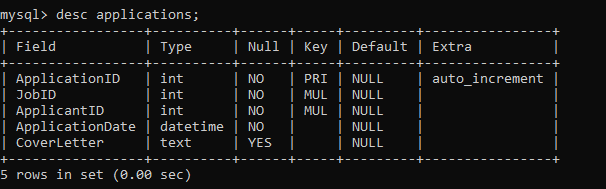
****

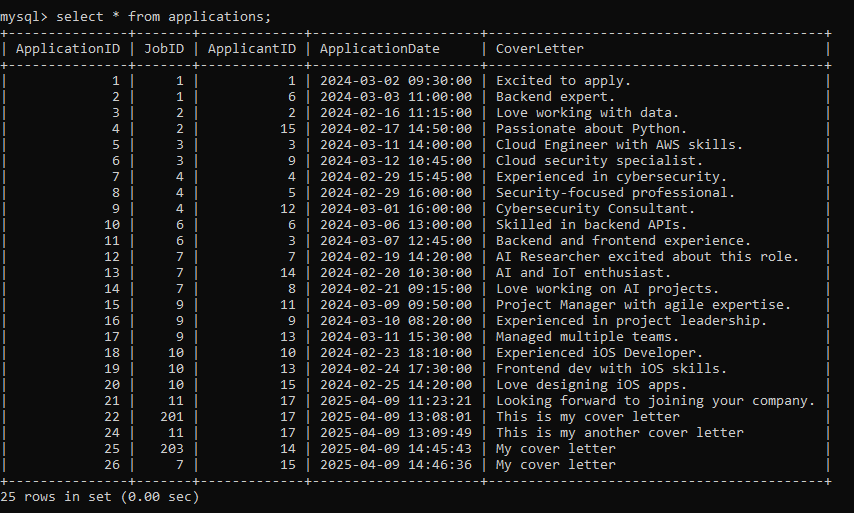
**Applicants table:**

****

****

**Applications table:**

****

****

**1. Create and implement the mentioned class and the structure in your application.**

**JobListing class:**

package entity.model;

import java.time.LocalDateTime;

public class JobListing {

private int jobID;

private int companyID;

private String jobTitle;

private String jobDescription;

private String jobLocation;

private double salary;

private String jobType;

private LocalDateTime postedDate;

public JobListing() {

super();

}

public JobListing(int jobID, int companyID, String jobTitle, String jobDescription, String jobLocation,

double salary, String jobType, LocalDateTime postedDate) {

super();

this.jobID = jobID;

this.companyID = companyID;

this.jobTitle = jobTitle;

this.jobDescription = jobDescription;

this.jobLocation = jobLocation;

this.salary = salary;

this.jobType = jobType;

this.postedDate = postedDate;

}

*@Override*

public String toString() {

return "JobID: " + jobID +

", Title: " + jobTitle +

", CompanyID: " + companyID +

", Location: " + jobLocation +

", Salary: " + salary +

", Type: " + jobType +

", Posted: " + postedDate;

}

public int getJobID() {

return jobID;

}

public void setJobID(int jobID) {

this.jobID = jobID;

}

public int getCompanyID() {

return companyID;

}

public void setCompanyID(int companyID) {

this.companyID = companyID;

}

public String getJobTitle() {

return jobTitle;

}

public void setJobTitle(String jobTitle) {

this.jobTitle = jobTitle;

}

public String getJobDescription() {

return jobDescription;

}

public void setJobDescription(String jobDescription) {

this.jobDescription = jobDescription;

}

public String getJobLocation() {

return jobLocation;

}

public void setJobLocation(String jobLocation) {

this.jobLocation = jobLocation;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public String getJobType() {

return jobType;

}

public void setJobType(String jobType) {

this.jobType = jobType;

}

public LocalDateTime getPostedDate() {

return postedDate;

}

public void setPostedDate(LocalDateTime postedDate) {

this.postedDate = postedDate;

}

}

**Company Class:**

package entity.model;

public class Company {

private int companyID;

private String companyName;

private String location;

public Company() {

super();

}

public Company(int companyID, String companyName, String location) {

super();

this.companyID = companyID;

this.companyName = companyName;

this.location = location;

}

*@Override*

public String toString() {

return "CompanyID: " + companyID +

", Name: " + companyName +

", Location: " + location;

}

public int getCompanyID() {

return companyID;

}

public void setCompanyID(int companyID) {

this.companyID = companyID;

}

public String getCompanyName() {

return companyName;

}

public void setCompanyName(String companyName) {

this.companyName = companyName;

}

public String getLocation() {

return location;

}

public void setLocation(String location) {

this.location = location;

}

}

**Applicant Class:**

package entity.model;

public class Applicant {

private int applicantID;

private String firstName;

private String lastName;

private String email;

private String phone;

private String resume;

public Applicant() {

super();

}

public Applicant(int applicantID, String firstName, String lastName, String email, String phone, String resume) {

super();

this.applicantID = applicantID;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phone = phone;

this.resume = resume;

}

*@Override*

public String toString() {

return "ApplicantID: " + applicantID +

", Name: " + firstName + " " + lastName +

", Email: " + email +

", Phone: " + phone;

}

public int getApplicantID() {

return applicantID;

}

public void setApplicantID(int applicantID) {

this.applicantID = applicantID;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getResume() {

return resume;

}

public void setResume(String resume) {

this.resume = resume;

}

}

**JobApplication Class:**

package entity.model;

import java.time.LocalDateTime;

public class JobApplication {

private int applicationID;

private int jobID;

private int applicantID;

private LocalDateTime applicationDate;

private String coverLetter;

public JobApplication() {

super();

}

public JobApplication(int applicationID, int jobID, int applicantID, LocalDateTime applicationDate,

String coverLetter) {

super();

this.applicationID = applicationID;

this.jobID = jobID;

this.applicantID = applicantID;

this.applicationDate = applicationDate;

this.coverLetter = coverLetter;

}

*@Override*

public String toString() {

return "ApplicationID: " + applicationID +

", JobID: " + jobID +

", ApplicantID: " + applicantID +

", Applied On: " + applicationDate +

", CoverLetter: " + coverLetter;

}

public int getApplicationID() {

return applicationID;

}

public void setApplicationID(int applicationID) {

this.applicationID = applicationID;

}

public int getJobID() {

return jobID;

}

public void setJobID(int jobID) {

this.jobID = jobID;

}

public int getApplicantID() {

return applicantID;

}

public void setApplicantID(int applicantID) {

this.applicantID = applicantID;

}

public LocalDateTime getApplicationDate() {

return applicationDate;

}

public void setApplicationDate(LocalDateTime applicationDate) {

this.applicationDate = applicationDate;

}

public String getCoverLetter() {

return coverLetter;

}

public void setCoverLetter(String coverLetter) {

this.coverLetter = coverLetter;

}

}

**2. Util package:**

**DBConnUtil.java**

package util;

import exception.DatabaseConnectionException;

import java.sql.Connection;

import java.sql.DriverManager;

public class DBConnUtil {

private static final String PROPERTIES\_FILE = "db.properties";

public static Connection getConnection() throws DatabaseConnectionException {

try {

String connectionString = DBPropertyUtil.getConnectionString(PROPERTIES\_FILE);

// DriverManager can auto-load the driver if JDBC 4.0+

return DriverManager.getConnection(connectionString);

} catch (Exception e) {

throw new DatabaseConnectionException("Failed to connect to database: " + e.getMessage());

}

}

}

**DBPropertyUtil.java**

package util;

import java.io.InputStream;

import java.util.Properties;

public class DBPropertyUtil {

public static String getConnectionString(String propertyFileName) {

try (InputStream input = DBPropertyUtil.class.getClassLoader().getResourceAsStream(propertyFileName)) {

if (input == null) {

throw new RuntimeException("Property file not found: " + propertyFileName);

}

Properties props = new Properties();

props.load(input);

String url = props.getProperty("url");

String username = props.getProperty("username");

String password = props.getProperty("password");

// Return full connection string

return url + "?user=" + username + "&password=" + password;

} catch (Exception e) {

throw new RuntimeException("Failed to load connection string from properties: " + e.getMessage());

}

}

}

**DB.properties:**

url=jdbc:mysql://localhost:3306/careerhub

username=root

password=@MrBTS07691234

driver=com.mysql.cj.jdbc.Driver

**3. DatabaseManager Class:**

**Methods implemented:**

* InitializeDatabase(): Initializes the database schema and tables.
* InsertJobListing(job: JobListing): Inserts a new job listing into the "Jobs" table.
* InsertCompany(company: Company): Inserts a new company into the "Companies" table.
* InsertApplicant(applicant: Applicant): Inserts a new applicant into the "Applicants" table.
* InsertJobApplication(application: JobApplication): Inserts a new job application into the "Applications" table.
* GetJobListings(): List: Retrieves a list of all job listings.
* GetCompanies(): List: Retrieves a list of all companies.
* GetApplicants(): List: Retrieves a list of all applicants.
* GetApplicationsForJob(jobID: int): List: Retrieves a list of job applications for a specific job listing.
* getJobsBySalaryRange(minSalary, maxSalary): Retrieves a list of jobs in between a specific salary range.

package manager;

import entity.model.\*;

import util.DBConnUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import exception.DatabaseConnectionException;

public class DatabaseManager {

public void initializeDatabase() {

try {

try (Connection conn = DBConnUtil.getConnection()) {

Statement stmt = conn.createStatement();

String companyTable = "CREATE TABLE IF NOT EXISTS Companies (" +

"CompanyID INT PRIMARY KEY," +

"CompanyName VARCHAR(100)," +

"Location VARCHAR(100))";

String applicantTable = "CREATE TABLE IF NOT EXISTS Applicants (" +

"ApplicantID INT PRIMARY KEY," +

"FirstName VARCHAR(100)," +

"LastName VARCHAR(100)," +

"Email VARCHAR(100)," +

"Phone VARCHAR(20)," +

"Resume TEXT)";

String jobTable = "CREATE TABLE IF NOT EXISTS Jobs (" +

"JobID INT PRIMARY KEY," +

"CompanyID INT," +

"JobTitle VARCHAR(100)," +

"JobDescription TEXT," +

"JobLocation VARCHAR(100)," +

"Salary DECIMAL(10,2)," +

"JobType VARCHAR(50)," +

"PostedDate DATETIME," +

"FOREIGN KEY (CompanyID) REFERENCES Companies(CompanyID))";

String appTable = "CREATE TABLE IF NOT EXISTS Applications (" +

"ApplicationID INT PRIMARY KEY," +

"JobID INT," +

"ApplicantID INT," +

"ApplicationDate DATETIME," +

"CoverLetter TEXT," +

"FOREIGN KEY (JobID) REFERENCES Jobs(JobID)," +

"FOREIGN KEY (ApplicantID) REFERENCES Applicants(ApplicantID))";

stmt.execute(companyTable);

stmt.execute(applicantTable);

stmt.execute(jobTable);

stmt.execute(appTable);

System.out.println("Database schema initialized.");

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void insertCompany(Company company) {

String sql = "INSERT INTO Companies VALUES (?, ?, ?)";

try (Connection conn = DBConnUtil.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, company.getCompanyID());

ps.setString(2, company.getCompanyName());

ps.setString(3, company.getLocation());

ps.executeUpdate();

System.out.println("Company inserted.");

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

}

public void insertApplicant(Applicant applicant) {

String sql = "INSERT INTO Applicants VALUES (?, ?, ?, ?, ?, ?)";

try (Connection conn = DBConnUtil.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, applicant.getApplicantID());

ps.setString(2, applicant.getFirstName());

ps.setString(3, applicant.getLastName());

ps.setString(4, applicant.getEmail());

ps.setString(5, applicant.getPhone());

ps.setString(6, applicant.getResume());

ps.executeUpdate();

System.out.println("Applicant inserted.");

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

}

public void insertJobListing(JobListing job) {

String sql = "INSERT INTO Jobs VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

try (Connection conn = DBConnUtil.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, job.getJobID());

ps.setInt(2, job.getCompanyID());

ps.setString(3, job.getJobTitle());

ps.setString(4, job.getJobDescription());

ps.setString(5, job.getJobLocation());

ps.setDouble(6, job.getSalary());

ps.setString(7, job.getJobType());

ps.setTimestamp(8, Timestamp.valueOf(job.getPostedDate()));

ps.executeUpdate();

System.out.println("Job listing inserted.");

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

}

public void insertJobApplication(JobApplication app) {

String sql = "INSERT INTO Applications VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DBConnUtil.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, app.getApplicationID());

ps.setInt(2, app.getJobID());

ps.setInt(3, app.getApplicantID());

ps.setTimestamp(4, Timestamp.valueOf(app.getApplicationDate()));

ps.setString(5, app.getCoverLetter());

ps.executeUpdate();

System.out.println("Job application submitted.");

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

}

public List<JobListing> getJobListings() {

List<JobListing> jobs = new ArrayList<>();

String sql = "SELECT \* FROM Jobs";

try {

try (Connection conn = DBConnUtil.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

JobListing job = new JobListing(

rs.getInt("JobID"),

rs.getInt("CompanyID"),

rs.getString("JobTitle"),

rs.getString("JobDescription"),

rs.getString("JobLocation"),

rs.getDouble("Salary"),

rs.getString("JobType"),

rs.getTimestamp("PostedDate").toLocalDateTime()

);

jobs.add(job);

}

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return jobs;

}

public List<Company> getCompanies() {

List<Company> companies = new ArrayList<>();

String sql = "SELECT \* FROM Companies";

try (Connection conn = DBConnUtil.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Company c = new Company(

rs.getInt("CompanyID"),

rs.getString("CompanyName"),

rs.getString("Location")

);

companies.add(c);

}

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

return companies;

}

public List<Applicant> getApplicants() {

List<Applicant> list = new ArrayList<>();

String sql = "SELECT \* FROM Applicants";

try (Connection conn = DBConnUtil.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Applicant a = new Applicant(

rs.getInt("ApplicantID"),

rs.getString("FirstName"),

rs.getString("LastName"),

rs.getString("Email"),

rs.getString("Phone"),

rs.getString("Resume")

);

list.add(a);

}

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

return list;

}

public List<JobApplication> getApplicationsForJob(int jobID) {

List<JobApplication> apps = new ArrayList<>();

String sql = "SELECT \* FROM Applications WHERE JobID = ?";

try (Connection conn = DBConnUtil.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, jobID);

ResultSet rs = ps.executeQuery();

while (rs.next()) {

JobApplication app = new JobApplication(

rs.getInt("ApplicationID"),

rs.getInt("JobID"),

rs.getInt("ApplicantID"),

rs.getTimestamp("ApplicationDate").toLocalDateTime(),

rs.getString("CoverLetter")

);

apps.add(app);

}

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

return apps;

}

public List<JobListing> getJobListingsBySalaryRange(double minSalary, double maxSalary) {

List<JobListing> jobs = new ArrayList<>();

String sql = "SELECT \* FROM Jobs WHERE salary BETWEEN ? AND ?";

try (Connection conn = DBConnUtil.getConnection(); PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setDouble(1, minSalary);

stmt.setDouble(2, maxSalary);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

JobListing job = new JobListing(

rs.getInt("JobID"),

rs.getInt("CompanyID"),

rs.getString("JobTitle"),

rs.getString("JobDescription"),

rs.getString("JobLocation"),

rs.getDouble("Salary"),

rs.getString("JobType"),

rs.getTimestamp("PostedDate").toLocalDateTime()

);

jobs.add(job);

}

} catch (DatabaseConnectionException e) {

System.err.println("Database connection error while inserting job listing: " + e.getMessage());

} catch (Exception e) {

e.printStackTrace();

}

return jobs;

}

}

**4. Interface implementation:**

**IServiceProvider interface:**

package dao;

import entity.model.Applicant;

import entity.model.JobListing;

import java.util.List;

public interface IServiceProvider {

void createProfile(String email, String firstName, String lastName, String phone, String resume);

void apply(int applicantID, String coverLetter);

void applyForJob(int jobID, int applicantID, String coverLetter);

void postJob(int companyID, String jobTitle, String jobDescription, String jobLocation, double salary, String jobType);

List<JobListing> getJobsBySalaryRange(double minSalary, double maxSalary);

List<Applicant> getApplicants();

List<JobListing> getJobs();

}

**ServiceProviderImpl.java**

package dao;

import entity.model.Applicant;

import entity.model.JobApplication;

import entity.model.JobListing;

import exception.InvalidEmailFormatException;

import exception.NegativeSalaryException;

import manager.DatabaseManager;

import java.time.LocalDateTime;

import java.util.List;

public class ServiceProviderImpl implements IServiceProvider {

private final DatabaseManager db;

public ServiceProviderImpl() {

this.db = new DatabaseManager();

}

@Override

public void createProfile(String email, String firstName, String lastName, String phone, String resume) {

try {

if (!email.matches("^.+@.+\\..+$")) {

throw new InvalidEmailFormatException("Invalid email format: " + email);

}

Applicant applicant = new Applicant(0, firstName, lastName, email, phone, resume);

db.insertApplicant(applicant);

} catch (InvalidEmailFormatException e) {

System.err.println(e.getMessage());

}

}

@Override

public void apply(int applicantID, String coverLetter) {

int jobID = getLatestJobID();

if (jobID == -1) {

System.out.println("No job available to apply to.");

return;

}

applyForJob(jobID, applicantID, coverLetter);

}

@Override

public void applyForJob(int jobID, int applicantID, String coverLetter) {

JobApplication application = new JobApplication(0, jobID, applicantID, LocalDateTime.now(), coverLetter);

db.insertJobApplication(application);

}

@Override

public void postJob(int companyID, String jobTitle, String jobDescription, String jobLocation, double salary, String jobType) {

try {

if (salary < 0) {

throw new NegativeSalaryException("Salary cannot be negative: " + salary);

}

JobListing job = new JobListing(0, companyID, jobTitle, jobDescription, jobLocation, salary, jobType, LocalDateTime.now());

db.insertJobListing(job);

} catch (NegativeSalaryException e) {

System.err.println(e.getMessage());

}

}

@Override

public List<Applicant> getApplicants() {

return db.getApplicants();

}

@Override

public List<JobListing> getJobs() {

return db.getJobListings();

}

@Override

public List<JobListing> getJobsBySalaryRange(double minSalary, double maxSalary) {

return db.getJobListingsBySalaryRange(minSalary, maxSalary);

}

private int getLatestJobID() {

List<JobListing> jobs = db.getJobListings();

if (jobs.isEmpty()) return -1;

return jobs.get(jobs.size() - 1).getJobID();

}

}

**5. Exceptions handling: Create and implement the following exceptions in your application.**

* **Invalid Email Format Handling:**

package exception;

public class InvalidEmailFormatException extends Exception {

public InvalidEmailFormatException(String message) {

super(message);

}

}

* **Salary Calculation Handling:**

package exception;

public class NegativeSalaryException extends Exception {

public NegativeSalaryException(String message) {

super(message);

}

}

* **File Upload Exception Handling:**

package exception;

public class FileUploadException extends Exception {

public FileUploadException(String message) {

super(message);

}

}

* **Application Deadline Handling:**

package exception;

public class ApplicationDeadlineException extends Exception {

public ApplicationDeadlineException(String message) {

super(message);

}

}

* **Database Connection Handling:**

package exception;

public class DatabaseConnectionException extends Exception {

public DatabaseConnectionException(String message) {

super(message);

}

}

**ExceptionHandlers.java class:**

package util;

import exception.\*;

import java.io.File;

import java.sql.\*;

import java.time.LocalDate;

import java.util.List;

public class ExceptionHandlers {

public void validateEmail(String email) throws InvalidEmailFormatException {

if (email == null || !email.contains("@") || !email.matches("^.+@.+\\..+$")) {

throw new InvalidEmailFormatException("Invalid email format: " + email);

}

}

public double calculateAverageSalary(List<Double> salaries) throws NegativeSalaryException {

double total = 0;

int count = 0;

for (double salary : salaries) {

if (salary < 0) {

throw new NegativeSalaryException("Invalid negative salary found: " + salary);

}

total += salary;

count++;

}

return count == 0 ? 0 : total / count;

}

public void uploadResume(File file) throws FileUploadException {

if (!file.exists()) {

throw new FileUploadException("File not found: " + file.getName());

}

if (file.length() > 2 \* 1024 \* 1024) { // 2MB size limit

throw new FileUploadException("File size exceeds 2MB: " + file.getName());

}

if (!file.getName().endsWith(".pdf") && !file.getName().endsWith(".docx")) {

throw new FileUploadException("Unsupported file format: " + file.getName());

}

// proceed with upload

}

public void checkApplicationDeadline(LocalDate deadline) throws ApplicationDeadlineException {

if (LocalDate.now().isAfter(deadline)) {

throw new ApplicationDeadlineException("Application deadline has passed: " + deadline);

}

}

public Connection connectToDatabase(String url, String user, String password) throws DatabaseConnectionException {

try {

return DriverManager.getConnection(url, user, password);

} catch (SQLException e) {

throw new DatabaseConnectionException("Database connection failed: " + e.getMessage());

}

}

}

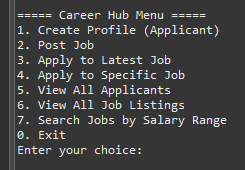
**6. Database Connectivity**

**Create and implement the following tasks in your application.**

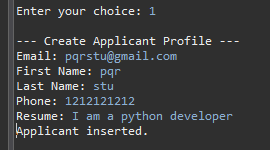
* Job Listing Retrieval
* Applicant Profile Creation
* Job Application Submission
* Company Job Posting
* Salary Range Query

All these functions are already implanted in ServiceProviderImpl.java

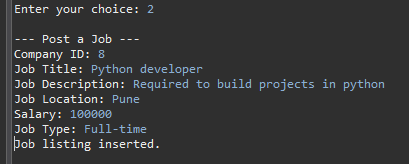
**7. Output:**

****

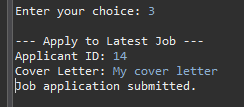
**Option 1 (Create profile) :**

****

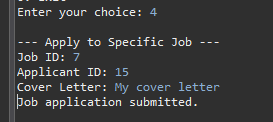
**Option 2 (Post job) :**

****

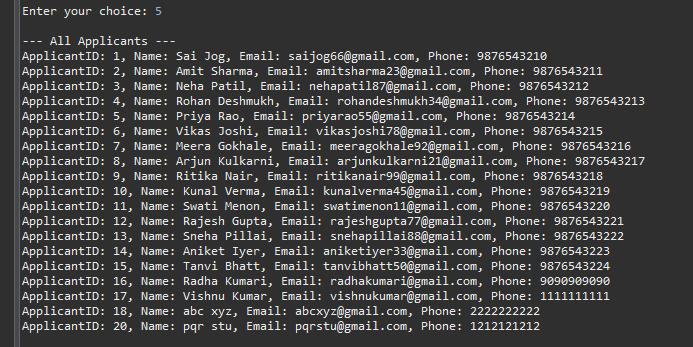
**Option 3 (Apply to latest job) :**

****

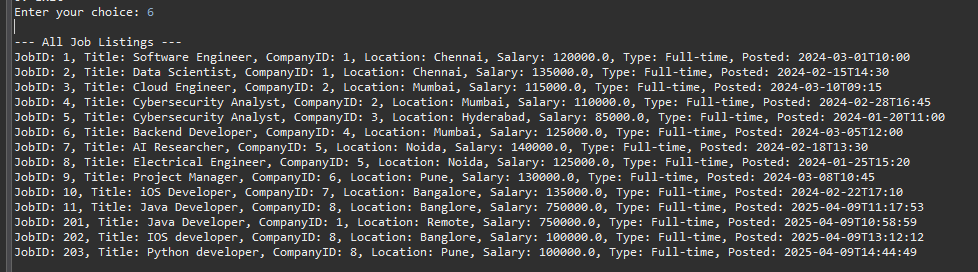
**Option 4 (Apply to specific job) :**

****

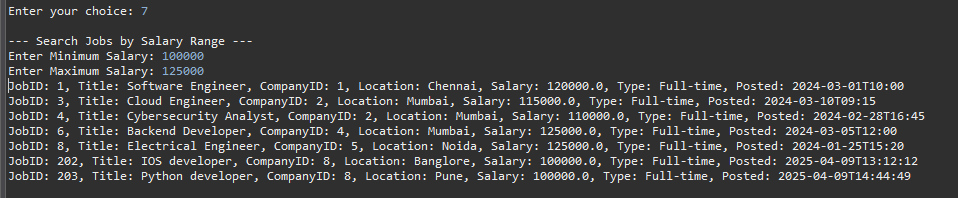
**Option 5 (View all applicants) :**

****

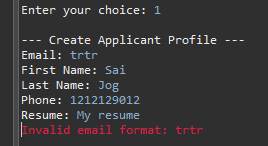
**Option 6 (View all job listings) :**

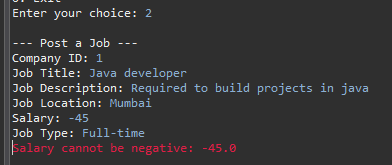
****

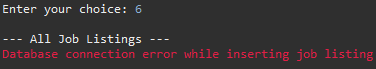
**Option 7 (Search jobs by salary range) :**

****

**Output of exceptions:**

****

****

****