Job Portal System - Full Project

src/main/resources/db.properties.sample

```
# db.properties.sample
db.driver=com.mysql.cj.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/job_portal?useSSL=false&serverTimezone=UTC
db.username=YOUR_DB_USERNAME
db.password=YOUR_DB_PASSWORD
```

sql/schema.sql

```
-- SQL schema for Job Portal (MySQL / MariaDB)

DROP TABLE IF EXISTS applications;
DROP TABLE IF EXISTS jobs;
DROP TABLE IF EXISTS employers;
DROP TABLE IF EXISTS users;
DROP TABLE IF EXISTS resumes;

CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  full_name VARCHAR(150),
  email VARCHAR(150) NOT NULL UNIQUE,
  role ENUM('JOBSEEKER','EMPLOYER','ADMIN') NOT NULL DEFAULT 'JOBSEEKER',
  phone VARCHAR(20),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE employers (
  employer_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL UNIQUE,
  company_name VARCHAR(200),
  company_description TEXT,
  website VARCHAR(255),
  CONSTRAINT fk_employer_user FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE jobs (
  job_id INT AUTO_INCREMENT PRIMARY KEY,
  employer_id INT NOT NULL,
  title VARCHAR(200) NOT NULL,
  description TEXT,
  location VARCHAR(150),
```

```
  employment_type ENUM('FULL_TIME','PART_TIME','CONTRACT','INTERNSHIP') DEFAULT 'FULL_TIME',
  salary_range VARCHAR(100),
  posted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  last_date DATE,
  is_active BOOLEAN DEFAULT TRUE,
  CONSTRAINT fk_job_employer FOREIGN KEY (employer_id) REFERENCES employers(employer_id) ON DELETE CASCADE
);

CREATE TABLE applications (
  application_id INT AUTO_INCREMENT PRIMARY KEY,
  job_id INT NOT NULL,
  user_id INT NOT NULL,
  cover_letter TEXT,
  resume_path VARCHAR(500),
  status ENUM('APPLIED','SHORTLISTED','REJECTED','HIRED') DEFAULT 'APPLIED',
  applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_app_job FOREIGN KEY (job_id) REFERENCES jobs(job_id) ON DELETE CASCADE,
  CONSTRAINT fk_app_user FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE resumes (
  resume_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  file_name VARCHAR(255),
  content_type VARCHAR(100),
  file_blob LONGBLOB,
  uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT fk_resume_user FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

src/main/java/com/yourname/jobportal/util/DBConnectionManager.java

```
package com.yourname.jobportal.util;

import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DBConnectionManager {
    private static final String PROPS_FILE = "/db.properties";
    private static String url;
    private static String username;
    private static String password;
    private static String driver;

    static {
        try (InputStream is = DBConnectionManager.class.getResourceAsStream(PROPS_FILE)) {
            Properties props = new Properties();
            if (is == null) {
                throw new RuntimeException("db.properties not found in classpath");
            }
            props.load(is);
            driver = props.getProperty("db.driver");
            url = props.getProperty("db.url");
            username = props.getProperty("db.username");
            password = props.getProperty("db.password");

            Class.forName(driver); // load driver
        } catch (IOException | ClassNotFoundException e) {
            throw new ExceptionInInitializerError("Failed to load DB properties: " + e.getMessage());
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, username, password);
    }
}
```

src/main/java/com/yourname/jobportal/util/PasswordUtil.java

```
package com.yourname.jobportal.util;

import org.mindrot.jbcrypt.BCrypt;

public class PasswordUtil {
    private static final int WORKLOAD = 10;

    public static String hashPassword(String plainTextPassword) {
        String salt = BCrypt.gensalt(WORKLOAD);
        return BCrypt.hashpw(plainTextPassword, salt);
    }

    public static boolean checkPassword(String plainTextPassword, String storedHash) {
        if (storedHash == null || !storedHash.startsWith("$2a$")) {
            throw new IllegalArgumentException("Invalid hash provided for comparison");
        }
        return BCrypt.checkpw(plainTextPassword, storedHash);
    }
}
```

src/main/java/com/yourname/jobportal/model/User.java

```
package com.yourname.jobportal.model;

import java.time.LocalDateTime;

public class User {
    private int userId;
```

```
    private String username;
    private String passwordHash;
    private String fullName;
    private String email;
    private String role;
    private String phone;
    private LocalDateTime createdAt;

    public User() {}

    // getters & setters...
    public int getUserId() { return userId; }
    public void setUserId(int userId) { this.userId = userId; }

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getPasswordHash() { return passwordHash; }
    public void setPasswordHash(String passwordHash) { this.passwordHash = passwordHash; }

    public String getFullName() { return fullName; }
    public void setFullName(String fullName) { this.fullName = fullName; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
}
```

src/main/java/com/yourname/jobportal/dao/UserDAO.java

```
package com.yourname.jobportal.dao;

import com.yourname.jobportal.model.User;
import java.sql.SQLException;
import java.util.Optional;

public interface UserDAO {
    int createUser(User user) throws SQLException;
    Optional<User> findByUsername(String username) throws SQLException;
    Optional<User> findById(int userId) throws SQLException;
    boolean updateUser(User user) throws SQLException;
}
```

src/main/java/com/yourname/jobportal/dao/impl/UserDAOImpl.java

```
package com.yourname.jobportal.dao.impl;

import com.yourname.jobportal.dao.UserDAO;
import com.yourname.jobportal.model.User;
import com.yourname.jobportal.util.DBConnectionManager;

import java.sql.*;
import java.time.LocalDateTime;
import java.util.Optional;

public class UserDAOImpl implements UserDAO {
    private static final String INSERT_USER_SQL =
            "INSERT INTO users (username, password_hash, full_name, email, role, phone) VALUES (?, ?, ?, ?, ?, ?)";

    private static final String SELECT_BY_USERNAME =
            "SELECT user_id, username, password_hash, full_name, email, role, phone, created_at FROM users WHERE use

    private static final String SELECT_BY_ID =
            "SELECT user_id, username, password_hash, full_name, email, role, phone, created_at FROM users WHERE use
```

```
private static final String UPDATE_USER =
        "UPDATE users SET full_name = ?, email = ?, phone = ? WHERE user_id = ?";

@Override
public int createUser(User user) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(INSERT_USER_SQL, Statement.RETURN_GENERATED_KEYS)) {

        ps.setString(1, user.getUsername());
        ps.setString(2, user.getPasswordHash());
        ps.setString(3, user.getFullName());
        ps.setString(4, user.getEmail());
        ps.setString(5, user.getRole());
        ps.setString(6, user.getPhone());

        int affected = ps.executeUpdate();
        if (affected == 0) {
            throw new SQLException("Creating user failed, no rows affected.");
        }

        try (ResultSet rs = ps.getGeneratedKeys()) {
            if (rs.next()) {
                int id = rs.getInt(1);
                user.setUserId(id);
                return id;
            } else {
                throw new SQLException("Creating user failed, no ID obtained.");
            }
        }
    }
}

@Override
public Optional<User> findByUsername(String username) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(SELECT_BY_USERNAME)) {

        ps.setString(1, username);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                User u = mapRowToUser(rs);
                return Optional.of(u);
            } else {
                return Optional.empty();
            }
        }
    }
}

@Override
public Optional<User> findById(int userId) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(SELECT_BY_ID)) {

        ps.setInt(1, userId);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                User u = mapRowToUser(rs);
                return Optional.of(u);
            } else {
                return Optional.empty();
            }
        }
    }
}

@Override
public boolean updateUser(User user) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(UPDATE_USER)) {

        ps.setString(1, user.getFullName());
        ps.setString(2, user.getEmail());
        ps.setString(3, user.getPhone());
        ps.setInt(4, user.getUserId());
```

```
            int updated = ps.executeUpdate();
            return updated > 0;
        }
    }

    private User mapRowToUser(ResultSet rs) throws SQLException {
        User u = new User();
        u.setUserId(rs.getInt("user_id"));
        u.setUsername(rs.getString("username"));
        u.setPasswordHash(rs.getString("password_hash"));
        u.setFullName(rs.getString("full_name"));
        u.setEmail(rs.getString("email"));
        u.setRole(rs.getString("role"));
        u.setPhone(rs.getString("phone"));
        Timestamp ts = rs.getTimestamp("created_at");
        if (ts != null) {
            u.setCreatedAt(ts.toLocalDateTime());
        }
        return u;
    }
}
```

src/main/java/com/yourname/jobportal/model/Job.java

```
package com.yourname.jobportal.model;

import java.time.LocalDate;
import java.time.LocalDateTime;

public class Job {
    private int jobId;
    private int employerId;
    private String title;
    private String description;
    private String location;
    private String employmentType;
    private String salaryRange;
    private LocalDate lastDate;
    private boolean isActive;
    private LocalDateTime postedAt;

    public Job() {}

    // getters & setters...
    public int getJobId() { return jobId; }
    public void setJobId(int jobId) { this.jobId = jobId; }

    public int getEmployerId() { return employerId; }
    public void setEmployerId(int employerId) { this.employerId = employerId; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public String getLocation() { return location; }
    public void setLocation(String location) { this.location = location; }

    public String getEmploymentType() { return employmentType; }
    public void setEmploymentType(String employmentType) { this.employmentType = employmentType; }

    public String getSalaryRange() { return salaryRange; }
    public void setSalaryRange(String salaryRange) { this.salaryRange = salaryRange; }

    public LocalDate getLastDate() { return lastDate; }
    public void setLastDate(LocalDate lastDate) { this.lastDate = lastDate; }

    public boolean isActive() { return isActive; }
    public void setActive(boolean active) { isActive = active; }

    public LocalDateTime getPostedAt() { return postedAt; }
    public void setPostedAt(LocalDateTime postedAt) { this.postedAt = postedAt; }
}
```

src/main/java/com/yourname/jobportal/dao/JobDAO.java

```
package com.yourname.jobportal.dao;

import com.yourname.jobportal.model.Job;

import java.sql.SQLException;
import java.util.List;
import java.util.Optional;

public interface JobDAO {
    int createJob(Job job) throws SQLException;
    Optional<Job> findById(int jobId) throws SQLException;
    List<Job> findAllActive(String keyword, String location) throws SQLException;
    boolean updateJob(Job job) throws SQLException;
    boolean deleteJob(int jobId) throws SQLException;
    List<Job> findByEmployerId(int employerId) throws SQLException;
}
```

src/main/java/com/yourname/jobportal/dao/impl/JobDAOImpl.java

```
package com.yourname.jobportal.dao.impl;

import com.yourname.jobportal.dao.JobDAO;
import com.yourname.jobportal.model.Job;
import com.yourname.jobportal.util.DBConnectionManager;

import java.sql.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class JobDAOImpl implements JobDAO {

    private static final String INSERT_JOB =
            "INSERT INTO jobs (employer_id, title, description, location, employment_type, salary_range, last_date,
            "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    private static final String SELECT_BY_ID =
            "SELECT * FROM jobs WHERE job_id = ?";

    private static final String SELECT_ACTIVE =
            "SELECT * FROM jobs WHERE is_active = TRUE " +
            "AND (? IS NULL OR (title LIKE ? OR description LIKE ?)) " +
            "AND (? IS NULL OR location LIKE ?) ORDER BY posted_at DESC";

    private static final String UPDATE_JOB =
            "UPDATE jobs SET title = ?, description = ?, location = ?, employment_type = ?, salary_range = ?, last_

    private static final String DELETE_JOB =
            "DELETE FROM jobs WHERE job_id = ?";

    private static final String SELECT_BY_EMPLOYER =
            "SELECT * FROM jobs WHERE employer_id = ? ORDER BY posted_at DESC";

    @Override
    public int createJob(Job job) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(INSERT_JOB, Statement.RETURN_GENERATED_KEYS)) {

            ps.setInt(1, job.getEmployerId());
            ps.setString(2, job.getTitle());
            ps.setString(3, job.getDescription());
            ps.setString(4, job.getLocation());
            ps.setString(5, job.getEmploymentType());
            ps.setString(6, job.getSalaryRange());
            if (job.getLastDate() != null) {
                ps.setDate(7, Date.valueOf(job.getLastDate()));
            } else {
                ps.setNull(7, Types.DATE);
            }
            ps.setBoolean(8, job.isActive());

            int affected = ps.executeUpdate();
            if (affected == 0) {
                throw new SQLException("Creating job failed, no rows affected.");
            }
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) {
                    int id = rs.getInt(1);
                    job.setJobId(id);
                    return id;
                } else {
                    throw new SQLException("Creating job failed, no ID obtained.");
                }
            }
        }
    }

    @Override
    public Optional<Job> findById(int jobId) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
```

```
         PreparedStatement ps = conn.prepareStatement(SELECT_BY_ID)) {
        ps.setInt(1, jobId);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                return Optional.of(mapRowToJob(rs));
            } else {
                return Optional.empty();
            }
        }
    }
}

@Override
public List<Job> findAllActive(String keyword, String location) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(SELECT_ACTIVE)) {

        if (keyword == null || keyword.trim().isEmpty()) {
            ps.setNull(1, Types.VARCHAR);
            ps.setNull(2, Types.VARCHAR);
            ps.setNull(3, Types.VARCHAR);
        } else {
            String like = "%" + keyword.trim() + "%";
            ps.setString(1, like);
            ps.setString(2, like);
            ps.setString(3, like);
        }

        if (location == null || location.trim().isEmpty()) {
            ps.setNull(4, Types.VARCHAR);
            ps.setNull(5, Types.VARCHAR);
        } else {
            String locLike = "%" + location.trim() + "%";
            ps.setString(4, locLike);
            ps.setString(5, locLike);
        }

        List<Job> list = new ArrayList<>();
        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                list.add(mapRowToJob(rs));
            }
        }
        return list;
    }
}

@Override
public boolean updateJob(Job job) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(UPDATE_JOB)) {

        ps.setString(1, job.getTitle());
        ps.setString(2, job.getDescription());
        ps.setString(3, job.getLocation());
        ps.setString(4, job.getEmploymentType());
        ps.setString(5, job.getSalaryRange());
        if (job.getLastDate() != null) ps.setDate(6, Date.valueOf(job.getLastDate()));
        else ps.setNull(6, Types.DATE);
        ps.setBoolean(7, job.isActive());
        ps.setInt(8, job.getJobId());

        return ps.executeUpdate() > 0;
    }
}

@Override
public boolean deleteJob(int jobId) throws SQLException {
    try (Connection conn = DBConnectionManager.getConnection();
         PreparedStatement ps = conn.prepareStatement(DELETE_JOB)) {
        ps.setInt(1, jobId);
        return ps.executeUpdate() > 0;
    }
}
@Override
```

```
    public List<Job> findByEmployerId(int employerId) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_BY_EMPLOYER)) {
            ps.setInt(1, employerId);
            List<Job> list = new ArrayList<>();
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    list.add(mapRowToJob(rs));
                }
            }
            return list;
        }
    }

    private Job mapRowToJob(ResultSet rs) throws SQLException {
        Job j = new Job();
        j.setJobId(rs.getInt("job_id"));
        j.setEmployerId(rs.getInt("employer_id"));
        j.setTitle(rs.getString("title"));
        j.setDescription(rs.getString("description"));
        j.setLocation(rs.getString("location"));
        j.setEmploymentType(rs.getString("employment_type"));
        j.setSalaryRange(rs.getString("salary_range"));
        Date ld = rs.getDate("last_date");
        if (ld != null) j.setLastDate(ld.toLocalDate());
        j.setActive(rs.getBoolean("is_active"));
        Timestamp ts = rs.getTimestamp("posted_at");
        if (ts != null) j.setPostedAt(ts.toLocalDateTime());
        return j;
    }
}
```

src/main/java/com/yourname/jobportal/model/Application.java

```
package com.yourname.jobportal.model;

import java.time.LocalDateTime;

public class Application {
    private int applicationId;
    private int jobId;
    private int userId;
    private String coverLetter;
    private String resumePath;
    private String status;
    private LocalDateTime appliedAt;

    public Application() {}

    // getters & setters...
    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 getUserId() { return userId; }
    public void setUserId(int userId) { this.userId = userId; }

    public String getCoverLetter() { return coverLetter; }
    public void setCoverLetter(String coverLetter) { this.coverLetter = coverLetter; }

    public String getResumePath() { return resumePath; }
    public void setResumePath(String resumePath) { this.resumePath = resumePath; }

    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }

    public LocalDateTime getAppliedAt() { return appliedAt; }
    public void setAppliedAt(LocalDateTime appliedAt) { this.appliedAt = appliedAt; }
}
```

src/main/java/com/yourname/jobportal/dao/ApplicationDAO.java

```
package com.yourname.jobportal.dao;

import com.yourname.jobportal.model.Application;

import java.sql.SQLException;
import java.util.List;
import java.util.Optional;

public interface ApplicationDAO {
    int createApplication(int jobId, int userId, String coverLetter, String resumePath) throws SQLException;
    List<Application> findByJobId(int jobId) throws SQLException;
    List<Application> findByUserId(int userId) throws SQLException;
    Optional<Application> findById(int applicationId) throws SQLException;
    boolean updateStatus(int applicationId, String status) throws SQLException;
}
```

src/main/java/com/yourname/jobportal/dao/impl/ApplicationDAOImpl.java

```
package com.yourname.jobportal.dao.impl;

import com.yourname.jobportal.dao.ApplicationDAO;
import com.yourname.jobportal.model.Application;
import com.yourname.jobportal.util.DBConnectionManager;

import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
```

```
public class ApplicationDAOImpl implements ApplicationDAO {

    private static final String INSERT_APP =
            "INSERT INTO applications (job_id, user_id, cover_letter, resume_path) VALUES (?, ?, ?, ?)";

    private static final String SELECT_BY_JOB =
            "SELECT * FROM applications WHERE job_id = ? ORDER BY applied_at DESC";

    private static final String SELECT_BY_USER =
            "SELECT * FROM applications WHERE user_id = ? ORDER BY applied_at DESC";

    private static final String SELECT_BY_ID =
            "SELECT * FROM applications WHERE application_id = ?";

    private static final String UPDATE_STATUS =
            "UPDATE applications SET status = ? WHERE application_id = ?";

    @Override
    public int createApplication(int jobId, int userId, String coverLetter, String resumePath) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(INSERT_APP, Statement.RETURN_GENERATED_KEYS)) {
            ps.setInt(1, jobId);
            ps.setInt(2, userId);
            ps.setString(3, coverLetter);
            ps.setString(4, resumePath);
            int affected = ps.executeUpdate();
            if (affected == 0) throw new SQLException("Creating application failed.");
            try (ResultSet rs = ps.getGeneratedKeys()) {
                if (rs.next()) return rs.getInt(1);
                else throw new SQLException("Creating application failed, no ID returned.");
            }
        }
    }

    @Override
    public List<Application> findByJobId(int jobId) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_BY_JOB)) {
            ps.setInt(1, jobId);
            List<Application> list = new ArrayList<>();
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) list.add(mapRow(rs));
            }
            return list;
        }
    }

    @Override
    public List<Application> findByUserId(int userId) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_BY_USER)) {
            ps.setInt(1, userId);
            List<Application> list = new ArrayList<>();
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) list.add(mapRow(rs));
            }
            return list;
        }
    }

    @Override
    public Optional<Application> findById(int applicationId) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
             PreparedStatement ps = conn.prepareStatement(SELECT_BY_ID)) {
            ps.setInt(1, applicationId);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) return Optional.of(mapRow(rs));
                else return Optional.empty();
            }
        }
    }

    @Override
    public boolean updateStatus(int applicationId, String status) throws SQLException {
        try (Connection conn = DBConnectionManager.getConnection();
```

```
             PreparedStatement ps = conn.prepareStatement(UPDATE_STATUS)) {
            ps.setString(1, status);
            ps.setInt(2, applicationId);
            return ps.executeUpdate() > 0;
        }
    }

    private Application mapRow(ResultSet rs) throws SQLException {
        Application a = new Application();
        a.setApplicationId(rs.getInt("application_id"));
        a.setJobId(rs.getInt("job_id"));
        a.setUserId(rs.getInt("user_id"));
        a.setCoverLetter(rs.getString("cover_letter"));
        a.setResumePath(rs.getString("resume_path"));
        a.setStatus(rs.getString("status"));
        Timestamp ts = rs.getTimestamp("applied_at");
        if (ts != null) a.setAppliedAt(ts.toLocalDateTime());
        return a;
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/RegisterServlet.java

```
package com.yourname.jobportal.web.servlet;

import com.yourname.jobportal.dao.UserDAO;
import com.yourname.jobportal.dao.impl.UserDAOImpl;
import com.yourname.jobportal.model.User;
import com.yourname.jobportal.util.PasswordUtil;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Optional;

@WebServlet("/register")
public class RegisterServlet extends HttpServlet {

    private final UserDAO userDAO = new UserDAOImpl();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.getRequestDispatcher("/jsp/register.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String username = req.getParameter("username");
        String rawPassword = req.getParameter("password");
        String fullName = req.getParameter("fullName");
        String email = req.getParameter("email");
        String role = req.getParameter("role");

        String error = null;
        if (username == null || username.trim().isEmpty()) error = "Username is required.";
        else if (rawPassword == null || rawPassword.length() < 6) error = "Password must be at least 6 characters."
        else if (email == null || email.trim().isEmpty()) error = "Email is required.";
        else if (!"JOBSEEKER".equals(role) && !"EMPLOYER".equals(role)) role = "JOBSEEKER";

        if (error != null) {
            req.setAttribute("error", error);
            req.getRequestDispatcher("/jsp/register.jsp").forward(req, resp);
            return;
        }

        try {
            Optional<User> existing = userDAO.findByUsername(username);
            if (existing.isPresent()) {
                req.setAttribute("error", "Username already exists.");
                req.getRequestDispatcher("/jsp/register.jsp").forward(req, resp);
                return;
            }

            User u = new User();
            u.setUsername(username);
            u.setFullName(fullName);
            u.setEmail(email);
            u.setRole(role);
            String hash = PasswordUtil.hashPassword(rawPassword);
            u.setPasswordHash(hash);

            int id = userDAO.createUser(u);

            resp.sendRedirect(req.getContextPath() + "/login?registered=true");
        } catch (SQLException e) {
            throw new ServletException("DB error during registration", e);
        }
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/LoginServlet.java

```
package com.yourname.jobportal.web.servlet;

import com.yourname.jobportal.dao.UserDAO;
import com.yourname.jobportal.dao.impl.UserDAOImpl;
import com.yourname.jobportal.model.User;
import com.yourname.jobportal.util.PasswordUtil;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Optional;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private final UserDAO userDAO = new UserDAOImpl();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.getRequestDispatcher("/jsp/login.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String username = req.getParameter("username");
        String rawPassword = req.getParameter("password");

        try {
            Optional<User> opt = userDAO.findByUsername(username);
            if (opt.isEmpty()) {
                req.setAttribute("error", "Invalid username or password.");
                req.getRequestDispatcher("/jsp/login.jsp").forward(req, resp);
                return;
            }
            User user = opt.get();
            boolean ok = PasswordUtil.checkPassword(rawPassword, user.getPasswordHash());
            if (!ok) {
                req.setAttribute("error", "Invalid username or password.");
                req.getRequestDispatcher("/jsp/login.jsp").forward(req, resp);
                return;
            }

            HttpSession session = req.getSession(true);
            session.setAttribute("currentUser", user);
            session.setAttribute("role", user.getRole());

            if ("ADMIN".equals(user.getRole())) {
                resp.sendRedirect(req.getContextPath() + "/admin/dashboard");
            } else if ("EMPLOYER".equals(user.getRole())) {
                resp.sendRedirect(req.getContextPath() + "/employer/dashboard");
            } else {
                resp.sendRedirect(req.getContextPath() + "/jobs");
            }
        } catch (SQLException e) {
            throw new ServletException("DB error during login", e);
        }
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/LogoutServlet.java

```
package com.yourname.jobportal.web.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
```

```
        HttpSession s = req.getSession(false);
        if (s != null) s.invalidate();
        resp.sendRedirect(req.getContextPath() + "/login?logout=true");
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/JobListServlet.java

```
package com.yourname.jobportal.web.servlet;

import com.yourname.jobportal.dao.JobDAO;
import com.yourname.jobportal.dao.impl.JobDAOImpl;
import com.yourname.jobportal.model.Job;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.SQLException;
import java.util.List;

@WebServlet("/jobs")
public class JobListServlet extends HttpServlet {
    private final JobDAO jobDAO = new JobDAOImpl();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String keyword = req.getParameter("q");
        String location = req.getParameter("location");

        try {
            List<Job> jobs = jobDAO.findAllActive(keyword, location);
            req.setAttribute("jobs", jobs);
            req.getRequestDispatcher("/jsp/job-list.jsp").forward(req, resp);
        } catch (SQLException e) {
            throw new ServletException("Unable to fetch jobs", e);
        }
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/employer/PostJobServlet.java

```
package com.yourname.jobportal.web.servlet.employer;

import com.yourname.jobportal.dao.JobDAO;
import com.yourname.jobportal.dao.impl.JobDAOImpl;
import com.yourname.jobportal.model.Job;
import com.yourname.jobportal.model.User;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.sql.SQLException;
import java.time.LocalDate;

@WebServlet("/employer/post-job")
public class PostJobServlet extends HttpServlet {
    private final JobDAO jobDAO = new JobDAOImpl();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.getRequestDispatcher("/jsp/employer/post-job.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        HttpSession session = req.getSession(false);
        if (session == null || session.getAttribute("currentUser") == null) {
            resp.sendRedirect(req.getContextPath() + "/login");
            return;
        }
        User u = (User) session.getAttribute("currentUser");
        if (!"EMPLOYER".equals(u.getRole())) {
            resp.sendError(HttpServletResponse.SC_FORBIDDEN, "Only employers can post jobs.");
            return;
        }
```

```
        String title = req.getParameter("title");
        String description = req.getParameter("description");
        String location = req.getParameter("location");
        String employmentType = req.getParameter("employmentType");
        String salaryRange = req.getParameter("salaryRange");
        String lastDateStr = req.getParameter("lastDate");

        Job job = new Job();
        job.setEmployerId(u.getUserId());
        job.setTitle(title);
        job.setDescription(description);
        job.setLocation(location);
        job.setEmploymentType(employmentType);
        job.setSalaryRange(salaryRange);
        if (lastDateStr != null && !lastDateStr.trim().isEmpty()) {
            job.setLastDate(LocalDate.parse(lastDateStr));
        }
        job.setActive(true);

        try {
            jobDAO.createJob(job);
            resp.sendRedirect(req.getContextPath() + "/jobs");
        } catch (SQLException e) {
            throw new ServletException("Unable to post job", e);
        }
    }
}
```

src/main/java/com/yourname/jobportal/web/servlet/ApplyServlet.java

```
package com.yourname.jobportal.web.servlet;

import com.yourname.jobportal.dao.ApplicationDAO;
import com.yourname.jobportal.dao.impl.ApplicationDAOImpl;
import com.yourname.jobportal.model.User;

import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.File;
import java.io.IOException;
import java.sql.SQLException;

@WebServlet("/apply")
@MultipartConfig(fileSizeThreshold = 1024 * 1024,
        maxFileSize = 5 * 1024 * 1024,
        maxRequestSize = 10 * 1024 * 1024)
public class ApplyServlet extends HttpServlet {

    private final ApplicationDAO applicationDAO = new ApplicationDAOImpl();

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        HttpSession s = req.getSession(false);
        if (s == null || s.getAttribute("currentUser") == null) {
            resp.sendRedirect(req.getContextPath() + "/login");
            return;
        }
        User user = (User) s.getAttribute("currentUser");
        if (!"JOBSEEKER".equals(user.getRole())) {
            resp.sendError(HttpServletResponse.SC_FORBIDDEN, "Only jobseekers can apply.");
            return;
        }

        String jobIdStr = req.getParameter("jobId");
        String coverLetter = req.getParameter("coverLetter");

        Part resumePart = req.getPart("resume");
        String resumePath = null;
        if (resumePart != null && resumePart.getSize() > 0) {
            String uploadsDir = getServletContext().getRealPath("/uploads");
            File userDir = new File(uploadsDir, String.valueOf(user.getUserId()));
            if (!userDir.exists()) userDir.mkdirs();

            String filename = System.currentTimeMillis() + "_" + resumePart.getSubmittedFileName();
            File f = new File(userDir, filename);
            resumePart.write(f.getAbsolutePath());
            resumePath = "uploads/" + user.getUserId() + "/" + filename;
        }

        try {
            int jobId = Integer.parseInt(jobIdStr);
            applicationDAO.createApplication(jobId, user.getUserId(), coverLetter, resumePath);
            resp.sendRedirect(req.getContextPath() + "/jobs?applied=true");
        } catch (NumberFormatException | SQLException e) {
            throw new ServletException("Unable to submit application", e);
        }
    }
}
```

JSP Pages (src/main/webapp/jsp/...)

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login - Job Portal</title>
</head>
<body>
<h2>Login</h2>

<c:if test="${not empty param.registered}">
    <p style="color:green">Registration successful. Please login.</p>
</c:if>

<c:if test="${not empty error}">
    <p style="color:red">${error}</p>
</c:if>

<form action="${pageContext.request.contextPath}/login" method="post">
    <label>Username: <input type="text" name="username" required></label><br>
    <label>Password: <input type="password" name="password" required></label><br>
    <button type="submit">Login</button>
</form>

<p>Don't have an account? <a href="${pageContext.request.contextPath}/register">Register</a></p>
</body>
</html>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Register - Job Portal</title>
</head>
<body>
<h2>Register</h2>

<c:if test="${not empty error}">
    <p style="color:red">${error}</p>
</c:if>

<form action="${pageContext.request.contextPath}/register" method="post">
    <label>Username: <input type="text" name="username" required></label><br>
    <label>Password: <input type="password" name="password" required></label><br>
    <label>Full name: <input type="text" name="fullName"></label><br>
    <label>Email: <input type="email" name="email" required></label><br>
    <label>Role:
        <select name="role">
            <option value="JOBSEEKER">Jobseeker</option>
            <option value="EMPLOYER">Employer</option>
        </select>
    </label><br>
    <button type="submit">Register</button>
</form>

<p>Already registered? <a href="${pageContext.request.contextPath}/login">Login</a></p>
</body>
</html>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Jobs - Job Portal</title>
</head>
<body>
<h2>Available Jobs</h2>

<form method="get" action="${pageContext.request.contextPath}/jobs">
    <input type="text" name="q" placeholder="Keyword" value="${param.q}"/>
```

```
    <input type="text" name="location" placeholder="Location" value="${param.location}"/>
    <button type="submit">Search</button>
</form>

<c:if test="${empty jobs}">
    <p>No jobs found.</p>
</c:if>

<ul>
    <c:forEach var="job" items="${jobs}">
        <li>
            <a href="${pageContext.request.contextPath}/job?id=${job.jobId}">${job.title}</a>
            <br/>
            <small>${job.location} — ${job.employmentType}</small>
        </li>
    </c:forEach>
</ul>

<p><a href="${pageContext.request.contextPath}/">Home</a></p>
</body>
</html>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    com.yourname.jobportal.model.Job job = (com.yourname.jobportal.model.Job) request.getAttribute("job");
%>
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>${job.title} - Job Detail</title>
</head>
<body>
<h2>${job.title}</h2>
<p><strong>Location:</strong> ${job.location}</p>
<p><strong>Type:</strong> ${job.employmentType}</p>
<p><strong>Salary:</strong> ${job.salaryRange}</p>
<p>${job.description}</p>

<c:choose>
    <c:when test="${not empty sessionScope.currentUser and sessionScope.role == 'JOBSEEKER'}">
        <form action="${pageContext.request.contextPath}/apply" method="post" enctype="multipart/form-data">
            <input type="hidden" name="jobId" value="${job.jobId}"/>
            <label>Cover letter:<br/><textarea name="coverLetter" rows="6" cols="60"></textarea></label><br>
            <label>Upload resume: <input type="file" name="resume"/></label><br>
            <button type="submit">Apply</button>
        </form>
    </c:when>
    <c:otherwise>
        <p><a href="${pageContext.request.contextPath}/login">Login as jobseeker to apply</a></p>
    </c:otherwise>
</c:choose>

<p><a href="${pageContext.request.contextPath}/jobs">Back to jobs</a></p>
</body>
</html>

<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Post Job - Employer</title>
</head>
<body>
<h2>Post a New Job</h2>

<form action="${pageContext.request.contextPath}/employer/post-job" method="post">
    <label>Title: <input type="text" name="title" required/></label><br/>
    <label>Description:<br/><textarea name="description" rows="8" cols="60"></textarea></label><br/>
    <label>Location: <input type="text" name="location"/></label><br/>
    <label>Employment Type:
        <select name="employmentType">
            <option value="FULL_TIME">Full-time</option>
            <option value="PART_TIME">Part-time</option>
            <option value="CONTRACT">Contract</option>
            <option value="INTERNSHIP">Internship</option>
        </select>
    </label><br/>
    <label>Salary Range: <input type="text" name="salaryRange"/></label><br/>
    <label>Last Date (YYYY-MM-DD): <input type="date" name="lastDate"/></label><br/>
    <button type="submit">Post Job</button>
</form>

<p><a href="${pageContext.request.contextPath}/employer/dashboard">Back to Dashboard</a></p>
</body>
</html>
```

WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <display-name>Job Portal</display-name>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>jsp/login.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yourname</groupId>
  <artifactId>job-portal</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.1.0</version>
    </dependency>

    <dependency>
      <groupId>org.mindrot</groupId>
      <artifactId>jbcrypt</artifactId>
      <version>0.4</version>
    </dependency>

    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>4.0.4</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>jakarta.servlet.jsp.jstl</groupId>
      <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
      <version>2.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.web</groupId>
      <artifactId>jakarta.servlet.jsp.jstl</artifactId>
      <version>2.0.0</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>job-portal</finalName>
  </build>
</project>
```