**Job Portal Backend - Project Requirements**

**Project Overview**

The goal of this project is to build a **backend system for a job portal** using **Spring Boot**, **Spring Data JPA**, and **MySQL**. The system will allow **admins**, **employers**, and **job seekers** to perform various tasks, such as posting jobs, applying for jobs, and managing applications. The backend will expose REST APIs, and the functionality will be tested using **Postman**.

**Functional Requirements**

**1. User Roles**

The system will have three types of users:

1. **Admin**:
   * Can view and delete users.
2. **Employer**:
   * Can post jobs, view applications for their jobs, and update application statuses.
   * Can add reviews for jobs they posted.
3. **Job Seeker**:
   * Can view all jobs, apply for jobs, and upload resumes.
   * Can view their own applications.

**2. API Endpoints**

The backend will expose the following REST APIs:

**Admin APIs**

* **Get All Users**
  + Pagination and filtering by role.
* **Delete User**
  + Only ADMIN can delete users.

**Employer APIs**

* **Post a Job**
  + Only EMPLOYER can post jobs.
* **Get All Jobs Posted by Employer**
  + Pagination and filtering by job title or location.
* **View Applications for a Job**
  + Pagination and filtering by application status.
* **Update Application Status**
  + Only the employer who posted the job can update the status.
* **a Review for a Job** **Add**
  + Only EMPLOYER can add reviews for jobs they posted.

**Job Seeker APIs**

* **Apply for a Job**
  + Only JOB\_SEEKER can apply for jobs.
* **Upload Resume**
  + Only JOB\_SEEKER can upload a resume.
* **Get All Applications by Job Seeker**
  + Pagination and filtering by job title or application status.
* **View All Jobs**
  + Pagination and filtering by job title, location, or employer.

**Review APIs**

* **Get Reviews for a Job**
  + Pagination and filtering by rating.

**3. Features to Implement**

1. **Pagination and Filtering**:
   * Implement pagination and filtering for listing jobs, applications, and reviews.
2. **Role-Based Access Control**:
   * Restrict access to APIs based on user roles (ADMIN, EMPLOYER, JOB\_SEEKER).
3. **DTOs and Mappers**:
   * Use DTOs (Data Transfer Objects) and mappers (e.g., MapStruct) to separate entity and API layers.
4. **Validation**:
   * Validate inputs for posting jobs, applying for jobs, and updating application statuses.

**4. Example Workflow**

1. **Admin** logs in and views all users.
2. **Employer** posts a job .
3. **Job Seeker** views all jobs and applies for a job (POST /api/applications).
4. **Employer** views applications for their job and updates the status
5. **Job Seeker** uploads a resume and views their applications
6. **Employer** adds a review for a job.

**5. Tools and Technologies**

* **Backend**: Spring Boot
* **Database**: MySQL
* **ORM**: Spring Data JPA
* **API Testing**: Postman
* **Mapper**: MapStruct or Manual DTO Mapping
* **Security**: Spring Security (optional for role-based access)

**6. Deliverables**

1. **Source Code**:
   * Complete Spring Boot project with all entities, repositories, services, and controllers.
2. **Postman Collection**:
   * A Postman collection with sample requests for all endpoints.
3. **Documentation**:
   * A README file explaining how to set up and run the project.

**7. Submission Guidelines**

1. **Source Code**:
   * Push your code to a GitHub repository and share the link.
2. **Postman Collection**:
   * Export the Postman collection and include it in your submission.
3. **README File**:
   * Provide clear instructions on how to set up and run the project.