Job Posting Application Documentation

1. Introduction

This is a RESTful web application for managing job postings. The application is built using Spring Boot,

Hibernate, and PostgreSQL. It allows CRUD operations on job posts, including searching posts by keywords.

2. Technologies Used

- Spring Boot
- Hibernate (JPA)
- PostgreSQL
- Maven
- Java 17
- React (Frontend)

3. Project Structure

- com.pratik.spring_boot_rest
  - controller
    - JobRestController: Handles API requests related to job postings.
  - model
    - JobPost: Represents the job post entity.
  - repo
    - JobRepo: Interface extending JpaRepository for database operations.
  - service

- JobService: Contains business logic for managing job posts.

4. Key Components

- JobRestController: Defines endpoints for CRUD operations and keyword search.
- JobService: Implements business logic and interacts with JobRepo.
- JobRepo: Handles database queries using Spring Data JPA.
- JobPost: Entity class annotated with @Entity representing a database table.

5. Database Design

The application uses PostgreSQL. The 'JobPost' table has the following structure:

- postId (Primary Key): Integer
- postProfile: String
- postDesc: String
- reqExperience: Integer
- postTechStack: List<String>

6. REST API Endpoints

- GET /jobposts: Fetch all job posts.
- GET /jobposts/{postId}: Fetch a specific job post by ID.
- GET /jobposts/keyword/{keyword}: Search job posts by keyword.
- POST /jobposts: Add a new job post.
- PUT /jobposts: Update an existing job post.
- DELETE /jobposts/{postId}: Delete a job post by ID.

7. Detailed Examples

1. Add a Job Post (POST /jobposts):

Request Body:

```
{
  "postId": 1,
  "postProfile": "Java Developer",
  "postDesc": "Experience in core Java.",
  "reqExperience": 2,
  "postTechStack": ["Java", "Spring Boot"]
}
```

2. Search by Keyword (GET /jobposts/keyword/Java):

Response:

```
[
  {
    "postId": 1,
    "postProfile": "Java Developer",
    "postDesc": "Experience in core Java.",
    "reqExperience": 2,
    "postTechStack": ["Java", "Spring Boot"]
  }
]
```

8. How to Run

1. Ensure PostgreSQL is installed and running.

2. Create a database named 'Job-App' and update credentials in application.properties.

3. Run the Spring Boot application using your IDE or Maven command: mvn spring-boot:run.

4. Access API endpoints via Postman or integrate with a frontend (e.g., React).

9. Improvements

1. Add authentication and authorization using Spring Security.

2. Implement pagination for job post listings.

3. Integrate a frontend for better user experience.

4. Add Docker support for containerization.