Spring Boot REST Application: Job Portal

Spring Boot REST Application: Job Portal

1. Application Overview

- **Name**: Spring Boot Job Portal REST API

- **Purpose**: Manage job postings, including retrieving, adding, updating, and deleting job posts.

- **Technology Stack**:

- Spring Boot (Spring Web, Spring Data)

- Java Collections for in-memory data storage

- Jackson for JSON/XML serialization

- Tools: Postman for API testing

2. Project Structure

- **Controller Layer (JobRestController)**:

Handles HTTP requests and maps them to service methods.

Supports JSON and XML data formats using content negotiation.

- **Service Layer (JobService)**:

Contains business logic and acts as an intermediary between the controller and repository layers.

- **Repository Layer (JobRepo)**:

Simulates a database using an in-memory list (List<JobPost>).

Provides CRUD operations.

- **Model Layer (JobPost)**:

Represents a job posting with attributes like postId, postProfile, postDesc, reqExperience, and postTechSta

- **Main Class (SpringBootRestApplication)**:

Entry point for the Spring Boot application.

3. Key Components and Flow

- **Endpoints in JobRestController**:

- `GET /jobposts`: Retrieves all job postings.

- `GET /jobposts/{postId}`: Retrieves a specific job post by its ID.

- `POST /jobposts`: Adds a new job post (only accepts XML input).

- `PUT /jobposts`: Updates an existing job post.

- `DELETE /jobposts/{postId}`: Deletes a job post by its ID.

- **Data Flow**:

- Controller -> Receives HTTP requests, delegates processing to the service layer.

- Service -> Contains business logic and interacts with the repository.

- Repository -> Performs CRUD operations on the in-memory list.

4. Features

- **Content Negotiation**:

- Supports JSON and XML formats for requests and responses.

- Example: Add `Accept: application/json` or `Accept: application/xml` in request headers.

- **In-Memory Storage**:

- Uses an ArrayList in JobRepo to simulate database operations.

- **Data Initialization**:

- Pre-populates JobRepo with 20 predefined job posts.

- **Dynamic Filtering**:

- Deletes job posts efficiently using `List.removeIf()`.

5. Example Endpoints

- **Retrieve All Job Posts**:

- Request:

```
GET /jobposts
Accept: application/json
```

- Response (Example):

```json
[
{
"postId": 1,
"postProfile": "Java Developer",
"postDesc": "Must have good experience in core Java and advanced Java",
"reqExperience": 2,
"postTechStack": ["Core Java", "J2EE", "Spring Boot", "Hibernate"]
}
]
```

- **Add a New Job Post (XML Only)**:

- Request:

```xml
POST /jobposts
Content-Type: application/xml
```

<JobPost>

<postId>21</postId>

<postProfile>AI Specialist</postProfile>

<postDesc>Develop AI models for various industries</postDesc>

<reqExperience>5</reqExperience>

<postTechStack>

<postTechStack>Python</postTechStack>

<postTechStack>AI</postTechStack>

</postTechStack>

</JobPost>

```

- Response: `201 Created`

- **Update a Job Post**:

- Request:

```json

PUT /jobposts

Content-Type: application/json

{

"postId": 1,

"postProfile": "Senior Java Developer",

"postDesc": "Expert in core and advanced Java",

"reqExperience": 5,

"postTechStack": ["Core Java", "Spring Boot"]

}

```

- Response: Updated job post details.
```

- **Delete a Job Post**:

- Request:

```

DELETE /jobposts/1

```

- Response: `Data has been deleted successfully`.

6. Notes for Improvement

- **Validation**: Add input validation using annotations like `@NotNull`, `@Size`.

- **Database Integration**: Replace in-memory storage with a real database (e.g., MySQL, H2).

- **Error Handling**: Implement global exception handling using `@ControllerAdvice`.

- **Logging**: Add proper logging using SLF4J or Logback.