TheKiranAcademy

Job Portal API Documentation

This document provides detailed technical guidance on using the Job Portal API, a RESTful service enabling seamless interactions between employers and job seekers via a modern job marketplace platform. It outlines the API's core features, technical stack, intended users, and endpoint organization. Developers integrating front-end applications or third-party systems will find comprehensive information on managing job postings, applications, skills, notifications, and analytics. Hosted locally and documented with Swagger UI, this API is designed for scalability, extensibility, and high performance.

Overview and Key Features

The Job Portal API is a scalable RESTful backend developed with **Spring Boot** and **MySQL 8**, optimized for efficient job marketplace functionalities. Key features include robust **Job Posting and Management** capabilities allowing employers to specify detailed listings including titles, descriptions, salary ranges, skills, and employment types. The **Bulk Skill Upload** functionality supports easy addition of required skills by name, automatically adding new entries to the system.

The API also manages the entire **Application Lifecycle** for job seekers, who can submit applications with cover letters and receive real-time status updates. Employers can track applications, update statuses such as Accepted or Rejected, and add **Review Comments** for comprehensive feedback.

Additional features like **Employer Analytics** provide critical insights on posting performance including view counts and applications received, aiding employers in optimizing their listings. The integrated **Notification System** ensures job seekers remain informed of application progress, with a dedicated endpoint for fetching updates.

Technical Architecture and Stack

The backend leverages **Spring Boot 2.5.6** as the primary framework, offering rapid development and built-in integration with various Java libraries and tools. Data persistence is managed via **MySQL 8**, interfaced through **Spring Data JPA** to enable seamless ORM and efficient query operations.

Validation of incoming requests and data integrity is enforced through the **Javax Validation API**. The API is built using **Java 17**, taking advantage of enhanced language features for improved maintainability and performance. Build automation and dependency management rely on **Maven** for consistency in development cycles.

Developers can interactively explore and test API endpoints using the integrated **Springdoc OpenAPI** interface, accessible at **http://localhost:8091/swagger-ui.html**. Manual testing and debugging during development is supported via **Postman**, ensuring fidelity prior to deployment.

API Endpoints and Resource Structure

The API endpoints are logically organized by resource domains to support modular interaction:

- **/jobs** — Manage job postings including creation, updates, and retrieval of job listings.
- **/applications** — Handle job applications submitted by seekers, enable status updates and review comments by employers.
- **/notifications** — Access application status notifications to inform job seekers of any progress or feedback.
- **/skills** — Create and manage individual and bulk skill entries ensuring comprehensive job qualification definitions.
- **/categories** (administrators) — Manage job categories for enhanced search filtering and classification.

Each endpoint supports standard RESTful HTTP methods such as GET, POST, PUT, and DELETE, enabling full lifecycle management. Response payloads are structured in JSON, adhering to well-defined schemas facilitated by OpenAPI specifications.

User Roles and Access

The Job Portal API serves multiple user roles, each with different responsibilities and access levels. **Employers** can create and manage job listings, review applications, update statuses, and access analytics relevant to their posted jobs. **Job Seekers** utilize the API to search, filter, and apply for jobs, as well as track the progress of their applications through real-time notifications.

Additionally, **Administrators** have elevated privileges to manage system-wide resources such as skills and categories to maintain data integrity and system scalability. **Developers** using the API as the backend for front-end applications benefit from clear documentation and interactive testing environments that promote rapid integration and seamless user experience development.

Role-based authentication and authorization mechanisms are expected to be integrated at the application layer to protect sensitive operations and maintain secure access. These may include token-based authentication methods such as JWT (JSON Web Tokens).

Search, Recommendations, and Notifications

The API offers powerful search capabilities enabling job seekers to discover relevant opportunities through multiple filters including keywords, job categories, locations, and required skills. This multi-faceted search system enhances user experience by narrowing down large job inventories to match precise candidate preferences.

Moreover, a **Recommended Jobs** feature (implementation assumed) leverages user skill profiles to suggest personalized job listings, improving engagement and application success rates.

To keep users informed efficiently, the API's notification subsystem delivers timely application status updates and review comments through a dedicated notifications endpoint. This enables front-end clients to poll or subscribe to updates, ensuring job seekers remain engaged and aware of their application progress in near real time.

Employer Dashboard Analytics

The Job Portal API provides valuable insights into job posting performance through the Employer Dashboard Analytics feature. Employers can access metrics such as the number of views per job listing, total applications received, and other engagement statistics. These data points help employers evaluate the effectiveness of their postings and make informed decisions to optimize job descriptions, adjust salary ranges, or refine skill requirements.

Such analytics contribute to an iterative improvement process that can increase recruitment efficiency by highlighting which jobs attract the most qualified candidates and which may need further promotion or adjustment.

Implementation considerations include aggregating data efficiently while maintaining low latency for real-time dashboard responsiveness. This feature complements the overall goal of creating a data-driven hiring process for employers.

API Access, Documentation, and Future Enhancements

The Job Portal API is accessible at **http://localhost:8091/api**, providing a stable interface for integration with web or mobile clients. Comprehensive API documentation is available through an interactive Swagger UI interface at **http://localhost:8091/swagger-ui.html**, detailing all available endpoints, request and response schemas, status codes, and example usage.

This documentation facilitates both exploratory testing and straightforward adoption by development teams and third-party integrators. Authentication, error handling, and data validation guidelines are also included to ensure robust and secure integration.

Future enhancements under consideration include advanced skill suggestion algorithms to streamline job posting, job status history tracking for deeper application lifecycle insights, and enriched analytics dashboards with predictive recruiting indicators. The API's modular and extensible architecture supports such iterative improvements without disrupting existing functionality.

Objective

The primary objective of the Job Portal API is to create a robust, scalable, and user-friendly backend system that streamlines the job search and hiring process for both employers and job seekers. By leveraging modern web technologies, the API aims to facilitate efficient job discovery, application management, and performance tracking, fostering a seamless connection between talent and opportunities. The system is designed to support a web or mobile application, providing a foundation for a dynamic job marketplace that enhances user experience and operational efficiency.

Employers Workflow

The Employers Workflow outlines the steps an employer takes to interact with the Job Portal API, from account creation to managing job listings and applications. This workflow leverages all implemented features, including **Bulk Skill Upload**, **Employer Dashboard Analytics**, **Application Review Comments**, and **Swagger UI**, to provide a comprehensive hiring experience. All API calls are made to http://localhost:8091/api and can be tested using the Postman collection "Job Portal API."

Workflow Steps

1. **Register as an Employer**
   - **Goal**: Create an employer account to access job posting features.
   - **API**: POST /api/users/register
   - **Postman Request**: Users > Register User
   - **Details**:
     - Body: {"email": "employer@example.com", "password": "password", "role": "EMPLOYER", "firstName": "John", "lastName": "Doe"}
     - Note the userId (e.g., 2) from the response for subsequent steps.
   - **Expected Outcome**: Employer account created with role: EMPLOYER.
2. **Create a Company**
   - **Goal**: Set up a company profile for job listings.
   - **API**: POST /api/companies
   - **Postman Request**: Companies > Create Company
   - **Details**:
     - Body: {"name": "Tech Corp", "description": "Innovative tech solutions", "website": "https://techcorp.com", "location": "New York"}
     - Note the companyId (e.g., 1).
   - **Expected Outcome**: Company created and associated with the employer.
3. **Create a Category (Optional)**
   - **Goal**: Define a job category if not already available.
   - **API**: POST /api/categories
   - **Postman Request**: Categories > Create Category
   - **Details**:
     - Body: {"name": "Software Development"}
     - Note the categoryId (e.g., 1).
   - **Expected Outcome**: Category created for job classification.
4. **Create a Location (Optional)**
   - **Goal**: Define a job location if not already available.
   - **API**: POST /api/locations
   - **Postman Request**: Locations > Create Location
   - **Details**:
     - Body: {"city": "New York", "state": "NY", "country": "USA"}
     - Note the locationId (e.g., 1).
   - **Expected Outcome**: Location created for job assignment.
5. **Create Skills (Optional, Manual)**
   - **Goal**: Add skills to the system if not using **Bulk Skill Upload**.
   - **API**: POST /api/skills
   - **Postman Request**: Skills > Create Skill
   - **Details**:
     - Body: {"name": "Java"}
     - Repeat for additional skills (e.g., Python).
     - Note skill IDs if needed.
   - **Expected Outcome**: Skills created for job requirements.
6. **Create a Job**
   - **Goal**: Post a new job listing with skill names.
   - **API**: POST /api/jobs
   - **Postman Request**: Jobs > Create Job
   - **Details**:
     - Body:

```
{
  "userId": 2,
  "title": "Software Engineer",
  "description": "Develop web applications",
  "companyId": 1,
  "categoryId": 1,
  "locationId": 1,
  "salaryRange": "$100,000 - $120,000",
  "employmentType": "FULL_TIME",
  "skillNames": ["Java", "Python", "NewSkill"]
}
```

     - Note the jobId (e.g., 1).
   - **Expected Outcome**: Job created, new skills (e.g., NewSkill) added automatically.
1. **View Job Listings**
   - **Goal**: Retrieve all jobs posted by the employer.
   - **API**: GET /api/jobs
   - **Postman Request**: Jobs > Get All Jobs
   - **Details**:
     - Query: ?page=0&size=10
     - Response includes the created job (ID 1).
   - **Expected Outcome**: Paginated list of jobs displayed.
2. **View Job Analytics**
   - **Goal**: Check the performance of a posted job (views, applications).
   - **API**: GET /api/jobs/{id}/analytics
   - **Postman Request**: Jobs > Get Job Analytics
   - **Details**:
     - URL: {{baseUrl}}/jobs/1/analytics
     - Response: {"jobId": 1, "views": 50, "applications": 10}
   - **Expected Outcome**: Analytics data for the job retrieved.
3. **View Applications for a Job**
   - **Goal**: Review applications submitted for a job.
   - **API**: GET /api/applications/job/{jobId}
   - **Postman Request**: Job Applications > Get Applications by Job
   - **Details**:
     - URL: {{baseUrl}}/applications/job/1
     - Response includes applications (e.g., applicationId: 1).
   - **Expected Outcome**: List of applications for the job.
4. **Update Application Status with Comment**
   - **Goal**: Update an application's status and add a review comment.
   - **API**: PUT /api/applications/{id}/status
   - **Postman Request**: Job Applications > Update Application Status with Comment
   - **Details**:
     - URL: {{baseUrl}}/applications/1/status?status=ACCEPTED&comment=Strong%20candidate
     - Triggers a notification to the job seeker.
   - **Expected Outcome**: Application status updated, comment saved, notification sent.
5. **Access API Documentation**
   - **Goal**: Explore API endpoints for reference.
   - **Action**: Open http://localhost:8091/swagger-ui.html
   - **Details**:
     - Browse endpoints like POST /api/jobs, PUT /api/applications/{id}/status.
     - Test endpoints directly in Swagger UI.
   - **Expected Outcome**: Interactive API documentation displayed.

Notes

- **Prerequisites**: Employer must have a valid userId with role: EMPLOYER.
- **Optional Steps**: Steps 3–5 (category, location, skill creation) are optional if data exists or **Bulk Skill Upload** is used.
- **Notifications**: Application status updates trigger notifications, viewable by job seekers.
- **Swagger**: Use Swagger UI to understand request/response formats and test endpoints.

Job Seekers Workflow

The Job Seekers Workflow outlines the steps a job seeker takes to interact with the Job Portal API, from account creation to applying for jobs and tracking application status. This workflow incorporates features like **Application Status Notifications**, **Application Review Comments**, and **Swagger UI** to enhance the job search experience. All API calls are made to `http://localhost:8091/api` and can be tested using the Postman collection "Job Portal API."

Workflow Steps

1. **Register as a Job Seeker**
   - **Goal**: Create a job seeker account to apply for jobs.
   - **API**: `POST /api/users/register`
   - **Postman Request**: `Users > Register User`
   - **Details**:
     - Body: `{"email": "seeker@example.com", "password": "password", "role": "JOB_SEEKER", "firstName": "Jane", "lastName": "Smith"}`
     - Note the `userId` (e.g., `1`) from the response.
   - **Expected Outcome**: Job seeker account created with `role: JOB_SEEKER`.
2. **Search for Jobs**
   - **Goal**: Find jobs matching criteria (e.g., keyword, location).
   - **API**: `GET /api/jobs/search`
   - **Postman Request**: `Jobs > Search Jobs`
   - **Details**:
     - Query: `?keyword=Software&categoryId=1&locationId=1&page=0&size=10`
     - Response includes jobs (e.g., `jobId: 1`, title: "Software Engineer").
   - **Expected Outcome**: Paginated list of matching jobs.
3. **View Job Details**
   - **Goal**: Review details of a specific job.
   - **API**: `GET /api/jobs/{id}`
   - **Postman Request**: `Jobs > Get Job`
   - **Details**:
     - URL: `{{baseUrl}}/jobs/1?userId=1`
     - Logs a view in **Employer Dashboard Analytics**.
   - **Expected Outcome**: Job details retrieved, view recorded.
4. **Get Recommended Jobs**
   - **Goal**: Discover jobs matching the job seeker's skills.
   - **API**: `GET /api/jobs/recommended`
   - **Postman Request**: `Jobs > Get Recommended Jobs`
   - **Details**:
     - Query: `?userId=1&page=0&size=10`
     - Assumes skill-based matching (implementation assumed).
   - **Expected Outcome**: Paginated list of recommended jobs.
5. **Apply for a Job**
   - **Goal**: Submit an application for a job.
   - **API**: `POST /api/applications`
   - **Postman Request**: `Job Applications > Create Application`
   - **Details**:
     - Body: `{"jobId": 1, "userId": 1, "coverLetter": "I am excited to apply..."}`
     - Note the `applicationId` (e.g., `1`).
   - **Expected Outcome**: Application submitted with status `PENDING`.
6. **View Own Applications**
   - **Goal**: Track submitted applications.
   - **API**: `GET /api/applications/user/{userId}`
   - **Postman Request**: `Job Applications > Get Applications by User`
   - **Details**:
     - URL: `{{baseUrl}}/applications/user/1`
     - Response includes applications (e.g., `applicationId: 1`, status: `PENDING`).
   - **Expected Outcome**: List of user's applications, including comments if any.
7. **View Notifications**
   - **Goal**: Check notifications for application status updates, including comments.
   - **API**: `GET /api/notifications`
   - **Postman Request**: `Notifications > Get User Notifications`
   - **Details**:
     - Query: `?userId=1`
     - Response includes notifications, e.g., "Your application for Software Engineer has been ACCEPTED. Comment: Strong candidate."
   - **Expected Outcome**: Notifications retrieved with status and comment details.
8. **Access API Documentation**
   - **Goal**: Explore API endpoints for job seeker actions.
   - **Action**: Open `http://localhost:8091/swagger-ui.html`
   - **Details**:
     - Browse endpoints like `POST /api/applications`, `GET /api/notifications`.
     - Test endpoints directly in Swagger UI.
   - **Expected Outcome**: Interactive API documentation displayed.

Notes

- **Prerequisites**: Job seeker must have a valid `userId` with `role: JOB_SEEKER`.
- **Notifications**: Triggered by employer actions (e.g., status updates with comments).
- **Analytics**: Job views are logged for employer analytics.
- **Swagger**: Use Swagger UI to understand request/response formats and test endpoints.

ERR Diagram