**Software Requirements Specification (SRS)**

**Project Title: Job Portal Application**

**Technology Stack: React (Frontend), Spring Boot (Backend), MySQL (Database)**

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the requirements for the development of a Job Portal Application. This application aims to connect job seekers with employers efficiently, providing interfaces to browse, search, and apply for job listings. The SRS outlines the functional and non-functional requirements, system architecture, interface definitions, and role-based responsibilities involved in the development and deployment of the system.

1.2 Document Conventions

* REST API endpoints are described in camelCase.
* Components and modules follow PascalCase naming conventions.
* Currency is denoted in USD ($).
* Dates follow the YYYY-MM-DD format.

1.3 Intended Audience and Reading Suggestions

* This document is intended for:
* Frontend Developers (React) to implement UI components as per requirements
* Backend Developers (Spring Boot)—to build RESTful services and database interaction
* Database Engineers - to manage schema, indexes, and stored procedures
* Project Managers & Testers -to understand application flow and use-case testing Stakeholders to monitor and evaluate whether business needs are fulfilled.

1.4 Product Scope

The Job Portal Application enables employers to post job listings and candidates to search and apply for jobs. The platform ensures efficient browsing through categorized listings, keyword-based search, and a detailed view of job descriptions. The backend provides secure REST endpoints, data persistence, and search functionalities. The application is scalable to accommodate role-based access for job seekers, employers, and administrators.

1.5 References

* Spring Boot Official Documentation
* React Documentation
* MySQL Documentation
* Axios for React
* ISO/IEC 25010:2011

2. Overall Description

2.1 Product Perspective

The Job Portal Application is a standalone web-based application developed using a React frontend, a Spring Boot REST API backend, and a MySQL database. It integrates several modules to manage job listings, search capabilities, job details, and user interactions. It follows an N-tier architecture:

• Presentation Layer: React components (JobListing, JobSearch, JobDetail).

• Business Logic Layer: Spring Boot services and controllers handling application logic.

• Persistence Layer: MySQL database for storing job data.

The application is modular and can be integrated with external services like email notifications or resume parsers in the future.

2.2 Product Functions

The major functions of the system include:

* Allowing job seekers to browse and search for jobs.
* Displaying job listings with essential details (title, company, type, location, date).
* Showing detailed job information upon user selection.
* Backend APIs to manage job creation and retrieval.
* Error handling for missing results or API failures.
* Role-based accessibility for future admin/employer modules.

2.3 User Classes and Characteristics

**User Class**

Job Seeker

Admin (Future)

Employer

(Future)

**Description**

Can browse, search, and view job details.

Manages job listings, approves/rejects entries (not covered in MVP version).

Posts and updates jobs (admin-assisted in MVP version).

2.4 Operating Environment

Component

Specification

Frontend

React 18+, JavaScript, HTML5, CSS3

Backend

Spring Boot 3+, Java 17+, Maven

Database

MySQL 8+

Browser

Hosting

Chrome, Edge, Firefox (latest versions)

Localhost/AWS EC2 / Spring Cloud/Vercel

2.5 Design and Implementation Constraints

* Must support CORS configuration to connect frontend with backend.
* Must follow RESTful API standards.
* Data initialization should happen during app startup.
* Use appropriate HTTP status codes in API responses.

·

APIs must be secure and reject malformed payloads.

2.6 User Documentation

* API documentation using Swagger/OpenAPI.
* README file with setup instructions for React and Spring Boot.
* Postman Collection for API testing.
* Screenshots or demo video for frontend interactions.

2.7 Assumptions and Dependencies

* All job data is inserted via backend API or preloaded in DB.
* React and Spring Boot apps are deployed independently.
* Network connectivity is stable between frontend and backend.
* The database is preconfigured and running before backend start.

**3.2 Functional Requirements**

FR1: Job Listing Display

Description:

The system must provide an intuitive interface that displays a comprehensive list of available job opportunities to users, enabling job seekers to browse through current openings efficiently.

Detailed Requirements:

* The job listing page shall fetch all job postings from the backend API endpoint /api/jobs.
* Each job entry shall display the job title, company name, location, job type (e.g., Full-time, Part-time), and posted date.
* Job listings shall be presented in reverse chronological order, showing the most recent postings first.

·

A loading indicator shall be visible while data is being fetched.

• If the backend API call fails, an error message such as "Unable to load job listings.

Please try again later." shall be shown to the user.

• Each job listing shall be clickable, allowing navigation to the job detail page.

FR2: Job Search Functionality

Description:

The system shall allow users to search for jobs using keywords, returning relevant results and providing appropriate feedback if no matches are found.

Detailed Requirements:

• A search input field shall be present on the job listing page.

• When a user submits a search, the system shall call

/api/jobs/search?keyword={searchTerm} to retrieve matching jobs.

• If matching jobs are found, the job listing shall update to display only the relevant results.

• If no matches exist, the system shall display the message "No jobs found matching your search criteria."

• Search results shall also show job title, company, location, type, and posted date. • API errors during search shall trigger an error message, such as "Search failed. Please try again."

FR3: Job Detail View

Description:

The system shall present detailed information for a selected job, allowing users to thoroughly understand the opportunity.

Detailed Requirements:

* Clicking on a job shall navigate the user to the job detail page.
* The job detail page shall retrieve data from/api/jobs/{jobid}.  
    
   Displayed information must include: job title, company name, location, job type,
* posted date, full job description, required skills (listed), salary range, and
* application deadline.
* A "Back to Listings" button shall be available for easy navigation back to the job list. • A loading indicator shall show during data retrieval.
* Errors fetching job details shall display a user-friendly message, e.g., "Unable to load job details. Please try again."

FR4: Job Application Submission

Description:

* The system shall provide a mechanism for users to apply for jobs directly through the portal.
* Detailed Requirements:
* An "Apply Now" button shall be displayed on the job detail page.
* Clicking the button shall open an application form requiring applicant name, contact info, resume upload, and cover letter.
* The form shall enforce validation of required fields before submission.
* Upon submission, the system shall POST application data to the backend API endpoint.
* Success messages (e.g., "Application submitted successfully") or error messages shall be displayed appropriately.

FR5: Backend Job Management API

Description:

The backend shall expose secure and efficient APIs for managing job postings and applications.

Detailed Requirements:

* The backend shall provide endpoints to create, retrieve, update, and delete job listings.
* Input data shall be validated server-side according to business rules before
* persistence.
* APIs shall return appropriate HTTP status codes and descriptive error messages.
* CORS policy shall be configured to restrict API access to the authorized frontend origin.

FR6: Database Schema for Job Storage

Description:

The system database shall efficiently store job-related data to support frontend display and search functionalities.

Detailed Requirements:

The database shall include a jobs table with columns: id (primary key), title, company, location, type, posted\_date, description, skills (JSON or related table), salary range, and application\_deadline.

• Indexing shall be implemented on commonly searched fields such as title and location.

• Constraints shall enforce required fields and data types to maintain integrity.

FR7: Data Initialization on Startup

Description:

The system shall preload sample job listings on initial deployment to facilitate testing and demonstration.

Detailed Requirements:

On application startup, the backend shall check if the job table is empty.

If empty, it shall insert at least five sample job listings with realistic, diverse data.

• Sample jobs shall cover various locations, job types, and industries.

FR8: Responsive User Interface

Description:

The frontend interface shall adapt to various screen sizes and devices, ensuring usability across desktops, tablets, and mobiles.

Detailed Requirements:

Layouts shall dynamically adjust to fit small, medium, and large screens.

Navigation menus, buttons, forms, and job listings shall remain fully functional and readable on all devices.

• No horizontal scrolling shall be required on mobile devices.

FR9: Error Handling and User Feedback

Description:

The system shall provide clear, informative feedback for successful operations, validation errors, and unexpected failures.

Detailed Requirements:

* Form fields shall show inline validation messages when inputs are missing or invalid.
* API failures shall trigger user-visible alert messages or banners.
* Successful actions such as job application submission shall show confirmation messages.
* Feedback messages shall be concise, polite, and instructive.

FR10: Security Measures

Description:

The system shall enforce security best practices to protect data integrity and user privacy.

Detailed Requirements:

• CORS shall restrict API access to known frontend origins only.

• All input data shall be sanitized to prevent injection attacks.

• Sensitive endpoints (e.g., job creation) shall be protected via authentication (if implemented).

• Passwords and sensitive data (if applicable) shall be securely stored.

FR11: Keyword Search Filtering

Description:

The job search shall filter results based on keywords found in job titles, company names, or locations.

Detailed Requirements:

• Searches shall be case-insensitive.

• Partial matches within any searchable field shall be considered a valid match.

• The backend search API shall implement efficient filtering to maintain responsiveness.

FR12: Job Sorting by Date

Description:

Jobs shall be sorted by the date posted, ensuring that the most recent job listings are prioritized.

Detailed Requirements:

• Default sorting order for all job lists and search results shall be descending by posted date.

• Sorting shall be enforced on the backend for consistency.

FR13: Display of Required Skills

Description:

Jobs shall display required skills prominently to help users quickly assess fit.

Detailed Requirements:

• Skills shall be shown as distinct tags or badges on listings and job detail pages.

• Skills data shall be fetched dynamically from the backend.

• The UI shall allow easy scanning of skills in a visually consistent manner.

FR14: Loading Indicators

Description:

The application shall provide visual cues while fetching data to enhance user experience.

Detailed Requirements:

Loading spinners or progress bars shall display during API calls for job lists, search, and job detail retrieval.

• Loading indicators shall disappear immediately upon data load or error.

FR15: Accessibility Compliance

Description:

The frontend shall meet accessibility standards to ensure usability for users with disabilities.

Detailed Requirements:

* UI components shall include appropriate ARIA labels and roles.
* Keyboard navigation shall be fully supported across all interactive elements.
* Color contrasts shall meet WCAG 2.1 AA requirements for readability.

5. Performance Requirements

PR1: Fast Data Retrieval

Description:

The system shall ensure quick loading times for job listings, search results, and job detail views to maintain user engagement.

Detailed Requirements:

Job listing API responses shall be delivered within 2 seconds under normal load conditions.

* Search queries shall return results within 1.5 seconds for typical keyword searches.
* Job detail page data shall load within 1.5 seconds upon selection.
* Backend shall implement indexing and caching strategies to optimize query performance.

PR2: Scalability

Description:

The application shall be scalable to handle increasing numbers of users and job listings without degradation in performance.

Detailed Requirements:

The backend shall be designed to support at least 10,000 concurrent users.

• Database schema and queries shall be optimized for large datasets exceeding 100,000 job entries.

• The frontend shall efficiently handle rendering large lists using techniques such as pagination or lazy loading.

PR3: Efficient Network Usage

Description:

The system shall minimize network bandwidth consumption to improve responsiveness, especially for users on limited data connections.

Detailed Requirements:

* API responses shall include only necessary data fields to reduce payload size.
* The frontend shall implement data caching where appropriate to avoid redundant API calls.
* Images and static assets (if any) shall be optimized for web delivery.

PR4: API Throughput

Description:

The backend API shall support high request throughput to accommodate spikes in user activity.

Detailed Requirements:

• The system shall handle a minimum of 500 API requests per second during peak hours.

• Load balancing and horizontal scaling strategies shall be considered in deployment architecture.

• Monitoring tools shall be implemented to track API performance and alert on degradations.

PR5: Response Time Under Load

Description:

The system shall maintain acceptable response times under heavy user load.

Detailed Requirements:

Under 80% server CPU utilization, 95% of API responses shall complete within 3 seconds.

• Slow queries shall be logged and analyzed for optimization.

PR6: Frontend Rendering Speed

Description:

The frontend shall render pages swiftly to provide a smooth user experience.

Detailed Requirements:

* Initial page load times shall be under 3 seconds on a standard broadband connection.
* Client-side rendering shall avoid blocking the UI thread to prevent lag.
* Components shall leverage React optimization techniques like memoization and code splitting.

PR7: Database Connection Pooling

Description:

The backend shall manage database connections efficiently to improve performance. Detailed Requirements:

* Connection pooling shall be configured to limit overhead and maximize throughput.
* Pool size and timeout settings shall be tuned based on expected load.

PR8: Background Processing for Heavy Tasks

Description:

Long-running tasks shall be processed asynchronously to avoid blocking user interactions.

Detailed Requirements:

• Operations like sending notifications (if implemented) or batch data imports shall run in background jobs.

• Users shall receive progress or completion notifications where applicable.

6. Design Constraints

DC1: Technology Stack

Description:

The system must be developed using the specified technologies to ensure consistency and maintainability.

Detailed Requirements:

* Frontend shall be implemented using React.js.
* Backend shall be developed using Spring Boot framework.
* The database shall use MySQL for data persistence.

DC2: API Design and Communication

Description:

The backend API shall follow RESTful principles to ensure clear and standardized communication with the frontend.

Detailed Requirements:

* Endpoints shall use appropriate HTTP methods (GET, POST, etc.).
* JSON shall be the standard format for request and response bodies.
* API versioning shall be maintained to allow backward compatibility.

DC3: Security Implementation

Description:

The system must implement security best practices for data protection and user privacy.

Detailed Requirements:

* All API calls shall be secured using JWT (JSON Web Token) based authentication.
* Sensitive data (e.g., user information) shall be encrypted in transit using HTTPS.
* Input validation shall be implemented on both frontend and backend to prevent injection attacks.
* CORS policy must be configured to restrict unauthorized origins.

DC4: Responsive Design

Description:

The frontend must provide a responsive interface usable on various devices and screen sizes.

Detailed Requirements:  
• UI components shall adapt layout for desktop, tablet, and mobile views. • Accessibility standards (WCAG 2.1) shall be considered to improve usability.

DC5: Database Schema Constraints

Description:

The database schema must enforce data integrity and relationships.

Detailed Requirements:

• Primary keys shall be unique and auto-incremented.

• Foreign key constraints shall enforce referential integrity between tables.

• Appropriate indexes shall be added on frequently queried fields (e.g., job title, company).

DC6: Deployment Constraints

Description:

The system shall be deployable on cloud environments with container support.

Detailed Requirements:

* The backend and frontend shall be containerized using Docker for portability.
* Deployment scripts and configuration files shall support common cloud providers like AWS, Azure, or GCP.
* Environment-specific configuration (dev, test, production) must be supported.

DC7: Third-party Libraries and Tools

Description:

Only approved and actively maintained third-party libraries shall be used.

Detailed Requirements:  
  
React libraries and Spring Boot dependencies shall be selected based on community support and security track record.

• Version updates must be managed carefully to avoid breaking changes.

DC8: Scalability and Maintainability

Description:

The system architecture shall allow for future growth and easy maintenance.

Detailed Requirements:

• Backend services shall be modular with clear separation of concerns (e.g., controllers, services, repositories).

• Frontend codebase shall follow component-based architecture with reusable components.

• Proper documentation shall be maintained for code and APIs.