
Design Document

ITSC 4155 – Software Development Project
Project Title: JobSearchHelper
Team:

Jaden Peterkin

Abhinav Koukuntla

Deep Trivedi

Andre Harper

Aida Samsombath

Julian Taylor

Date: April 25, 2025
Tool Acknowledgment: Design document generated with assistance from OpenAI's ChatGPT (gpt-4)

I. Introduction

The JobSearchHelper application is a comprehensive full-stack system built using Java 17 and Spring Boot. It assists users in browsing, filtering, and applying to job listings, managing their applications, receiving notifications, and even preparing for interviews through AI-based chat simulations. The front-end is powered by Thymeleaf templates, and the system leverages MongoDB for database persistence.

The project features modular development using the MVC (Model-View-Controller) pattern, REST APIs, and microservice-oriented designs like external API integration (OpenAI) for enhanced interview preparation.

II. Requirements

Functional Requirements

- Users can view, search, and apply to job postings.
- Users can sign up, log in, and manage their profiles.
- Applications can be tracked (applied, already applied views).
- Users receive notifications related to job activities.
- AI-based chat system to simulate interview preparation.
- Administrators can create, edit, and delete job postings.

Non-Functional Requirements

- Built with Java 17, Spring Boot, Maven.
- Uses MongoDB for data persistence.
- Follows MVC and layered architecture (Controllers → Services → Repositories).
- RESTful APIs with JSON data exchange.
- Thymeleaf for dynamic HTML rendering.
- Modular and scalable codebase.
- Embedded Tomcat server for local deployment.

III. System Architecture

Main Components

- **Model:** Entities like `User`, `Posting`, `Application`, `Notification`, `ChatMessage`.
- **View:** Thymeleaf templates for job postings, login/signup pages, profiles, and interview chats.
- **Controller:** REST endpoints mapped to business logic (e.g., `PostController`, `LoginController`, `InterviewController`).
- **Service Layer:** Handles core business logic (e.g., `UserService`, `PostingService`, `OpenAIService`).
- **Repository Layer:** Interfaces for MongoDB operations (`UserRepository`, `PostingRepository`, etc.).

External Integrations

- OpenAI API for chat-based interview simulations.
- Apache Tika for resume parsing (future extension).

IV. Class Design

Major Domain Classes

- **User.java:** User profiles, login credentials, and saved skills.
- **Posting.java:** Represents job postings with title, company, location, description.
- **Application.java:** Links users to applied jobs.
- **Notification.java:** Stores notifications for user activities.
- **ChatMessage.java, ChatRequest.java:** Support AI-based interview chats.

Key Services

- **UserService:** Business logic for user authentication and profile management.
- **PostingService:** Handles CRUD operations for job postings.
- **ApplicationService:** Manages job application submissions and status.
- **NotificationService:** Sends and manages user notifications.
- **OpenAIService:** Manages API calls to OpenAI for interview questions.

Controllers

- **HomeController:** Directs homepage activities.
- **PostControllers (Create, Edit, Delete, View):** CRUD operations on job postings.
- **LoginController/SignUpController/LogoutController:** User authentication routes.
- **ApplyController/ApplicationsController:** Job application tracking.
- **InterviewController:** Chat system for mock interviews.
- **NotificationController:** User notification management.

Database

- MongoDB collections for Users, Postings, Applications, Notifications.

V. Data Flow & API Logic

Example: Applying to a Job

1. User navigates to a job posting.
2. `ApplyController` handles POST request to submit an application.
3. `ApplicationService` saves new application via `ApplicationRepository`.
4. User receives a notification via `NotificationService`.

Example: Chat Interview Simulation

1. User starts a chat session on the interview page.
2. `InterviewController` sends chat request to `OpenAIService`.
3. `OpenAIService` formats and forwards request to OpenAI API.
4. Response is parsed and displayed to user dynamically.

Example: Login Process

1. User submits login credentials.
2. `LoginController` invokes `UserService` to authenticate.
3. Successful login initiates a session via `SessionHandler`.

VI. Key Logic & Algorithms

- **Session Management:** Basic session handling for login/logout flow.
- **REST API Interaction:** External OpenAI integration for real-time communication.
- **Dynamic Rendering:** Thymeleaf templates loop and conditionally render job and application information.
- **MongoDB Integration:** Repositories use Spring Data annotations to interact with MongoDB collections.

VII. Known Issues & Future Work

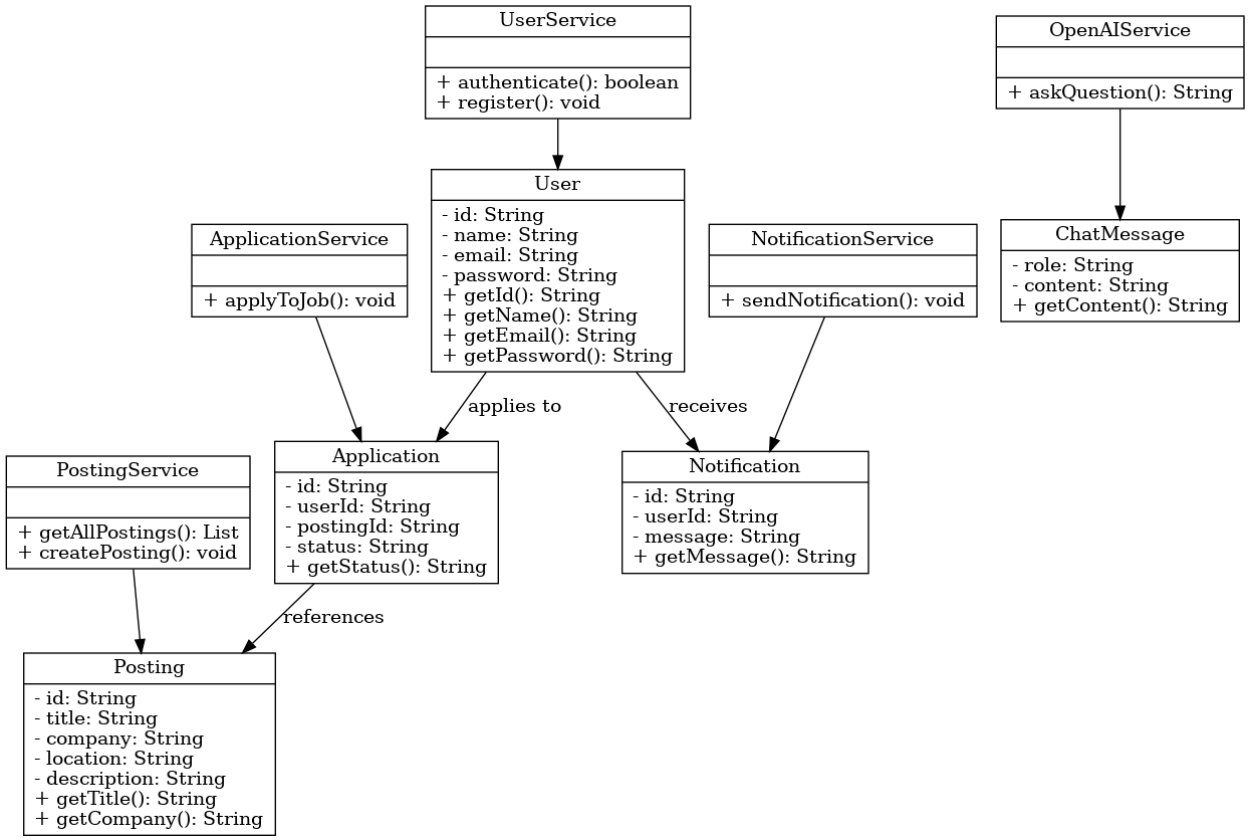
Known Issues

- Resume parsing functionality (TikaHandler) is not fully implemented.
- Limited error handling in external API communications.
- No multi-user roles (admin/user) formally enforced yet.

Future Work

- Implement full resume parsing and skill extraction.
- Add filtering, keyword search, and advanced sorting of job listings.
- Implement user roles (admin dashboard for job posting management).
- Integrate third-party job APIs (e.g., LinkedIn Jobs, Indeed).
- Improve UI/UX with responsive design improvements.
- Harden security (password hashing, OAuth2 integration).

VIII. UML Design



VIII. AI Tool Citation

This design document was generated using OpenAI's ChatGPT (gpt-4), based on detailed analysis of the project source code, directory structure, and configuration files uploaded on April 25, 2025.

Document Version: Updated April 25, 2025 based on current project state.