ResumeLink API Documentation

Overview
ResumeLink is a Spring Boot-based backend system designed to fetch and provide JSON information about user profiles from various web applications, especially social media platforms. This information is retrieved from a database and passed through different layers to facilitate rendering in frontend applications.

Modules and Dependencies
1. Validation API (2.0.1.Final)
Description: Provides a set of APIs for Bean Validation (JSR 380) to ensure data integrity and validation within the application.
2. Spring Data Commons (2.0.0)
Description: A core module of the Spring Data project, offering common functionality and abstractions for data access operations.
3. Joda-Time (2.10.10)
Description: A library for date and time handling, enhancing and providing a more feature-rich alternative to the standard Java Date and Calendar classes.
4. JWT (JJWT 0.9.1)
Description: JSON Web Token (JWT) implementation for secure data transmission, commonly used for authentication and information exchange.
5. ModelMapper (2.3.8)
Description: A powerful, convention-based object mapping library that simplifies the conversion of complex data types between different layers of the application.
6. JAXB API (2.3.1)
Description: Java Architecture for XML Binding (JAXB) API, used for mapping Java objects to and from XML representations.
7. Commons Collections4 (3.2.1)
Description: A library that extends and enhances the Java Collections Framework, providing additional functionality and utilities for working with collections.
Functionality
The ResumeLink backend performs the following key functions:
Profile Information Retrieval:
Fetches user profile information from the database based on requests.
Data Validation:
Utilizes the Validation API for ensuring the integrity and validity of the data retrieved.

JSON Payload Generation:
Constructs JSON payloads containing user profile information for further processing.
Security Token Handling:
Implements JWT (JJWT) for secure token-based communication and authentication.
Object Mapping:
Utilizes ModelMapper for efficient and customizable object mapping between different layers of the application.
Configuration
The application can be configured using properties files, environment variables, or configuration classes, allowing customization of database connections, security settings, and other parameters.
Pipeline Components
1. Source Code Repository
The ResumeLink source code is stored in a version control system (e.g., Git). Developers push changes to the repository, triggering the deployment pipeline.
2. Continuous Integration (CI)
The CI system automates the build, testing, and validation processes to maintain code quality.
CI Workflow:
Code Build: Pulls the latest code from the repository.
Dependency Installation: Installs project dependencies (e.g., Maven dependencies) using a build tool.
Code Compilation: Compiles the Spring Boot application.
Unit Testing: Executes unit tests to ensure code correctness.
Static Code Analysis: Utilizes tools like SonarQube for static code analysis.
Artifact Generation: Creates a deployable artifact (JAR file).
3. Containerization
The deployable artifact (JAR file) is containerized using Docker for consistent deployment across different environments.

Containerization Steps:
Docker Image Build: Constructs a Docker image containing the Spring Boot application and its dependencies.
Image Tagging: Tags the Docker image with a version or unique identifier.
Push to Registry: Pushes the Docker image to a container registry (e.g., Docker Hub, Amazon ECR).
4. Continuous Deployment (CD)
The CD pipeline automates the deployment and configuration of the ResumeLink backend on target environments.
CD Workflow:
Artifact Retrieval: Pulls the Docker image from the container registry.
Environment Provisioning: Sets up the target environment, including databases and other required services.
Configuration Management: Applies environment-specific configurations (e.g., database connections, security settings).
Container Deployment: Deploys the Docker image on the target environment.
Smoke Testing: Performs basic tests to ensure the deployed application is functioning.
Security Token Configuration: Configures and manages JWT secret keys for secure communication.
5. Monitoring and Logging
Integrates monitoring and logging tools to track the health and performance of the ResumeLink backend.
Monitoring and Logging Components:
Application Insights: Gathers insights into application behavior and performance.
Log Aggregation: Utilizes tools like ELK Stack (Elasticsearch, Logstash, Kibana) for centralized logging.
Configuration Management
Utilizes configuration management tools (e.g., Ansible, Puppet) to manage environment-specific configurations, ensuring consistency across different deployment environments.
Environment Variables
Configures environment variables for runtime parameters, including database connections, security keys, and other settings.

Dependencies:

spring-boot-starter-data-mongodb == 3.3.3

Spring Boot Starter Data MongoDB is a Spring Boot module that provides an easy way to integrate MongoDB, a popular NoSQL document database, into Spring Boot application. It simplifies the process of connecting to a MongoDB database, performing CRUD operations, and using advanced features like data mapping, query derivation, and more.

To use spring-boot-starter-data-mongodb version 3.3.3 in our resumelink application, we perform the following steps: We first create SpringBoot application, then we define domain model class. After that we create a repository framework. Then we configure MongoDB in application.properties. We then create a service class and controller class. **In our application** the spring-boot-starter-data-mongodb dependency is added to the project's build configuration (e.g., pom.xml for Maven or build.gradle for Gradle). This dependency provides the necessary libraries and configurations to integrate MongoDB into the Spring Boot application.

Spring framework == 6.0.15

The Spring Framework is a comprehensive Java framework facilitating the development of enterprise applications by promoting modularity, flexibility, and ease of integration. Its core container provides features such as dependency injection and inversion of control, simplifying component management. Spring MVC offers a robust model-view-controller architecture for building web applications, while Spring Data streamlines data access and manipulation

across various data stores. Spring Boot further accelerates development with auto-configuration and opinionated defaults, enabling rapid application setup and deployment.

In our resumelink application the spring-boot-starter-web version 6.0.15 dependency is used in Spring Boot applications inherited through spring framework to quickly set up web applications. It includes dependencies required for building web applications using Spring MVC, including embedded servlet containers such as Tomcat, Spring Web, and other utilities. A basic example of how to use spring-boot-starter-web in a Spring Boot application is to add dependency to our project build configuration (e.g. 'pom.xml' for Maven or build.gradle for Gradle)

Jsonwebtoken == 8.5.0

io.jsonwebtoken is a Java library used for handling JSON Web Tokens (JWT). JWT is a compact, URL-safe means of representing claims to be transferred between two parties. These claims are typically used to transmit information about an authenticated user or authorization data.

To use JSON Web Tokens (JWTs) in our resumelink application, we utilize the io.jsonwebtoken version 8.5.0 library. First, we add the necessary dependencies to our project's build configuration file (pom.xml for Maven or build.gradle for Gradle). Then, we create and validate JWTs using the library's APIs. This typically involves setting claims such as user ID, username, and expiration time when generating JWTs, and validating them upon receiving requests. Additionally, we integrate JWT authentication with Spring Security by implementing a custom filter to validate JWT tokens sent in requests.

modelmapper == 2.4.2

The ModelMapper library is often used in Java applications to simplify the process of mapping data between different object models, such as mapping data from a database entity to a Data Transfer Object (DTO) or vice versa.

For our resumelink application we use sub dependencies of modelmapper version 2.4.2 such as com.h2database which is a lightweight, in-memory SQL database for development and testing purposes. It provides features such as JDBC API compatibility, support for SQL syntax, and an embedded web-based console for database management. Furthermore, in our context of our resumelink application, ModelMapper and com.h2database are often used together to simplify data management tasks. For example, ModelMapper is used to map entities retrieved from the H2 Database to DTOs used in RESTful endpoints. This combination allows for efficient data manipulation and transformation within the application, enabling us to focus on implementing business logic rather than dealing with low-level data handling intricacies.