EEET2582/ISYS3461 Software Engineering: Architecture and Design

DEVision – Job Applicant Subsystem

Project Milestone 1

Squad Mulan - Team Job Applicant:

Ho Quoc Thai	s3927025
Phan Trong Nguyen	[insert Sid here]
Nguyen Ngoc Luong	[insert Sid here]
Nguyen Hoang	s3929351

Course Coordinator

Tri Huynh Minh	tri.huynh5@rmit.edu.vn

Submission Date November 28th, 2025

Table of Contents

Introduction

This document serves as the comprehensive technical specification and architectural blueprint for the Job Applicant sub system of the DEVision platform. Its primary purpose is to document the critical design decisions, conceptual data models and system architecture required to build a robust career advancement and building platform for Computer Science professionals. This report outlines the structural framework necessary to support the system functional requirements, providing justifications for technology selection and design patterns. It acts as a guide for stakeholders and the development team to ensure the application achieves the goals of scalability, maintainability, and security while effectively bridging the gap between technical talent and potential employers.

The Job Applicant team is responsible for implementing the following core functionalities, designed to empower users to establish a solid career foundation:

- **Registration and Authentication**: Secure user onboarding via email and password registration with strict validation, as well as SSO (Single Sign-On) integration. Security is enforced through JSON Web Encryption (JWE) tokens and brute-force protection mechanisms.
- **Profile Management**: A dynamic interface allowing applicants to manage core contact information (name, email, address…) along with their educational history, work experience and their technical skills as tags. The page also includes support for uploading multimedia portfolios (images/videos) and user avatars.
- **Job Search and Discovery**: Advanced search capabilities utilizing Full Text Search (FTS) across job titles, descriptions, and required skills. Applicants can filter results by employment type, salary range, and location, with efficient data retrieval optimized via database sharding.
- **Application Submission**: A streamlined process for applicants to submit applications to specific job posts, including the ability to upload CV files and cover letters.
- **Premium Subscription Services**: A monetization feature that allows applicants to subscribe for premium features. This will benefit applicants with real-time notifications when new jobs matching their specific criteria are posted.
- **Administration**: A dedicated admin panel for managing applicant and company accounts on the platform, as well as handle job posts and skill tags distribution, ensuring platform integrity by deactivating problematic accounts or content.

The Job Applicant team has decided to implement the above features with the architectures as follows:

- Frontend Architecture: Headless UI architecture, implemented using a React-based framework (Vite)
- Backend Architecture: Distributed Microservices Architecture implemented using Spring Boot with per-service databases, Redis for caching and Apache Kafka for event streaming.

Squad Data Model

Data Model

Appendix A: Squad Data Model (file name: Squad_DataModel_Diagram.png)

To support Identity and Access Management, the model uses UserCredential and AuthProvider entities to store authentication credentials. This structure supports both standard email and password authentication logins and SSO integrations by linking external provider data to the user. The UserProfile entity includes the userCountryCode attribute, which functions as the sharding key required in the system specification to partition user data based on location.

For profile management, the model uses separate entities for education and work experience of each user such as UserEducation and UserWorkExperience, and UserSkill to store the respective lists of applicant's skills, which then exposed to the Job Manager sub system to provide the applicant details.

The job application and search features are primarily supported by the JobApplication entity, which tracks the lifecycle of an application by linking a userId to a jobPostId and recording statuses such as "Submitted" or "Rejected." This entity directly references the File entity to associate specific Curriculum Vitae (CV) and Cover Letter documents with each submission. Regarding search functionality, the system does not store job data locally; instead, it consumes the Job Post Data API provided by the Job Manager subsystem. The Job Applicant system passes dynamic user criteria (e.g., title, salary range) to this API to retrieve and display available job listings. Conversely, our system provides the Job Application API, enabling the Job Manager subsystem to retrieve the list of applicants and their attached documents for specific job posts.

The premium subscription and notification features utilize the UserSubscription entity to record the start and end date of a user's premium status. TO support automated job matching, the model utilized JobSearchProfile, JobSearchSkill, and JobSearchEmployment to persist the applicant's specific preferences, such as salary range, job titles, skills, and employment types. These stored the attributes function as the filter parameters for cross system data retrieval. The system reads these criteria to construct targeted requests to the Job Post Data API provided by the Job Manager sub system, allowing the application to fetch a list of job posts that specifically fit the applicant's defined profile. Additionally, these entities are used to validate incoming Kafka event for real time notifications. Financial transactions processed via the Payment API are recorded in the PaymentTransaction entity.

Sharding (Nguyen)

If you plan to do sharding, describe your sharding strategy in detail. Specify the shard key, what data will you shard, and how can your backend know which shard to collect the data.

Squad Container Diagram

System Overview

1. Appendix B: Job Applicant Frontend Container Diagram (file name: JA_FE_Container.png)
2. Appendix C: Job Manager Frontend Container Diagram (file name: JM_FE_Container.png)
3. Appendix D: Job Applicant Backend Container Diagram (file name: JA_BE_Container.png)
4. Appendix E: Job Manager Backend Container Diagram (file name: JM_BE_Container.png)

The Job Manager frontend is built using a React-based framework, adopting headless UI architecture. Their design explicitly separates presentation from logic, evidenced by their structure of headless UI component group paired with hooks and services components. On the backend server side, job manager team also utilize the distributed microservices architecture powered by Spring boot, with Redis for caching and Apache Kafka for streaming of events. This alignment in backend architecture facilitates consistent deployment and scaling strategies across the entire DEVision ecosystem.

The integration between Job Applicant and Manager sub system is critical to the functionality of DEVision platform. To manage the data exchange described in our data model while adhering to the principles of a distributed microservices architecture, we have established a dual channel communication strategy.

The first channel utilizes REST APIs to handle direct data retrieval between the two systems. These functions are based on the backend-to-backend communication, where Job Applicant backend will communicate with Job Manager backend.

The second channel utilizes Kafka messages to facilitate one way event propagation. Where one sub system will act as a producer and publish events to a designated Kafka topic. A consumer which subscribes to the same topic will receive the event data. This data then be used to trigger functions or logics on the consumer side.

Job Applicant Frontend Container Diagram

[Insert or link to diagram and provide description as per guidelines above.]

For your frontend diagrams, describe which technology Stack is used in the Frontend, its architecture style, the purpose of core containers (Security, State Management, Browser Router, etc.), and how your frontend communicates with the backend. [Detail your stack, e.g., Technology: React with Redux; Style: Component-based; Core containers: Security for auth tokens, State Management for global data; Communication: Via HTTP requests to backend endpoints.]

Job Applicant Backend Container Diagram

[Insert or link to diagram(s) and provide description(s) as per guidelines above. If combined: Describe unified backend. If separate: Use subheadings like 3a/ Job Applicant Backend and 3b/ Job Manager Backend.]

Backend Component Diagram (Hoang)

List down component diagrams of your microservice/module/layer containers here. Summarize the purpose of each container and describe any incoming/outgoing communication to/from this container. [For each component: Insert or link to diagram, then summarize, e.g., Authentication Microservice: Handles user login; Incoming: API calls from frontend; Outgoing: Tokens to other services.]

Frontend Component Diagram (Luong)

List down component diagrams of your Headless UI (and Modularized Components)/Modularized Component/Layer containers here. Summarize the purpose of each container and describe how the components interact with each other to provide the features. [For each component: Insert or link to diagram, then summarize, e.g., Dashboard Component: Displays user data; Interactions: Fetches data from State Management container via hooks.]

Architecture Rationale (Nguyen)

Whenever applicable, justify the following properties of your **Data Model, Frontend, and Backend Architecture**. For every property, explain **both the advantages and disadvantages with examples or arguments related to the DEVision context**.

- **Maintainability:** Enables the system to be easily understood, updated, and tested independently by current and future Development Teams.

- **Extensibility:** Enables new features to be added with minimal impact on existing components. This approach allows for future enhancements and adaptations to changing user needs or market demands.

- **Resilience:** The system's ability to continue functioning correctly in the face of unexpected conditions, such as component failures or high loads.

- **Scalability:** The capacity of a system to handle increased load or user demand without sacrificing performance.

- **Security:** The measures taken to protect a system and its users from unauthorized access, data breaches, and other threats.

- **Performance:** Performance relates to how quickly and efficiently a system responds to user requests and processes data.

Data Model Justification

[For each property above, provide justifications specific to your data model, discussing pros and cons]

Frontend Architecture Justification (Thai/Luong)

[For each property above, provide justifications specific to your frontend]

Backend Architecture Justification (Nguyen/Hoang)

[For each property above, provide justifications specific to your backend]

Conclusion

- **Summarize the features provided by your system.**

- **Summarize the key advantages in your system.**

- **Summarize the key disadvantages and flaws in your system design** based on the rationales above.

References

Appendix

Appendix A: Squad Data Model