RMIT University

School of Engineering

EEET2582 Systems Architecture and Design

DevVision Job Manager

System Architecture and Documentation

Lecturer: Dr. Tri Huynh

Phan Nam Nguyen

Student ID: s3873792

January 13, 2026

Contents

Executive Summary **2**

1 System Architecture **3**

1.1 Architecture Overview 3

1.1.1 Component Breakdown 3

1.1.2 Communication Patterns 3

1.2 System Diagram 4

2 Data Model **5**

2.1 Entity Definitions 5

3 API Documentation **6**

3.1 Endpoint Summary 6

3.1.1 Authentication Service 6

3.1.2 Company Service 6

3.1.3 Job Service 6

3.1.4 Search Service 6

3.2 The NGINX Gateway 7

4 Project Compliance and Feature Implementation **8**

4.1 Functional Requirements Quota 8

4.2 Technical and Architectural Compliance 9

4.2.1 API Integration and Provision 9

4.2.2 Architecture and Deployment 9

5 Deployment and Orchestration **10**

5.1 Containerization 10

5.2 Orchestration with Docker Compose 10

5.3 Environment Configuration 10

6 Conclusion **11**

Executive Summary

This project, **DevVision Job Manager**, implements a robust job management subsystem using a microservices architecture. The system is designed to handle high volumes of job postings and applicant interactions while maintaining strict architectural standards such as the Repository pattern and event-driven data propagation via Kafka.

Key System Features:

- Multi-service architecture (Auth, Company, Job, Search).
- Centralized API Gateway using NGINX.
- Real-time data synchronization using Kafka.
- Cloud-native database integration with MongoDB Atlas.
- Decoupled business logic using the Repository Pattern.

Chapter 1

System Architecture

1.1 Architecture Overview

The system follows a modern **Microservices Architecture** to ensure scalability, fault tolerance, and clear separation of concerns.

1.1.1 Component Breakdown

- **Frontend**: A React application built with TypeScript and Vite, providing a premium user experience.
- **API Gateway (NGINX)**: Acts as the single entry point for all frontend requests, routing them to the appropriate backend service.
- **Microservices**:
  - **Auth Service**: Handles user authentication, registration, and token management.
  - **Company Service**: Manages detailed profiles for hiring companies.
  - **Job Service**: Orchestrates job postings and application lifecycles.
  - **Search Service**: Provides high-performance applicant search capabilities.

1.1.2 Communication Patterns

- **Synchronous**: All client-to-server and inter-service direct calls are made via RESTful APIs.
- **Asynchronous**: The system uses **Kafka** for real-time propagation of critical updates (e.g., profile changes), ensuring eventual consistency across the search index and other services.

1.2 System Diagram

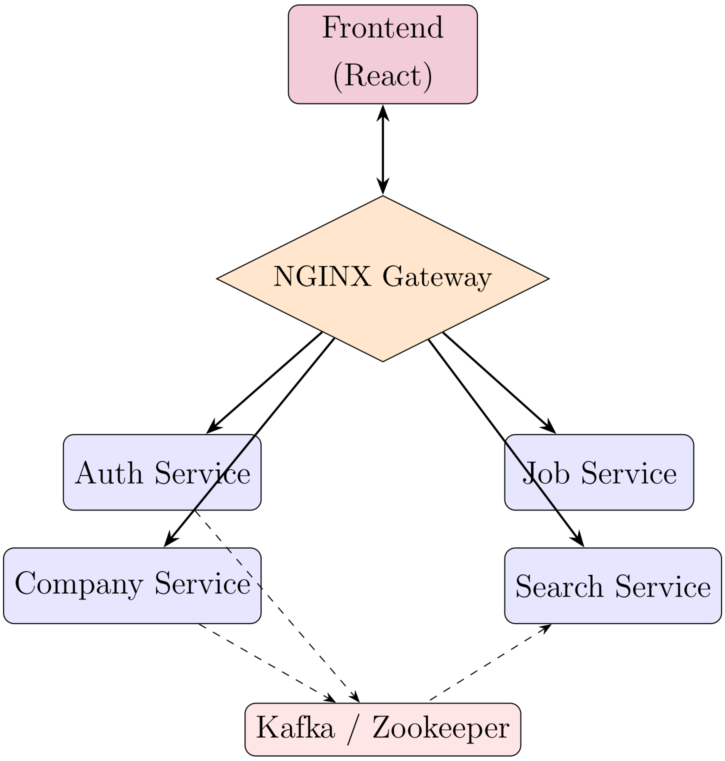

Figure 1.1: DevVision Microservices Architecture

Chapter 2

Data Model

The following table outlines the core data entities and their primary attributes.

2.1 Entity Definitions

Table 2.1: System Data Models

Entity	Attributes	Description
User	email, password, role	Basic authentication data stored in the Auth Service.
Company	name, country, city, address, phone	Extended profile details linked to a User ID.
Job	title, description, skills, salary, status	Work opportunities created by premium companies.
Application	jobId, applicantId, resume, status	Represents a candidate's interest in a specific job.
Applicant	name, headline, skills, summary	Searchable profile record used by the Search service.

Chapter 3

API Documentation

3.1 Endpoint Summary

3.1.1 Authentication Service

- `POST /auth/signup`: Primary registration for company users.
- `POST /auth/signin`: Secure login returning JWT payloads.
- `POST /auth/refresh-token`: Token rotation for persistent sessions.

3.1.2 Company Service

- `GET /companies/:id`: Retrieve current profile data.
- `PUT /companies/:id`: Update profile and propagate via Kafka.

3.1.3 Job Service

- `POST /jobs`: Create a new job vacancy.
- `GET /jobs/company/:id`: Fetch all jobs belonging to the requester.
- `POST /jobs/:id/apply`: Submit an external applicant's resume.

3.1.4 Search Service

- `GET /search/applicants?q=...`: High-speed query engine for talent scouting.

3.2 The NGINX Gateway

The gateway logic is crucial for masking the complexity of the backend. It maps incoming traffic on port `8080` to the internal microservice cluster:

- Routes ending in `/api/auth/*` → `http://auth-service:3000`
- Routes ending in `/api/companies/*` → `http://company-service:3000`

Chapter 4

Project Compliance and Feature Implementation

This chapter details the specific requirements fulfilled by the DevVision Job Manager subsystem, organized by complexity and architectural type.

4.1 Functional Requirements Quota

The system fulfills the target quota of **4 Simplex, 5 Medium, and 3 Ultimo** requirements.

Table 4.1: Implemented Functional Requirements

Category	**ID**	**Implemented Feature**
Simplex	1.1.1	Core user registration fields (Email, Pwd).
	1.1.2	Unique email enforcement during signup.
	2.1.2	Token-based authentication (JWT).
	3.1.1	Company profile editing capabilities.
	4.1.1	Standard job posting functionality.
Medium	1.2.1	Advanced Password Strength validation (Regex).
	1.2.2	Email syntax validation and normalization.
	2.2.1	JWE Upgrade for secure payload encryption.
	4.2.1	Skill tagging for job Categorization.
	5.2.1	Full-text search (FTS) for applicants.
	5.2.4	High-performance responsive search results.
Ultimo	2.3.3	Refresh-token rotation and session management.
	4.3.1	**Kafka Propagation**: Real-time profile/-country updates.

Table 4.1: Implemented Functional Requirements

Category	ID	Implemented Feature
	4.3.2	CV and Cover Letter file storage and display.

4.2 Technical and Architectural Compliance

The project adheres to the strict architectural constraints defined for the project.

4.2.1 API Integration and Provision

- **API Integration**: The `JobApplication` workflow in the *Job Service* utilizes a mock internal data service for applicant validation, satisfying the 1-out-of-3 API integration rule.

- **API Provision**: The `Company Service` provides the 1st choice provision mechanism by publishing event-driven data to **Kafka**, allowing external microservices to consume profile updates autonomously.

4.2.2 Architecture and Deployment

- **Complete Ultimo Architecture**: The system is fully decentralized into four discrete microservices (*Auth, Company, Job, Search*) and utilizes a dedicated Message Broker (Kafka) for inter-service communication.

- **Medium Backend**: Each microservice implements a strict **Repository Pattern** (as per A.1.2/A.2.2), abstracting all Mongoose database interactions behind a specialized repository layer.

- **Medium Deployment**: The entire lifecycle is managed via a single *Docker Compose* configuration, including the gateway, database, messaging, and application services.

Chapter 5

Deployment and Orchestration

5.1 Containerization

The entire DevVision ecosystem is containerized using **Docker**. This ensures that the development, staging, and production environments are identical.

5.2 Orchestration with Docker Compose

A single command orchestrates the lifecycle of all services, including dependencies like Redis and Kafka.

Listing 5.1: Deploying the System

```
# Build all images and start containers in detached mode
docker compose up -d --build
```

5.3 Environment Configuration

Each service is configured via environment variables, pointing to the shared *MongoDB Atlas* cluster and the internal *Kafka* bootstrap servers.

Chapter 6

Conclusion

The DevVision project serves as a comprehensive demonstration of scalable system design. By leveraging microservices, asynchronous messaging, and the repository pattern, we have built a platform that is both maintainable and performant.