SOFTWARE ENGINEERING PROGRAM

SWENG 894 – CAPSTONE EXPERIENCE

Team 2

Software Architecture and Design

11 August 2024

Jack Taylor
Javier A. Rodriguez
Julio Aira IV

Table of Contents

1. Introduction

1.1 Purpose and Scope of the Architecture

This document outlines the software architecture and design of the resume analysis tool, providing a comprehensive view of its structure, components, and interactions. It serves as a guide for developers, system architects, and stakeholders to understand how the tool is built, how its components work together, and how it fulfills its intended purpose of automating resume screening and comparison.

The architecture of this tool encompasses both the front-end and back-end components, including the user interface, client-side processing using JavaScript, optional server-side processing, and data storage. It covers the integration of NLP techniques for analyzing resumes and job descriptions, and the deployment strategies to ensure scalability, security, and performance.

The primary business goal motivating this tool's development was to streamline and enhance the recruitment process's efficiency. By automating the initial screening of resumes against job descriptions using NLP, the tool aims to reduce the time and effort required by recruiters, improve candidate matching, and ensure a more objective evaluation process, ultimately leading to better hiring decisions.

Stakeholders can use this documentation as a roadmap for understanding the system's architecture, making informed decisions about future enhancements, and ensuring that the system aligns with business goals. Developers can reference it to guide implementation, while project managers and architects can use it to assess scalability, maintainability, and integration with other systems. It also serves as a communication tool between technical and non-technical stakeholders.

The architecture encompasses the entire system, including the front-end interface built with HTML, CSS, and JavaScript, the optional backend powered by Node.js, and the integration of NLP components using the Compromise.js library. It also includes database interactions for storing and retrieving user data, resumes, and job descriptions. Explanation of the business goals that motivated the development of the system.

2. System Overview

2.1 System Context Diagram

The system is visually represented as a multi-layered architecture consisting of the user interface, frontend processing, optional backend services, and data storage. The UI layer interacts directly with users through a web browser, while the frontend handles client-side logic and initial NLP processing. The backend, when implemented, manages server-side operations and interacts with a database for data persistence. This architecture is deployed on cloud platforms, ensuring accessibility and scalability within the operational environment.

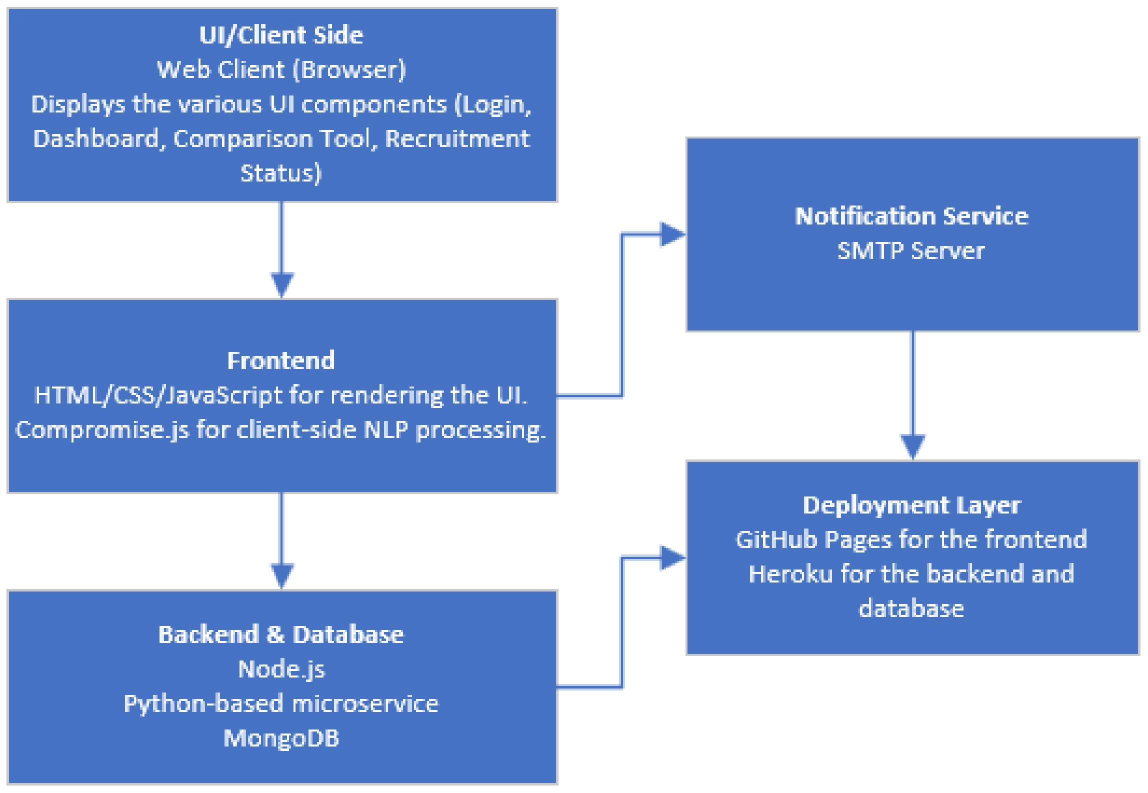
UI/Client Side
Web Client (Browser)
Displays the various UI components (Login, Dashboard, Comparison Tool, Recruitment Status)
Frontend
HTML/CSS/JavaScript for rendering the UI.
Compromise.js for client-side NLP processing.
Backend & Database
Node.js
Python-based microservice
MongoDB
Notification Service
SMTP Server
Deployment Layer
GitHub Pages for the frontend
Heroku for the backend and database

Figure 1. Software Architecture Context Diagram.

The system interacts with users primarily through the web interface, where they can log in, upload resumes, and view analysis results. Users trigger actions that initiate frontend and optional backend processing. The system may interact with external APIs for sending email notifications to candidates regarding status changes. Additionally, it can integrate with other HR systems through RESTful APIs to fetch job data or submit candidate information, facilitating seamless data exchange and enhancing recruitment workflows.

2.2 Stakeholders

Recruiters and hiring managers use the system to streamline the resume screening process and make informed hiring decisions. Developers and system architects are responsible for building, maintaining, and scaling the tool. Candidates, while indirect stakeholders, benefit from a more efficient and transparent hiring process, receiving timely feedback and updates on their application status.

The system is designed to meet the specific needs of its stakeholders by automating and enhancing the resume screening process. For recruiters and hiring managers, it reduces manual effort and increases the accuracy of candidate matching, leading to more efficient hiring decisions. Developers benefit from a scalable and modular architecture that supports easy maintenance and extension. For candidates, the system provides a more responsive application experience, ensuring they receive feedback and updates promptly, which improves their overall satisfaction with the recruitment process.

3. Architectural Background

3.1 Motivation and Design Decisions

The architecture is shaped by business goals such as improving recruitment efficiency, reducing manual screening efforts, and enhancing candidate-job matching accuracy. Engineering objectives focused on creating a scalable, maintainable, and modular system that supports both client-side and server-side processing. The use of NLP techniques was prioritized to automate the analysis of resumes and job descriptions, providing recruiters with data-driven insights. The architecture was also influenced by the need for seamless integration with existing HR systems and tools.

Key quality attributes for this system include performance, security, and scalability. Performance is critical to ensure fast processing of resumes and quick responses to user interactions, especially during peak usage. Security is essential to protect sensitive candidate data and ensure compliance with privacy regulations. Scalability is necessary to handle an increasing number of users and large volumes of data as the system grows. The architecture incorporates these attributes by using efficient algorithms, secure data handling practices, and cloud-based deployment strategies that can easily scale.

The system is designed to address several key scenarios in the recruitment process. These include automating the screening of resumes against job descriptions, ranking candidates based on their fit for a role, and notifying candidates about their application status. It also supports recruiters in comparing multiple candidates and making informed decisions quickly. The tool is flexible enough to handle various job types and industries, making it a versatile solution for different recruitment needs. By addressing these scenarios, the system aims to streamline the hiring process, reduce time-to-hire, and improve the quality of hires.

3.2 Architectural Drivers

3.2.1 Non-Functional Requirements (NFRs)

The architecture design was heavily influenced by several non-functional requirements (NFRs) that are critical to the system's success. Performance was a top priority, requiring the system to process large volumes of resumes quickly and deliver near-instant feedback to users. Scalability was essential to accommodate growing user bases and data loads, particularly during peak hiring seasons. Security was paramount, ensuring that all user and candidate data is protected against unauthorized access, with compliance with data protection regulations like GDPR. Maintainability and extensibility were also key, necessitating a modular design that allows for easy updates and the integration of new features. Usability was another important NFR, ensuring the system is user-friendly for both recruiters and candidates. Finally, reliability was crucial to provide consistent and accurate analysis without downtime.

3.2.2 Constraints

Several constraints influenced the architecture, including the chosen technology stack. The decision to use JavaScript, with frameworks like Node.js for the backend and Compromise.js for NLP, dictated certain performance characteristics and integration capabilities. Additionally, the architecture had to be designed with cloud deployment in mind, leveraging platforms like GitHub Pages and Heroku, which introduced constraints related to platform-specific features and scalability limits. Regulatory requirements like GDPR and data privacy laws also imposed constraints, particularly in how data is stored, processed, and transmitted, necessitating robust encryption and secure data handling practices.

3.2.3 Architectural Decisions

Key architectural decisions were made to address the identified drivers and constraints. The choice to implement a client-server architecture allows for separation of concerns, with the frontend handling immediate user interactions and the backend managing complex data processing and storage. The decision to use Compromise.js for client-side NLP was driven by the need for quick, in-browser processing, reducing the load on the server and improving responsiveness. For the backend, using Node.js and Express.js was chosen to leverage JavaScript's asynchronous capabilities, which are crucial for handling multiple concurrent user requests efficiently. The database choice, opting for MongoDB, was influenced by its flexibility in handling unstructured data like resumes. Finally, the decision to deploy on cloud platforms like GitHub Pages and Heroku ensures scalability and ease of deployment, while also aligning with the constraints of the chosen technology stack and regulatory requirements.

4. Architectural Specification

4.1 Use Cases and Functional Responsibilities

The system supports several key use cases critical to the recruitment process. Resume Upload and Parsing allows users to upload resumes, which are then parsed to extract relevant information such as skills, experience, and education. Job Description Analysis enables recruiters to input or upload job descriptions, which the system analyzes to identify required qualifications. Candidate Matching and Ranking is a core use case, where the system compares resumes against job descriptions using NLP techniques to rank candidates based on fit. Status Tracking and Notifications let recruiters update candidate statuses and automatically notify candidates via email. Responsibilities are allocated across different system components to ensure efficient operation. The frontend handles user interactions, presenting data, and initial client-side processing, while the backend manages more complex tasks like data storage, API integration, and advanced NLP processing. The database stores user data, resumes, job descriptions, and analysis results. The notification service handles automated communications with candidates.

4.2 Architectural Constraints

Several constraints influenced the architectural design, including hardware limitations and cloud deployment restrictions. Since the system is designed to run in a web environment, it needed to be lightweight and optimized for performance, particularly in environments with limited processing power or bandwidth. The choice of JavaScript and related technologies was partly driven by these considerations, as they allow for efficient execution in the browser. Legacy system integration posed another constraint, particularly for organizations using older HR systems. The architecture had to be flexible enough to integrate with these systems via APIs or data export/import functionalities. Additionally, the need to comply with data privacy regulations such as GDPR imposed constraints on data handling, storage, and transmission.

4.3 Architectural Drivers

The architectural drivers were primarily focused on scalability, performance, security, and usability. Scalability was a critical drive, as the system needed to handle varying loads, especially during peak recruitment periods. This led to the decision to implement a cloud-based architecture that could scale horizontally as needed. Performance was another key driver, influencing the decision to use asynchronous processing in the backend and client-side processing for NLP tasks to reduce server load and improve responsiveness. Security considerations drove the adoption of secure data storage practices, encryption, and compliance with regulatory standards. Usability was also a significant driver, ensuring that the system’s user interface was intuitive and accessible, enabling recruiters to quickly and easily perform tasks like resume screening, candidate comparison, and status updates. These drivers collectively shaped the system’s modular design, the choice of technologies, and the overall approach to deployment and integration.

5. Architecture Representation

5.1 High-Level Architecture

The high-level architecture of the system is decomposed into several key components or modules, each responsible for specific functions within the system. The User Interface (UI) Layer serves as the entry point for users, providing access to features like resume upload, job description analysis, candidate matching, and recruitment status tracking. This layer is built using HTML, CSS, and JavaScript, ensuring responsiveness and user-friendly interactions.

Beneath the UI layer lies the Frontend Logic Module, where client-side processing occurs. This module includes components like Compromise.js, which handles basic NLP tasks directly in the browser, allowing for immediate feedback and processing without the need to constantly interact with the server. This enhances the system's responsiveness and improves user experience.

The Backend Services Layer is responsible for more complex processing tasks. If implemented, this layer typically runs on a Node.js/Express.js stack, managing data persistence, handling API requests, and performing advanced NLP tasks that are too resource-intensive for client-side execution. The backend also serves as the intermediary between the frontend and the Database Module, which is responsible for storing user data, resumes, job descriptions, and analysis results. This data is securely stored and can be accessed as needed to support the system's various functions.

Finally, the Notification Service Module handles automated communications, such as sending email notifications to candidates regarding changes in their recruitment status. This service is typically integrated with external email services like SendGrid, ensuring reliable and secure messaging.

The high-level architecture of the system is decomposed into several key components or modules, each responsible for specific functions within the system. The User Interface (UI) Layer serves as the entry point for users, providing access to features like resume upload, job description analysis, candidate matching, and recruitment status tracking. This layer is built using HTML, CSS, and JavaScript, ensuring responsiveness and user-friendly interactions.

Beneath the UI layer lies the Frontend Logic Module, where client-side processing occurs. This module includes components like Compromise.js, which handles basic NLP tasks directly in the browser, allowing for immediate feedback and processing without the need to constantly interact with the server. This enhances the system's responsiveness and improves user experience.

The Backend Services Layer is responsible for more complex processing tasks. If implemented, this layer typically runs on a Node.js/Express.js stack, managing data persistence, handling API requests, and performing advanced NLP tasks that are too resource-intensive for client-side execution. The backend also serves as the intermediary between the frontend and the Database Module, which is responsible for storing user data, resumes, job descriptions, and analysis results. This data is securely stored and can be accessed as needed to support the system's various functions.

Finally, the Notification Service Module handles automated communications, such as sending email notifications to candidates regarding changes in their recruitment status. This service is typically integrated with external email services like SendGrid, ensuring reliable and secure messaging.

5.2 Low-Level Architecture

The class/package diagrams illustrate how responsibilities are systematically allocated across different classes within the module, ensuring a clear demarcation of functionalities. The UI Module contains the LoginComponent, responsible for handling user authentication and password management, which is essential for secure access control. The attributes include username and password, both of which are fundamental for login operations. The operations authenticate() and resetPassword() provide the necessary functionality to verify user credentials and assist in password recovery, respectively.

Class 2, labeled as a generic User class, acts as the foundation for user management within the system. It stores basic user information (userID, email, role) and includes operations like updateEmail() and changeRole(), allowing for dynamic updates to user profiles. This class serves as the base for specialized user types, demonstrated by Inherited Class 1 and Inherited Class 2.

Inherited Class 1, conceptualized as a Recruiter, extends Class 2 by adding recruiterID and managedPositions, enabling recruiters to manage job positions effectively. The assignPosition() operation allows them to allocate jobs to candidates or postings. Inherited Class 2, identified as a Candidate, extends from the generic User class as well, incorporating specific attributes like candidateID and resume, and operations such as upload Resume() and applyToPosition(), which are pivotal for job application processes.

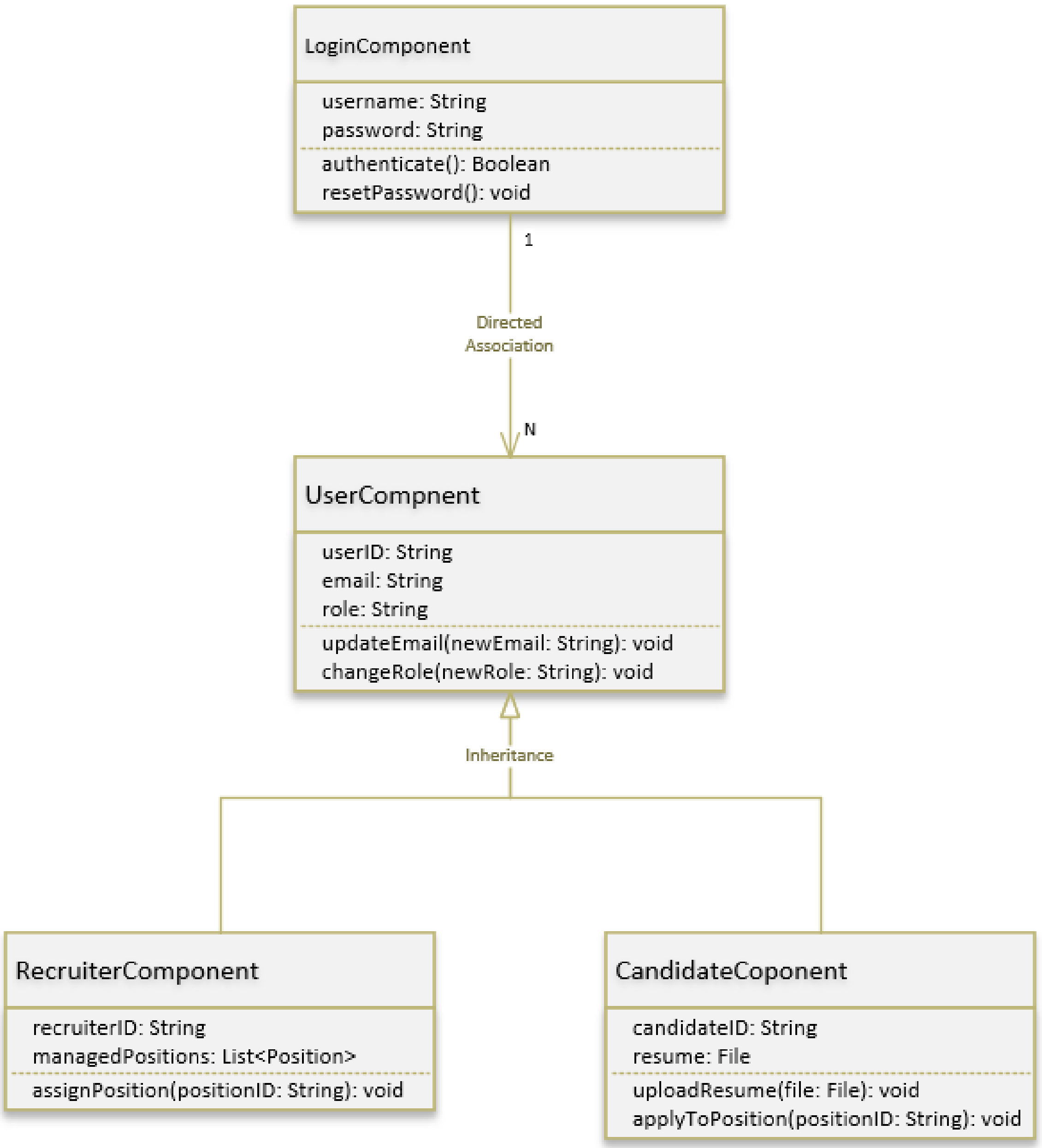

Figure 2. Class Diagram

For instance, when a candidate applies for a position, the Candidate class instance interacts with the Recruiter class instance through methods like applyToPosition(), triggering the Recruiter to evaluate the application using assignPosition(). This interaction is vital for the recruitment process, demonstrating direct communication between different user roles within the system.

The mapping between high-level architecture and low-level architecture is clarified through these diagrams and class definitions. The high-level architecture's focus on user interaction and data management is translated into specific classes with defined responsibilities in the low-level architecture. This mapping ensures that each component in the high-level design is effectively represented in the low-level implementation, maintaining consistency and coherence across the system. The high-level user management and authentication mechanisms are instantiated through the LoginComponent and User class in the lower-level, ensuring that the system's architecture supports both broad functionalities and detailed operations necessary for comprehensive user management and role-specific activities. This methodical mapping fosters a robust system architecture that is both scalable and maintainable, addressing the complex needs of user interactions within a dynamic recruitment environment.

5.3 Data Flow

The data flow within the system is designed to ensure efficient, secure, and intuitive interactions between different components, from user inputs through the UI to data processing and storage in the backend. Here's a detailed description of how data moves through the system:

1. User Interaction and Data Entry:
   - Starts at UI Module: Users interact with the system primarily through the UI, where they enter data such as usernames, passwords, and resume files. Recruiters might input job descriptions or select positions to manage.
   - LoginComponent: Here, user credentials (username and password) are entered for authentication. The component handles the data to verify user identities or initiate password resets.
2. Authentication and Session Management:
   - Data to Backend: Once entered, credentials are sent to the backend via secure API calls where they are checked against stored credentials in the database.
   - Session Token: If the authentication is successful, the backend generates a session token which is returned to the user's browser for maintaining a logged-in state.
3. Resume and Job Description Processing:
   - Resume Upload: Candidates upload resumes through the UI, which are then sent to the backend.
   - Processing: NLP algorithms process these resumes in the backend to extract relevant data like skills and experience. Similarly, job descriptions input by recruiters are processed to identify required qualifications and skills.
4. Data Storage and Retrieval:
   - Database Interaction: Extracted data from resumes and job descriptions are stored in the database under respective user profiles or job entries.
   - Retrieval: When a recruiter needs to view candidate information or compare candidates, the system retrieves this data from the database, processes it if necessary, and sends it back to the front end for display.
5. Candidate Matching and Notifications:
   - Matching Process: The backend performs a matching process between the job requirements and candidate profiles using stored data. This involves comparing extracted skills and qualifications.

    - Notification System: Once a match is determined, or a status update is required, the system triggers notifications which are sent to candidates via email. This uses the email addresses stored in the database.
6. Status Updates and Feedback:
    - Interactive Updates: Recruiters can update the recruitment status of candidates through the UI. These updates are sent to the backend, processed, and stored in the database.
    - Feedback to Users: Changes in status or important notifications are relayed back to users via the UI, ensuring they are always informed of their application status or any required actions.

6. Architecture Design Decisions and Trade-offs

6.1 Design Decisions

The design of the resume analysis tool involved several key decisions to optimize the system's functionality and efficiency. One such decision was the integration of client-side NLP processing using Compromise.js, chosen for its lightweight nature and ease of use, allowing for rapid parsing and analysis directly within the user's browser. This approach was selected over server-side processing to reduce server load and enhance responsiveness. Additionally, the use of MongoDB for data storage was decided based on its flexibility with unstructured data, making it ideal for storing diverse resume formats and contents. The modular architecture using Node.js and Express.js was designed to facilitate scalability and maintainability, enabling easy updates and expansion.

6.2 Architecture Trade-offs

During the architecture design process, several trade-offs were necessary to balance system requirements and performance. A significant trade-off was the decision to handle certain NLP tasks client-side, trading computational load on the server for increased dependence on the client's processing power. This decision was influenced by the need to provide real-time feedback to users, prioritizing responsiveness and user experience. Another trade-off involved using a NoSQL database over a traditional SQL database, prioritizing flexibility and scalability over the complex querying capabilities of relational databases. These decisions impacted on the system's scalability and maintainability, allowing for easier scaling and updates at the cost of potentially increased client-side resource requirements.

7. Quality Attributes Addressed by the Architecture

7.1 Overview of Quality Attributes

The architecture of the resume analysis tool prioritizes several critical quality attributes, including performance, security, and scalability. Performance is enhanced by distributing processing tasks between the client and server, ensuring that the system remains responsive even under load. Security measures include the use of HTTPS for secure communications, encrypted storage for sensitive data, and rigorous input validation to prevent injection attacks. Scalability is addressed through the system’s modular design, allowing components like the database and backend services to scale independently based on demand. This architectural approach ensures that the system can handle growing numbers of users and data volumes without degradation in performance or user experience.

8. Conclusion

8.1 Summary of Architectural Design

The architectural design of the resume analysis tool has been structured to optimize both functionality and user experience through strategic decisions across the front-end and back-end components. Key decisions include the use of Compromise.js for client-side NLP processing to reduce server load and enhance interactivity, and MongoDB for flexible, scalable data storage suitable for dynamic resume data. The choice of a modular architecture with Node.js and Express.js supports scalability and facilitates maintenance, ensuring the system can evolve easily over time. These decisions significantly impact the system by enhancing performance, ensuring scalability, and maintaining a high level of security.

8.2 Final Assessment

The architecture of the resume analysis tool effectively meets the established business goals and engineering objectives. By automating resume screening and enhancing the recruitment process, the system helps organizations efficiently identify the best candidates, aligning with the business goal of improving hiring quality. The architecture supports essential quality attributes like performance, security, and scalability, crucial for maintaining user trust and handling increasing loads. Overall, the architectural choices have proven sound, providing a robust foundation for current functionalities and future expansions.

8.3 Future Recommendations

For future architectural improvements, it is recommended to explore advanced NLP capabilities by integrating more sophisticated AI models, potentially through cloud services, to enhance entity recognition and sentiment analysis accuracy. Considering the expansion to a microservices architecture could further enhance scalability and allow more granular scaling of specific functionalities, such as real-time data processing and more complex analytics. Additionally, continuous evaluation of database solutions should be conducted to ensure the storage capabilities continue to meet the system’s needs as data volume and query complexity increase. These enhancements will ensure the architecture remains cutting-edge and can adapt to emerging recruitment technologies and data analytics trends.