Oregon State University

CS.025 Design Document

OPEN RESPONSE - OSU CS SENIOR CAPSTONE PROJECT

AUTHORS

Evan Baumann - 934-076-910
Sean Gibson - 934-356-114
Sage Morgillo - 934-070-648
Tran Cong Son Nguyen - 934-529-215
Nathan Rumsey - 934-129-273
Nathaniel Wood - 934-143-984

2025-03-02

Change Log

Table 1: Winter, 2025 Change Log

Date	Version	Change Description	Reason for Change	Author/Initiator
11/17/2024	1.0	Initial version of the Requirements Document.	Enumerate project functional and non-functional requirements, scope, and timeline.	Team
02/26/2025	1.1	Converted document from Word to LaTeX.	Standardized formatting for professionalism and easier maintenance.	Nathan Rumsey
03/02/2025	1.2	Removed section 4.9 (Cloud and Hosting).	IaC tools removed due to budget constraints.	Nathan Rumsey
03/02/2025	1.3	Updated section 5 (Deployment View).	Removed Terraform references to match revised deployment strategy.	Nathan Rumsey
03/02/2025	1.4	Removed section 7.4 (Infrastructure as Code).	IaC is no longer planned for implementation.	Nathan Rumsey
03/02/2025	1.5	Removed mentions of RabbitMQ.	The performance concerns it addressed are no longer relevant to project scope.	Nathan Rumsey
03/02/2025	1.6	Updated system architecture.	Reflects API consolidation, Docker Swarm usage, and RabbitMQ removal.	Nathan Rumsey
03/02/2025	1.7	Minor formatting adjustments to sections 2, 3, and 4.	Improved readability.	Sean Gibson

Contents

1 Introduction

1.1 Description

The Open-Source Classroom Polling Project is designed to be a free, accessible alternative to commercial classroom polling software. This project aims to provide a platform where educators can conduct live polls, collect real-time feedback, and assess student's understanding during lectures. By eliminating the financial burden for students, this project makes interactive classroom polling more accessible.

1.2 Goals

The primary objective is to create a user-friendly and responsive platform that allows instructors to conduct live polling during lectures, enabling educators to gauge students' understanding and adjust their teaching dynamically, fostering a more interactive and adaptable learning environment. Specifically, the project will focus on developing essential user-interface features to facilitate live polling, support multiple-question formats, provide automatic grading, and offer real-time feedback for instructors. Additionally, the platform will incorporate key features such as account creation for various roles (administrators, instructors, and students), course and lecture management, and live in-classroom polling.

To ensure the platform meets the needs of the dynamic classroom environment, we will prioritize incorporating scalable, accessible, and secure design principles.

2 Constraints

This is an open source project, where we are continuing the work of previous teams, as well as passing on the project to another team once we complete our work. We are constrained to slightly more than one academic term of full-time development, and even then development time is constrained within the context of all classes for each group member. However, this time constraint is not as critical, since we can reliably hand off our work to a future team. We have extremely limited funds available to handle hosting expenseses.

3 Context and Scope

3.1 Project Scope

The goal of this project is to create a free accessible alternative to commercial solutions like Top Hat and Poll Everywhere. The primary objective is to create a user-friendly and responsive platform that allows instructors to create and conduct live polling during lectures. We aim to implement poll creation, and real-time responses using socket.io, response collection, and result viewing. Users should be able to select roles (student, instructor), create classes, lectures with questions, and hold live lecture sessions. Students should be able to view and respond to live questions. Instructors should be able to assign points to questions and the grades from the lecture should be able to be exported to CSV and uploaded to Canvas or other Learning Management Systems (LMS) grade books.

3.2 External Systems

Our project will rely on several external systems to ensure smooth functionality and integration with existing platforms. The database will use MySQL to store and manage application data, including user accounts, poll responses, and other essential information. Additionally, we will have our notification service which communicates with external systems such as the web browsers or email clients to deliver notifications.

4 Solution Strategy

To build a scalable, efficient, and user-friendly class polling software, we will utilize a robust stack of modern technologies and frameworks designed to address specific needs across the application lifecycle:

4.1 Containerization and Deployment

- **Docker:** Docker will be used to containerize the application, ensuring consistency and reproducibility across development, testing, and production environments. Its ability to isolate dependencies and configurations facilitates collaboration among team members and ensures that environments remain identical regardless of the underlying host system.
- **Docker Swarm:** For orchestration, Docker Swarm will manage Docker containers, ensuring high availability, scalability, and fault tolerance. Swarm will handle tasks such as service discovery, load balancing through its built-in routing mesh, and service replication, making the deployment process streamlined and efficient.

4.2 Frontend Development

- **React:** The front-end will leverage React for building a responsive, interactive, and dynamic user interface. React's component-based structure supports reusability, which accelerates development while maintaining a clean and maintainable codebase. Features such as hooks and React's state management enable efficient data flow and integration with real-time back-end updates.
- **Vite:** Vite will serve as the build tool and development server for React. Its fast builds, hot module replacement, and efficient development process make it an excellent choice for projects requiring rapid iteration and testing. Vite's modern architecture ensures compatibility with the latest JavaScript standards, providing a streamlined development experience.

4.3 Backend Development

- **Node.js:** Node.js will act as the runtime environment for back-end services, offering scalability and efficiency for handling concurrent requests. Its non-blocking I/O model is ideal for real-time polling features.
- **Express.js:** For building RESTful APIs, Express.js will be used as a lightweight and flexible framework, simplifying the process of routing and middleware integration.

4.4 Real-Time Communication

- **WebSockets:** Real-time features, such as live polling updates and collaborative functionalities, will use WebSockets servers to maintain low-latency bidirectional communication between the client and server.

4.5 Database Management

- **MySQL:** MySQL will serve as the primary relational database for storing structured data such as user profiles, poll results, and application metadata. It ensures high performance, reliability, and ACID compliance, essential for data integrity.
- **Redis:** Redis will be used as an in-memory data store for managing transient real-time data like live poll states and user sessions. Its high-speed data operations make it well-suited for scenarios requiring quick access and temporary storage.

4.6 Development and Build Tools

- **ESLint and Prettier:** These tools will enforce code consistency and quality across the development team, reducing the likelihood of bugs and improving maintainability.
- **GitHub Actions:** Continuous Integration and Deployment (CI/CD) workflows will use GitHub Actions for automating testing, building, and deployment pipelines.

4.7 Monitoring and Performance Tools

- **k6:** Performance testing will leverage k6 to simulate concurrent user loads, identifying performance limitations and ensuring the application meets its scalability targets.

4.8 Accessibility and Testing

- **axe-core:** Automated accessibility testing will use axe-core to identify and resolve compliance issues with WCAG standards, ensuring an inclusive user experience.
- **Cypress:** Cypress will be employed for end-to-end testing to validate entire user workflows in a simulated environment, guaranteeing functional correctness.
- **Vitest and Jest:** These tools will handle unit and integration tests for the front-end and back-end, ensuring robust and defect-free code.

4.9 Conclusion

By integrating these technologies, the class polling software will deliver high performance, scalability, and an excellent user experience while adhering to modern development standards. Each tool and framework plays a critical role in addressing functional and non-functional requirements efficiently.

5 Deployment View

To deploy our application, we are going to use AWS Lightsail. We chose AWS Lightsail for its essential webhosting feature selection, in addition to its competitive hosting pricing that fits within our budget. If the project partner later decides to pursue with an on-premise deployment solution versus a cloud-based one, provided the deployment solution is a Linux system, our scripts and documentation will still function, making deploying the application consistent and reproducible no matter the desired deployment destination.

6 Crosscutting Concepts

6.1 Security

Ensuring the security of the software is a critical priority, particularly in the context of managing sensitive student data. The system is designed to align with industry standards and comply with the Family Educational Rights and Privacy Act (FERPA). This section details the measures taken to safeguard data, enforce proper access controls, and ensure that privacy and security requirements are met at all levels.

6.1.1 FERPA Compliance

FERPA mandates that student information, including grades and responses, be protected and used only for authorized educational purposes. To comply with these regulations, the software incorporates the following strategies:

1. **Data Privacy**:
    - Student responses and personal data are encrypted both in transit and at rest using AES-256 encryption.
2. **Access Restrictions**:
    - Personal data is only accessible to instructors and authorized personnel through secure, authenticated access.
    - Sensitive data fields, such as names and grades, are hidden from other students and external users at all times.
3. **Data Minimization**:
    - Only the minimum necessary student data is collected and stored to facilitate polling, classroom management, and grade book integration.

6.1.2 Role-Based Access Control (RBAC)

The system employs a robust Role-Based Access Control (RBAC) mechanism to limit user permissions based on their roles:

1. **Defined Roles**:
    - *Instructors*: Authorized to create polls, manage classes, view aggregated student responses, and export grades.

- *Students*: Limited to participating in polls, viewing results, and linking accounts to class rosters.

2. **Permission Enforcement**:
   - Each role is assigned a distinct set of permissions to ensure access is limited to only the functionalities necessary for that role.
   - All data access requests are validated against RBAC rules, and unauthorized access attempts are logged and blocked.

6.1.3 Encryption and Secure Communication

Encryption is a cornerstone of the software's security model, protecting data during transmission and storage:

1. **Data in Transit**:
   - All communication between clients and servers is secured using SSL/TLS protocols to prevent eavesdropping and man-in-the-middle attacks.
2. **Data at Rest**:
   - Sensitive data, such as student responses and account credentials, is encrypted using industry-standard AES-256 encryption.
   - Passwords are hashed using bcrypt with a strong salt to ensure secure storage and prevent unauthorized access.

6.1.4 Authentication and Session Management

The software employs secure authentication and session management protocols to ensure only authorized users access the system:

1. **Secrue Authentication**:
   - All user accounts require strong password protection, and passwords are securely hashed and stored.
2. **Session Management**:
   - Session tokens are generated securely and expire after a predefined period of inactivity to minimize risk.

6.1.5 Logging, Monitoring, and Incident Response

To support detection and response to security threats, logs from each microservice will be kept for later analysis by server maintainers.

6.2 CI/CD Pipeline

To prepare for future deployment by other institutions or developers the code base is designed to be deployment-agnostic, whether end-users deploy this project on-premises or using cloud compute resources. Our CI/CD setup will allow for easy adaptation by others, with scripts and processes, this will enable future users or institutions to quickly establish their own CI/CD pipeline if they choose to fork the repository to maintain automated deployments.
We will incorporate the following GitHub workflows in our CI/CD pipeline:

1. **Continuous Integration**:
    - GitHub workflows run automated tests, including unit, integration, end-to-end, and accessibility tests. Automated testing follows the Testing Strategy below.
    - Ensures code quality and reliability through comprehensive testing.
2. **Continuous Deployment**:
    - GitHub Actions automate the build and deployment process by building Docker images and pushing the updated containers to a registry.
3. Performance testing is conducted outside of the CI pipeline to ensure accurate load testing in production-like environments.
4. **Testing Strategy**:
    - **Unit Testing**:
        - **Frontend**: Using `Vitest` to test utility functions and data transformations. This ensures early detection of bugs in foundational logic independent of the DOM.
        - **Backend**: Using `Jest` for high coverage of individual methods and core logic in the backend. Tests focus on validating functionality such as data processing and utility methods.
    - **Integration Testing**:
        - **Frontend**: Utilizing `jsdom` and `Vitest` to simulate the DOM environment. This validates how components interact and ensures workflows such as form submissions and navigation function correctly.

  - **Backend**: Conducting integration tests to verify interactions between APIs, the database, and external services. These tests ensure correctness in complex, real-world scenarios. Other than specific cases, backend testing may be sparse, based on the level of coverage needed to be achieved through other test types.

- **End-toEnd (E2E) Testing**:
  - Using `Cypress` to simulate real-world scenarios from a user's perspective. E2E tests validate workflows such as creating polls, submitting responses, and navigating the platform, ensuring seamless functionality across all integrated components.
- **Performance Testing**:
  - Conducted with `k6` to measure the platform's responsiveness under load. This includes simulating multiple concurrent users and assessing metrics such as response times, resource usage, and data throughput.
- **Automated Accessibility Testing**:
  - Leveraging tools like `axe` within the CI/CD pipeline to ensure compliance with WCAG 2.1 Level AA. These tests automatically detect common accessibility issues, such as improper color contrast and missing alternative text.

6.3 Accessibility Standards

To ensure that our application is accessible to all users, we will be adhering to the Web Content Accessibility Guidelines (WCAG) version 2.1 at the AA level. This standard provides a comprehensive set of requirements to enhance usability and accessibility for individuals with disabilities. By following WCAG 2.1 AA, we commit to designing an interface that is perceivable, operable, understandable, and robust for users with various needs. This includes implementing clear and consistent layouts, supporting keyboard navigation, and ensuring sufficient color contrast. Meeting these guidelines will help us create an inclusive application that accommodates users with visual, auditory, motor, and cognitive disabilities, thus improving the overall user experience and ensuring compliance with industry best practices.

7 Architecture Concepts

The design of the project architecture reflects a shift from a monolithic architecture design from the previous group to a hybrid microservices and monolithic based approach, aligning with the project partner's goals of scalability, maintainability, and performance optimization.

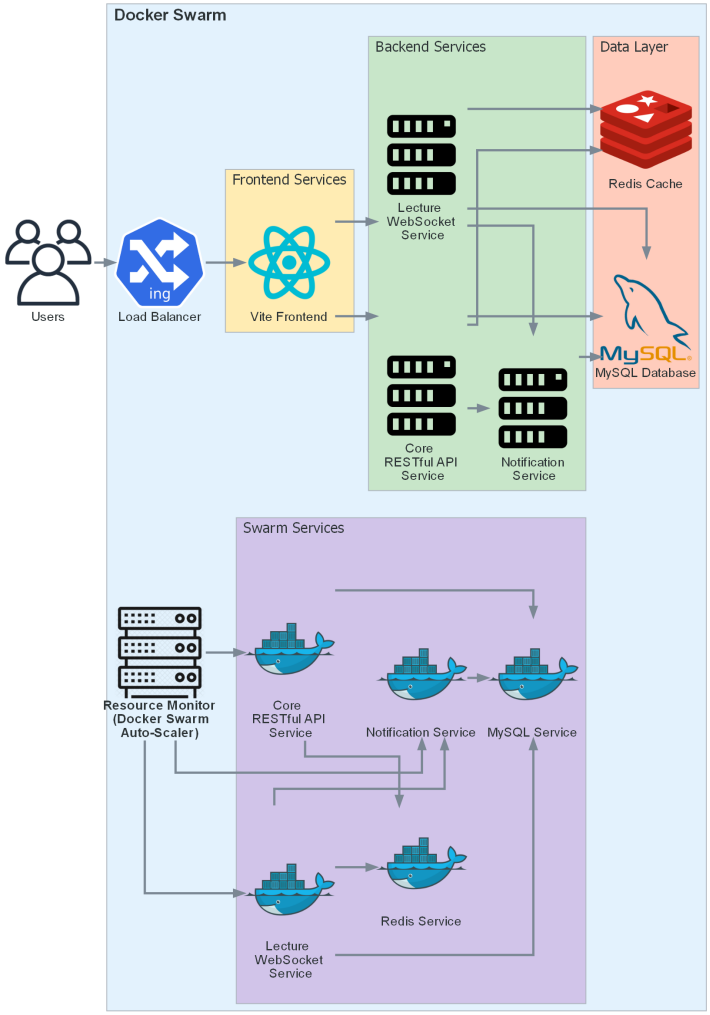

Figure 1: Open-Response Docker Swarm Microservices Architecture: A Scalable and Resilient System with Auto-Scaling, Load Balancing, and Efficient Service Communication.

Figure 1 contains an overview of the general microservice architecture. This system architecture leverages Docker Swarm for deployment, scaling, and resource management of the services within Docker containers.

7.1 Architecture Design

1. **Microservices Architecture**:
    - The backend is restructured into multiple, independently deployable microservices to enhance scalability and fault isolation.
    - Each service focuses on a specific domain (e.g., Authentication, Core API, LMS Integration) and communicates via RESTful APIs and WebSockets.
2. **Key Microservices**:
    - **Frontend Service**:
        - A React-based web application built with Vite.
        - Handles user interactions and communicates with backend services via RESTful APIs and WebSockets.
        - Passes authentication details to the Core RESTful API and includes JWT in subsequent API calls.
    - **Core RESTful API**:
        - Centralized management of users, classes, lectures, questions, and grade books.
        - Manages user authentication using a combination of sessions and JSON Web Tokens (JWTs) and Single Sign-On (SSO) methods.
        - Issues and verifies JWTs upon successful login for stateless, scalable authentication.
        - Handles notifications via the Notification Service.
        - Stores persistent data in the MySQL database.
    - **Notification Service**:
        - Handles user notifications (e.g., emails, browser notifications) using third-party services like Courier.
        - Triggered by the Core API and Lecture Service for user-facing notifications.
    - **Lecture WebSocket Service**:
        - Manages real-time lecture states using WebSockets (e.g., socket.io) for connected users and comptuers.
        - Stores live states in Redis for efficient in-memory processing.
        - Persists final state data to the MySQL database post-lecture.

3. **Databases**:

   - **MySQL Database**: Stores persistent data related to users, classes, lectures, questions, grades, and other entities.
   - **Redis Datastore**: Utilized for in-memory storage of transient, real-time lecture states and other short-term data.

7.2 Relational Database Schema

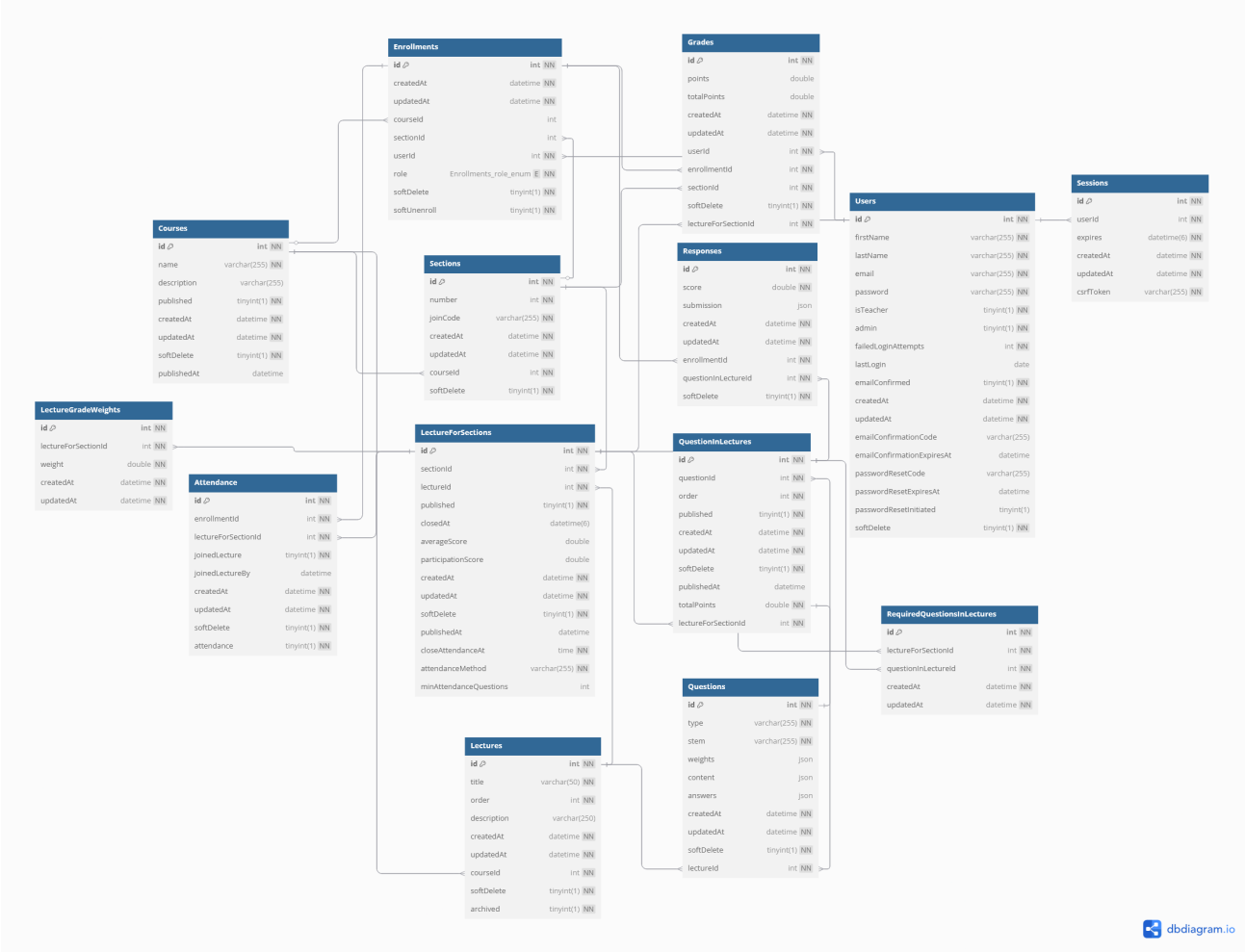

Figure 2: Database schema representing the relationships between courses, users, enrollments, lectures, and associated data.

7.3 Containerization and Orchestration

To ensure consistency across development and production environments, containerization and orchestration tools are leveraged. This approach simplifies deployment, scaling, and management of services.

1. **Containerization**:
    - Each microservice is encapsulated in its own Docker container.
    - Multi-stage Docker builds are used to optimize image sizes and streamline dependency management.
2. **Orchestration with Docker Swarm**:
    - Each service runs as a Docker service with multiple replicas for high availability and scalability. Docker Swarm handles container orchestration, leveraging:
        - **Ingress Routing Mesh** for efficient load balancing and internal service communication.
        - **Auto-scaling via service scaling commands** to adjust hte number of replicas based on demand.
3. **Persistent Volumes**:
    - The MySQL database is backed by persistent volumes to ensure data durability across container restarts.
    - Plans to evaluate cloud-native solutions for database management during cloud deployment.

7.4 Deployment Automation

Scripts will be provided to automate set up of Docker, Docker Swarm, cloning of the GitHub Repository, MySQL, and other dependencies in cloud environments.

8 Quality Requirements

Performance and scalability are key to ensuring a seamless user experience and meeting project goals. Success metrics include 99.9% uptime, a maximum response time of 500ms during live lectures, and the ability to support up to 1k-5k simultaneous users during peak times.

1. **Performance Optimization**:
    - **Caching**: Use Redis for caching frequently accessed data and managing live state during lectures to minimize database queries.
    - **Load Testing**: Conduct performance tests to identify bottlenecks and optimize service performance under simulated peak loads.
2. **Scalability Measures**:
    - **Horizontal Scaling**: Kubernetes' Horizontal Pod Autoscaler adjusts the number of service pods based on resource usage, ensuring the system can handle varying loads.
    - **Vertical Scaling**: Allocate additional resources (CPU, memory) to services and databases as needed to support higher loads.
    - **Database Optimization**: Indexing and query optimization ensure efficient database performance even during high traffic.
3. **High Availability**:
    - **Load Balancing**: Distribute traffic evenly across multiple replicas to prevent overloading any single instance.
    - **Monitoring and Alerts**: Implement monitoring to automatically restart services in case of unexpected failures or errors.

This architecture ensures a robust, scalable, and maintainable system that supports high availability and performance while allowing for future extensibility and integrations.

9 Risks and Technical Debt

9.1 Risks

If deployment is restricted to cloud instances with severely limited compute and memory due to budget constraints, such as our budget constraints during development of this project, the application may fail to operate effectively. This constraint can result in performance bottlenecks, excessive request latency, or outright service failure under minimal traffic, impeding the project's ability to function as intended.

Another significant risk arises from the exclusion of advanced security and monitoring capabilities from the project scope as dictated by the proejct partner. Without important security features such as Single Sign-On (SSO) and Multi-Factor Authentication (MFA), the system may not be suitable for production deployment, particularly in environments requiring stringent access controls. Furthermore, the absence of logging, monitoring, and incident response tools (e.g., the Grafana stack) reduces the system administrator's ability to detect, diagnose, and respond to security threats or poor performance in real time, increasing the likelihood of undetected system failures or exploitation. These features are excluded from the project scope due to limited development time.

Additionally, while this project's code may successfully implement all desired features and interactivity, the lack of advanced scalability mechanisms beyond what is already present in the software design poses a risk under high user loads. Without optimizations such as offline computing capabilities or database sharding and replication, the system may struggle to scale beyond a limited user base. In combination with constrained hosting resources, this limitation could lead to poor performance, long response times, and eventual service disruption when concurrent user counts reach the thousands or more. This tradeoff must be carefully evaluated to align development priorities with long-term usability and operational feasibility.

9.2 Technical Debt

The inherited codebase presents significant technical debt, directly impacting the scope of feasible implementations within the project timeline. A substantial portion of development time has been, and will continue to be, allocated to resolving foundational issues before meaningful feature development can proceed. The project was handed off in a non-functional state, with broken authentication and deployment mechanisms, requiring development time before any further progress could be made.

Beyond these issues, the codebase suffers from poor architectural design. While refactoring would improve maintainability and future extensibility, the time investment required is prohibitive, forcing development to proceed under suboptimal conditions. This imposes an ongoing burden on project team members, as new features must be integrated into an inefficient and poorly structured foundation.

Additionally, the backend documentation is incomplete, inconsistent, and in several instances, outright incorrect. Essential details regarding system operations, and architectural decisions are either missing or inadequately explained. The development environment setup instructions, in particular, contained errors that stalled initial project setup, including an undocumented Docker/Node.js compatibility issue that required downgrading Node.js to a specific version for code to run in Docker containers at all. Addressing these documentation gaps is necessary to reduce friction and ensure future teams can navigate the codebase effectively.

List of Figures

List of Tables