SP-8 Volunteer Manager

Final Report

Course number: 4850
Section: 01
Semester: Fall
Year: 2024
Date: April 10, 2024
GitHub Link: https://github.com/SP8VolunteerManager
Website: https://sp8volunteermanager.github.io/
Number of Lines of Code in the Project: 5-10 Thousand.
Number of Component: 5

Project Team

Roles	Name	Major responsibilities
Team leader	Cade Percival	Product, Requirements, and Deliverables.
Team members	Bryce Moore	Developer, QA, and Research.
	Thomas Ness	Developer, QA, and Research.
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.

Table of Contents

1 Introduction

This document serves as the final report for the SP 8 Volunteer Manager Application. It covers the design considerations, architectural strategies, system architecture, systems design, challenges, narrative discussion, design constraints, requirements and test plans aiming to provide a comprehensive report for the application.

1.1 Outline

This final report document is organized into several key sections:

- Introduction: Provides an overview of the document, including its purpose and scope.
- Design Considerations: Discusses factors that have influenced design decisions, including system constraints, software and hardware considerations, and design methodologies.
- System Architecture and Components: Describes the architectural design of the system, including major components and their interactions.
- Policies and Tactics: Describes the details of the system that don't affect the overall system in a major way but still affect the system enough to be mentioned.
- Detailed System Design: Elaborates on the design of specific system components, data models, and workflows.
- Challenges: Discusses the challenges that the team experienced when developing the final product of the application.
- Narrative Discussion: Discusses how this project was developed and what we did to accomplish the final product.
- Design Constraints: Provides an overview of the application and what is not possible with the current environment.
- Requirements: Covers all the required functions that the developers had to finish before the project was complete.
- Test Plan and Test Report: Provides explanation and documentation on the various test done on the application to ensure the readiness of the application.
- Conclusion: Ties everything together in a short explanation.

1.2 Description

The purpose of this document is to provide a summary of the SP8 Volunteer Manager application progress. It aims to cover everything that was required to make the project successful in it's design and completion.

1.2.1 Introduction

Purpose of This Document: The purpose of this final report is to thoroughly document the development process and design details of the SP8 Volunteer Manager application. It serves as a comprehensive overview, aligning our team's efforts with the project's strategic objectives and the specific needs of our users.

Scope of This Document: This report spans the full range of the software's design, from the initial architectural framework to the intricate design of individual components. It underscores the security protocols, data

management strategies, and the methodologies and technologies that have been pivotal in bringing this project to completion.

Intended Audience: This document is primarily targeted at the project's core team members, including software engineers, designers, and project managers. Additionally, it is a valuable resource for stakeholders and potential future developers, providing an in-depth look at the system's design principles and functional mechanics.

Background Documentation: This report follows on from the project plan document, which provides an overarching outline and insight into the project's goals and expected scope—a foundational piece that set the stage for this detailed final report.

Document Summary: Organized into key sections, this report methodically examines each aspect of the software's design. It starts with an exploration of design considerations, progresses through the architectural framework, and concludes with an elaborate examination of the system's design components.

1.2.2 System Overview

The SP8 Volunteer Manager application is designed to automate and streamline the management of volunteer data for organizations. It provides an intuitive web platform for volunteers to submit their information and for managers to approve, manage, and report on volunteer activities efficiently. The system supports account creation, data submission, manager review, automatic database updates, and data export functionalities. By simplifying these processes, the application aims to enhance the operational efficiency of organizations and improve the volunteering experience.

2 Considerations

The design considerations section describes many issues which need to be addressed or resolved before attempting to devise a complete design solution for the project. Many considerations must be made to ensure that the final product is effect and seamless for people using the product.

2.1 Assumptions and Dependencies

It’s essential to outline the assumptions and dependencies that could affect the software’s development, development, and usage. These factors ensure that the design and implementation phase consider the necessary conditions for the software to operate as intended.

Software and/or Hardware:

- Web Browsers: This application will assume that the user will have access to the internet and access to web browsers such as Chrome, Firefox, Safari, or Edge for optimal performance and compatibility.
- Server infrastructure: It is assumed that our application will be hosted on a scalable cloud infrastructure, such as AWS or Azure, to manage varying loads effectively.
- Database: The application is designed with PostgreSQL as its primary database system. Compatibility with other database systems is not assumed without modification.

Operating Systems:

- Compatibility: The application is designed to be platform-agnostic, with full functionality available on Windows, macOS, and Linux through web browsers. No specific operating system features are assumed to be available.
- Server OS: For backend deployment, a Unix/Linux-based operating system is assumed for its stability and performance benefits.

End-User Characteristics:

- Digital Literacy: It is assumed that end-users (volunteers and managers) possess basic digital literacy, including the ability to navigate web interfaces and use common web forms.
- Accessibility: The application design assumes the need to accommodate users with disabilities, adhering to WCAG 2.1 guidelines for accessibility.

Possible and/or Probable Changes in Functionality:

- Scalability: The application architecture assumes future scalability needs, both in terms of user base growth and the addition of new functionalities.
- Regulatory Compliance: Changes in data protection and privacy regulations may necessitate adjustments in how user data is managed and protected. The application's design includes flexible data handling processes to accommodate such changes.
- Integration Capabilities: While the initial release may not support extensive third-party integrations, the system architecture assumes future expansions to include API access for integrations with other software used by volunteer organizations, such as CRM systems or social media platforms.

By acknowledging these assumptions and dependencies, the development team can anticipate potential challenges and design the SP8 Volunteer Manager application with a clear understanding of the necessary conditions for its successful deployment and use. This foresight helps in mitigating risks associated with external factors and ensures a smoother development process and user experience.

2.2 General Constraints

In the Software Design Specifications for the SP8 Volunteer Manager application, recognizing global limitations and constraints is crucial for shaping the system's architecture and functionalities. These constraints can significantly impact various aspects of the software's design and implementation. Below are some of the key constraints and their associated impacts:

Hardware or software environment: The application’s design must ensure compatibility across various hardware and software environments with requiring specific, high-end hardware, limiting its accessibility. This requires a responsive, lightweight web interface and efficient backend services to accommodate users with older or less powerful devices.

End-user environment: Varied digital literacy levels among end-users requires a user interface that is intuitive and easy to navigate for all user demographics. This constraint influences the design to prioritize simplicity and clarity, possibly limiting the complexity of features that can be introduced without impacting usability.

Availability of Resources: Resource constraints, especially in terms of budget and development time, impact the scope of the product features. The design must be modular to allow incremental updates and feature additions without significant rework.

Interoperability Requirements: The need to integrate with existing systems (e.g., volunteer databases or CRM software) requires the application to be designed with flexible APIs and data exchange protocols, impacting the architecture, and potentially increasing the complexity of the system.

Interface/Protocol Requirements: Adhering to specific interface standards or protocols for data communication can restrict the choice of technologies and libraries, impacting development speed and potentially increasing costs. However, with a $0 budget this might prove to be difficult to accomplish.

Data Repository and Distribution Requirements: The need for robust, scalable data storage and distribution mechanisms influences the choice of database systems and data handling strategies, impacting system performance and scalability.

Security Requirements: Strict security requirements necessitate comprehensive encryption, secure coding practices, and regular security assessments, which can increase development time and complexity, especially in designing user authentication and data protection features.

Memory and Other Capacity Limitations: Capacity limitations of the server infrastructure may constrain the application's scalability and performance, especially in handling large numbers of concurrent users, necessitating efficient code and database optimization.

Performance Requirements: High performance requirements necessitate the optimization of front-end and back-end code, possibly limiting the use of certain feature-rich but resource-intensive frameworks or libraries.

Network Communications: The application's reliance on internet connectivity may limit its accessibility in areas with poor network infrastructure, impacting the design decision to make offline functionality a potential future requirement.

Verification and Validation Requirements: Rigorous testing requirements can extend development timelines but are critical for ensuring the reliability and stability of the application. This necessitates the incorporation of automated testing frameworks from the outset of development.

Addressing Quality Goals: Achieving high-quality goals may require additional resources and time for testing, user feedback collection, and iterative design improvements, impacting project timelines and budgets.

These constraints underscore the importance of strategic planning and design to mitigate potential impacts on the SP8 Volunteer Manager application's development. Balancing these limitations with the project's goals and user needs is crucial for successful software delivery.

2.3 Goals and Guidelines

The goals, guidelines, principles, or priorities that guide the design of a system's software is essential for aligning the development team and stakeholders with the project's overarching vision. Here are several key design principles for the SP8 Volunteer Manager application, along with the rationale for their importance:

User-Centric Design:
Goal: Prioritize ease of use and accessibility for all users, ensuring that the application is intuitive for volunteers and managers alike.
Rationale: A user-centric design ensures that the application can be easily adopted by its target audience, reducing the need for extensive training or support. By focusing on the needs and preferences of end-users, the application aims to enhance user satisfaction and engagement, crucial for volunteer-driven organizations.

Scalability and Flexibility:
Goal: Design the application with scalability in mind, allowing for easy expansion in terms of user base, functionality, and integration capabilities.
Rationale: As volunteer organizations grow and evolve, the software must be able to accommodate an increasing number of users and an expanding scope of functionalities without significant rework. Scalability ensures the application remains a valuable tool over time, and flexibility allows for the incorporation of new technologies and standards as they emerge.

Security by Design:
Goal: Embed security considerations into every aspect of the application design, from data storage to user interactions.
Rationale: Given the sensitivity of volunteer information and potential regulatory compliance requirements, prioritizing security from the outset minimizes vulnerabilities and protects against data breaches. This approach builds trust among users and stakeholders, critical for the application's success.

Efficient Resource Utilization:
Goal: Optimize the application for efficient use of server resources and bandwidth.
Rationale: Efficient resource utilization ensures that the application can handle high volumes of user activity without causing excessive operational costs. This principle is particularly important for non-profit and volunteer organizations that often operate with limited budgets.

Adherence to Standards and Best Practices:
Goal: Follow industry standards and best practices in software development, including coding standards, architectural patterns, and accessibility guidelines.
Rationale: Compliance with established standards and best practices not only enhances the quality and maintainability of the software but also ensures compatibility with other systems and technologies. This creates easier integration, updates, and regulatory requirements.

By adhering to these goals, guidelines, and principles, the SP8 Volunteer Manager application aims to deliver a robust, user-friendly, and secure platform that meets the needs of both volunteer organizations and their volunteers. These guiding principles form the foundation of software design, influencing decision-making throughout the development process.

2.4 Development Methods

The software design approach for the SP8 Volunteer Manager application is based on a combination of Agile development principles and User-Centered Design (UCD) methodology, with considerations for scalability, security, and maintainability. This hybrid approach ensures flexibility, responsiveness to user feedback, and robustness in the application's architecture and features.

Agile Development Principles

Description: Agile methodology emphasizes iterative development, where requirements and solutions evolve through collaboration between cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, encouraging rapid and flexible responses to change.

Adoption Reason: Agile was chosen for its flexibility in accommodating changes in requirements and its focus on delivering functional software quickly. It allows the project team to incorporate user feedback into the development process continuously, ensuring the final product closely aligns with user needs and expectations.

User-Centered Design (UCD)

Description: User-Centered Design is a framework of processes in which usability goals, user characteristics, environment, tasks, and workflow are considered at every stage of the design process. UCD places the user at the center of the design and development, ensuring that the product is tailored to meet their needs and preferences.

Adoption Reason: The SP8 Volunteer Manager application directly interfaces with volunteers and organizational managers, making UCD critical for ensuring the application's usability, accessibility, and effectiveness. This approach helps in creating a product that is not only functional but also intuitive and satisfying for its users.

Considered Methods:

Waterfall Model: While the sequential nature of the Waterfall model provides clear milestones and deliverables, it was deemed too rigid for this project due to its inability to accommodate changes easily after the design phase.

DevOps: Although DevOps practices, especially its emphasis on automation, continuous integration, and continuous deployment, were considered valuable, the initial setup and resource requirements were beyond the scope of the project's early stages. However, integrating DevOps principles into the Agile framework remains a long-term goal to improve deployment and operational efficiency.

By blending Agile development principles with User-Centered Design, the project team aims to create a responsive, user-friendly, and adaptable application. This combined approach ensures that the SP8 Volunteer Manager application not only meets the current needs of its users but is also well-positioned to evolve with future requirements and technologies.

3 Architectural Strategies

The architectural strategies for the SP8 Volunteer Manager application are designed to ensure a robust, scalable, and user-friendly system. These strategies are closely aligned with the project's design goals and principles, such as user-centric design, scalability, security, and efficient resource utilization. Below, we discuss key architectural decisions and strategies, including the rationale behind each and the trade-offs considered.

Programming Language and Framework Selection for Backend

- Decision: Adopt Java for the backend development, leveraging frameworks such as Spring Boot for creating the RESTful API.

- Rationale: Java is known for its robustness, scalability, and extensive ecosystem, making it a suitable choice for enterprise-level applications. Spring Boot simplifies the development of new services by providing a range of out-of-the-box functionalities for web development, security, data access, and more. This choice supports the need for a scalable, secure, and maintainable backend architecture.
- Impact on Design Goals:
    - Scalability and Performance: Java's strong performance characteristics and thread management capabilities align with the goal of building a scalable backend that can handle a high volume of requests efficiently.
    - Security: Java and Spring Boot offer extensive security features and configurations, which aligns with the priority of embedding security into every layer of the application.
    - Development Efficiency: While Java may have a steeper learning curve compared to JavaScript/Node.js, the mature ecosystem and comprehensive documentation of Java and Spring Boot can offset this, especially when building complex, enterprise-grade applications.

Reuse of Existing Software Components

- Decision: Incorporate open-source libraries and frameworks wherever appropriate, such as the Spring framework for java on the server side.
- Rationale: Leveraging existing, well-tested software components accelerates development, ensures reliability, and reduces the risk of bugs. It allows the team to focus on application-specific features rather than foundational elements.
- Alternatives Considered: Building custom solutions for routing, authentication, and database management. This approach was deemed inefficient given the availability of robust open-source alternatives.

User Interface Paradigms

- Decision: Adopt a Single Page Application (SPA) paradigm using React.
- Rationale: SPAs provide a fluid and interactive user experience, reducing page reloads and making the application feel more like a native desktop or mobile app. This approach aligns with the goal of creating an intuitive and efficient interface for volunteers and managers.
- Alternatives Considered: Traditional multi-page applications. Rejected due to the inferior user experience and slower interactions compared to SPAs.

Error Detection and Recovery

- Decision: Implement comprehensive client-side and server-side validation, along with clear error messaging and logging.
- Rationale: Robust error handling improves application reliability and user trust by providing clear feedback and minimizing disruptions in the user experience. Logging errors facilitate quick debugging and recovery.
- Alternatives Considered: Relying primarily on server-side validation. This was rejected due to the potential for a poor user experience with delayed feedback.

Memory Management Policies

- Decision: Utilize stateless server architecture and efficient caching mechanisms.

- Rationale: Stateless servers improve scalability by not storing user session data on the server, allowing requests to be distributed easily across multiple servers. Caching frequently accessed data reduces database load and improves response times.
- Alternatives Considered: Stateful server architecture. Rejected due to scalability concerns in handling a large number of concurrent user sessions.

External Databases and Data Storage Management

- Decision: Use PostgreSQL for structured data storage, with considerations for future integration of NoSQL databases for unstructured data.
- Rationale: PostgreSQL offers robust data integrity features, extensive support for complex queries, and reliability. It's suitable for the structured data needs of the application, such as volunteer records and activity logs.
- Alternatives Considered: Solely using NoSQL databases like MongoDB. While offering scalability and flexibility, they were deemed less suitable for the relational data aspects of the application.

Distributed Data or Control over a Network

- Decision: Design the system with a microservices architecture, where different functionalities can be developed, deployed, and scaled independently.
- Rationale: This approach allows for greater flexibility and scalability, enabling the system to adapt to changing requirements and workloads efficiently.
- Alternatives Considered: Monolithic architecture. Rejected due to potential scalability and flexibility issues as the application grows.

Communication Mechanisms

- Decision: Use RESTful APIs for client-server communication, with WebSockets for real-time features.
- Rationale: RESTful APIs provide a standard and scalable way of building web services, while WebSockets facilitate real-time communication between the client and server, enhancing the user experience for dynamic features.
- Alternatives Considered: SOAP for web services. Rejected due to its complexity and less suitability for web-based, real-time applications.

Each of these decisions was made after careful consideration of the project's goals, the development team's expertise, and the desire to create a scalable, secure, and user-friendly application. By choosing these strategies, the team aims to balance functionality, development efficiency, and future growth potential.

4 System Architecture

The SP8 Volunteer Manager application is designed with a microservices architecture to ensure scalability, flexibility, and the ability to rapidly develop and deploy features. This high-level system architecture aims to partition the functionality into distinct services or components, each with a focused responsibility. The decision to adopt this architecture is driven by the need for a system that can evolve over time, handle varying loads

efficiently, and simplify the development and maintenance process by allowing teams to work on different components independently.

4.1 Decomposition and Responsibilities

The system is decomposed into several top-level components, each designed to handle specific aspects of the application:

User Management Service:

- Responsibilities: Handles user registration, authentication, and profile management.
- Collaboration: Interacts with the Volunteer Management Service for accessing and updating volunteer profiles and with the Notification Service for sending emails or notifications upon registration or profile updates.

Volunteer Management Service:

- Responsibilities: Manages volunteer data, including hours logged, tasks performed, and organizations involved.
- Collaboration: Works closely with the User Management Service to link volunteer activities to specific user profiles and with the Reporting Service for generating volunteer activity reports.

Organization Management Service:

- Responsibilities: Manages information related to organizations, including organization profiles, volunteer opportunities, and manager accounts.
- Collaboration: Integrates with the Volunteer Management Service to associate volunteers with organizations and opportunities.

Reporting Service:

- Responsibilities: Generates reports for volunteer activities, organization contributions, and overall system usage statistics.
- Collaboration: Gathers data from the Volunteer Management Service and Organization Management Service to compile comprehensive reports.

Notification Service:

- Responsibilities: Sends notifications and communications to users, including email alerts and system notifications.
- Collaboration: Supports other services by providing a unified approach to sending out communications.

API Gateway:

- Responsibilities: Serves as the entry point for clients, routing requests to the appropriate services and aggregating responses.

- Collaboration: Interfaces with all services, providing a seamless integration point for the frontend application.

Rationale for Decomposition

The microservices architecture was chosen over a monolithic architecture to allow for independent development, scaling, and deployment of each service. This approach enables the team to implement updates or new features to one service without impacting others, reducing the risk of system-wide failures and facilitating quicker release cycles.

Alternatives Considered

A monolithic architecture was considered but ultimately rejected due to its potential for creating a tightly coupled system that is difficult to scale and maintain as the application grows. Similarly, a serverless architecture was evaluated for its scalability and cost-efficiency but was deemed premature for the project's current stage, given the uncertainties around the exact load patterns and the potential complexity of managing a serverless architecture across multiple cloud providers.

Design Patterns Employed

API Gateway Pattern: Centralizes request handling and provides a single entry point for all client requests, facilitating security measures like authentication and rate limiting.

Service Registry and Discovery: Enables services to dynamically discover and communicate with each other in a distributed environment, improving resilience and scalability.

Circuit Breaker Pattern: Protects the system from cascading failures by failing fast or providing fallbacks when a service is unavailable.

This architectural strategy ensures that the SP8 Volunteer Manager application can meet current and future demands, providing a robust foundation for delivering a high-quality, scalable, and user-friendly volunteer management platform.

4.2 Subsystem Architecture

SP8 Volunteer Manager application has a complex and multifaceted nature, certain components merit a more detailed discussion to understand their architecture and interactions within the broader system. one such component is the Volunteer Management Service, which is pivotal for managing volunteer data, including hours logged, tasks performed, and organizations involved. This service is crucial for the core functionality of the application and therefore deserves a closer look.

Volunteer Management Service Architecture

Overview

The Volunteer Management Service is designed as a microservice, part of the larger microservices architecture of the SP8 Volunteer Manager application. It handles all data and processes related to volunteers, including registration, activity logging, and task assignment.

Subcomponents

- Volunteer Profile Management:
  - Handles creation, updates, and retrieval of volunteer profiles.
  - Interacts with the User Management Service for authentication and user data.
- Activity Logging and Tracking:
  - Manages the recording of volunteer hours and activities.
  - Provides interfaces for volunteers to log their activities and for managers to review and approve them.
- Task Assignment and Management:
  - Facilitates the assignment of tasks to volunteers by organization managers.
  - Tracks the status of tasks, including completion and verification.
- Integration Layer:
  - Serves as the communication hub with other services, such as the Organization Management Service and the Reporting Service.
  - Ensures data consistency and facilitates the aggregation of data for reporting.

Interactions and Relationships

- Volunteer Profile Management directly interacts with the User Management Service to ensure that volunteer profiles are linked with their corresponding user accounts.
- Activity Logging and Tracking relies on data from Volunteer Profile Management to accurately record volunteer contributions.
- Task Assignment and Management uses information from both Volunteer Profile Management and Activity Logging and Tracking to assign appropriate tasks to volunteers and track their completion.
- The Integration Layer enables seamless data flow between the Volunteer Management Service and other system components, ensuring that all parts of the application have access to up-to-date and consistent volunteer data.

Rationale for Decomposition

The decomposition into these subcomponents was driven by the need to separate concerns within the Volunteer Management Service, allowing for focused development and easier maintenance. This structure supports scalability, as each subcomponent can be independently scaled based on demand. Additionally, it facilitates the implementation of specific security and data protection measures relevant to each subcomponent's functionality.

5 Policies and Tactics

Engineering Trade-offs

Database Performance vs. Consistency: For the Volunteer Management Service, the choice of database technology (PostgreSQL) and the use of transactional integrity mechanisms ensure data consistency, even though this may introduce performance trade-offs under high load conditions. Techniques like caching and database optimization are employed to mitigate potential performance issues.

Flexibility vs. Complexity: The microservices architecture offers flexibility and scalability but at the cost of increased complexity in deployment and management. The use of containerization and orchestration tools like Docker and Kubernetes is a tactic to manage this complexity effectively.

Specific Product Choices

Framework: Spring Boot was chosen for developing the Volunteer Management Service due to its extensive support for building microservices, including web services, data access, security, and messaging.

Communication: RESTful APIs are used for synchronous communication, with JSON as the data format for ease of use and compatibility with the frontend. For asynchronous communication and events, Apache Kafka is considered to decouple service interactions, enhancing system resilience.

This detailed discussion on the Volunteer Management Service provides insight into its critical role within the SP8 Volunteer Manager application, illustrating how it's architected to fulfill its responsibilities effectively while aligning with the system's overall design principles and goals. Further details and low-level design specifics would be covered in the Detailed System Design section or dedicated design documents for complex subsystems.

6 Detailed Systems Design

The web application includes 4 subsystems: Client application, REST API, Database, and Cloud Instance

6.1 Classification

Client application: A subsystem within the web application that runs in the client’s browser.

Database: Backend data storage system.

6.2 Definition

- Client application: A Single Page Application (SPA) using React, responsible for the presentation layer and client-side logic.
- Database: A relational database management system using PostgreSQL to store and manage application data.

6.3 Responsibilities

- Client application:
    - Rendering user interfaces for shift reports, manager dashboards, and profile management.
    - Handling client-side navigation and state management.

    - Communicating with the backend via RESTful APIs.
- Database:
    - Persistently storing user, group, and shift report data.
    - Providing data integrity and transaction management.

6.4 Constraints

- Client application:
    - Requires a modern web browser with JavaScript enabled.
    - Dependent on RESTful API responses for dynamic content.
- Database:
    - Scaled according to the storage and performance requirements of the application.
    - Dependent on the backend Java application for data access and manipulation.

6.5 Composition

- Client application:
    - Components for user authentication, forms for shift reporting, dashboards, and profile management interfaces.
    - Utilizes React Router for navigation.
    - Integrates with Axios or Fetch API for HTTP requests.
        - Example of using Axios to attach a JSON web token to a request

```
axios.interceptors.request.use(config => {

    const token = localStorage.getItem('token');

    config.headers.Authorization = `Bearer ${token}`;

    return config;

});
```

- Database:
    - Tables for users, groups, user_groups associations, and shift reports

6.6 Uses/Interactions

- Client application:
    - Interacts with the Java REST API for user authentication, data retrieval, submission, and updates.
    - Uses JWT (JSON Web Token) for secure authentication and session management .
        - When a user logs in, their credentials are sent to and validated by the Java backend. A JWT is sent in response and stored in the user's local storage. The JWT is then used in the authorization header of subsequent requests.
        - If the token expires or the user logs out, the token will be removed from local storage and the user's state will be updated to match their permissions, most likely by redirecting them to the login.
- Database:

    - Interacts with the backend Java application for CRUD operations.
    - Supports SQL queries for data retrieval and manipulation.

6.7 Resources

- Client application:
    - Browser's local or session storage for JWT.
    - External libraries like bootstrap for UI components
- Database:
    - Disk storage for database files.
    - Requires regular backups and maintenance for data integrity.

6.8 Processing

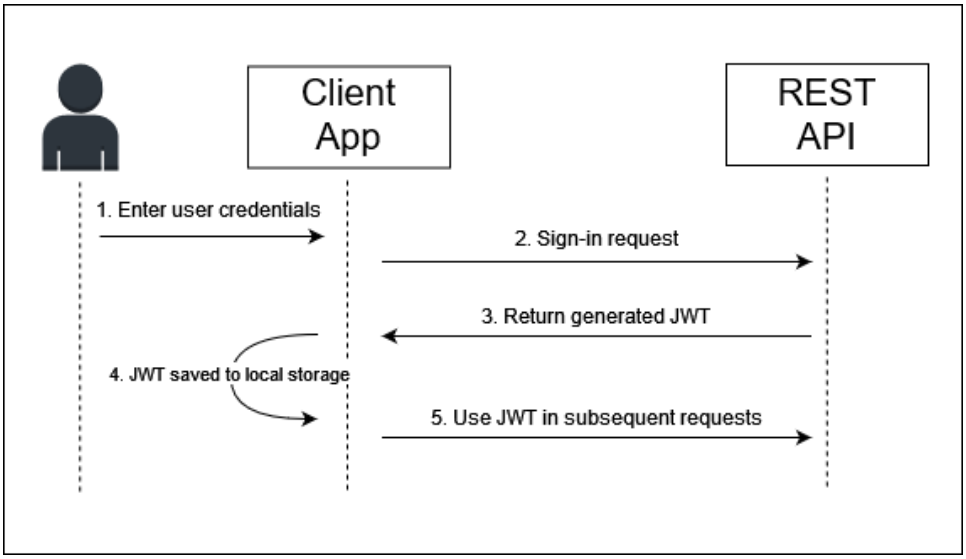
- Client application:
    - Implements client-side logic and rendering.
    - Handles form submissions and user inputs.
    - Manages state with React's useState.
- Database:
    - Processes SQL queries and transactions.
    - Manages indexing, caching, and query optimization.

6.9 Interface/Exports

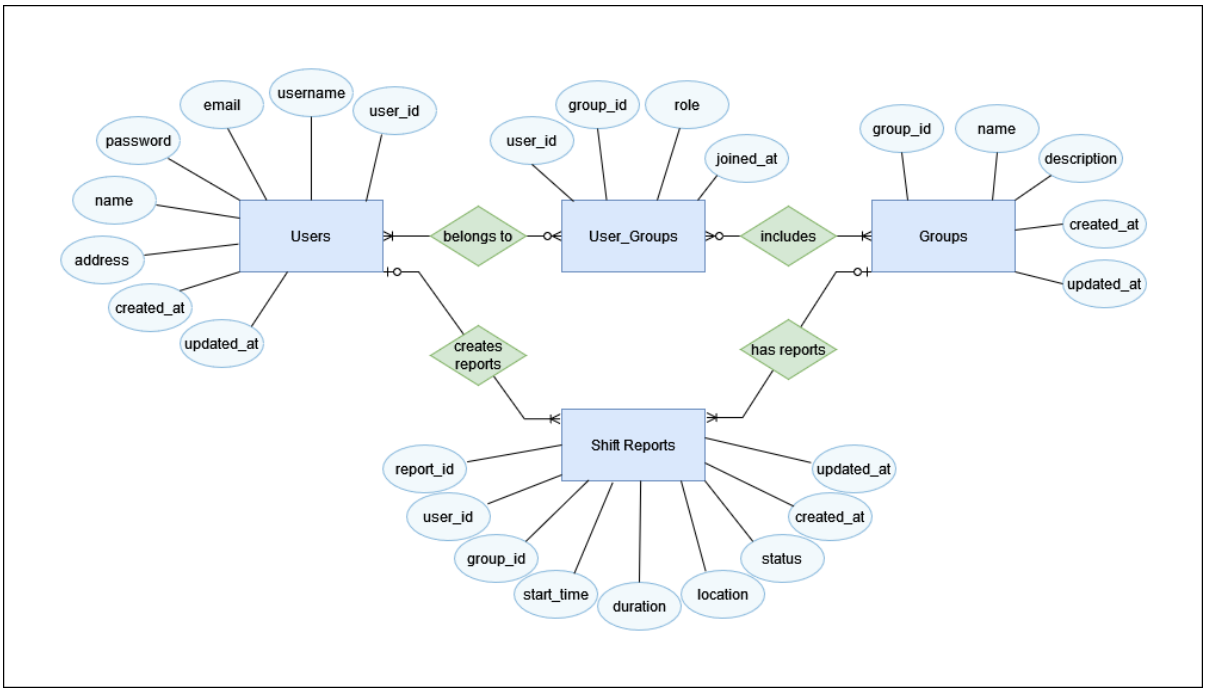
- Client application:
    - Exports various React components and hooks for use across the application.
- Database:
    - Provides SQL interface for data manipulation and retrieval.
    - Supports connections from the backend Java application.

6.10 Detailed Subsystem Design

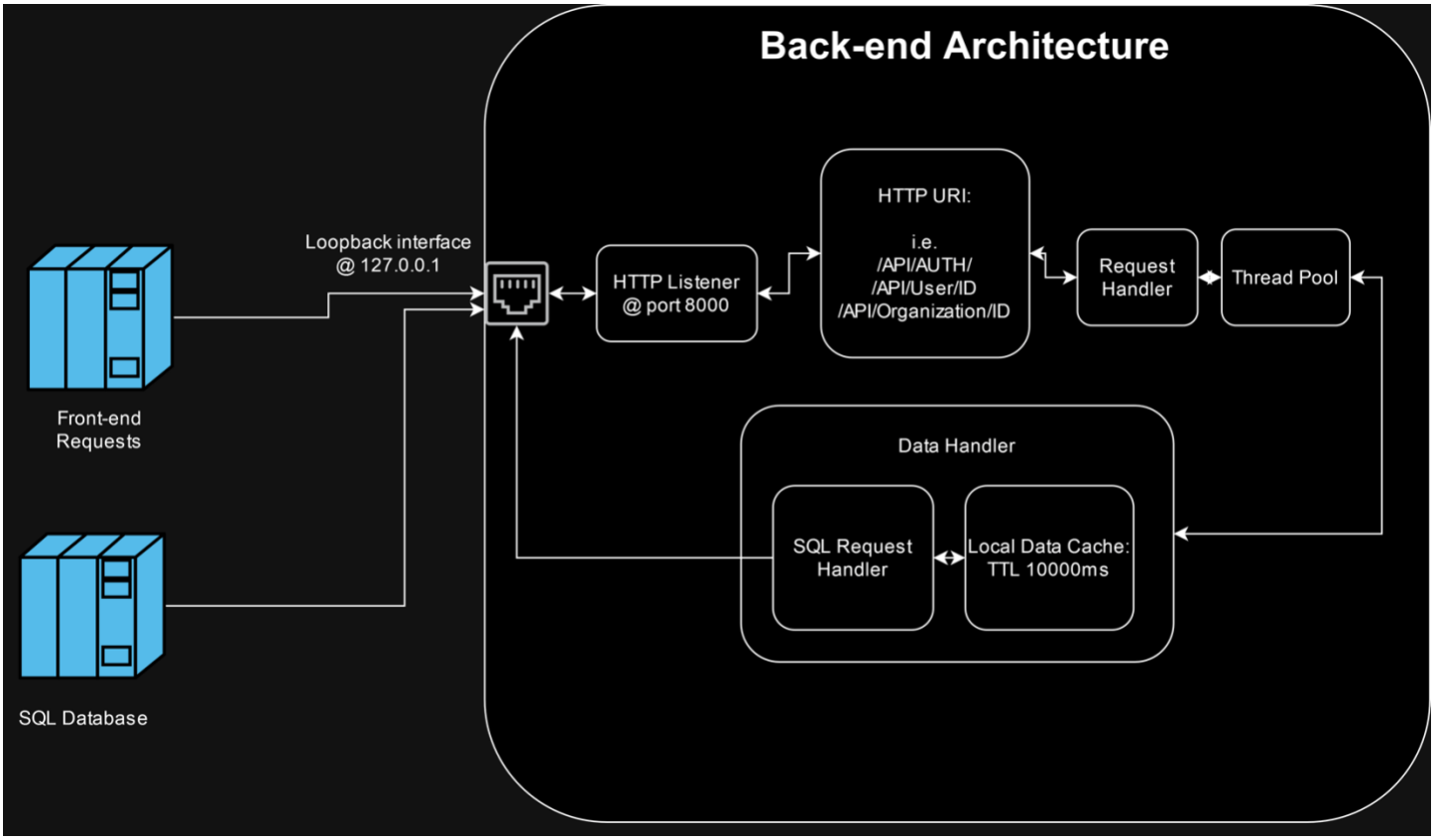
- A diagram showing the JWT authorization flow



- An Entity-Relationship Diagram



- Backend Architecture



7 Constraints

7.1 Environment

- **Web-Based Platform**: The application will be accessible exclusively through web browsers and will require design considerations for various browsers (Chrome, Firefox, Safari).
- **Internet Dependency**: Constant internet connectivity is required for all functionalities, which may limit usage in areas with poor internet connectivity.
- **Hosting and Scalability**: The application will be hosted on a GCP Compute instance to ensure scalability and high availability.

7.2 User Characteristics

- **Diverse User Base**: Considering users with varying levels of technical proficiency, the design must be intuitive and straightforward.
- **Device Compatibility**: Ensuring compatibility with various devices, including desktops, tablets, and smartphones, with a responsive design to adapt to different screen sizes.

7.3 System

The system will support user authentication, data management, and report generation, operating primarily on a cloud-based database.

8 Requirements

8.1 Functional

1. **User Registration and Authentication**
    - **3.1.1** The system shall allow volunteers to register by providing their name, phone number, email address, and password.
    - **3.1.2** The system shall allow managers to register with additional privileges.
    - **3.1.3** The system shall provide a secure login process for both volunteers and managers.
    - **3.1.4** The system shall offer a password recovery option.
2. **Volunteer Profile Management**
    - **3.2.1** Volunteers shall be able to view and edit their personal profile information.
    - **3.2.2** Managers shall be able to view and edit volunteer profiles under their management.
3. **Shift Submission and Management**
    - **3.3.1** Volunteers shall be able to submit shift details including date, start and end time, and a description of tasks completed.
    - **3.3.2** The system shall allow managers to view submitted shifts for approval or denial.
    - **3.3.3** Upon approval by a manager, the shift data shall be logged into the system's database.
4. **Reporting Functionality**
    - **3.4.1** The system shall allow managers to generate and export reports detailing volunteer activities and shift data.
    - **3.4.2** Reports can be exported in formats such as CSV or PDF.
5. **Manager Dashboard**
    - **3.5.1** The system shall provide a dashboard for managers to view pending shift approvals, volunteer activity, and other relevant information.
    - **3.5.2** Managers shall be able to add or remove volunteers from their organization within the system.
6. **Data Validation and Error Handling**
    - **3.6.1** The system shall validate input data in forms for errors and inconsistencies.
    - **3.6.2** The system shall provide user-friendly error messages and guidance for correction.
7. **Security and Data Protection**
    - **3.7.1** The system shall ensure the confidentiality and integrity of user data.
    - **3.7.2** The system shall implement measures to prevent unauthorized access to sensitive data.

8.2 Non-Functional

8.2.1 Security

- **4.1.1** The system shall implement role-based access control to ensure users can only access information relevant to their role.

8.2.2 Capacity

- **4.2.1. The system shall be scalable to support many concurrent users (at least 1000 concurrent sessions).**
- **4.2.2** The database shall be capable of efficiently handling large volumes of data without performance degradation.
- **4.2.3** The system shall maintain a response time of under 3 seconds for all user interactions under normal load conditions.

8.2.3 Usability

- **4.3.1** The user interface shall be intuitive and easy to navigate for users with varying levels of technical expertise.
- **4.3.2** The system shall provide clear, concise, and helpful error messages.
- **4.3.3** User interfaces shall be responsive and compatible with different devices and screen sizes, including tablets and smartphones.

8.2.4 Other

- **4.4.1 Performance**: The system shall have an uptime of 99.9%, excluding scheduled maintenance.
- **4.4.2 Compatibility**: The application shall be compatible with the latest versions of major web browsers including Chrome, Firefox, Safari, and Edge.
- **4.4.3 Maintainability**: The codebase shall be well-documented to facilitate easy maintenance and future updates.

8.3 External Interface

8.3.1 User Interface

- **5.1.1 Design and Layout**
  - The application shall have a clean, modern design with a neutral color palette to enhance user experience.
  - The interface shall maintain consistency across different sections for ease of navigation.
- **5.1.2 Standard Buttons and Functions**
  - Each screen shall include standard buttons like 'Home' and 'Log out'.
  - A 'Search' function shall be available on all pages where applicable.
- **5.1.3 Visual Indicators and Tooltips**

    - The application shall use color-coded indicators for different states (e.g., green for approved, red for denied).
    - Tooltips shall be provided for all icons and buttons to guide users on hover or tap.
- **5.1.4 Error Message Display**
    - Error messages shall be displayed in a clear, concise manner, indicating the nature of the error and suggested corrective actions.
    - Error messages shall appear in a consistent location on the screen, preferably near the top or at the center.
- **5.1.5 Responsive Design**
    - The application shall use responsive design to ensure usability across various devices and screen sizes.
    - Screen elements shall dynamically adjust to the resolution and orientation of the device.
- **5.1.6 User Interface Components**
    - User Registration: Interface for inputting personal details with validation feedback.
    - Volunteer Profile Management: Interface with editable fields and submission options.
    - Shift Submission: A form-based interface for entering shift details.
    - Reporting: Interface for selecting parameters and generating reports.

8.3.2 Hardware Interface

- **5.2.1** The primary hardware requirement for an end-user is an external internet connection.
- **5.2.2** Compatibility with standard input devices (keyboard, mouse, touchscreen) is required.

8.3.3 Software Interface

- **5.3.1** The application shall be compatible with the latest versions of major web browsers: Chrome, Firefox, Safari, and Edge.
- **5.3.2** The application shall interface with a cloud-based SQL database for data storage and retrieval.
- **5.3.3** The application shall use RESTful APIs for communication between the front-end and back-end systems.

8.3.4 Communication

- **5.4.1** The application shall support data export features in CSV or PDF formats via email or direct download.
- **5.4.2** HTTP REST API is required for real-time data communication between the front-end client application and back-end server.
- 5.4.3 TCP on port 1433 is required for interface between back-end server and SQL database.

9 Tests

In this part of the report, we lay out the comprehensive testing framework that was essential for ensuring the SP8 Volunteer Manager application's functionality and reliability. It outlines the various testing methods we applied, from unit tests to user acceptance testing, and discusses how each contributed to the refinement of the

application. The results compiled in this test report reflect the application's readiness for deployment and its adherence to the quality standards we set forth at the project's inception. This section serves as both a validation of our application's capabilities and a benchmark for its operational excellence.

9.1 Test Plan

This test plan details the developer-led testing approach for the SP8 Volunteer Manager application. Recognizing the dynamic and fast-paced nature of our development process, this plan prioritizes a flexible and iterative testing methodology, engaging developers directly in the validation of functionalities and system integrity.

9.1.1 Objectives

The primary objectives of this test plan are to:

- Ensure all features meet the defined requirements and function correctly.
- Identify and resolve defects prior to deployment.
- Validate the performance, usability, and security of the application.

9.1.1 Scope

The scope of testing includes:

- All new features and updates to the application.
- Backend integrations and data persistence layers.
- Frontend user interface and user experience elements.
- Security protocols and data protection measures.
- Performance under simulated load conditions.

9.1.1 Responsibilities

Developers are responsible for:

- Conducting tests on their code prior to merging into the main branch.
- Collaborating on peer reviews to identify potential issues.
- Documenting any identified issues and the steps taken to resolve them.
- Participating in regular code integration and testing sessions.

9.1.1 Test Environment

Testing will be conducted in a dedicated development environment that mirrors the production setup as closely as possible. This includes:

- A local testing environment on the developer’s machine.
- A continuous integration (CI) pipeline for automated building and testing.
- Staging environments for integration and performance testing.

9.1.1 Testing Tools

Developers will use a variety of tools for different testing purposes:

- Unit Testing: Frameworks such as JUnit (for Java) or Jest (for React components).
- Integration Testing: Tools like Postman for API testing and Selenium for end-to-end testing.
- Performance Testing: Tools such as JMeter or LoadRunner to simulate user load.
- Security Testing: Static code analysis tools and vulnerability scanners.

9.2 Test Report

Test Case	Expected Outcome	Results
Login with Account	Logs in if an account exists.	Pass
Login without Account	Rejects login if no account is existed.	Pass
Create an Account	Creates an account that the user can log in with.	Pass
Volunteer Audit Message	Sending an audit from a volunteer to the manager	Pass
Volunteer Accept	The audit should change the Database	Pass
Volunteer Decline	The audit should not change the database	Pass
Shift Cancel	The User should be able to cancel a shift	Pass
Shift Accept	The user should be able to accept a shift and it appear on their calendar.	Pass
Shift Search	The user should be able to find the shift they were looking for in the search.	Pass
Delete Account	The admin should be able delete volunteers	Pass
Incorrect Credentials	Incorrect Credentials will stop the user from logging in	Pass

10 Version Control

Version control is a critical system in software development that manages changes to a project's codebase. For the SP8 Volunteer Manager application, the purpose of version control is multifaceted:

Historical Record Keeping:

It provides a complete history of all changes made to the code, including what was changed, why it was changed, who changed it, and when. This historical record is invaluable for understanding the evolution of the codebase over time.

Collaboration:

Version control is essential for team collaboration. With multiple developers working on the project, it helps coordinate efforts, ensuring that changes by one developer don't overwrite changes made by others. It allows team members to work on different features simultaneously without conflict.

Branching and Merging:

The project can maintain multiple lines of development simultaneously using branches. For instance, a branch can be used for new feature development, bug fixes, or experimentation without affecting the main codebase. Once development on a branch is complete and tested, it can be merged back into the main branch.

Release Management:

Version control enables the team to manage releases through tagging specific points in the codebase as release versions. This makes it easier to deploy a particular version of the software or revert to a previous stable state if needed.

Backup:

While not a substitute for a backup system, version control does act as a de facto backup; every check-in serves as a snapshot of the project's state at a point in time, which the team can revert to if necessary.

Accountability:

Every change made in the codebase is attributed to a specific developer. This accountability helps in both tracking contributions and resolving issues that may arise from a particular change.

Streamlining Development Process:

Version control helps in streamlining the development process by facilitating continuous integration and continuous deployment (CI/CD) pipelines. This allows the team to automate the testing and deployment processes, ensuring that any new code submissions do not break the existing application.

For the SP8 Volunteer Manager application, employing a version control system like Git, with repositories hosted on platforms like GitHub or GitLab, is integral. It not only provides the team with the tools needed to manage the application's development effectively but also ensures the stability and continuity of the software development lifecycle.

11 Development

The narrative discussion offers a chronological account of our journey from the SP8 Volunteer Manager application's initial concept to its final realization. This is a story of creativity, collaboration, and learning, highlighting key development milestones and the decision-making processes that shaped them. It provides a behind-the-scenes look at the project's evolution, shedding light on the collaborative efforts of our team and the iterative approach that was instrumental in sculpting the application's final form. Through this narrative, we aim to encapsulate the essence of the project's developmental saga and the collective efforts that brought it to life.

The SP 8 Volunteer Manager Project started with the documentation and idea creation of the project plan. This was a document that presented the project in a small form where we discussed the goals and other features that we would like to see in the application. Once approved the projects next document was the System Requirements Specifications (SRS) this covered what exact features needs to be implemented, such features included login pages, account creations, dashboards, and other necessary features. Once completed the System Design Documentation needed to be created, this is a document that explores the application on a higher level where we discuss the exact tools, frameworks, languages, databases, and more app specific features. These were then submitted to be approved for the development process of the application. This was the majority of the set up for the project, next was to develop and implement these features.

The development process of the application would take around 7-8 weeks to complete. This is having the developers create the application that was described in the previous documents. The application started by spinning up a new GitHub repository and the making that repository an organizational repository. There were documents that helped with converting a regular repository into an organizational repository. These could be found online or just by searching in GitHub for it. Once the GitHub was created, we needed to create the React app itself, this can be done by just typing ‘npm create-react-app’ into the terminal and the designated location. This command (if you have npm installed) will create a template of the application. Just delete the items that won’t be important for the application and start adding your modules. There are many ways to organize your repository, but an organized repository is an organized app, so this step is very important. We separated pages, components, roots, and more modules that made it easy to find parts of the application for fixing or implementing. Once the initial app is created push the changes to the Git and invite everyone. This is where the version control and team collaboration start.

Once everyone accepts invitation, the real progression can start for the application. Each team member was assigned a task, I (Cade Percival) was the team lead and the UI designer and developer, Bryce Moore was the Front-end developer and Thomas Ness was the Back-end developer. Each member picked their own spot for the application because that is what everyone was comfortable with.

UI Development was important for this application because it allows the users to understand the app without having to be taught about the app. Figma was used to help with the creation of the UI because it allows designers to easily create their ideas and then implement the using CSS in React. I would create the initial idea on Figma then once I was satisfied, I would go to the app and Bryce would have created the framework of the application and I would go in and add the CSS. This type of programming was helpful because it allowed the Front-end developer to create the roots and connect the Back-end and Front-end and allow the UI designer to only worry about the design and that is it. This would go on until the application was complete on the front-end.

The Back-end was created by Thomas Ness (Tom), Bryce would give him end points that we needed for our front end and Tom would the create a RESTful API that we can use to get the data that we needed. This was done by adding the java API to the server that was created and the users can use our front end to interact with the backend.

Once the app was done it was able to allow users to sign up and use our application. This means that they can create organizations, schedules, and more. The applications purpose was accomplished and now you can manager your volunteers.

12 Summary

This final report brings together the comprehensive details of the SP8 Volunteer Manager application development journey. It outlines the project's design considerations, architectural strategies, system architecture, detailed system design, challenges faced, narrative discussions, design constraints, requirements, testing strategies, and final conclusions.

The project adopted a user-centric design and agile methodology to ensure flexibility and responsiveness to user feedback. We chose a robust Java API for the backend, React for the frontend, and PostgreSQL for the database, creating a scalable, secure microservices architecture. The system was designed with an API gateway to facilitate communication between services, and each service was developed with specific roles and responsibilities to streamline the volunteer management process.

Challenges throughout the project were met with strategic problem-solving and adaptability. The team employed development testing, avoiding traditional test cases, in favor of a more iterative and dynamic testing approach integrated into the development cycle.

Version control played a vital role in the project, utilizing Git for historical record-keeping, collaboration, branching, merging, release management, and as a backup measure. It ensured a smooth and coordinated workflow, even with a distributed development team.

The project's success was underpinned by the adherence to design principles and the efficient resolution of engineering trade-offs. Throughout the development, careful consideration was given to balancing the needs for rapid development, high performance, security, and a seamless user experience.

In summary, this final report is not only a testament to the technical accomplishments of the SP8 Volunteer Manager application but also a reflection of the collaborative spirit, technical proficiency, and methodical approach that the development team maintained throughout the project's lifecycle.

13 Appendix

13.1 Appendix

JSON Web Token (JWT): An open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

Microservices Architecture: A design approach that structures an application as a collection of loosely coupled services, which implement business capabilities. Each service is self-contained and should implement a single business function.

API (Application Programming Interface): A set of protocols, routines, and tools for building software applications, specifying how software components should interact.

RESTful API: An architectural style for designing networked applications, relying on stateless communication and standard HTTP methods to facilitate interactions between clients and servers.

Spring Boot: A Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production-ready spring applications.

PostgreSQL: An open-source, object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

Docker: A platform as a service (PaaS) product that uses OS-level virtualization to deliver software in packages called containers.

Kubernetes: An open-source system for automating deployment, scaling, and management of containerized applications.

CI/CD (Continuous Integration/Continuous Deployment): A method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous deployment, and continuous delivery.

OAuth: An open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.

Single Page Application (SPA): A web application or website that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from the server.

User-Centric Design: An iterative design process in which designers focus on the users and their needs in each phase of the design process.

WebSocket: A computer communications protocol, providing full-duplex communication channels over a single TCP connection.

Scalability: The capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth.

Containerization: A lightweight alternative to full machine virtualization that involves encapsulating an application in a container with its own operating environment.

Agile Methodology: An approach to project management and software development that emphasizes incremental delivery, team collaboration, continual planning, and continual learning, instead of trying to deliver it all at once near the end.

13.2 Training

13.2.1 Cade Percival (Team Lead)

I completed the React Native tutorial. - Cade Percival

13.2.1 Bryce Moore (Front-End Dev)

I completed the React Native tutorial. - Bryce Moore

13.2.1 Thomas Ness (Back-End Dev)

I completed the React Native tutorial. - Tom Ness