GCISLFullStackApp

Final Report

Sponsor: WSU Granger Cobb Institute for Senior Living (GCISL)

Mentor: Parteek Kumar

Team: StackD

Members:

Justin Keanini,

Teni Olugboyega,

Naomi Dion-Gokan

CptS 423 Software Design Project II

Spring 2025

TABLE OF CONTENTS - (Estimated)

I. Introduction

The Granger Cobb Institute for Senior Living (GCISL) at Washington State University focuses on research, education, and community engagement in senior living. As part of this mission, GCISL uses gciConnect, a web-based volunteer report tracking system designed to help faculty, staff, and students manage volunteer reports that contribute to ongoing research.

Despite its usefulness, gciConnect faces usability, reliability, and administrative challenges that impact its efficiency. The goal of this project is to enhance the system's user interface, ensure its reliability through testing and logging, and improve administrative capabilities, ultimately creating a more intuitive and effective tool.

The following is our Client and Mentor information:

Client: Washington State University's Granger Cobb Institute for Senior Living (GCISL)
Client Contact: Mrs. Darcie Bagott, Program Specialist
Mentor: Parteek Kumar

II. Team Members & Bios

Justin Keanini is a senior at Washington State University pursuing a degree in Computer Science. With a strong focus on frontend development, he has experience building responsive and user-friendly interfaces using HTML, CSS, and JavaScript. He brings a solid foundation in programming, gained through coursework in C/C++/C#, and certifications in Python and R for data analysis and visualization. For this project, he expanded his skills by working with React for dynamic UI development and MongoDB for backend data integration. Justin contributed to designing and implementing intuitive user interfaces while collaborating with the team to integrate and refine system features, ensuring the application meets both functional and usability goals.

Teni Olugboyega is a senior at Washington State University majoring in computer science graduating in Spring 2025. He has an interest in full-stack development, Artificial intelligence and Machine Learning. He has a strong background in Java, React and MERN full stack development which is essential for the completion of the project. For this project Teni played a role in constructing visually appealing pages that follows the clients specified theme, connecting frontend pages to the backend and database to make use of data stored ,and retrieve to add more definition to some pages .

Naomi Dion-Gokan is a senior at Washington State University, majoring in Computer Science and set to graduate in Spring 2025. She is passionate about web development, especially front-end engineering, and is excited to expand her skills into back-end work with the goal of becoming a full-stack software engineer. Naomi has developed expertise in C#, React, JavaScript, and SQL, particularly in building scalable full-stack solutions—skills she earned over two years during her internship. For this project, Naomi's responsibilities include setting up the

database, managing deployment, and facilitating team communication. Her technical toolkit also includes HTML, CSS, C++, and REST APIs, giving her a strong foundation for both project organization and technical implementation.

III.Project Requirements Specification

III.1. Project Stakeholders

Our Primary Stakeholder is GCISL (faculty, staff, and students) and our Secondary Stakeholders are the Senior living residents and volunteers who interact with the system. The key needs areImproved usability, enhanced reliability, and a more robust administrative portal.

III.2. Use Cases

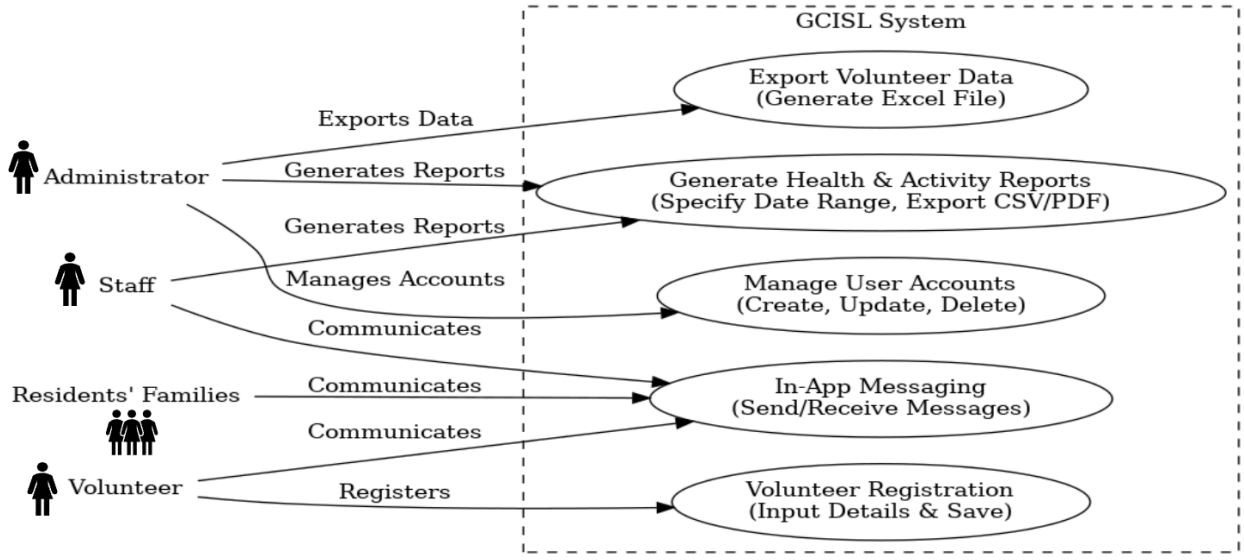


Figure 1: Use Case Diagram for gciConnect System

Shows the primary interactions between actors (e.g., administrators, staff, volunteers) and the system's functionalities, including user management, report generation, in-app messaging, and volunteer registration.

Use Case 1	
Actors	Administrator
Description	An administrator manages user accounts by creating, updating, and deleting profiles. This ensures that only authorized individuals can access the system, and their

	information is always up to date.
Preconditions	The user is logged in as an administrator and has access to the User Management Module.
Main Flow	The administrator selects the “Manage Users” option from the admin dashboard. The administrator selects “Add New User” and inputs required details such as name, contact information, and role. The administrator can also select an existing user, update their information, or delete the user from the system. The system saves the changes and updates the user database.
Post Conditions	The system updates the user list, reflecting the newly added, updated, or deleted profiles.
Exceptions	If the user does not have the required admin access, the system denies access to user management functionality.
Related Requirements	User Management Module, Requirement 1: User profile management (creation, updating, deletion). User Management Module, Requirement 2: Role-based access control.

Use Case 2	Generating Resident Health and Activity Reports
Actors	Administrator, Staff
Description	An administrator or staff member generates detailed reports of resident health and activity based on data collected over time.

	This helps in tracking trends and making informed decisions about care.
Preconditions	The administrator or staff has access to the Data Analysis and Reporting Modules.
Main Flow	The user selects the “Generate Report” option from the reporting dashboard. The user selects a specific resident or group of residents, and specifies the date range and type of data (health or activity). The system compiles the data and generates the report. The user can view the report on-screen or choose to export it as a CSV or PDF file.
Post Conditions	The report is displayed and optionally exported in the selected format.
Exceptions	If no data is available for the selected parameters, the system displays an error message.
Related Requirements	Data Analysis and Reporting Module, Requirement 1: Generate resident health and activity reports. Data Analysis and Reporting Module, Requirement 2: Export reports in CSV or PDF.

Use Case 3	Volunteer Registration
Actors	Volunteers, Administrator
Description	Volunteers input their personal information through a form on the website, enabling GCISL to maintain an updated volunteer database.

Preconditions	The volunteer has access to the Volunteer Management Module.
Main Flow	The volunteer accesses the registration form from the system's website. The volunteer enters their name, contact details, availability, and other requested information. The system validates the input and saves the volunteer's information to the database. Administrators can access the information through the volunteer management dashboard.
Post Conditions	The volunteer's information is stored in the system and accessible to administrators.
Exceptions	If the form is incomplete or contains errors, the system prompts the user to correct the input.
Related Requirements	Volunteer Management Module, Requirement 1: Input volunteer information.

Use Case 4	Exporting Volunteer Data
Actors	Administrator
Description	The administrator exports the volunteer database into an Excel file to analyze the data or share it with external parties.
Preconditions	The administrator has access to the Volunteer Management Module.
Main Flow	The administrator selects "Export Volunteer Data" from the admin dashboard. The system generates an

	Excel file with the details of all registered volunteers. The administrator downloads the file for further use.
Post Conditions	The Excel file is downloaded, containing the latest volunteer data.
Exceptions	If there is a system error during export, the system displays an error message and logs the issue.
Related Requirements	Volunteer Management Module, Requirement 2: Export volunteer information in Excel format.

III.3. Functional Requirements

1. User Management Module

Requirement 1: The system must allow for the creation, updating, and deletion of user profiles.

Description	The system will provide an interface for administrators to manage user accounts. This includes adding new users, editing their information (such as name, contact details, and role), and deleting users when they are no longer part of the system. This functionality is essential to ensure that only authorized users have access to the system and that user information remains accurate.
Source	GCISL
Priority	Priority Level 0: Essential and required functionality

Requirement 2: The system must support different user roles with corresponding access levels.

Description	The system will differentiate between user roles, such as administrators, staff, and volunteers. Each role will have specific permissions regarding what they can view and modify in the system. For example, administrators will have full access, while volunteers may only be able to update their own information. This ensures secure access control based on user responsibility.

Source	GCISL
Priority	Priority Level 0: Essential and required functionality

2. Data Analysis and Reporting Module

Requirement 1: The system must generate reports based on resident date, including health and activity information.

Description	The system will collect and organize resident data (e.g., health reports, activity logs) and allow administrators to generate detailed reports. These reports will help staff monitor resident health trends and daily activities, enabling better decision-making and care planning.
Source	GCISL
Priority	Priority Level 1: Desirable functionality

Requirement 2: The system must allow exporting of reports in CSV and PDF formats.

Description	The system will include a feature for exporting generated reports into widely used file formats, such as CSV and PDF. This will allow administrators to easily share data with external parties, such as healthcare providers, or for offline record-keeping.
Source	GCISL
Priority	Priority Level 0: Essential and required functionality

3. Communication Module

Requirement 1: The system must support in-app messaging between staff and stakeholders.

Description	The system will provide a messaging feature that enables real-time communication between staff members and external stakeholders (such as volunteers or residents' families). This will streamline internal communication and reduce the need for external messaging platforms, improving the efficiency of communication workflows.
Source	Internal requirements elicitation among members of the team
Priority	Priority Level 2: Extra features or stretch goals

4. Volunteer Management Module

Requirement 1: The system must provide a form on the website for volunteers to input their information, including name, email, and contact details.

Description A dedicated web form will be available for volunteers to register their details, including personal contact information and their availability for volunteering. This will allow GCISL to maintain an organized database of volunteers, ensuring that they can be contacted and scheduled for tasks easily.

Source	GCISL
Priority	Priority Level 0: Essential and required functionality

Requirement 2: The system must allow administrators to export the collected volunteer information into an Excel file.

Description	Administrators will have the ability to export the volunteer database into an Excel file, providing flexibility for data analysis, reporting, and manual editing. This feature ensures that volunteer data can be easily shared with other departments or used for event planning and coordination.
Source	GCISL
Priority	Priority Level 0: Essential and required functionality

6. Other Modules

System Reliability Improvements

Description	Implement tests to check for errors and ensure continuous functionality during updates.
Source	Developer best practices
Priority	Priority Level 0: Essential

Error Logging:

Description	Add error tracking and logging for rapid debugging by administrators.
Source	GCISL administrators
Priority	Priority Level 0: Essential

III.4. Non-Functional Requirements

Reliability
The system shall be reliable and operational most of the time. It must handle user requests without crashing or becoming unresponsive. This should be GCISL's requirement for a stable system to support ongoing research and volunteer tracking.

Response Time
The system shall process and respond to user actions such as login, task assignments within a few seconds. For larger operations, like generating reports, the system shall complete the task within less than 10 seconds. Usability needs for staff and volunteers.

Scalability
The system shall support scalability, accommodating a large number of concurrent users without degradation in performance. This ensures that future expansions to other institutions or larger datasets will be manageable. It would be GCISL's potential expansion to other senior living institutions.

Usability
The system shall be intuitive and easy to use, requiring no more than a few hours of training for an administrator or volunteer to perform basic tasks such as assigning tasks, submitting volunteer reports. The need for accessibility and ease of use for staff, volunteers, and senior residents.

IV. Software Design - From Solution Approach

This section should describe the final design of your software system.

Include sections III, IV, and V from your revised "Solution Approach" document. Please review these sections and make the necessary updates to reveal the changes in your project design since your revision.

IV.1. Architecture Design

Revise and include Section II from your Solution Approach report here. Provide the block diagram of your architecture and give a brief description of it.

IV.1.1. Overview

Our team has adopted a **component-based architectural pattern** for the gciConnect platform to support the volunteer management and research efforts of the Granger Cobb Institute for Senior Living (GCISL). This pattern ensures modularity, allowing each subsystem to function independently while seamlessly interacting with others. The platform's architecture consists of three layers: User Interface Layer, Backend Layer, and Data Layer. Each layer plays a specific role in ensuring the platform remains scalable, maintainable, and reliable.The User Interface Layer handles interactions between users and the system. The **UI Handler** manages layouts, page transitions, and user navigation, ensuring an accessible experience for all users. Working alongside the UI Handler, the **Inputs Handler** validates input data, such as form submissions, and routes it to the appropriate backend subsystems.

The Backend Layer manages system logic. The User Management Handler oversees accounts and permissions, ensuring proper access to features. The Volunteer Management Handler assigns tasks to volunteers and tracks their activities, sending status updates to the Reporting Handler for logging and to the Communication Handler for notifications. The Communication Handler sends notifications and logs inquiries received from the Contact Us page.The Data Layer ensures efficient data storage and retrieval. The Database Handler stores key data in MongoDB Atlas and logs inquiries in Google Sheets. This combination allows for both structured data storage and real-time tracking of communication.

Below is Figure 2 showing the system's structure. This design will allow future development without disrupting other components. In the next sections, detailed descriptions of the individual subsystems, algorithms, and services will be provided.

Subsystem Decomposition

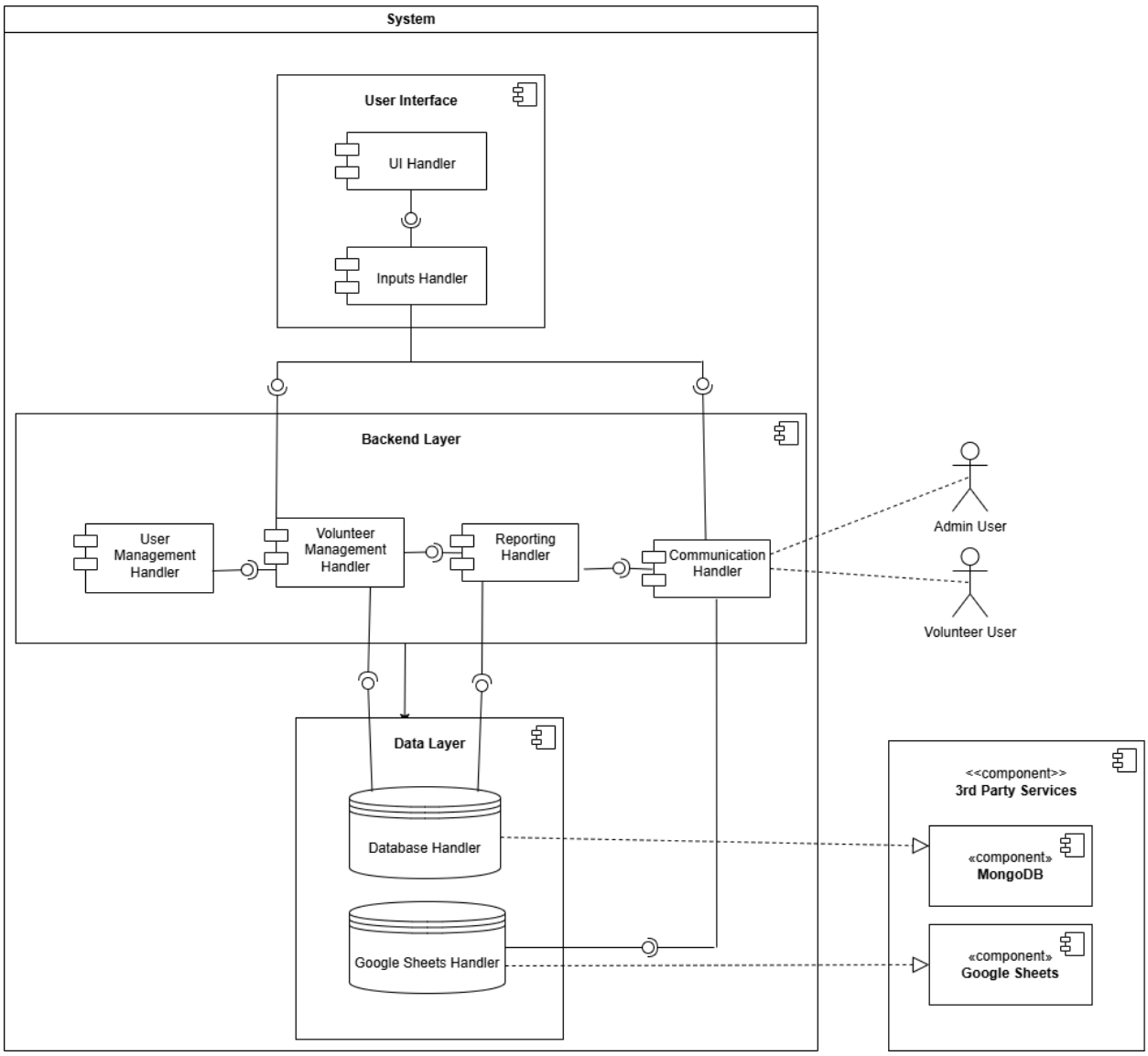


Figure 2: System Architecture of gciConnect Platform

Illustrates the interactions between the User Interface, Backend, and Data layers, showcasing how tasks, data, and notifications are managed.

IV.1.2.1 [User Management Handler]

a) Description

The User Management Handler manages user accounts, roles, and permissions. It ensures that only authorized users can access specific features, maintaining security and data integrity.

b) Concepts and Algorithms Generated

This subsystem applies Role-Based Access Control (RBAC) to manage different user roles (e.g., volunteers, staff, and administrators) and uses session management algorithms to validate user access during login.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
User Account Management	Volunteer Management Handler	Provides user account data (such as roles and permissions) to facilitate the assignment of tasks to volunteers.
Role Verification	Communication Handler	Verifies roles and permissions to ensure notifications and messages are routed to the appropriate administrators and users.

Services Required: None (self-contained).

IV.1.2.2 [Volunteer Management Handler]

a) Description

The Volunteer Management Handler manages volunteer tasks and engagement, ensuring that volunteers are assigned tasks and their activities are tracked.

b) Concepts and Algorithms Generated

This handler implements a task scheduling algorithm to assign tasks based on volunteer availability and a tracking system to log hours and monitor task progress.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
Task Management and Tracking	Reporting Handler	Sends task completion data and volunteer activity logs to the Reporting Handler for record-keeping and report generation.
Status Notification	Communication Handler	Sends updates on task status to the Communication Handler to notify administrators and relevant users about task progress or issues.

Services Required:

Service Name	Service Provided From
Account and Role Management Description: Uses user data to assign tasks to volunteers.	User Management Handler

IV.1.2.3 [Communication Handler]

a) Description

The Communication Handler manages notifications and internal communication, ensuring administrators are informed of important updates or new submissions through the Contact Us page.

b) Concepts and Algorithms Generated

This handler applies notification algorithms to send automated emails to administrators and stores message logs in Google Sheets through integrated forms.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
Notification Management	Admin Users via Email Notifications	Sends notifications upon task updates or message submissions and stores logs in Google Sheets.

Services Required:

Service Name	Service Provided From
Account and Role Management Description: Ensures notifications are routed to the correct administrators.	User Management Handler

IV.1.2.4 [Reporting Handler]

a) Description

The Reporting Handler aggregates data from other subsystems and generates reports for administrators. It allows exporting of volunteer data to Excel for analysis and sharing.

b) Concepts and Algorithms Generated

This handler uses data aggregation algorithms to compile logs and export functionality to generate downloadable Excel files for offline use.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
Data Reporting and Export	Admin Users	Aggregates volunteer activity logs and allows exporting to Excel for reporting purposes.

Services Required:

Service Name	Service Provided From
Task Management and Tracking Description: Uses volunteer data to generate reports.	Volunteer Management Handler

IV.1.2.5 [Database Handler]

a) Description

The Database Handler manages persistent storage for the platform using MongoDB Atlas and Google Sheets integration. It ensures seamless storage and retrieval of data across all subsystems.

b) Concepts and Algorithms Generated

MongoDB is used to store users, tasks, and volunteer logs, while Google Sheets stores Contact Us submissions. The handler ensures consistent data flow between the frontend and backend.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description

Data Storage	Volunteer Management Handler	Stores volunteer activity data and task assignments to ensure seamless management and updates across the system.
Data Retrieval	Reporting Handler	Provides access to stored task and activity logs to generate reports for administrators. Also, export data into Excel files.

Services Required: None (directly integrated with other handlers).

IV.1.2.6 [UI Handler]

a) Description

The UI Handler manages all user-facing elements, including page layouts, icons, and navigation. It ensures that the user interface is responsive and intuitive for all users.

b) Concepts and Algorithms Generated

The UI Handler applies dynamic layout algorithms to adjust the interface for different devices and users. It also handles navigation flow across various sections of the platform.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
Layout Management	Inputs Handler	Provides layout updates to align input elements and navigation flow.

Services Required: None (interacts directly with the frontend interface).

IV.1.2.7 [Inputs Handler]

a) Description

The Inputs Handler processes data input from users, such as form submissions and interactive elements. It ensures that data is validated and routed to the appropriate handlers.

b) Concepts and Algorithms Generated

This subsystem applies input validation algorithms to ensure correct data entry and routes the inputs to backend services such as task assignment or notifications.

c) Interface Description

Services Provided:

Service Name	Service Provided To	Description
Input Processing	Communication Handler	Task Input Routing
Task Input Routing	Volunteer Management Handler	Processes input related to task assignments and routes it to the Volunteer Management Handler for scheduling and tracking.

Services Required:

Service Name	Service Provided From
Layout Management Description: Uses layout updates to ensure inputs are aligned with the user interface.	UI Handler

IV.2. Data design

For this application, two primary data structures are utilized, both interacting with external subsystems: MongoDB and Google Sheets. Each plays a crucial role in data management and retrieval for different parts of the system.

MongoDB Data Structures

MongoDB serves as the primary database for managing system users, volunteer profiles, activities, and reports. The database is structured around collections that store documents in a flexible, JSON-like format. Below is an overview of the key collections and their internal data structures:

userId	Unique identifier for each user.
name	Full name of the user.
email	Email address for communication.
role	User role determining access level.
contact	Phone number or other contact details.
createdAt / updatedAt	Timestamps for tracking the user's record creation and updates.

User Collection

This collection stores user accounts, including administrators, staff, and volunteers, with role-based access control

Google Sheets Data Structure

Google Sheets serves as a complementary tool for Contact Us form submissions and volunteer activity tracking. It allows administrators to access certain data quickly and interact with it via familiar spreadsheet interfaces.

Contact Us Submissions Sheet

Stores information submitted by volunteers via the contact form.

Name	Full name of the person submitting the form.
Email	Contact email.
Message	The content of the message or inquiry.
Date	Date of submission.
Usage:	Admins access this sheet to review submissions and respond directly via email if required. This data is not stored long-term in MongoDB, as it mainly serves as a communication channel.

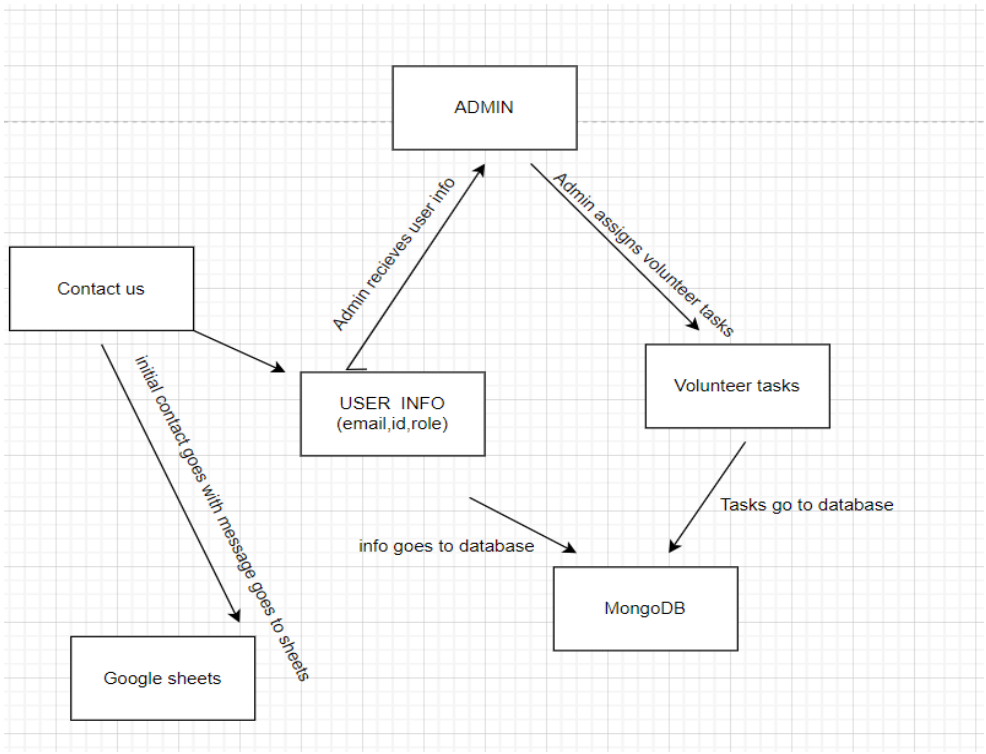


Figure 3: Data Design and Management Framework

Illustrates the interaction between MongoDB and Google Sheets for data management. MongoDB handles primary system data (e.g., users, volunteer profiles, and activity logs), while Google Sheets is used for temporary storage and quick access to Contact Us submissions and volunteer tracking summaries.

IV.3. User Interface Design

The GCISL Full Stack Application features a user interface designed to be intuitive, accessible, and user-friendly for administrators, staff, volunteers, and other stakeholders. The UI focuses on simplicity and efficiency, ensuring that users can navigate the system and perform essential tasks with minimal effort. Each page has been designed with accessibility and clarity in mind, ensuring that users of all technical backgrounds, particularly those in the senior living community, can easily interact with the system.

Home Page:
The Home Page serves as the entry point for all users, providing a brief overview of the system's features and quick access to important sections. It includes:

Navigation Bar: Links to the main sections of the website, such as Get Involved, Contact Us, About, Login, and Register.
Welcome Message: A brief introduction to GCISL's mission and purpose.

Featured Opportunities/News Feed: A section highlighting current volunteering or research involvement opportunities, with quick links to more detailed information.

Image 1: Home Page

Get Involved Page:
The Get Involved Page serves as the primary hub for volunteers and community members looking to contribute to GCISL activities. It includes three main sections:

Research Involvement: Provides opportunities for volunteers to participate in ongoing research projects.
Education and Mentorship: Lists opportunities for users to contribute through educational programs and mentorship roles.
Outreach and Charitable Contributions: Allows users to engage in charitable activities and community outreach programs.

This page is designed to be engaging and easy to navigate, with large, interactive buttons and concise descriptions to guide users through available opportunities. Users can quickly explore the roles they are interested in and sign up with minimal effort.

Contact Us Page:
The Contact Us Page is designed to streamline communication between users and the administration. It includes:

Use Case: Contact Us
Description: Users submit inquiries or feedback to GCISL administration.
Interface Interaction:

Users fill out the contact form on the Contact Us Page with their name, email, and message. After submitting the form, the message is sent to administrators, and users receive an automated confirmation.
This page is simple enough for users of all technical skill levels to navigate, with a clean and straightforward layout. It ensures that users can easily get in touch with GCISL staff for any inquiries or issues they may have.

About Page:
The About Page provides an overview of the GCISL organization, its mission, and its past activities. The page includes:

FAQ (Frequently Asked Questions): Answers to common questions regarding the organization and its services.
Picture Gallery: A visual gallery of images from past events and activities at GCISL.

This page is visually appealing and informative, designed to give visitors a comprehensive understanding of the organization's purpose and initiatives. The combination of text and images ensures a user-friendly and engaging experience.

Login Page:
The Login Page provides a straightforward mechanism for users to access the system. It includes:

Use Case: User Login
Description: Users log into their accounts to access personalized features and services.
Interface Interaction:
Users navigate to the Login Page, where they enter their username and password.
The system authenticates the credentials and redirects users to the Home Page.
This page has been designed with accessibility in mind, featuring large text fields and a clear error message system that provides immediate feedback in case of invalid login attempts. The design uses bold, easily readable fonts to ensure that the login process is seamless for all users.

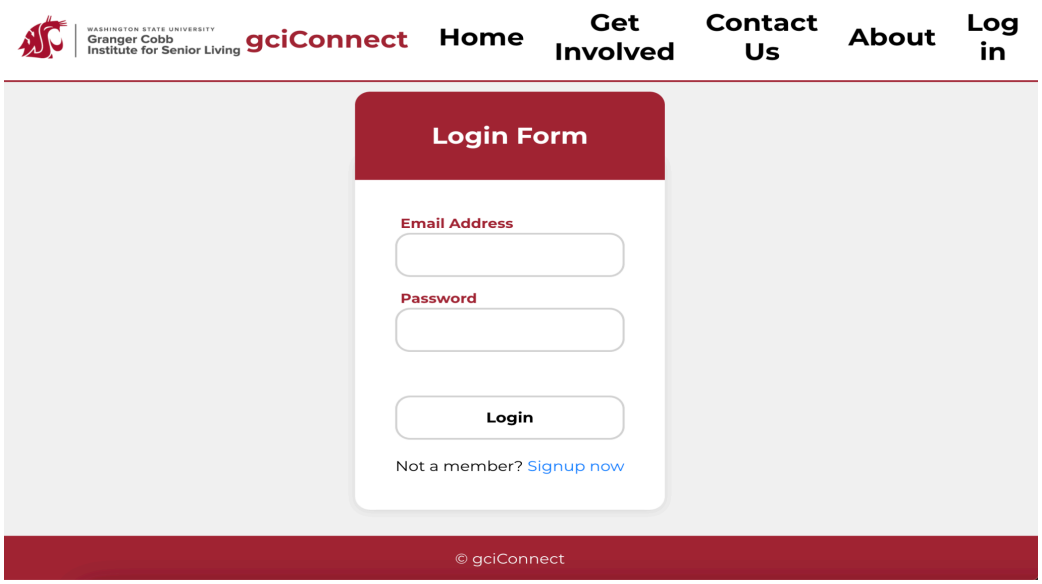


Image 2: Login Page

Register Page:
The Register Page allows new users to create an account and join the GCISL community. It includes:

Use Case: User Registration
Description: New users can create an account to join the GCISL community.
Interface Interaction:
Users access the Register Page to fill out personal information fields (name, contact details, password).
Upon submission, the system creates a user profile and sends a confirmation email.
The registration process is designed to be quick and easy, with large input fields and clear instructions. The page is simple, ensuring that new users can sign up without encountering difficulties.

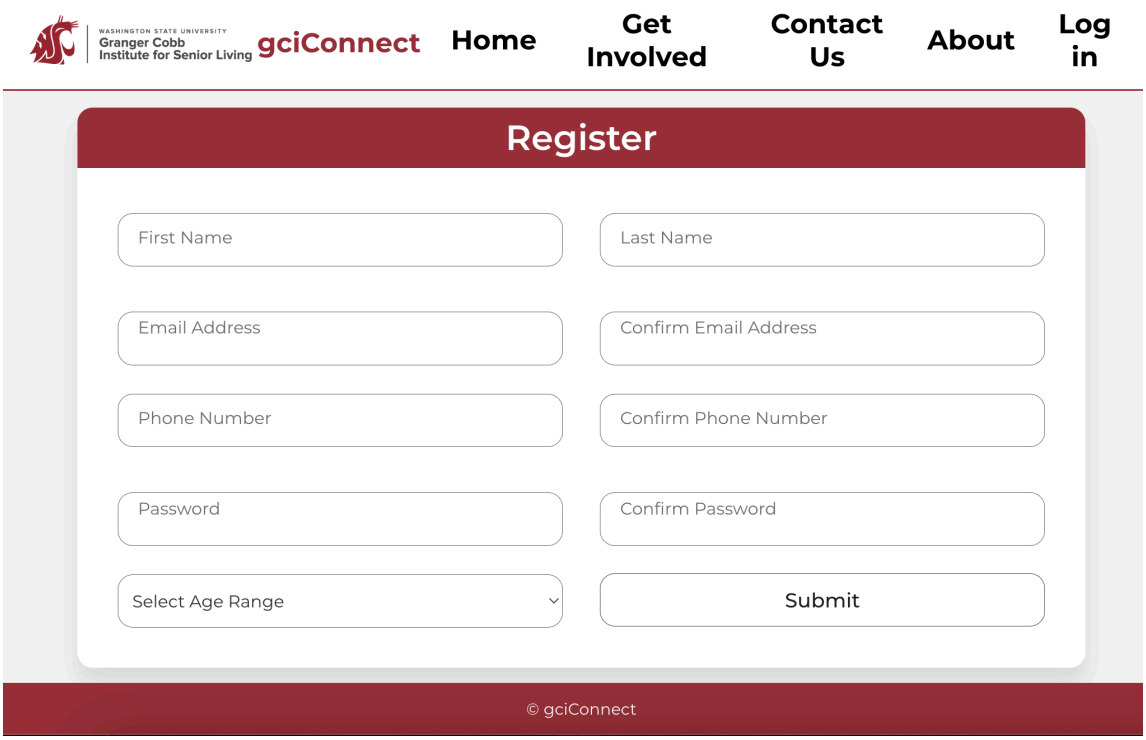


Image 3: Register Page

Admin View Page:
The Admin View Page is the central hub for administrators to manage volunteers, tasks, and reports within the GCISL system. After logging in, administrators are presented with a dashboard overview of key activities and controls for managing the platform. The page is designed to provide a clear and concise overview of all admin functions, ensuring efficient workflow and management.

Use Case: Admin Task Management
Description: Admins can view, assign, and manage tasks for volunteers, ensuring efficient allocation of resources.
Interface Interaction:
Admins access a list of all current tasks, with options to assign or reassign tasks to specific volunteers.
They can add new tasks with relevant descriptions and deadlines, as well as mark tasks as complete or update their statuses.
The page provides a summary of active, completed, and pending tasks, making it easy for admins to track volunteer contributions and project progress.

Use Case: Volunteer Reporting
Description: Admins can generate detailed reports on volunteer activity and task completion to assess engagement and productivity.
Interface Interaction:

The Admin View Page includes options for filtering reports by time period, volunteer, or task type.
Reports can be exported in various formats (e.g., CSV, PDF) to allow for offline analysis and sharing with other stakeholders.

Use Case: System Notifications
Description: Admins receive real-time notifications about critical system events, such as new volunteer registrations or task completion updates.
Interface Interaction:
Notifications appear as alerts on the Admin View dashboard, allowing admins to quickly address any issues or take action on pending items.
This feature ensures that admins are kept up-to-date with important activities within the system, improving response times and overall efficiency.

Accessibility and Usability:
The GCISL Full Stack Application is designed with accessibility in mind to meet the specific needs of the senior living community. Key features include:

High Contrast and Large Fonts: These design elements ensure that the interface is easy to read, even for users with visual impairments.
Clear Navigation: Simple, intuitive navigation ensures that users can move between different sections of the site without confusion.
Feedback and Error Handling: The system provides real-time feedback through error messages and confirmation prompts to guide users through tasks and prevent mistakes.

V. Test Case Specifications and Results

V.1. Testing Overview

Our testing strategy for the GCISL project will include fully automated tests for core functions, making sure they run smoothly through a Continuous Integration (CI) pipeline. Additionally, we'll use Continuous Delivery (CD) for easier and automated deployment processes. The key areas for testing include user authentication, CRUD operations on tasks and user data, and role-based access control. Non-core functions may also be tested, although they might not be prioritized due to time limits. Our testing process is divided into two main parts: the developer testing process and the playtesting process.

Developer Testing Process

1. Developer Writes Code: The process begins with adding the feature or fixing a bug. The implementation is considered complete when the feature achieves the required functionality.
2. Determine Test Cases: For each core function, at least two test cases will be created: one for expected behavior and one for unexpected or invalid behavior. Non-essential functions might include only test cases for expected behavior.
3. Run Tests: The developer runs all tests, including newly added ones, ensuring the new code doesn't break existing functions.
4. Fix Issues: If any tests fail, the developer finds and fixes the issues, rerunning tests until all pass.
5. Developer Pushes Code to Remote: The developer pushes their code to the remote repository. The code will be run through a CI pipeline, showing the results. Only branches that pass all tests in the CI are allowed to merge.
6. Developer Makes a Pull Request Against Main: The developer creates a pull request against the main branch and adds at least one reviewer. The branch cannot be merged until approved by the reviewer(s).
7. Merge Branch: Once approved, the branch is merged into the main branch. If a Continuous Delivery (CD) pipeline is set up, it will deploy a new release if needed.

Playtesting Process

1. Create Playtest Build: A developer creates a playtest build, either fresh or with modified content to focus on specific test areas.
2. Deliver Build & Allow Time for Testing: The build is delivered, allowing several weeks for thorough testing.
3. Deploy Feedback Forms: Testers are asked to fill out feedback forms about their experience, including enjoyment, learning, and retention.
4. Collect Feedback Data: Feedback is gathered and analyzed for insights on improvements.
5. Make Any Necessary Changes: Based on feedback, necessary changes to content, UI, or user experience are made for the next version.

Not using CI/CD would mean relying on manual testing and deployment, which is time-consuming, prone to errors, and less efficient. For a project aiming for continuous improvement and quick updates, CI/CD is essential. By implementing this testing strategy, we

can ensure strong, reliable, and efficient testing and delivery of all project components for the GCISL project.

V.2. Environment Requirements

Our testing environment is predominantly self-contained, utilizing modern web development tools compatible with both backend and frontend testing. Testing will be conducted through local environments as well as through the GitHub CI/CD pipeline for consistency and efficiency.

For testing purposes, unit and integration tests will utilize Jest and React Testing Library for JavaScript modules and React components, respectively. Additionally, Supertest (Node.js library used for testing HTTP endpoints – for API testing) will be used for testing backend API endpoints to ensure expected interactions with MongoDB.

If available, the team will incorporate GitHub Actions for continuous integration (CI) to automate tests on each commit and merge, guaranteeing that tests are consistently run and passed before changes are integrated into the main branch.

There are no specific hardware requirements.

V.3. Test Results

Aspect being tested	Expected Result	Observed Result	Test Result	Test Case Requirements
User login with valid credentials	The user logs in successfully, receiving a token.	The login succeeds, and a token is generated.	Pass	A test user must be created before executing authentication-based test cases. The test user should have: email: testuser@example.com password: password123 A valid JWT token should be obtained via login and used in authenticated requests.
Login with incorrect email	The login attempt fails with an "Invalid email or password" error.	Error message: "Invalid email or password."	Pass	A test user must be created before executing authentication-based test cases. The test user should have:

				email: testuser@example.com password: password123 A valid JWT token should be obtained via login and used in authenticated requests.
Login with incorrect password	The login attempt fails with an "Invalid email or password" error.	Error message: "Invalid email or password."	Pass	A test user must be created before executing authentication-based test cases. The test user should have: email: testuser@example.com password: password123 A valid JWT token should be obtained via login and used in authenticated requests.
Login with missing password	The API should return a 400 error with a message indicating required fields.	The server returned a 500 error due to an illegal argument in bcrypt.compare	Fail	A test user must be created before executing authentication-based test cases. The test user should have: email: testuser@example.com password: password123 A valid JWT token should be obtained via login and used in authenticated requests.
Registering a new user	The user should be created successfully.	User registered with message: "User registered successfully! "	Pass	N/A

Registering with an existing email	The API should return with "Email is already in use"	The API returned a different error message: "Email is already in use."	Pass	A test user with an existing account and the same email.
Registering with missing fields	The API should return a 400 error stating required fields are missing.	Error was returned.	Pass	N/A
Creating a task	The API should return 201 and create a task with the provided details.	The API returned 400 due to missing creationDate validation.	Fail (Bug: creationDate field is required but missing)	Task creation tests require a properly formatted request body containing: title: 'Test Task' duration: '01/01/2023 - 01/31/2023' document: 'test-document.pdf' color: 'blue' status: 'None'
Updating a task	The API should update the task and return 200	The API returned 405 (Method Not Allowed)	Fail (Bug: HTTP method not supported or incorrect routing)	Task creation tests require a properly formatted request body containing: title: 'Test Task' duration: '01/01/2023 - 01/31/2023' document: 'test-document.pdf' color: 'blue' status: 'None'
Deleting a task	The API should delete the task and return 204	The API returned 405 (Method Not Allowed)	Fail (Bug: Incorrect method handling	Task creation tests require a properly formatted request body containing:

			for deletion)	title: 'Test Task' duration: '01/01/2023 - 01/31/2023' document: 'test-document.pdf' color: 'blue' status: 'None'
Clearing all assignees from a task	The API should return 200 and remove all volunteers from a task.	The API returned 400 (Bad Request).	Fail (Bug: Issue in clearing task assignees)	At least one volunteer user must be registered and assigned to a task before clearing/removing them. The volunteer should be assigned to a valid taskId before calling the /tasks/clear or /tasks/remove endpoints.'
Removing a volunteer from a task	The API should return 200 and remove the specified volunteer.	The API returned 400 (Bad Request).	Fail (Bug: Issue in removing a volunteer from a task)	At least one volunteer user must be registered and assigned to a task before clearing/removing them. The volunteer should be assigned to a valid taskId before calling the /tasks/clear or /tasks/remove endpoints.

VI. Projects and Tools used

Tools/ Library/ Framework	Purpose
Git	Used for cloning the repository and version control.
Node.js & npm	Required for running both frontend and backend, managing dependencies.
Express.js	Backend web framework for building APIs and handling server-side logic.
MongoDB	Databases used for storing application data, can be hosted locally or on MongoDB Atlas.
Mongoose	ODM (Object Data Modeling) library for MongoDB, providing schema-based structure to data.
Bcrypt	Used for hashing passwords, ensuring secure storage of user credentials.
JSON Web Token (JWT)	Securely transmits authentication information between frontend and backend.
CORS	Middleware to enable Cross-Origin Resource Sharing, allowing frontend-backend communication across different origins.
dotenv	Loads environment variables from a .env file for managing sensitive information securely.
serverless-http	Wraps the Express app to run as serverless functions, suitable for Vercel deployment.
React	Used to build the interactive and dynamic user interface, ensuring a responsive and efficient frontend.

Languages Used in Project			
HTML5	JavaScript	CSS	MongoDB Query Language (MQL)

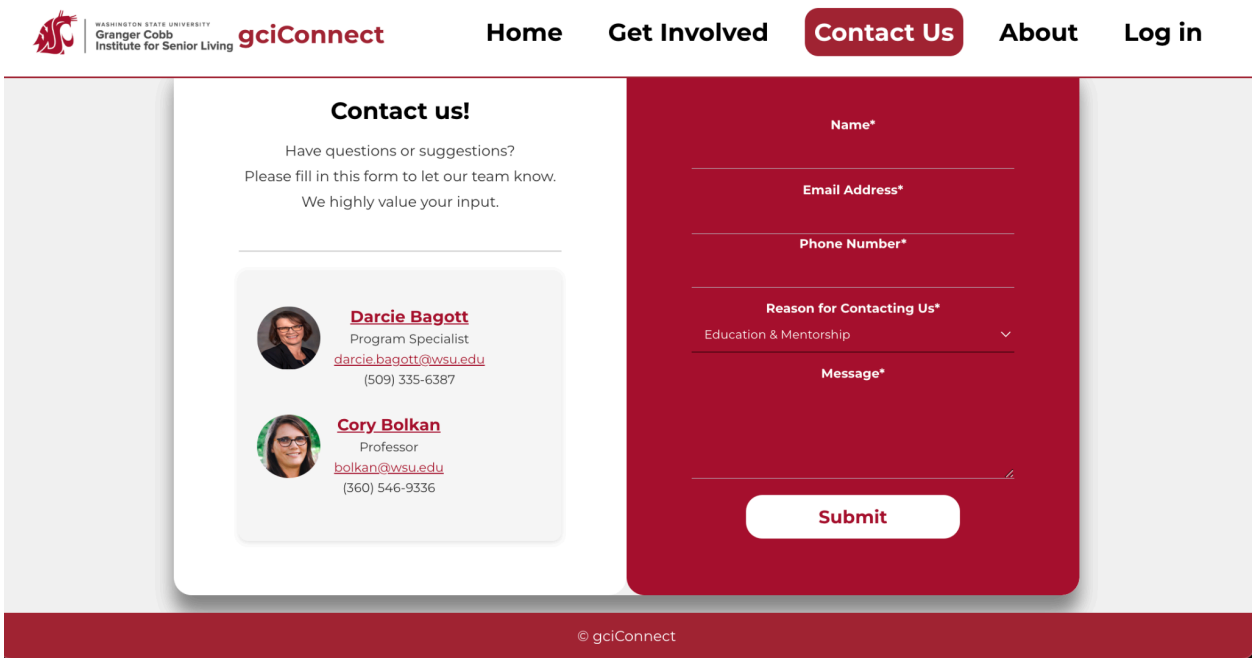
VII. Description of Final Prototype

The final prototype o**f gciConnect co**nsists of a web-based platform designed to facilitate communication, research engagement, and volunteer task management. It provides features accessible to both general visitors and authenticated users (administrators and volunteers).

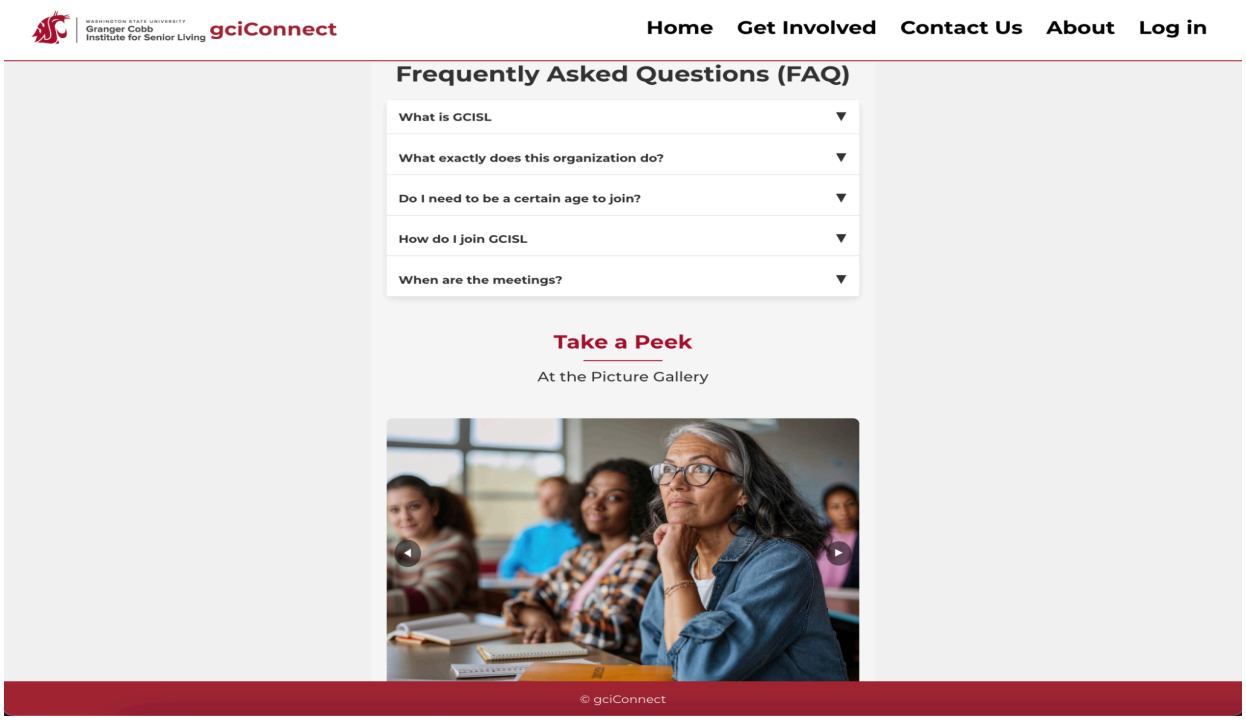
Public users can explore key sections of the site, while logged-in users have access to task management functionalities.

The prototype is divided into two main areas: the Final Prototype Summary that covers general features accessible to all visitors and the Major Use Cases that focuses on key actions admins and volunteers can perform once logged in.

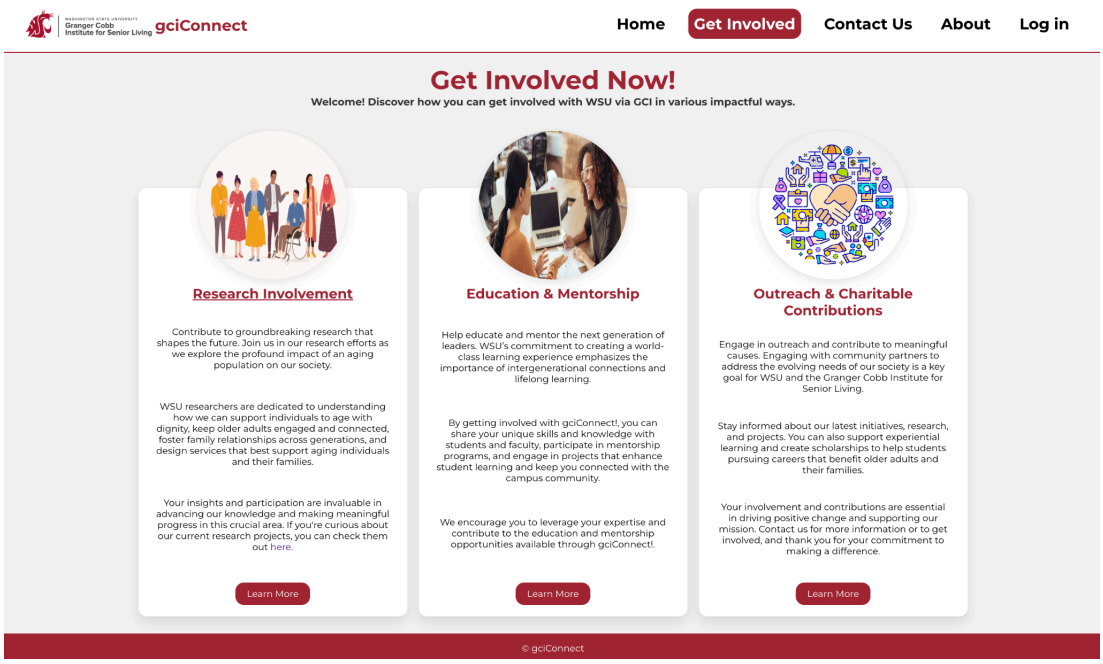
a) Final Prototype Summary



The Contact us page allowusers to communicate with the Gci personnel by sending them emails based on the specific concerns they have.

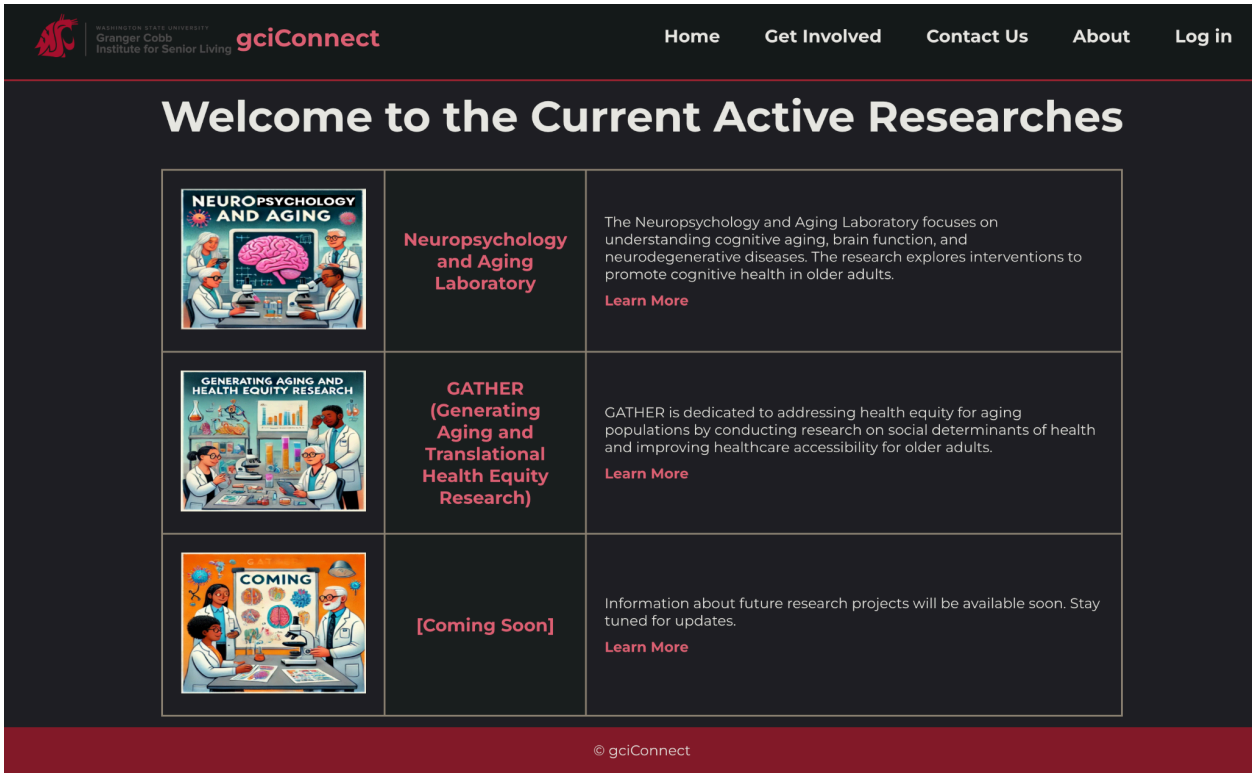


The FAQ page that gives a few answers to the questions that people tend to ask.



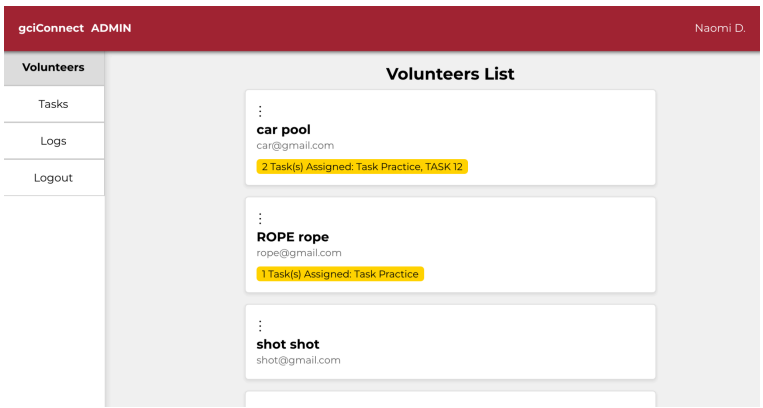
The Get involved page links various ways to be a part of GCI: Research Involvement, Education & Mentorship and Outreach & Charitable Contributions. Additionally those sections lead to pages with more information. When clicking on learn more (for the Research Involvement), the user is directed to the Contact Us page, where the field “Reason for Contacting Us” is already

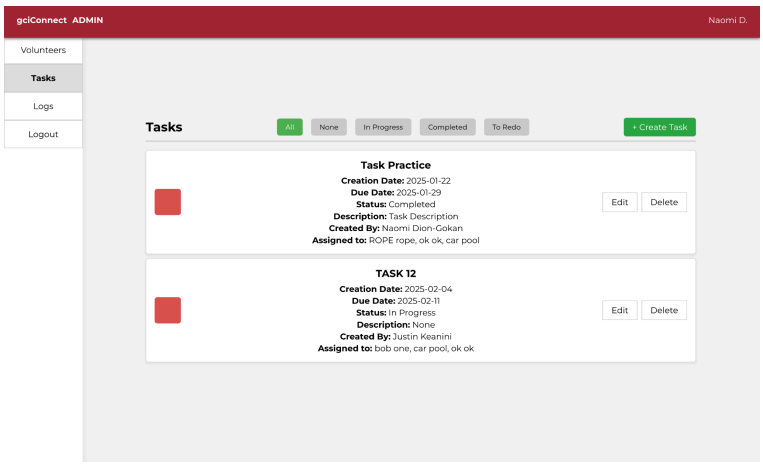
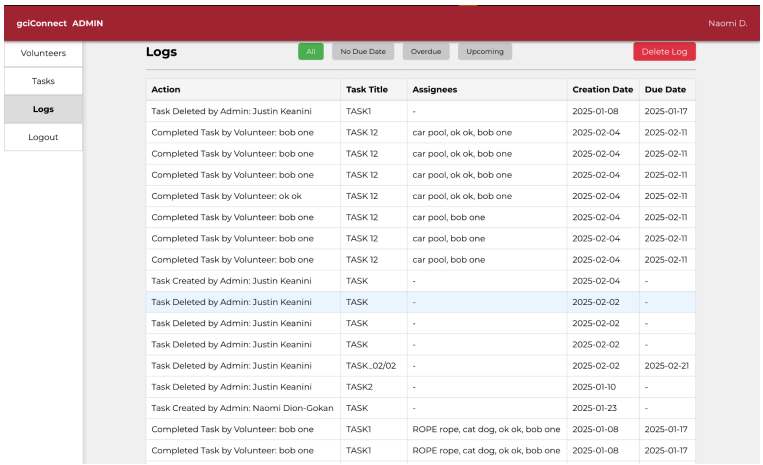
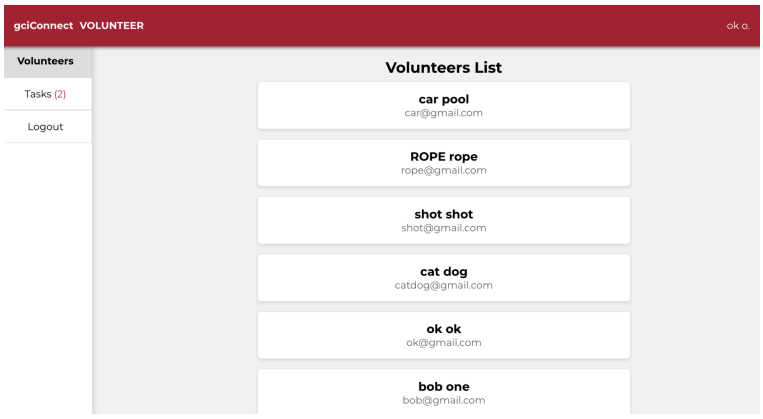
filled based on the section they are previously clicked on. Also by clicking on the “here” link, the user is led to the Research page.

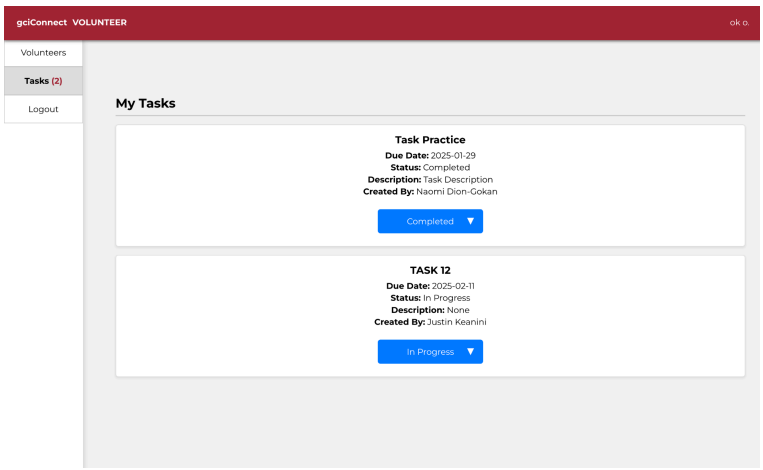


The Research page shows all the current researches available, their descriptions and links to know more about them.

b) Major Use Case

1	Shows the administrator’s dashboard, which provides the volunteer list showing all the volunteers, their emails and the tasks assigned to them.	

2	Shows the administrator's dashboard, which provides an overview of tasks. The ability to filter the tasks by status (All, None, In Progress, Completed, To Redo).	
3	Shows the administrator's dashboard, which provides an overview to logs to keep track of the activities that occur within the system. The ability to delete the log and filter it by due date (No Due Date, Overdue, Upcoming).	
4	Shows the volunteer dashboard, which provides the volunteer list showing all the volunteers names and their emails.	

5	Shows the volunteer's dashboard, which provides an overview of tasks. The ability to change the tasks status to "In Progress" and "Completed".	

VIII. Product Delivery Status

The final project is deployed on Vercel. It is available as a web application accessible to users online. Before deployment, the team worked closely with clients Darcie Bagott and Cory Bolkan, refining features based on feedback to ensure usability. The project has been demonstrated to the clients and it will be delivered to them on Friday, March 8th.

Project Location information:

1) The latest release of the project is available under Releases in the team's GitHub: https://github.com/awishto-write/GCISL as well as on Vercel: https://gciconnect.vercel.app/
2) The main branch of the team's GIHub will contain a stable version of the application
3) The materials needed to rebuild your project from the ground up are listed below

Set Up Instructions:

Given the nature of the gciConnect project, there will be only the developer sset up.

Prerequisites: Git, Node.js (14.x or higher), and a MongoDB instance (local or Atlas) and Visual Studio Code.

Add-Ons: There are no add-ons

Installation Steps:

- Clone the project
- Install dependencies and set the necessary environment variables
- Redirect to the front end and api folders to start the web application and run backend server

For more specific steps, check the GitHub repository installation steps:
https://github.com/awishto-write/GCISL/blob/main/README.md#installation-steps

IX. Conclusions and Future Work

IX.1. Limitations and Recommendations

a. **Single Task Status for Multiple Volunteers**

Currently, when multiple volunteers are assigned to the same task, any one of them can update its status, which may cause miscommunication. There is no built-in way to notify other

assignees of status changes. A possible solution is to introduce subtasks so that each volunteer has an individual task to update while still contributing to the main task. Adding a notification system for status updates would also improve coordination.

b. **Admin Task View Lacks Separation Between Assigned and All Task**

The current admin dashboard displays all tasks but does not distinguish between tasks assigned to the logged-in admin and those created by others. To improve task management, the dashboard should include two sections: “My Tasks”, showing only the logged-in admin’s tasks, and “All Tasks”, displaying tasks created by other admins. Both should be automatically sorted by the most recent creation date.

c. **Volunteer Task View and Log System**

Volunteers currently do not have a dedicated task view or a log to track tasks marked "To Redo" by an admin. Implementing a Volunteer Task Dashboard would allow them to see assigned tasks and track progress. Additionally, a Task Log Section should display tasks marked for rework, with admin notes for reference. For tasks with multiple assignees, an indicator should show when other volunteers have completed their subtasks.

IX.2. Future Work

The future work on this project will focus on expanding its functionality and user experience.One planned extension is the development of an in-app messaging feature to enable real-time communication between all the users (administrators and volunteers). This would streamline coordination, allowing for direct interactions within the platform and reducing reliance on external messaging tools. Additionally, a dark and light mode option will be introduced to provide users with a customizable interface, improving accessibility and usability for different preferences. Additionally, the system could be adapted for student organizations or other groups that rely on volunteers and need a structured system for assigning and tracking tasks.

X. Acknowledgements

For this section, the team would like to make a few acknowledgments. First, we would like to thank our clients, Darcie Bagott, a long-standing program specialist, and Cory Bolkan, a professor leading research for gciConnect . They have been instrumental in helping us understand the research topics that interest senior living residents. Their guidance ensures that our site is user-friendly and provides essential information for visitors. Secondly, we would like to thank our project mentor, Parteek Kumar, for his continuous support and valuable feedback, which helped ensure we exceeded client expectations by anticipating needs they may not have initially considered.

XI. Glossary

gciConnect: A web-based system used by GCISL for tracking and managing volunteer reports.

Automated Testing: A process where software tools are used to automatically run tests on the system to ensure its functionality, detect bugs, and improve reliability.

Application Performance Monitoring (APM): Tools and processes used to monitor and manage the performance and availability of software applications in real-time.

Admin Portal: A web-based interface used by system administrators to manage users, track performance, and troubleshoot issues within the system.

User Interface (UI): The part of a software application that users interact with, which should be designed to be intuitive and accessible.

Volunteer Report Tracking System: A system that helps track and manage the reports submitted by volunteers working in senior living environments, primarily for research purposes.

RESTful APIs: An architectural style for designing networked applications. It stands for Representational State Transfer, and it's used to allow communication between systems over the web.

Database: An organized collection of structured information or data, typically stored electronically in a computer system, that can be easily accessed, managed, and updated.

Scalability: The capability of a system to handle an increasing number of users or transactions without performance degradation.

System Reliability: The ability of a system to consistently perform its intended functions without failure over time.

Component-based architecture(CBA): an architectural style where systems are built by assembling pre-built and reusable software components

Role-Based Access Control (RBAC): A system of managing user permissions based on their role within an organization (e.g., volunteer, staff, administrator).

User Interface (UI) Layer: The part of the application responsible for managing interactions between users and the system, such as page layouts and navigation.

Backend Layer: Manages the system's core functionality, including user and volunteer management, task assignments, and notifications.

MongoDB Atlas: A cloud-based NoSQL database used for storing user, task, and volunteer information in a flexible document format.

Task Scheduling Algorithm: A method used by the Volunteer Management Handler to assign tasks to volunteers based on availability and need.

Session Management Algorithm: Ensures user sessions are validated and managed securely throughout the platform.

UAT (User Acceptance Testing) Participants: They are real end-users or people who represent the target audience of the application. They help confirm that the system meets the intended requirements and is easy to use.

Postman: it is a popular API testing tool that lets developers send HTTP requests to their backend and check responses, making it easier to validate and debug APIs.

CI/CD (Continuous Integration/Continuous Deployment): A set of practices that automate code integration and deployment processes. CI/CD pipelines run tests automatically upon each code push, ensuring continuous validation of code changes and quick deployment of updates.

Jest: A JavaScript testing framework primarily used for testing React applications. It enables developers to run unit and integration tests on components and functions to verify correctness.

React Testing Library: A tool that enables testing of React components by interacting with the application the way a user would, focusing on component behavior rather than implementation details.

Supertest: A Node.js library for testing HTTP endpoints. It allows for testing API interactions, verifying that backend services function correctly and respond with the expected results.

Unit Testing: A software testing method where individual components or functions of an application are tested in isolation to ensure they work as expected.

Integration Testing: Testing that verifies multiple components of the system work together. In this context, it ensures that front-end elements interact correctly with backend APIs and databases.

System Testing: Testing the entire application as a whole to validate that it meets functional and non-functional requirements, ensuring overall system integrity.

Functional Testing: A type of testing that validates the features and operations of an application, ensuring that they align with specified requirements and that the system behaves as expected in various scenarios.

Performance Testing: Tests that evaluate the application's speed, scalability, and stability under load, focusing on metrics such as response time, memory usage, and throughput.

Stress Testing: A type of performance test where the system is subjected to extreme workloads to identify its breaking point and measure how it handles high traffic or data volumes.

MongoDB: A NoSQL database used for storing and retrieving data in the GCISL application. Known for its scalability and flexibility, MongoDB stores data in JSON-like documents.

XII. References

WSU Granger Cobb Institute for Senior Living, "Volunteer Report Tracking System Enhancement," internal document, ACME8-GCISL FullStackApp, 2023 "

MongoDB Documentation," MongoDB Inc., 2023. [Online]. Available:

https://docs.mongodb.com "

Google Sheets API Documentation," Google LLC, 2023. [Online]. Available: https://developers.google.com/sheets/api

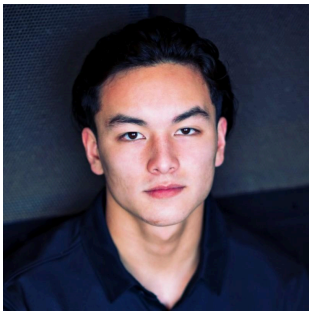
"React Documentation," Meta Platforms, Inc., 2024. [Online]. Available: https://reactjs.org/docs/getting-started.html

Gillis, Alexander S. "What Is User Acceptance Testing (UAT)?: Definition from TechTarget." Software Quality, TechTarget, 14 Mar. 2022. [Online]. Available: www.techtarget.com/searchsoftwarequality/definition/user-acceptance-testing-UAT. [Accessed: 02-Nov-2024]

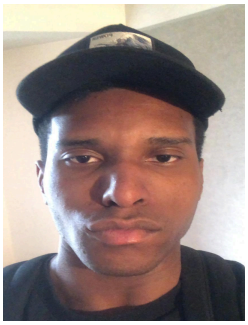
Postman. [Online]. Available: https://www.postman.com/postman/test-examples-in-postman/overview. [Accessed: 02-Nov-2024]

XIII. Appendix A – Team Information

List your team members here and provide a team photo.



Justin



Teni

Naomi

XIV. Appendix B - Example Testing Strategy Reporting

```
 PASS  tests/routes.test.js
  API Routes
    ✓ GET /api/user - Should fetch current user details (28 ms)
    ✓ POST /api/tasks - Should create a new task (31 ms)
    ✓ GET /api/tasks - Should fetch all tasks (29 ms)
    ✓ GET /api/volunteer-tasks - Should fetch tasks assigned to the current volunteer (25 ms)
    ✓ PUT /api/tasks/:id - Should update an existing task (29 ms)
    ✓ DELETE /api/tasks/:id - Should delete a task (28 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        1.813 s, estimated 2 s
Ran all test suites.
```

```
A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try runn
ing with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.
Test Suites: 2 failed, 2 total
Tests:       8 failed, 21 passed, 29 total
Snapshots:   0 total
Time:        3.902 s, estimated 4 s
Ran all test suites.
```

Testing Approach

The team implemented automated API testing to ensure the reliability and functionality of key backend components. The focus was on validating core API operations, response times, and error handling.

Requirements Being Tested

- Correct handling of user and task-related API routes
- Data integrity and response accuracy
- Performance (ensuring requests complete within an expected time frame)

Automated Testing

The backend was tested using Jest, covering the following API endpoints:

- GET /api/user – Fetches current user details
- POST /api/tasks – Creates a new task
- GET /api/tasks – Retrieves all tasks
- GET /api/volunteer-tasks – Fetches tasks assigned to volunteers
- PUT /api/tasks/:id – Updates existing tasks
- DELETE /api/tasks/:id – Deletes tasks

Each test validated expected behavior and edge cases, ensuring the system handles requests efficiently.

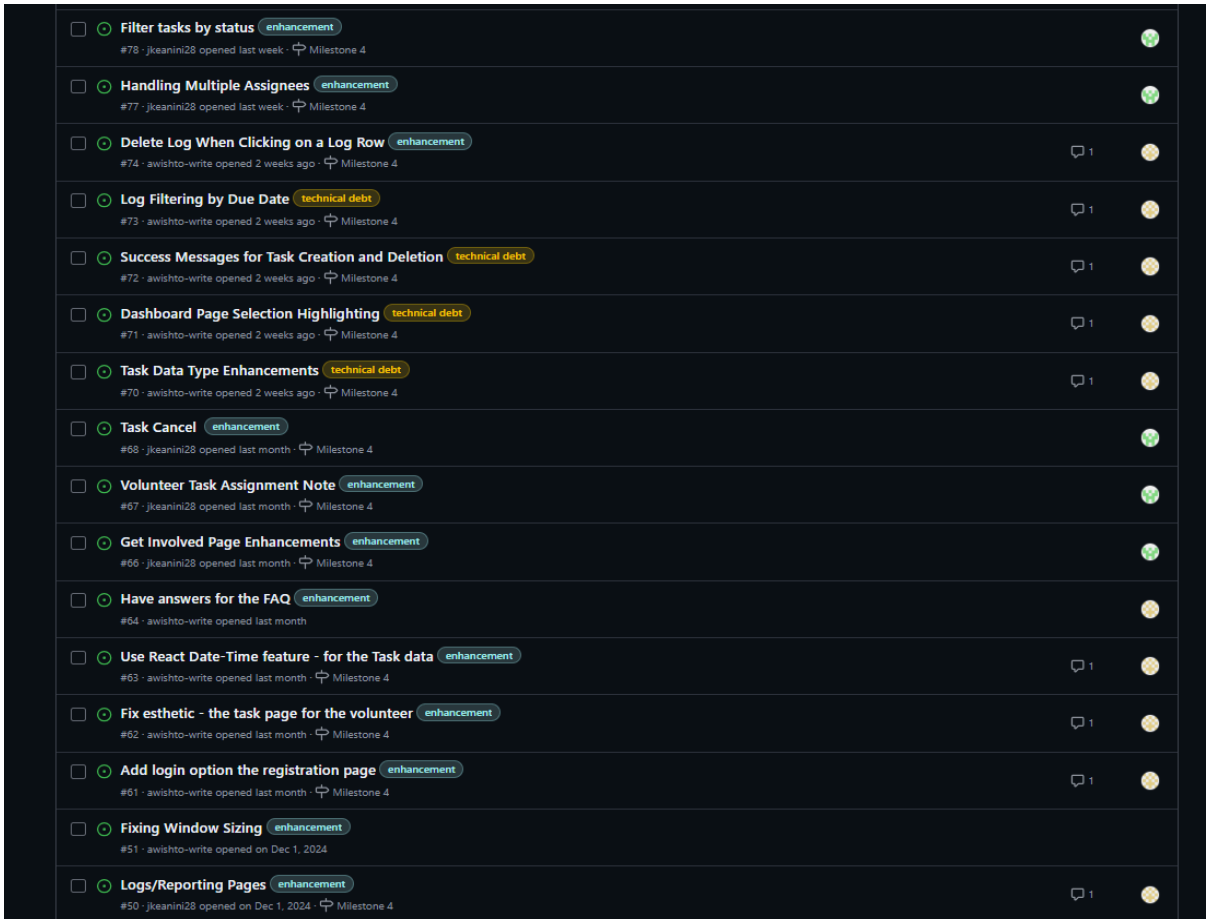
Results

All 6 tests passed successfully, confirming that API endpoints are functioning correctly, with response times averaging 25–31ms. The results reinforce confidence in the system's stability, accuracy, and performance.

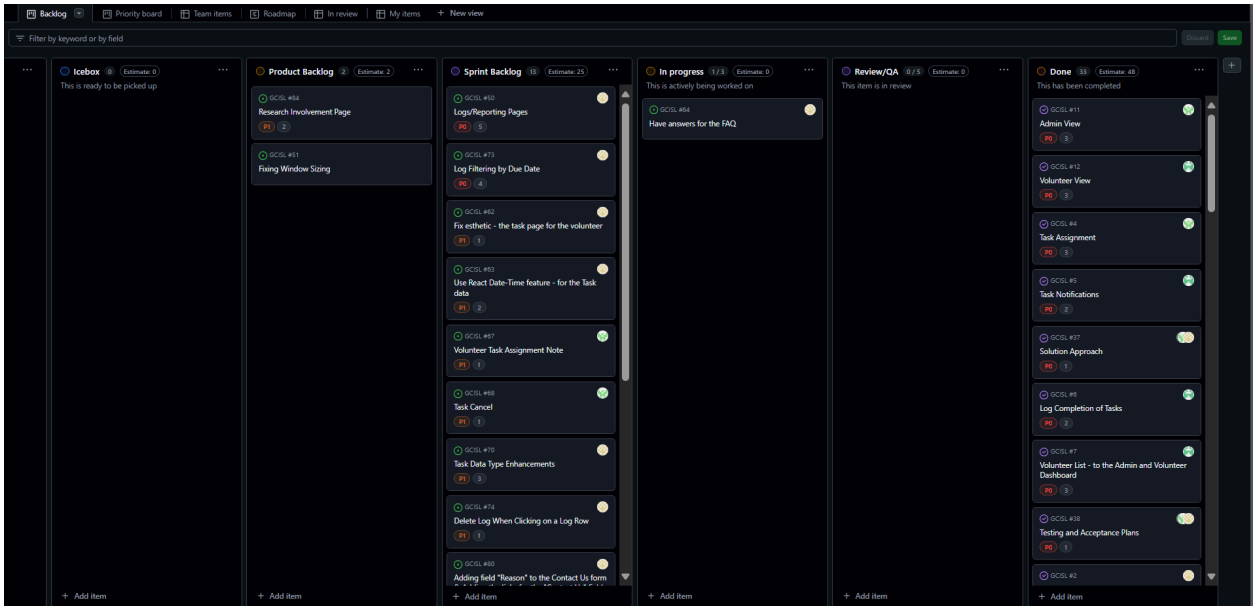
XV. Appendix C - Project Management

The team follows a weekly schedule that includes internal meetings to review tasks for the upcoming week and anticipate what will need attention in the following week. Every other week, the team meets with the client, Program Specialist Darcie Bagott, to present a demo of the team’s progress and gather her feedback. Additionally, the team holds regular check-ins with their mentor, Parteek Kumar, to assess the project’s progress and address any concerns.

Team Meetings:
The team gathers to review the issues for the upcoming sprint, discussing which team member will handle each task and setting timelines for completion. Weekly check-ins are held to monitor progress, and throughout the week, team members communicate about any challenges that arise during development. These meetings take place over Zoom. The team uses issue tracking to outline tasks (image 1), and a Kanban board to monitor tasks through the stages of “New Issues”, “Icebox”, “Product Backlog”, “Sprint Backlog”, “In Progress”, “Review Q/A”, and “Completed” during the sprint (image 2).


Filter tasks by status enhancement
#78 · jkeanini28 opened last week · Milestone 4
Handling Multiple Assignees enhancement
#77 · jkeanini28 opened last week · Milestone 4
Delete Log When Clicking on a Log Row enhancement
#74 · awishto-write opened 2 weeks ago · Milestone 4
Log Filtering by Due Date technical debt
#73 · awishto-write opened 2 weeks ago · Milestone 4
Success Messages for Task Creation and Deletion technical debt
#72 · awishto-write opened 2 weeks ago · Milestone 4
Dashboard Page Selection Highlighting technical debt
#71 · awishto-write opened 2 weeks ago · Milestone 4
Task Data Type Enhancements technical debt
#70 · awishto-write opened 2 weeks ago · Milestone 4
Task Cancel enhancement
#68 · jkeanini28 opened last month · Milestone 4
Volunteer Task Assignment Note enhancement
#67 · jkeanini28 opened last month · Milestone 4
Get Involved Page Enhancements enhancement
#66 · jkeanini28 opened last month · Milestone 4
Have answers for the FAQ enhancement
#64 · awishto-write opened last month
Use React Date-Time feature - for the Task data enhancement
#63 · awishto-write opened last month · Milestone 4
Fix esthetic - the task page for the volunteer enhancement
#62 · awishto-write opened last month · Milestone 4
Add login option the registration page enhancement
#61 · awishto-write opened last month · Milestone 4
Fixing Window Sizing enhancement
#51 · awishto-write opened on Dec 1, 2024
Logs/Reporting Pages enhancement
#50 · jkeanini28 opened on Dec 1, 2024 · Milestone 4

(image 1)



(image 2)

Client meetings:
Every other week the team meets with Program Specialist Darcie Bagott to discuss the fullstack app's progress and record her comments and concerns. These meetings are conducted over Zoom communicated by Email.

Mentor meetings:
Every month the team meets with Parteek Kumar to discuss the progress of the project and documentation. These meetings are conducted in-person.