Google Workspaces Alternative Application

Alpha Prototype Project Report

Wahkiakum School district, Naselle School District and Wahkiakum 4-H

Team Toto

Albert Lucas

Ben Kaufmann

Daniel Semenko

TABLE OF CONTENTS

I. Introduction **5**

II. Team Members - Bios and Project Roles **6**

III. Project Requirements **6**

III.1. System Requirements 6

III.1.1. Use Cases 6

III.1.2. Functional Requirements 27

III.1.2.1. User Authentication and Permissions 27

III.1.2.2. User Records 28

III.1.2.3. Administrative Moderation 28

III.1.2.4. Posts 29

III.1.2.5. Post Spaces 29

III.1.2.6. Training Materials 29

III.1.2.7. Commenting 30

III.1.2.8. Private Messaging 30

III.1.2.9. Searching Post Spaces 30

III.1.2.10. Home Post Space 30

III.1.2.11. Post Space Organization 31

III.1.3. Non-Functional Requirements 31

III.1.3.1. Easy to Use 31

III.1.3.2. Maintainable 31

III.1.3.3. Visually Appealing 31

III.1.3.4. Quick Response Time 31

III.1.3.5. Cross-Platform Availability 31

III.1.3.6. Security 32

III.1.3.7. CIPA Compliant 32

III.2. System Evolution 32

IV. Solution Approach **32**

IV.1. System Overview 32

IV.2. Architecture Design 33

IV.2.1. Overview 33

IV.2.2. Subsystem Decomposition 33

IV.2.2.1. Controller 33

IV.2.2.1.1. Description 33

IV.2.2.1.2. Concepts and Algorithms Generated 33

IV.2.2.1.3. Interface Description 33

IV.2.2.2. Authentication 34

IV.2.2.2.1. Description 34

IV.2.2.2.2. Concepts and Algorithms Generated 34

IV.2.2.2.3. Interface Description 34
IV.2.2.3. Computational Logic 34
IV.2.2.3.1. Description 34
IV.2.2.3.2. Concepts and Algorithms Generated 34
IV.2.2.3.3. Interface Description 35
IV.2.2.4. Data Layer 36
IV.2.2.4.1. Description 36
IV.2.2.4.2. Concepts and Algorithms Generated 36
IV.2.2.4.3. Interface Description 36
IV.2.2.5. Data Access Logic 36
IV.2.2.5.1. Description 36
IV.2.2.5.2. Concepts and Algorithms Generated 36
IV.2.2.5.3. Interface Description 36
IV.2.2.6. Firebase 37
IV.2.2.6.1. Description 37
IV.2.2.6.2. Concepts and Algorithms Generated 37
IV.2.2.6.3. Interface Description 37
IV.2.2.7. Presentation Layer 37
IV.2.2.7.1. Description 37
IV.2.2.7.2. Concepts and Algorithms Generated 38
IV.2.2.7.3. Interface Description 38
IV.2.2.8. IIS 38
IV.2.2.8.1. Description 38
IV.2.2.8.2. Concepts and Algorithms Generated 39
IV.2.2.8.3. Interface Description 39
IV.2.2.9. User 39
IV.2.2.9.1. Description 39
IV.2.2.9.2. Concepts and Algorithms Generated 39
IV.2.2.9.3. Interface Description 39
IV.3. Data Design 39
IV.4. UI Design 40
IV.4.1. Login Page 40
IV.4.2. Forgot Password Page 41
IV.4.3. Superuser Login Page* 41
IV.4.4. Main Home Page 41
IV.4.5. Side Tab Extended 41
IV.4.6. Mouse-over any Button 42
IV.4.7. Join/Create Class Space Popouts 42
IV.4.8. Mouse-over Profile Image 42
IV.4.9. View any profile popout 42
IV.4.10. Class Space Page 42

IV.4.11. View Post Page 43
IV.4.12. Delete Post (or Post Space) Popout 43
IV.4.13. Mouse-over Member Count 43
IV.4.14. View Member List Popout 43
IV.4.15. Remove User Popout 44
IV.4.16. Search for/add User Popout 44
IV.4.17. Edit Profile Page 44
IV.4.18. Change Password Popout 44
IV.4.19. Change Profile Picture Popout 45

V. Testing Plan 45
V.1. Testing Strategy 45
V.2. Testing Plans 46
V.2.1. Unit Testing 46
V.2.2. Integration Testing 46
V.2.3. System Testing 47
V.2.3.1. Functional testing: 47
V.2.3.2. Performance testing: 47
V.2.3.3. User Acceptance Testing: 48
V.3. Environment Requirements 48

VI. Alpha Prototype Description 48

VII. Alpha Prototype Demonstration 48

VIII. Future Work 49

IX. Glossary 49

X. References 50

XI. Appendices 51
XI.1. Appendix A - Software Architecture 51
XI.2. Appendix B - Data Design 52
XI.3. Appendix C - Initial User Interface Design 53

I. Introduction

This document is a record of Team Toto's solution design for Wahkiakum School district, Naselle School district, and Wahkiakum 4-H. It briefs the reader on the approach to building a communicative platform available for ChromeOS and Android. It dives deep into the blueprint of the application and projects how a future prototype should portray data and interact with users. This document is intended for developers, faculty, and anyone that is curious about how the application was constructed.

Wahkiakum School district, Naselle School district, and Wahkiakum 4-H are in need of an application that allows student users to communicate with each other across classrooms, clubs, groups, etc. To satisfy these district needs, our team is planning on analyzing, designing, and building a stand-alone application from scratch to improve student learning and connection.

II. Team Members - Bios and Project Roles

- Major
- Interests
- Experience
- Skills
- Responsibilities

Ben Kauffman is a computer science major with interests

Albert Lucas is a computer science student with interests in web development and cybersecurity. He has experience in systems programming, mobile application development, and mentoring fellow computer science students in the Compass Mentor Program at Washington State University. Albert is proficient in C/C++ and Python and is familiar with Windows, Linux, and mobile platforms. In this project, his responsibilities entail note-taking, documentation and developing in tests.

Daniel Semenko is a computer science major, who is interested in front-end development and web development. He has experience in mobile application development, front end development, systems programming, data science and more. He is proficient in C/C++, C#. Python, Flutter/Dart, and is familiar with Windows, Linux, and mobile platforms. For this project, Daniel's responsibilities include general documentation work, general programming, and a focus on frontend development.

III. Project Requirements

III.1. System Requirements

This section specifies the software product's requirements. This section specifies all the software requirements to a level of detail sufficient to enable designers to design a software system to satisfy those requirements and to enable testers to test that the software system satisfies those requirements.

In project management, the clients are the ones who define the requirements of the project and set the parameters such as budget and deadlines. Therefore, for Team Toto's communication application, the clients are the school district who are instating the use of this project. They need this online platform to allow for various methods of communication between students and faculty. These faculty members are the ones setting the requirements and specifications that the program must follow. They also help oversee the deadline for completion.

The users, on the other hand, would be anyone who actively uses the application. In the case of this project, the potential users could be students or faculty. Students can use the application to join class spaces and communicate with faculty or other users through direct or voice communication. The faculty, on the other hand, are the ones creating, using, and moderating these class spaces, while also being able to monitor student communications. Faculty can get an overview of the students who are in their spaces.

Finally, the stakeholders are anyone who is affected by the implementation of the project. The point of this project from the beginning is to connect faculty and students. Therefore, the potential stakeholders would include the faculty, who are able to create, use, and monitor class spaces and personal chats. Students are also stakeholders, as they are given the opportunity to join classes and participate in class space activities, and communicate with faculty or other students through private or voice communications. In addition, there could be administrators who aren't connected to any one class space, but through the implementation of the program, are able to moderate class spaces as needed.

III.1.1. Use Cases

Use Case #1

Name	Create Student Account
Users	Faculty
Rationale	People should not be allowed into the application if they are not verified students. Thus, faculty should go through and create accounts for students, with their school ID as part of their account. Once a student

	account is created, students can change their password to whatever they want.
Triggers	The faculty clicks the “Create Student Account” button on the sidebar.
Preconditions	Faculty is on any page with a view of the sidebar.
Actions	1. The faculty is taken to “Student Creation” page 2. The faculty fills out the minimum required information for student account creation (ID, username, temporary password) 3. Student account information is stored in the database for later updating and retrieval. 4. The Faculty is returned to the main home page
Alternative Paths	If the user’s registration information already exists, then don’t add the user to the database and display an appropriate error message on redirect.
Postconditions	The student account exists in the database.
Acceptance Tests	The student account is found in student listings and can join spaces or private communications.

Use Case #2

Name	Create Faculty Account
Users	Faculty
Rationale	Regular account creation, but it must have added verification to confirm that faculty accounts are actually linked to a faculty member. Faculty, with proper verification, can then create and fill the rest of their account.
Triggers	The faculty clicks “Create Faculty Account” button on the login screen.
Preconditions	Faculty is on the “Login” page.

Actions	1. User fills out all fields with their contact information 2. The faculty information is stored for later retrieval. 3. The user is redirected to the login page, with a message indicating that they registered successfully.
Alternative Paths	If the user's registration information already exists, then don't add the user to the database and display an appropriate error message on redirect.
Postconditions	The faculty exists in the database.
Acceptance Tests	The faculty is found in faculty listings and can create class spaces, utilize class space functions, and monitor and edit class posts/comments/communications.

Use Case #3

Name	User Login
Users	Students, Faculty
Rationale	As long as the user's account exists in the database, users should be able to log on with a user and password. Users need exclusive access to the application.
Triggers	The user clicks the "Login" button after entering their username and password.

Preconditions	User is not logged in, user account already exists
Actions	1. User enters username and password and presses the “log in” button. 2. User is authenticated, then taken to the main page.
Alternative Paths	User account does not exist yet, can press “Create Faculty Account” to go to the Faculty Account Creation page. User may provide incorrect username or password, returned to empty log in page with corresponding message.
Postconditions	User is logged into the correct user-type page. That is, a student will be taken to the student view page, while the faculty will be taken to the faculty view page.
Acceptance Tests	A student member can log in and be taken to their home page, but can not create class spaces. A faculty member can log in and be taken to their home page and is able to see their class spaces or create new ones.

Use Case #4

Name	Logout
Users	Students, Faculty
Rationale	Users should be able to log out.
Triggers	User clicks “Log Out” button.
Preconditions	User is logged in, on any page.

Actions	1. User is logged out. 2. User is taken to login screen 3. “Successful logout” message is displayed on the screen.
Alternative Paths	None
Postconditions	The user is successfully logged out.
Acceptance Tests	After logging out, no user is currently logged in.

Use Case #5

Name	View Account
Users	Student, Faculty
Rationale	Any user should be able to view their own account, or any other user’s account (student or faculty). They should be able to find a user through a class space, or through the user search on the sidebar.
Triggers	User clicks another user’s profile block.

	User clicks their own profile block.
Preconditions	User is logged in, on any page with the sidebar.
Actions	1. User clicks another user's profile block, or their own "My Profile" button. 2. User is taken to the page, where their/the users information is displayed.
Alternative Paths	None
Postconditions	The queried user profile page is opened up, where profile information is visible. The user's own profile page is opened up, where their personal profile information is visible.
Acceptance Tests	The correct account information is displayed.

Use Case #6

Name	Edit Account
Users	Students, Faculty
Rationale	Users shall be able to edit their account details, such as profile description, user image, contact information, etc. Faculty can edit student accounts (in case moderation is needed), but only modify profile description and profile picture.

Triggers	User clicks “Edit Profile” button on own profile page. Faculty clicks “Edit Profile” button on student profile page.
Preconditions	User is logged in, user on own profile page. Faculty is logged in, viewing student profile page.
Actions	1. Taken to Edit Profile page. Student users can only have their profile picture and description modified. Faculty can change everything else, except for log in information. 2. After clicking “save”, updated information is stored in database. 3. User is returned to previous profile page.
Alternative Paths	If user presses “revert changes,” or leaves the page without saving the changes, return to the profile page without updating the database.
Postconditions	The updated information exists in the database.
Acceptance Tests	Users should open their profile, make and save changes, and see those changes updated. Faculty should open a student profile, make and save changes, and see those changes updated.

Use Case #7

Name	Delete Account
Users	Faculty
Rationale	Only Faculty can delete either a student’s account, or their own account. A student should not be able to randomly delete their account.

Triggers	Faculty clicks “Delete Profile” button on profile page.
Preconditions	Faculty should be on their own profile page, or viewing a student's profile page.
Actions	1. After clicking “Delete Profile” button, faculty should have to confirm selection (no accidental deletions). 2. All posts, comments, and/or related class spaces belonging to the deleted user should be deleted. 3. Students that are part of a class of a now deleted faculty are redirected to their home page, with a message stating that the class has been deleted.
Alternative Paths	If user presses “no” on the confirmation, user returns to previous page.
Postconditions	Account is no longer visible in user listings, not in the database.
Acceptance Tests	Logging in as the faculty, and deleting a students account should make their account and all their posts and comments not visible anymore, nor should it be able to be logged into.

Use Case #8

Name	Create Post
Users	Faculty and Students
Rationale	Faculty and Students must be able to create a post on Post Space

	page
Triggers	The user clicks “Create Post”
Preconditions	The user is in logged in
Actions	1. The use fills out post field, detailing what they would like to post publicly in space 2. Click submit to create a publicly visible post
Alternative Paths	None
Postconditions	There is a Space in the data for the user to post in. Users are redirected to the new page
Acceptance Tests	Post is found in “Post Space” page and can comment/react on post

Use Case #9

Name	View Post
Users	Faculty and Students

Rationale	Faculty and Students
Triggers	The user clicks on an existing post on a “space” page
Preconditions	The user is logged in
Actions	1. The user is on a “space” page 2. The user clicks a post, and the post reveals the content comments within that Post page
Alternative Paths	None
Postconditions	The post is now viewable, showing the post content, reactions and comments
Acceptance Tests	Post contents are able to be viewed

Use Case #10

Name	Edit Student’s Post
Users	Faculty. Only Faculty can Edit a Student’s Post

Rationale	Faculty Users must be able to Edit any Student user's post from Post Spaces.
Triggers	Faculty users click the "Edit Post" button to modify the content within their post. Once the Post is edited, the users can click the "Submit Edit" button.
Preconditions	The Faculty user is logged in, is in the Post Space and there exists a post created by a Student user
Action	1. The Faculty user clicks on a post they would like to edit in a Post Space 2. The user clicks "Edit Post" button to edit the contents of a post 3. Once the post content is modified, the user clicks the "Submit Edit" button to repost a new post.
Alternative Paths	None
Postconditions	The new modified Post content is posted the Post Space
Acceptance Tests	Viewing an edited Post reflects the new changes made in the edit.

Use Case #11

Name	Delete Student's Post
Users	Faculty. Only Faculty can Edit a Student's Post

Rationale	Faculty Users must be able to Delete any Student user's post from Post Spaces.
Triggers	Faculty users click the "Delete Post" button after clicking a post to remove any unwanted Student user posts from the Post Space.
Preconditions	The Faculty user is logged in, is in the Post Space and there exists a post created by a Student user.
Action	1. The Faculty user clicks on a post they would like to remove from a Post Space 2. The user clicks "Delete Post" button 3. Once the button is pressed, the post is removed from the database and from the Post Space page
Alternative Paths	None
Postconditions	The post is no longer available on the Post Space page
Acceptance Tests	The Post Space no longer contains the deleted Post.

Use Case #12

Name	Comment on Post

Users	Faculty and Students
Rationale	Faculty and Students must be able to comment on a post on a Post Space page
Triggers	The user clicks on “Comment” button
Preconditions	The user is logged in and there exists a post on a Post Space page
Action	1. The user clicks on a post or “View Post” button 2. The user clicks on the “Comment” button and fills out the comment area box for the comment content to be posted 3. Users presses Enter button or “Submit Comment” to submit content to database and be posted on the Post page
Alternative Paths	None
Postconditions	The comment is visible on the post after clicking the “View Post” button
Acceptance Tests	Viewing a Post displays the comment made on it.

Use Case #13

Name	Edit Student's Comment
Users	Faculty. Only Faculty can Edit a Student's Post
Rationale	Faculty Users must be able to Edit any Student user's comment from Post Spaces.
Triggers	Faculty users click the "Edit Comment" button to modify the content of their comment. Once the comment is edited, the users can click the "Submit Edit" button.
Preconditions	The Faculty users is logged in, is in a Post Space and there exists a comment on a post
Action	1. The user clicks on a post or "View Post" button 2. The users find a comment they would like to edit and click the "Edit Comment" button to modify the content within the comment area box 3. Once the edits are finalized, the users clicks on the "Submit Edit" button to submit the modified content to the database and to post the new modifications on the comment
Alternative Paths	None
Postconditions	The new modified Comment content is posted on the post
Acceptance Tests	Viewing an edited Comment reflects the new changes made in the edit.

Name	Delete Student's Comments
Users	Faculty. Only Faculty can delete another Student's Post
Rationale	Faculty users must be able to delete and Student user's comment from Post Spaces
Triggers	Faculty users click the "Delete Comment" button to remove a
Preconditions	The Faculty user is logged in, is in the Post Space and there exists a comment on a post created by a Student user.
Action	1. The user clicks on a post or "View Post" button 2. The users find a comment they would like to remove and click the "Delete Comment" button 3. Once the button is pressed, the comment is removed from the database and from the Post page
Alternative Paths	None
Postconditions	The comment is removed
Acceptance Tests	The comment is no longer visible on the Post it was made under.

Name	Create Post Space
Users	Faculty.
Rationale	Faculty need to be able to create Post Spaces to organize the Posts together. Students should not be able to do this since they are unable to moderate the Post Space.
Triggers	Faculty user clicks the “Create Post Space” button from the left sidebar.
Preconditions	Faculty user is logged in.
Actions	1. Faculty user fills out Post Space fields. 2. The Post Space information is created and stored for later retrieval.
Alternative Paths	If the new Post Space already exists, then don’t add it to the database and display an appropriate error message on redirect back to the original page.
Postconditions	The new Post Space is now created. The user is redirected to the new Post Space page.
Acceptance Tests	Post Space is found in the database.

Name	Delete Post Space
Users	Faculty.
Rationale	Faculty users need to be able to delete a Post Space so the database does not get cluttered/disorganized.
Triggers	Faculty user clicks “Delete Post Space” button from either the “Individual Post Space” page or the “View all Post Spaces” page.
Preconditions	Faculty user is logged in and has joined the Post Space they intend to delete
Actions	1. Faculty user is prompted with “Are you sure you want to do this?” prompt.
Alternative Paths	If a Faculty user changes their mind when asked for reverification, they are redirected to the previous page they were on.
Postconditions	Post Space is deleted from the database. Faculty user is redirected to the previous page. If the previous page no longer exists, the Faculty user is redirected to home.
Acceptance Tests	Post space is no longer able to be found in the database.

Name	Add User to Post Space
Users	Faculty.
Rationale	Faculty users need to invite other users to a Post Space so groups can be formed.
Triggers	Faculty user clicks the “Invite” button from the “Individual Post Space” page.
Preconditions	Faculty user must be logged in and a part of the Post Space they are trying to invite users to.
Actions	1. Faculty user is prompted with a search bar meant for ID or username to enter. 2. “Add” button displays next to each search result that is not a part of the Post Space already. 3. Faculty user closes the prompt when they are done inviting users.
Alternative Paths	None.
Postconditions	Users are added to the Post Space. Faculty user is redirected to the previous page.
Acceptance Tests	User is able to see the Post Space and appears on the Post Space user list.

Name	Remove Student from Post Space
Users	Faculty.
Rationale	Faculty users must be able to remove unwanted Student users from Post Spaces.
Triggers	Faculty user clicks the “Remove Student From Post Space” option under the “Post Space Settings” button.
Preconditions	Faculty user is logged in, is in the Post Space and the Student user they intend to remove is also in the Post Space.
Actions	1. Faculty user is displayed every Student user in the Post Space with a “Remove” button next to each of their names. 2. Faculty user closes the prompt when they are done removing users.
Alternative Paths	None.
Postconditions	Student Users are removed from Post Space. Faculty user is redirected to the previous page.
Acceptance Tests	User is no longer able to see the Post Space and does not appear on the Post Space user list.

Name	Join Discord Voice Channel
Users	Faculty, Students.
Rationale	Having Voice Channels allows for online group meetings. Users must be able to join these Voice Channels from the application.
Triggers	User clicks the “Join Voice Channel” button from the “Individual Post Space” page.
Preconditions	User must be logged in and in the Post Space of the Voice Channel they are trying to join.
Actions	1. User is redirected to Discord. 2. User joins the dedicated Discord Voice Channel for that Post Space. 3. User disconnects from Voice Channel.
Alternative Paths	Something is wrong with opening Discord (logging in, opening the app, etc.) and the user is redirected to the previous page.
Postconditions	User is redirected to the previous page.
Acceptance Tests	User is successfully redirected to the Discord Voice Channel.

Name	Request to join a Post Space
Users	Faculty, Students. Faculty do not need authorization to join a Post Space
Rationale	It would be convenient to the users if they could search through all the visible Post Spaces to join instead of relying on invites only.
Triggers	User clicks the “Request to Join” button next to the desired Post Space in the “View All Post Spaces” page.
Preconditions	User is logged in and not already a part of the Post Space they are attempting to request to join.
Actions	1. Request is sent to the Faculty users within a Post Space. 2. Faculty user is prompted with a notification to accept or deny the request.
Alternative Paths	Request is denied and now the user can only be added by a Faculty user to a Post Space.
Postconditions	Request is accepted and the user is added to the Post Space.
Acceptance Tests	The request is received by the Faculty users of the Post Space. Once the request is approved, the User is able to see the Post Space and appears on the Post Space user list.

III.1.2. Functional Requirements

III.1.2.1. User Authentication and Permissions

Authentication: Student and Faculty identities need to be verified before the creation of both faculty and student accounts. Users should have student or faculty ID numbers to which their accounts will correspond. Faculty accounts with verified identities will be able to create student accounts and have moderative abilities.

Source: Team Toto development team originated this requirement. The requirement is necessary for the security of the application.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.2. User Records

Title: The databases created during development will be responsible for storing all user, space, post, comment, and other data. Users need to be able to view a list of user records. Faculty need to be able to create these functions, have access to them, and allow access for students to access them.

Source: Team Toto development team originated this requirement. The requirement is necessary for the development of the application, and for application usage.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.3. Administrative Moderation

Comment and Post Review: Faculty accounts with verified identities will be able to moderate student activity through the editing and deleting of student activity. Additionally, Faculty users will be able to change the permissions of Student users within each Post Space to allow/disallow certain actions. This includes student posts, comments, profile information, and other communications.

Source: The client originated this requirement. The requirement is necessary for faculty users and CIPA Compliance.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.4. Posts

Post Creation and Usage: Posts can be created by all users given this permission in a Post Space. Each Post also has an individual settings page that can allow/disallow comments and pin the Post within the Post Space. All users are able to edit, delete and change the settings of their own posts, while Faculty users can edit, delete and change the settings of the Posts made by Student Users.

Source: The client originated this requirement. The requirement is necessary for communication within the application.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.5. Post Spaces

Post Space Creation and Usage: Post Spaces are the text channels in which Posts are created and displayed to the rest of the users within that Post Space. Faculty users are the only ones who can create Post Spaces. After creation, Faculty users can add other users to the Post Space and manage their permissions to Post and Comment within the Post Space. Other managerial duties include the ability to remove users, delete the Post Space, hide the Post Space from the search.

Source: The client originated this requirement. The requirement is necessary for communication within the application.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.6. Training Materials

Training for Students and Teachers: The application must have training materials to train students, faculty, and administrators on how to use the application. Since this application will be unfamiliar to all users, provided training materials must clearly demonstrate how to use the application. The application is focused on usage by students in high school and younger, and by Faculty. The student training materials must be easy to read and learn how to do.

Source: The client originated this requirement. The requirement is necessary for both student and faculty users.

Priority: Priority Level 0: Essential and required functionality

III.1.2.7. Commenting

Comment Creation and Usage: Comments can be created in a thread underneath Posts by all users given this permission in a Post Space. Users can edit and delete their own comments, while Faculty users can edit and delete the comments of other Students.

Source: Team Toto development team originated this requirement. The requirement is necessary for communication within the application.

Priority: Priority Level 0: Essential and required functionality.

III.1.2.8. Private Messaging

Private Messaging between users: Users are able to privately message other users, setting up a Text message-like conversation.

Source: Team Toto development team originated this requirement. The requirement is desirable for communication between Users.

Priority: Priority Level 1: Desirable functionality.

III.1.2.9. Searching Post Spaces

Searching: Users are able to search (by keyword) for Post Spaces. They are then able to request to join a Post Space, where a Faculty user can either accept or deny them.

Source: Team Toto development team originated this requirement. The requirement is desirable for ease of use.

Priority: Priority Level 1: Desirable functionality.

III.1.2.10. Home Post Space

Personalized Home Post Space: Users are able to select a Home Post Space that they are greeted with upon application startup.

Source: Team Toto development team originated this requirement. The requirement is an extra feature for the development and usage of the application.

Priority: Priority Level 2: Extra features or stretch goals.

III.1.2.11. Post Space Organization

Organization: Users can organize the Post Spaces they are a part of into folders.

Source: Team Toto development team originated this requirement. The requirement is an extra feature for the development and usage of the application.

Priority: Priority Level 2: Extra features or stretch goals.

III.1.3. Non-Functional Requirements

III.1.3.1. Easy to Use

The application will be used by both students and faculty, who will be new to the software. It must be easy to use and accessible to all in order for users to actually want to use the application. If teachers struggle to use the application's features, its uses are useless.

III.1.3.2. Maintainable

When the project is completed and delivered to the client, the school districts will become the owners and maintainers of the product. Thus, this application should be easy for them to maintain.

III.1.3.3. Visually Appealing

As the application is focused on usage by late elementary-age to high school-age students, the application must be visually appealing enough to keep the attention of students and get them to want to use it.

III.1.3.4. Quick Response Time

The server should respond to page accesses in a reasonable time. On a fast connection, 3s response time is reasonable.

III.1.3.5. Cross-Platform Availability

The application would work from platform to platform. Students utilize Chromebooks through school checkouts, but will also utilize applications on mobile devices. The application should run and look consistent regardless of platform.

III.1.3.6. Security

The application must be secure in regard to application security and user integrity. The accounts of users shall stay secure, and a password change shall be required every 90 days. User accounts shall be verified before being imputed into the database.

III.1.3.7. CIPA Compliant

The application shall follow all required guidelines and stay compliant with the required CIPA guidelines. This entails record keeping of all user interactions, and administrative oversight.

III.2. System Evolution

The main fundamental assumption we are making is the fact that these requirements are all CIPA-compliant. Functionality such as Posts and Commenting should be safe since Faculty users will be able to edit and delete inappropriate content. However, functionality such as Private Messaging and especially Discord Voice Communication might need to change over time to adjust to CIPA.

When it comes to Discord Voice Communication, several ideas have been discussed to comply with CIPA. The Voice Channels will likely need to be private, meaning the only way to join them is through the Post Space. This way, users outside of the Post Space will be unable to join. If all voice calls need to be monitored, bots can be created that record the audio conversations. This could potentially lead to more issues however and needs future discussion. Discord allows for private messaging between accounts that are unmonitored. Disabling this functionality is an obvious option, but may not be achievable. If our team finds out that using Discord is not a realistic option, then an alternative plan will be put in place. Using a different platform (Google Voice, Telegram) and creating our own voice channels are all options up for consideration. Overall, our team believes the benefits of using Discord outweigh the risks it jkjdevices. Our team does not have experience working with Flutter, but experienced developers say that it is easy to learn and similar to Django. Flutter allows for different types of architectural patterns, so we will be using the Business Logic Component pattern since we have all worked with it before. In terms of the User Interface, we decided to model our application like similar applications such as Discord and Google Spaces. That means having the user's post spaces accessible from anywhere in the app using a sidebar. Overall, we are taking a cautious approach in developing the system and are sticking with things we have experience with.

III.3. Architecture Design

III.3.1. Overview

The Team Toto development team has decided on a Bloc, Business Logic Component, and architectural pattern for this application. The team concluded with the justification that this application is meant to be a Cross-Platform Application. Thus, there will be a user view, a database model, and a business logic-driven control. We believe that given the context of this application, any other architectural design pattern doesn't make sense.

(See Appendix A Image 1)

Our team is using this diagram to base our development around, and it was used for subsystem decomposition, which is expanded upon in the next section. Here, we will provide an overview of every component within the diagram, with expanded details to follow in the next section.

III.3.2. Subsystem Decomposition

III.3.2.1. Controller

III.3.2.1.1. Description

In general, the Bloc design pattern, and any Bloc-based application, will wait and listen for user inputs and will send this data in the form of an Event to the Business Logic Layer of the application. From there, Bloc will send requests to the database and will receive a response in an asynchronous manner just as it did from the user end. Bloc will then transmit the response to emit a state back to the user end of the application. Specific to this application, the Bloc subsystem will represent the connection between the user on the front end of the application, the database, and the state of the application. It will utilize business logic in order to create posts, comments, spaces, and more.

III.3.2.1.2. Concepts and Algorithms Generated

The Bloc subsystem will be composed of a Bloc class, with some smaller systems as part of the greater Bloc subsystem.

III.3.2.1.3. Interface Description

We will leave the Bloc Interface Description empty, as the Bloc subsystem consists of the Authentication and Computational Logic sections, and we can go into more detail within those individual subsystems.

III.3.2.2. Authentication

III.3.2.2.1. Description

The authentication subsystem is tied to the Bloc and the User Interface subsystems. It is a smaller subsystem, in charge of the constant authentication of user permissions, log-in checks, superuser privileges, and more. Linked closely to the User Interface subsystem, since it transmits states to the User Interface subsystem, and sends back events to the Bloc.

III.3.2.2.2. Concepts and Algorithms Generated

This subsystem has been created to encapsulate any behavior or functions in regard to authentication, permission handling, and password checking. It is necessary so that the subsystems that utilize authentication don't need to do their own authentication checks, and can instead focus on their more specific functions. This helps promote encapsulation.

III.3.2.2.3. Interface Description

III.3.2.2.3.1. Services Provided:

1. *Service name:* Login
   Service Provided to: Bloc
   Description: This service will authenticate the user's provided credentials with those present in the Firebase database. After authentication, a new user session is created.

2. *Service Name:* Logout
   Service Provided to: Bloc
   Description: This service will end the user session.

3. *Service Name:* hasPermissionTo
   Service Provided to: Bloc
   Description: This service, provided to many functions throughout the application, will state whether a given user has any given permission to any given requested resource or action. The permissions are pulled from the Firebase database, dependent on the specific function from the queried table.

III.3.2.2.3.2. Services Required:

1. Firebase database

III.3.2.3. Computational Logic

III.3.2.3.1. Description

The logic handling and running both the data layer and the presentation layer processes. This is where a majority of the code is, as it must handle the connection between the Presentation subsystem and the Data subsystem. Part of the greater Bloc subsystem.

III.3.2.3.2. Concepts and Algorithms Generated

This subsystem will have methods to handle the many different requests for the many different tables we have. These functions will retrieve data through the data access subsystems in order to run the application. There may be add<>(), update<>(), or delete<>() functions in which data comes from the User Interface to Bloc to update in the Data Layer. There may be get<>() functions in which data comes from the Data Layer to Bloc to display in the User Interface.

III.3.2.3.3. Interface Description

III.3.2.3.3.1. Services Provided:

1. *Service name:* GET

Service provided to: Presentation Layer
Description: A GET request, or a get<> function, is used to retrieve/request data from a database or server, in this application, from the Firebase database. It retrieves whatever information is identified within the request.

2. *Service name:* POST
   Service provided to: Data Access Logic
   Description: A POST request, or a post<> function, is used to create a resource. In this application, it creates a new resource within one of the tables, within the Firebase database.

3. *Service name:* DELETE
   Service provided to: Data Access Logic
   Description: A DELETE request, or a delete<> function, is used to delete an existing resource. In this application, it deletes an existing resource from a specific table, within the Firebase database.

4. *Service name:* PATCH
   Service provided to: Data Access Logic
   Description: A PATCH request, or a patch<> function, is used for updating an existing resource. In this application, it updates an existing resource within the specific table, within the Firebase database.

5. *Service name:* PUT
   Service provided to: Data Access Logic
   Description: A PUT request, or a put<> function, is used for checking if a resource already exists then updating, or otherwise creating a new resource if it doesn't already exist. In this application, it either updates an existing resource within the specific table or creates a new resource within one of the tables, within the Firebase database.

III.3.2.3.3.2. Services Required:

1. Firebase database
2. User Interface, aka Presentation Layer

III.3.2.4. Data Layer

III.3.2.4.1. Description

The Data Layer subsystem will be composed of a Model class, and some smaller systems part of the greater Data Layer subsystem. The general purpose of the model class is to connect and communicate with the database/data provider, take that information and parse/translate it, and send it to the specific Bloc function called from the Bloc subsystem.

III.3.2.4.2. Concepts and Algorithms Generated

The Data Layer subsystem consists of the Data Access Logic, the Firebase, and the DartKB subsystems. These subsystems work together to allow for the transfer, updating and deleting of information between the Bloc, and the Database.

III.3.2.4.3. Interface Description

We will leave the Data Layer's Interface Description empty, as the Data Layer's subsystem consists of the Data Access Logic, DartKB, and Firebase sections, and we can go into more detail within those individual subsystems.

III.3.2.5. Data Access Logic

III.3.2.5.1. Description

The logic allows for data updating and retrieval. Also known as the Data Access Layer, this subsystem is housed within the Data Layer subsystem and is responsible for providing access to data stored in persistent storage. In this example, the persistent storage is our Firebase database, within which our data is stored.

III.3.2.5.2. Concepts and Algorithms Generated

The Data Access Logic generates queries to add or retrieve data to the USER, POST, COMMENT, POST SPACE, and PERMISSIONS tables. It must instantiate objects for these data types when necessary and add new entries to the appropriate tables.

III.3.2.5.3. Interface Description

III.3.2.5.3.1. Services Provided:

1. *Service name:* Firebase Protocol
   Service provided to: Firebase database
   Description: This service provides a connection between the Firebase database and the Data Access Logic.

2. *Service name:* AddEntry
   Service provided to: Firebase database
   Description: This service adds any given new entry to the appropriate model table, using data received from the Bloc subsystem.

3. *Service name:* UpdateEntry
   Service provided to: Firebase database
   Description: This service updates any given entry in its corresponding table, using data received from the Bloc subsystem.

4. *Service name:* RemoveEntry
   Service provided to: Firebase database
   Description: This service removes a given entry from its corresponding table, provided by the Bloc subsystem. .

5. *Service name*: returnData
   Service provided to: Firebase database
   Description: This service receives data from the Firebase database using getData (see below), parses and translates the data to the correct object, and returns it to the process it was called within.

III.3.2.5.3.2. Services Required:

1. *Service name*: getData
   Service Provided from: Bloc

III.3.2.6. Firebase

III.3.2.6.1. Description

Firebase is a Backend-as-a-Service acquired and backed by Google known for its realtime database. Firebase provides a variety of tools for developing, handling, and enhancing applications, allowing the application to be stored online.

III.3.2.6.2. Concepts and Algorithms Generated

This subsystem allows for the connection between the application and the database, which would be stored in Google Cloud.

III.3.2.6.3. Interface Description

The group is not 100% on how the database will be utilized, between cloud hosting vs server hosting. As we are awaiting a decision from the client, and throughout development will be using Firebase on Google Cloud, we will be leaving the Interface Description empty.

III.3.2.7. Presentation Layer

III.3.2.7.1. Description

What the user is physically seeing. The Presentation subsystem controls and handles the application's data representation and user interactions.The Presentation Layer consists of a tree of widgets, originating from the route widget, and extending with more and more as it goes deeper into the application. The widgets interact together, along with some methods from the Bloc subsystem, to run the page that is sent to the client through IIS. The view will be similar, but slightly different for the Mobile view and for the Chromebook view. Both are handled through the same subsystem.

III.3.2.7.2. Concepts and Algorithms Generated

The Presentation's many functions are dispersed across many smaller subsystems, similar to that of the Data and the Bloc subsystems. However, since it is more general in the Presentation subsystem compared to the others, these will all be handled under the general Presentation subsystem.

III.3.2.7.3. Interface Description

III.3.2.7.3.1. Services Provided:

1. *Service name*: DisplayPage
   Service provided to: IIS
   Description: This service displays all the widgets of the requested page.

2. *Service name*: UpdatePage
   Service provided to: IIS, Computational Logic System
   Description: This service updates the requested page's widgets after receiving an update from the Computational Logic System.

3. *Service name*: DisplayPopup
   Service provided to: IIS
   Description: This service displays a popup with the corresponding requested popup.

4. *Service name*: UpdatePopup
   Service provided to: IIS, Computational Logic System
   Description: This service updates the requested popup after receiving an update from the Computational Logic System.

5. *Service name*: ClosePopup
   Service provided to: IIS
   Description: This service closes the requested popup.

6. *Service name*: sendGetRequest
   Service provided to: Computational Logic subsystem
   Description: This service takes user-updated information and sends it to the Computational Logic subsystem to be handled.

III.3.2.7.3.2. Services Required:

1. IIS
2. Bloc Subsystem

III.3.2.8. IIS

III.3.2.8.1. Description

The Internet Information Service is how the user gets their application page loaded from the program to their screen. Basically, users have to be connected to the internet in order to access the application, and the application's back end also has to be connected to the internet to serve users.

III.3.2.8.2. Concepts and Algorithms Generated

In terms of our project, this subsystem is a gateway/buffer between the user and the Presentation subsystem.

III.3.2.8.3. Interface Description

III.3.2.8.3.1. Services Provided:

1. *Service name*: sendData
   Service provided to: Presentation
   Description: This service sends data in the form of an event from the User to the Bloc subsystem.

2. *Service name*: receiveData
   Service provided to: Computational Logic
   Description: This service receives state data from the Bloc subsystem, to then display to the User.

III.3.2.8.3.2. Services Required:

1. Presentation
2. Computational Logic

III.3.2.9. User

III.3.2.9.1. Description

This subsystem is self-explanatory, but the user is anyone using the application. They have to be connected to the internet in order to use the application, and their device has to be an allowed device.

III.3.2.9.2. Concepts and Algorithms Generated

There are no concepts or algorithms present in the User subsystem.

III.3.2.9.3. Interface Description

The Interface Description is extremely simplified in this “subsystem”. There is user input and the user’s connection to the IIS.

III.3.3.

III.4. Data Design

(See Appendix B Image 1)

The **User** entity will hold every created user (both Faculty and Students) and will distinguish the different types of users with the **userType** attribute. This entity includes basic profile attributes such as **firstName**, **lastName, email**, **password**, and **profilePicture**. The **parentEmail** will be required for students as their parents will be constantly receiving a report of their child’s activity in the application. The **email** attribute is the primary key that the user will be able to log in with. The **User** entity has a many-to-many relationship with **PostSpace**. This relationship has a **Permissions** composite attribute, meaning every **User** will have unique **Permissions** in each **PostSpace**. These permissions include **canComment**, **canInvite**, **canEdit**, **canPost**, **canRemove**, and **canDelete**. The **User** entity also has a one-to-many relationship with **Post.**

The **PostSpace** entity is distinguished by the **spaceName** attribute and has basic attributes such as **spacePicture** and **spaceDescription**. The **isPrivate** attribute indicates

whether the Post Space can be discovered by the “search post space” function where users can request to join. The **PostSpace** entity has a one-to-many relationship with **Post**.

The **Post** entity is a weak entity that uses the **timePosted** attribute as a weak key. It uses **isPinned** and **canComment** to hold the settings of each post. Other attributes include **title** and **contents**. The **Post** entity has a one-to-many relationship with **Comment**. Since the **Post** entity is a weak entity, it is identifiable by the primary keys of the **User** and **PostSpace** entities along with its weak key.

The **Comment** entity is a weak entity that uses the **timePosted** attribute as a weak key. The only other attribute is **contents**. Since the **Comment** entity is a weak entity, it is identifiable by the primary keys of the **User** and **PostSpace** entities along with the weak keys of itself and **Post**.

III.5. UI Design

NOTE: Many of the designs here have now been depreciated over time during the implementation of the application. Over time, the client’s requests and the development team’s input have changed how the application looks/works.

Our team has created UI Mock-Ups for most of the different pages and boxes for our application. While these can give an idea of what the application may look like, pages are all subject to change as the development team or as the client deems fit. The UI Mock-ups are used to help guide development over deciding on the look of the application. Additionally, these mock-ups only show the Chromebook view, not the mobile view - they will look similar. It help establishes and determines links in a way that’s easier to come up with, from that visual perspective. Thus, the mockups should give you an idea of how the application will be used.

III.5.1. Login Page

(See Appendix C Image 1)

First, we have the login page. This will be the landing page when users first open up the application. The background is images of Wahkiakum County’s landscape but can be any image determined by the client. It can be any particular image, a simple graphic, a blank colored background, or a slideshow of many images. Regardless of which one is used, the images must not contrast in a way that is too distracting from the login block. The application name can be displayed in the corner.

Users will log in with their school-associated email, and a password. Students (whose accounts are created for them) will have to change their password from a temporary password to their own password, provided it meets all password requirements. As long as the email and password match in the database, they can log in. If incorrect, they return to the same page, with a corresponding message of “Incorrect username or password. Please try again”. Users can also press “Forgot your password” to go to a password reset page (see item V.2). Superusers can press “Superuser Login” to go to the Superuser Login page (see item V.3).

III.5.2. Forgot Password Page

(See Appendix C Image 2)

Users will come to this page from the Main Login Page (see item V.1). Users will be able to enter their school-associated email, and submit it to reset their password. We do not know which method we will take for this yet, but it will most likely end up back on the application and look like the password reset page (see item V.18).

III.5.3. Superuser Login Page*

(See Appendix C Image 3)

Users will come to this page from the Main Login Page (see item V.1). Users who have superuser privileges (IT, Ron, etc.) will be provided with a Superuser Key which allows them administrative privileges, and a different Superuser main page view. We do not have UI mock-ups of these pages as they are more specific, and will be completed in the second half (or later) of the development lifecycle.

III.5.4. Main Home Page

(See Appendix C Image 4)

The Main page is where you come after a user successfully logs in, from the Main Log In page (see item V.1). The application is split between the current page, the sidebar on the left side, and pop-outs for specific functions. The current page of the Main home page is the landing page of all users. It will contain the personalized Mecha-Mules home page, containing links that the client will provide. It can open those links into new browser tabs. The sidebar will contain a Home button (to return here), buttons for all the class spaces the logged-in user is a part of (see item V.10), and the user's profile page button (see item V.5).

III.5.5. Side Tab Extended

(See Appendix C Image 5)

Here, we can see what happens when you press the logged-in user's profile picture (in the bottom left). It will extend the sidebar (from taking ~1/8th to ~1/3rd of the screen). In this view, the user will see their profile picture, name, a logout button, a link to the notifications page, a link to edit their profile page, and a link to the settings page.

III.5.6. Mouse-over any Button

(See Appendix C Image 6)

Here, we can see what it looks like when you hold the mouse over any clickable link or button. It will display some text pertaining to its function. In this example, the mouse is over the New Class Space button, which when clicked will take the user to the Join/Create Class Space Popout (see item V.7).

III.5.7. Join/Create Class Space Popouts

(See Appendix C Image 7)

Here, we arrive here when the user clicks the New Class Space button from the Sidebar (see item V.6). The user can see one of these two different popouts, depending on the permissions given to their account. Faculty (or those with permissions) will see the right image, that is, they can create a class space in addition to joining a class space. To join a class space, users must have a class space code, which will be provided to them when invited by the space owner (see item V.14).

III.5.8. Mouse-over Profile Image

(See Appendix C Image 8)

Similar to Item V.6, this is a view of when the user mouses over their profile picture, in the extended profile sidebar (see item V.5). If they click this, they go to the profile popout (see item V.9), specifically their own.

III.5.9. View any profile popout

(See Appendix C Image 9)

We arrive here when the user clicks their own Profile Image through the extended sidebar, or any other user's profile image. The Profile Page popout is a popout that shows a user's profile picture, their name, their description, the class spaces they are a part of, and an edit profile button (if they are viewing their own), or edit/deletes profile aspects buttons (if they are a faculty/superuser viewing a student's profile).

III.5.10. Class Space Page

(See Appendix C Image 10)

We arrive at the Class Space when a user clicks the corresponding Class Space button from the sidebar. Here, users can see information about the class space, and the post history, where posts are placed. Posts have some comments (all can be viewed in the post view), and this view shows all posts filtered by the most recent. Any user can create a post. All users can

click view post, bringing them to the View Post Page (see item V.11). Users with administrative privileges can edit or delete any post. We are still considering if we want these buttons to be visible on this page, or only on the view post page.

III.5.11. View Post Page

(See Appendix C Image 11)

Users arrive at the View Post Page when a user clicks “View Post” from the Class Space Page (see item V.10). Here, they can see Post details such as the title and post date, the full description, and all comments. Additionally, there is a Post Comment text box, where users can post a comment response to the current post. Users with administrative privileges can edit or delete any comment, or the current post (see item V.12).

III.5.12. Delete Post (or Post Space) Popout

(See Appendix C Image 12)

Users can delete their own posts, post space, or comment. Users with administrative privileges can delete any post, post space, or comment. Users arrive here when a user clicks “Delete Post”, “Delete Comment”, or Delete “Post Space”. Here, the users have to confirm the deletion before the database is updated and the post/post space/comment is deleted.

III.5.13. Mouse-over Member Count

(See Appendix C Image 13)

Similar to Item V.6, this is a view of when the user mouses over the member count in the Class Space view (see item V.10). If they click this, they go to the view member list pop out (see item V.14).

III.5.14. View Member List Popout

(See Appendix C Image 14)

Users arrive here when they click the member count from the Class Space view. This displays the list of members in the class. Here, any user can view the profile popout of any other user, or message them. Users with administrative privileges can invite/add new users, or remove users from the space.

III.5.15. Remove User Popout

(See Appendix C Image 15)

Users with administrative privileges arrive here when they click “Remove User” from the View Member List Popout. They have to confirm that they want to remove the user from the space.

III.5.16. Search for/add User Popout

(See Appendix C Image 16)

Users with administrative privileges arrive here when they click “Add User” from the View Member List Popout. They get to a list of all users in the database, where they can add users to their space, or view user profiles. The list can be filtered (by year, by name, etc.).

III.5.17. Edit Profile Page

(See Appendix C Image 17)

Users arrive at this page when they click “Edit Profile Page” from the Extended Side Tab (see item V.5). This allows users to change their profile picture, their description, and their password, in addition to text talking about how their posts/comments/profile pictures/profile descriptions can be edited by users with administrative privileges at any time. There is a “Change Password” button (see item V.18) and a “Change Profile Picture” button (see item V.19). Administrative users who are editing a student’s profile page will have a similar page.

III.5.18. Change Password Popout

(See Appendix C Image 18)

Users arrive at this popout when they click “Change Password” from the Edit Profile Page (see item V.17). They must provide their old password, their new password, and their new password again to update it. Checks will be performed to verify all this information. Users can click Save Changes. If there are mistakes or errors, a corresponding error message will be displayed. If no errors or mistakes, users return to the previous page, and the database is updated.

III.5.19. Change Profile Picture Popout

(See Appendix C Image 19)

Users arrive at this popout when they click “Change Profile Picture” from the Edit Profile Page (see item V.17). They can upload a picture file to display as their profile picture. Users

click save changes to update their profile picture in the database and return to the previous page. Administrative users who are editing a student's profile page will have a similar page.

IV. Testing Plan

IV.1. Testing Strategy

To achieve the most success in meeting our testing objectives, we will loosely follow a general testing lifecycle, as shown below:

IV.1.1. **Developer writes code:** This will include the implementation of the various subsystems of our software project. These include the Controller, Authentication, Computational Logic, Model, Data Access Logic, Google Cloud SQL, and View subsystems.

IV.1.2. **Developer writes tests for code:** Here, the actual tests for all the functions of the software subsystems are written.

IV.1.3. **Developer runs tests for code:** Test programs are run on the program to assess the effectiveness of the code against the requirements of the project.

IV.1.4. **If tests fail, fix bugs in code and retest, otherwise continue:** If a test case fails, that means the code has bugs, and it is not ready for development. Developers will repeat steps 1-3 until no bugs are found. Once all test cases pass, developers can proceed to the next phase.

IV.1.5. **Developer pushes code to GitHub, where CI/CD testing occurs. CI runs code through its pipelines to test if it is eligible to continue to the next step:** In this next step, Continuous Integration and Continuous Delivery occur, to ensure newly developed functionality or features do not conflict with previous changes. If CI/CD testing fails, developers repeat steps 1-4, until it passes, and can move to the next phase.

IV.1.6. **Developer creates a merge request to the master branch from their development branch, with all team members added as reviewers:** Developer creates a merge request through GitHub, and the other team members are added as reviewers.

IV.1.7. **Requests require at least 1 other team member to approve before the merge can be complete:** The feature branch must be approved by at least one other team member before it can be merged back into the main branch. If no team members approve, the developer must repeat steps 1-6.

IV.1.8. **Upon approval, code is merged into the master branch and deployed by the CD, if there are no pipeline errors:** Code is merged from the development branch into the main branch, and is deployed through CI/CD.

During testing, if a developer encounters a new bug, they must create a GitHub issue describing the bug. The same developer is responsible for fixing the bug and creating new tests relating to the bug. This allows those developers who are most familiar with certain system functions to specialize their efforts into those specific functions.

We will be utilizing the CI/CD development pipelines strategy, in which there is continuous development, testing, and delivery of new code.

IV.2. Testing Plans

IV.2.1. Unit Testing

The primary goal of unit testing is to take the smallest unit of testable software in the application, isolate it from the remainder of the code, and test it for bugs and unexpected behavior.

For our application, it seems like the best approach for unit testing is the standard approach. We can have two or more tests for each important/non-trivial function in the code. Potential tests can include valid function usage, and invalid function usage, among other function-specific tests or edge-case tests. Obviously, the more complex the function, the more unit tests should be linked to it. Conversely, trivial functions, such as doing very simple operations or calling other methods, should not require any testing.

We should factor in relevance and complexity when determining how many unit tests per function, thus we will use these two metrics as described: Relevant methods are those used a lot, or that if broken, would result in a severe decrease in Confidentiality/Integrity/Availability of the application. Complex methods are those with many potential uses and many internal execution paths. It will be up to a developer's individual discretion to determine how many tests a function they wrote deserves.

IV.2.2. Integration Testing

Integration testing detects faults that have not been detected during unit testing by focusing on small groups of components. Two or more components are integrated and tested, and when no new faults are revealed, additional components are added to the group.

In regard to our application, integration testing would be more complex than the basic and general unit tests. We will generate integration tests based on entry points. These entry points are the beginning of some application functions, and the test is testing that everything works as intended. An example would be logging in. The entry point is logging in, yet the tests are much more complex since a multitude of methods and functions are performed to complete the process of logging in.

Choosing where these entry points are, and which entry points are more important (based on relevance and complexity) will be up to the discretion of the development team. A more important entry point may be one where there are multiple tier 0 system requirements being fulfilled.

With the integration tests, it seems more important to test whole classes rather than individual methods. This helps by reducing the complexity of creating test cases, while still promoting high code coverage.

IV.2.3. System Testing

System testing is a type of black box testing that tests all the components together, seen as a single system to identify faults with respect to the scenarios from the overall requirements specifications. The entire system is tested as per the requirements.

During system testing, several activities are performed:

IV.2.3.1. Functional testing:

Test of functional requirements (from requirements specification). The goal is to select those tests that are relevant to the user and have a high probability of uncovering a failure.

In regard to our application, we will rely on our Requirements and Specifications document. In that document, we specified all the functional and non-functional requirements of our application. For functional testing, we will simply take all our functional requirements, and build functional system tests around each of those requirements. This will be logged in a document, including the steps taken in the system to carry out that function. All events of system restructuration that change the operations of functions will be written in this document. All data of failed tests will be recorded in the Results section with relevant information that identifies the problem within the function test. The developer that performed this test should then create an issue within the team's repository and assign a team member who is responsible for that specific task. That team member will engineer a solution towards that task by writing standardized tests to exterminate error results that could occur in the future.

IV.2.3.2. Performance testing:

Performance tests check whether the nonfunctional requirements and additional design goals from the design document are satisfied. In stress testing, the system is stressed beyond its specifications to check how and when it fails.

Performance testing will be executed in a similar manner to functional testing. We will be stressing the speed, scalability, and stability of the system under a specified workload. The tests should be based on specified variables in the Requirement and Specifications document and will meet the non-functional requirements of the system. The developer should be able to analyze the results of these requirements and give feedback using their own intuition. A document will be created with three sections: Non-functional requirements, steps to test/satisfaction criteria, and results. In the case that a developer feels that the system fails one of these requirements, they must identify what needs to be changed within the application to meet that requirement. Like a functional test, that developer must create an issue within the team repository and assign it to a member that's responsibilities fall under this type of error.

IV.2.3.3. User Acceptance Testing:

User acceptance testing will be exercised constantly throughout the development of the project. It will be similar to black box testing where two or more end users are stressing the application in a practical manner. This will allow the clients to evaluate the system and validate the flow of the application. Developers must also have expectations within mind that are based on the requirements in the Requirements and Specifications documents. The developer will produce a task agenda to ensure that these requirements are satisfied and recorded for improvement or fixing. This testing will also let clients explore new avenues of testing that have not already been performed by the developers. As user acceptance testing continues, the developers should show-and-tell less and let the client use the application until they have a question or request that will be documented. The goal is to have a staff member of a school district be able to train others on how to use the application by themselves. Like the previous testing, if a bug is determined or a request for improvement is made, the developer must create an issue within their team repository and assign it to the most relevant developer. That developer will be responsible for fixing that bug or creating that feature.

IV.3. Environment Requirements

Our testing environment will utilize a few different tools to enable Team Toto to best test the project. For unit testing, we will make use of Flutter testing packages, including the test and flutter_test packages. These two allow unit and integration tests to be created for any functions needed for testing.

In addition, GitHub has Continuous Integration available which our team will utilize to make sure that all of our tests are run, and passed, prior to merging any commits made by the team.

Finally, the hardware that will be used for the testing environment will be a Chromebook with standard school specifications. Since the application will run mostly on Chromebooks, we can physically test on Chromebooks to ensure that requirements falling within the User Requirements section are satisfied.

V.Alpha Prototype Description

As of now, we are making good progress on our project. A bulk of our project and an initial roadblock to expanding the application was the creation of its foundation, taking us longer than expected since we also had to learn how to use the chosen frameworks, tools, and libraries. This included setting up the database, writing our data, and configuring how everything is stored in the database. We also finished Login, Logout, and registration of user accounts. The UI is in progress, but will continue updating as progress on the backend is made.

The Firebase database, its connection to our application, and all the entity relationships and model tables are all connected, interact with each other, and store user data and space data.

In addition, the creation of class Spaces has been implemented. While the viewing, editing, and deleting of the class space is still in progress, the space is being created in the back-end, and being stored in the database

V.1. Create Student Account

V.1.1. Functions and Features Implemented

The ability to create student accounts has been implemented it. As of now, there are two ways to create student accounts. First, you can manually create and add a student account directly through Firebase. Here, you can enter in name, email, and password, and it will be added into the database, for use by the application. Another way to create student accounts is through the application, from an Superuser account. These accounts have extra options in the sidebar, including a “Create student account” button. Once clicked, they are taken to a window that makes them create a student account, with the same fields, and adds the new user account to the database.

(See Appendix D - Images 2, 3)

V.1.2. Preliminary Tests

To unit test this feature, we use a test case where the expected result is a successful creation of a student account in the database and compare it to the actual event sent to the business logic.

V.2. Create Faculty Account

V.2.1. Functions and Features Implemented

The ability to create faculty accounts has been implemented. Since these accounts have special administrative privileges, we decided to only let this function be accessible directly through the database.You can create the account in the database, and they will be added into the database, with special application privileges.

V.2.2. Preliminary Tests

To unit test this feature, we use a test case where the expected result is a successful creation of a faculty account in the database and compare it to the actual event sent to the business logic.

V.3. User Login

V.3.1. Functions and Features Implemented

The ability for users to log into the program has been implemented. Users must provide the email and password, these are checked against database values and other checks, and if all is successful, logs the user in.

(See Appendix D - Image 1)

V.3.2. Preliminary Tests

Since the feature was created using the BLoC pattern, we used the bloc_test dart package to test it. In this test, we check if the correct state is emitted from the business logic section to the presentation layer by using specific test cases of the email and password to send the expected event to the business logic. Depending on the test case of the login, the application sends an event back to the business logic and the test compares it to the expected results which should be in the form of ‘SubmittingForm’, ‘SubmissionSuccessful’ and ‘SubmissionFailed’. These are located in the test folder to the login folder, labeled as login_bloc_test.dart, login_event_test.dart and login_state_test.dart.

V.4. User Logout

V.4.1. Functions and Features Implemented

The ability for users to logout of the program has been implemented. When logged in users press the logout button, their account is signed out, and they are taken to the login screen.

(See Appendix D - Image 1, 2)

V.4.2. Preliminary Tests

Since the User Logout feature is already implemented in the LoginBloc it is already tested along with the initial login feature.

V.5. Delete Account

V.5.1. Functions and Features Implemented

The ability to delete users from the database is present and doable within the database. Since this is a function with consequences for the application, it will only be doable within Firebase. We do not want a teacher or space creator to somehow have the ability to delete a user from within the application.

V.5.2. Preliminary Tests

To unit test this feature, we use a test case where the expected result is a successful deletion of the account in the database and compare it to the actual event sent to the business logic.

V.6. Create Space

V.6.1. Functions and Features Implemented

The ability to create a space has been implemented. Users can create a Space, which is created and sent to the database. The user is automatically linked to the space as the owner.

So far, we have only implemented the creation of spaces, and not much else. In (Appendix D - Image 2) you can see that we are displaying the id of the space, created by the currently-logged in user.

V.6.2. Preliminary Tests

For the create space feature, we do this by bloc_testing the createSpace bloc and verify the submissions are successively created. There are test cases where the test expects a submissionFailed or submissionSuccessful. We then unit test the createSpace_event and createSpace_state to verify that the states and events are correctly routed.

V.7. Administrative Privileges

V.7.1. Functions and Features Implemented

The ability for administrative differences to be displayed to the application has been implemented, and is a function that will continue to be implemented further during the development of the application.

(See Appendix D - Image 4)

V.7.2. Preliminary Tests

We unit test the administrative privileges by using test cases where events sent to the business logic are compared to the expected events.

V.8. Application Side Bar

V.8.1. Functions and Features Implemented

The visual sidebar for every page within the application has been added. This sidebar contains buttons and links to the Home Page, Create a Student Profile (Superuser only), Create a Space (Superuser only), Notifications (No function yet), Settings (No function yet), and Logout. In addition, the sidebar will contain buttons/links to the user's spaces that they are connected to. This functionality is currently in progress, as we only display the firebase ID of the space, with nothing else visually.

(Appendix D - Image 2)

V.8.2. Preliminary Tests

We widget test the application side bar by testing the onTap() function of the navigaition_drawer.dart to verify that the correct event state is sent to the business logic. These functions are CreateSpaceView(), HomeView(), LoginView() and RegisterView().

V.9. Home Page

V.9.1. Functions and Features Implemented

This functionality, allowing for the home page to be linked to the actual web page of the client's school district site has been implemented. It uses a flutter library which allows it to connect to any given webpage, and embed it within the home page of the application. This is able to be changed.

V.9.2. Preliminary Tests

To test this feature, we use the flutter_test dart package provided by Flutter. We widget test homeView.dart file. We compare the event sent to the business logic to the expected link page state. We also test that the submission is valid to ensure that the feature is functioning. This is also tested in application side bar test.

V.10. Firebase

V.10.1. Functions and Features Implemented

Firebase is the (cloud-based) database we are using to house the data of our application. While we have to modify it and backend code every time we implement a new feature, it is fully updated and working with the rest of our functions. It stores data, their properties, and the dependencies to other data.

V.10.2. Preliminary Tests

For the database, we unit test the data model of the application. The repositories of the authorization data and space are already tested in the login and createSpace features. The userData is tested by verifying if the submitted user data is modified and sent to the data model of the application.

VI. Alpha Prototype Demonstration

VI.1. Mentor Demonstration

Our mentor meeting and demonstration occurred on December 6th. At that point of development, we had completed Login, Logout, Account Registration, Firebase setup and implementation, along with the UI to demonstrate it all.

The mentor stated that our progress was satisfactory. He also reminded our group of the requirements for the testing portion of the application.

VI.2. Client Demonstration

The meeting between our development group and our client occurred on December 9th. At this point, in addition to all previous work, we also finished the implementation of the web page view on the home page, and the creation of Spaces (again, only the creation).

The mentor and the team discussed the plan for the rest of the year, and going into the next semester. This plan included our development plan for sprint 4/over break, and how we could obtain the chromebook (for physical testing on the desired platform).

The client appeared to approve and like the design and the progress of our application, and hoped to have a solid version of the application for hands-on testing by Februrary.

VII. Future Work

We have a primary function that still needs to be completed, which is Post Spaces, Posts, and Comments, to be completed in that order. Post spaces will hold Posts, which will hold comments. Due to the complexity of these, a bulk of our time will be spent working on this aspect. Our plan is to work on this over break.

Other remaining features include Settings pages, profile pages, administrative pages/privileges, and other smaller things throughout the application.

VIII.Glossary

Administrative Moderation - Faculty users with administrative accounts have extra permissions that student users do not, such as editing and deleting other users' posts. These accounts also have permission to moderate the behavior of student users.

Authentication - A form of validating a user's ID and password are within the database and correct. Once validated, users can log in.

Back-End - The server side; Everything necessary for an application that users do not see, including code/scripting, databases, software architecture, security, etc.

Cross-platform - Software that is designed to work on several computing platforms. In this application, software must work on both Chromebooks and mobile devices.

Chromebooks - Runs using Chrome OS. The main platform students use and the main platform for this project.

CIPA Compliance - The Children's Internet Protection Act our application must comply with.

Database - An organized collection of structured information, or data, typically stored electronically in a computer system.

Discord Voice Communication - A very popular VoIP (Voice over Internet Protocol) and instant messaging social platform that the school previously used.

Front-End - The client-side; everything that users actually see on an application, including the web pages, performance, UI, etc.

Full-Stack Application - The entire depth of an application, both front-end and back-end.

Software Architecture - The fundamental structures of a software system and the discipline of creating such structures and systems

Stakeholders - Individuals, groups, or organizations directly involved with, or indirectly affected by, a project, product, service, or enterprise.

UI (User Interface) - The space where interactions between humans and applications occur; all the physical buttons, text boxes, etc. that users interact with within an application.

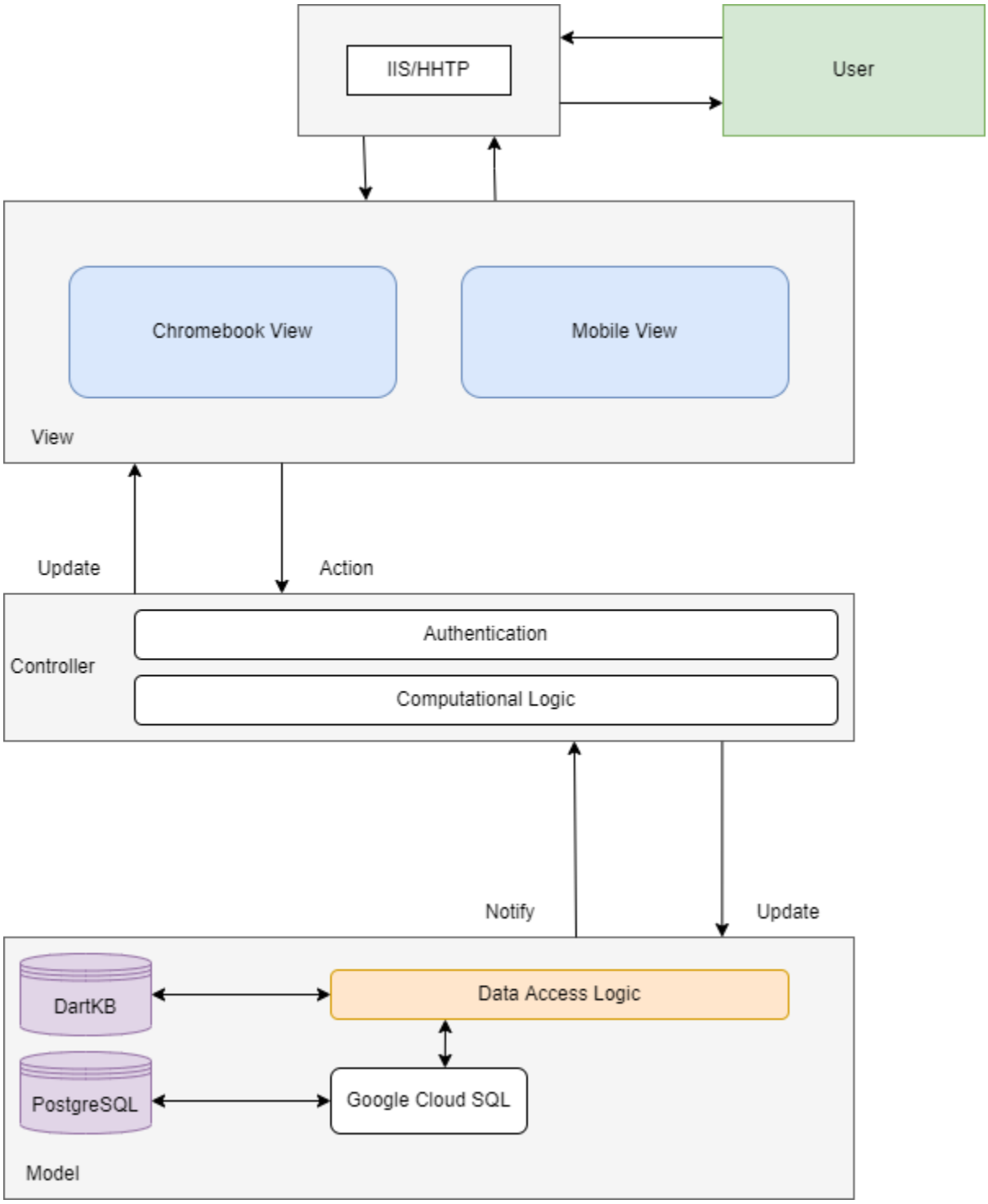
User - Any entity that has an account on the application. Users will have specific permissions depending if they are faculty or student.

User Records - Data of user's posts, comments, space, and profile. These are all stored on a database used for functions throughout the application.

IX. References

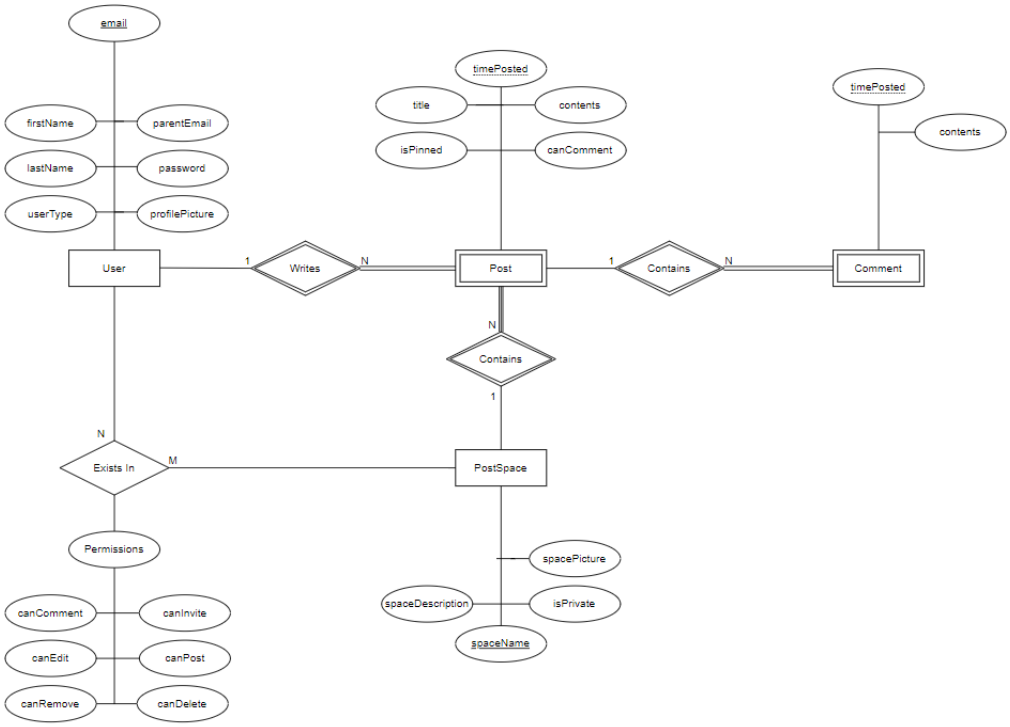
X. Appendices

X.1. Appendix A - Software Architecture



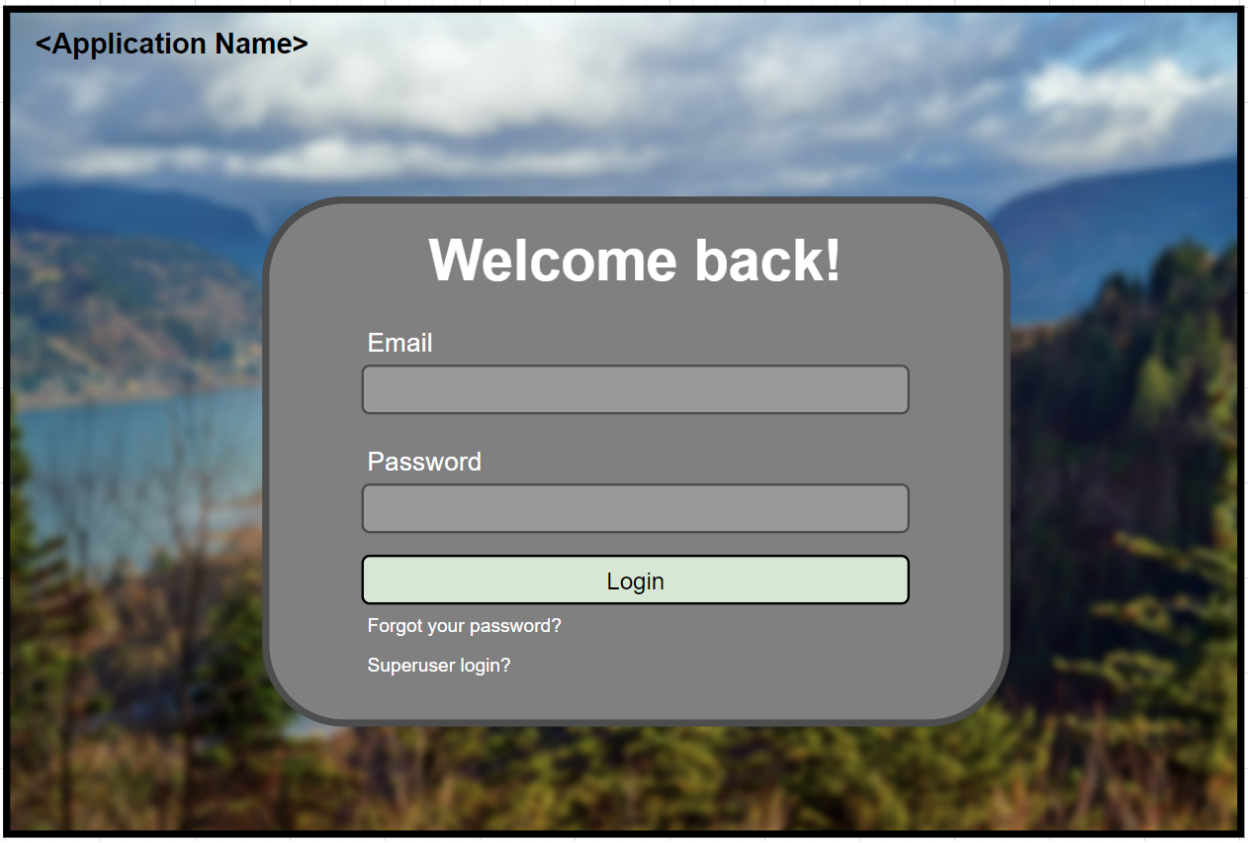
(Image 1)

X.2. Appendix B - Data Design

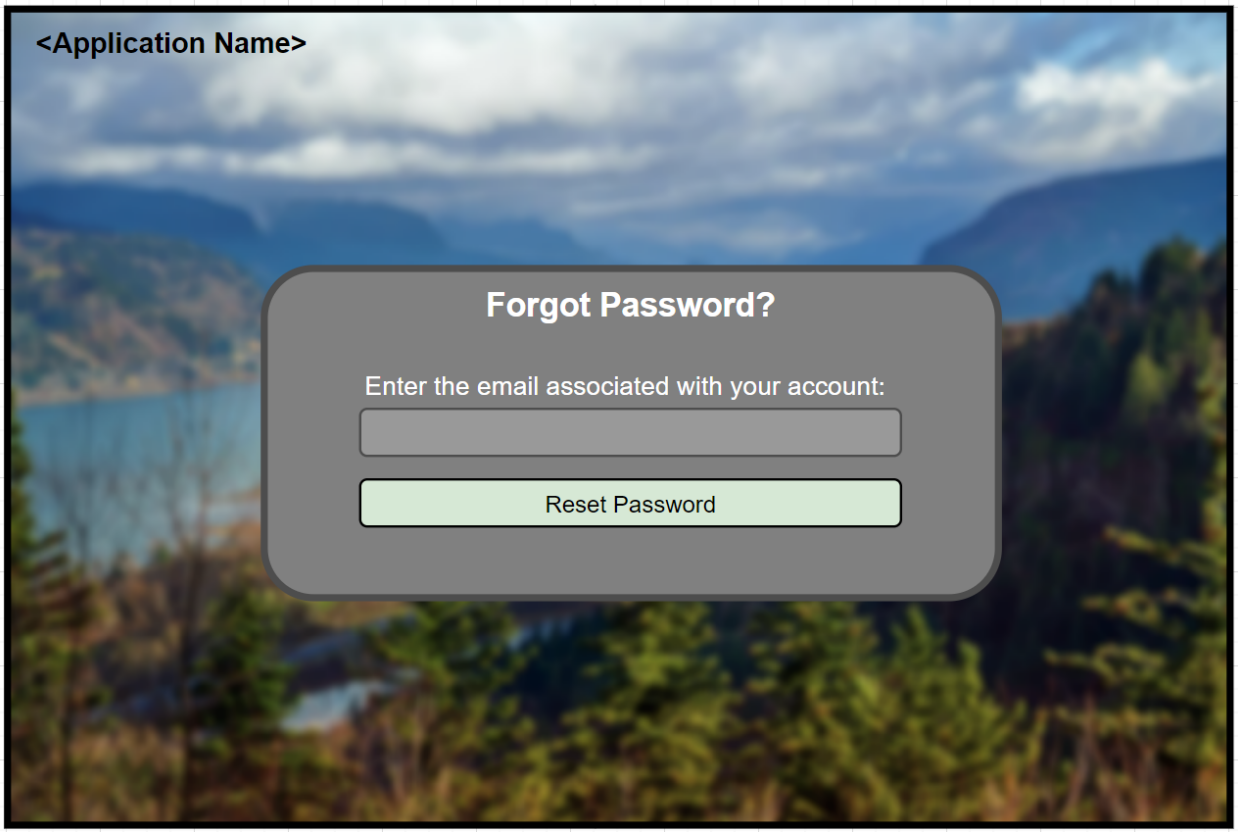


(Image 1)

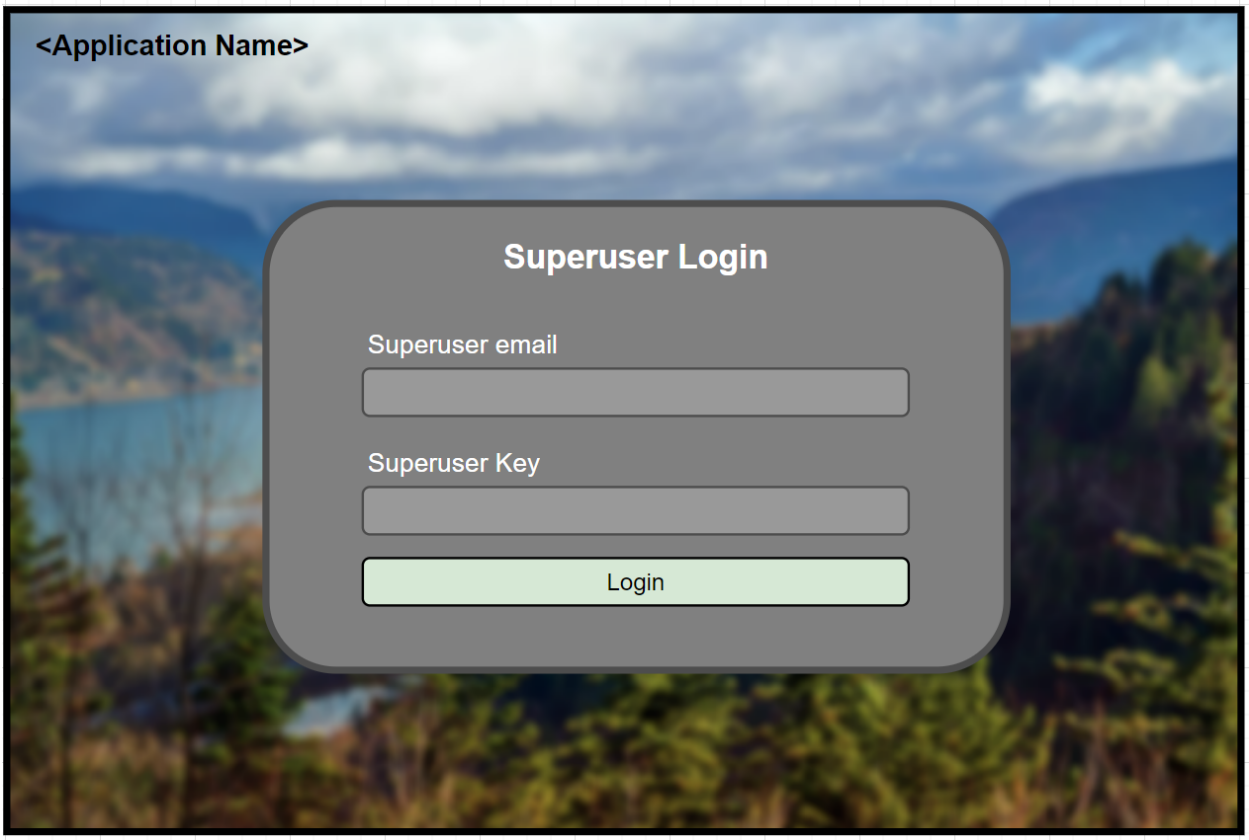
X.3. Appendix C - Initial User Interface Design



(Image 1)


<Application Name>
Forgot Password?
Enter the email associated with your account:
Reset Password

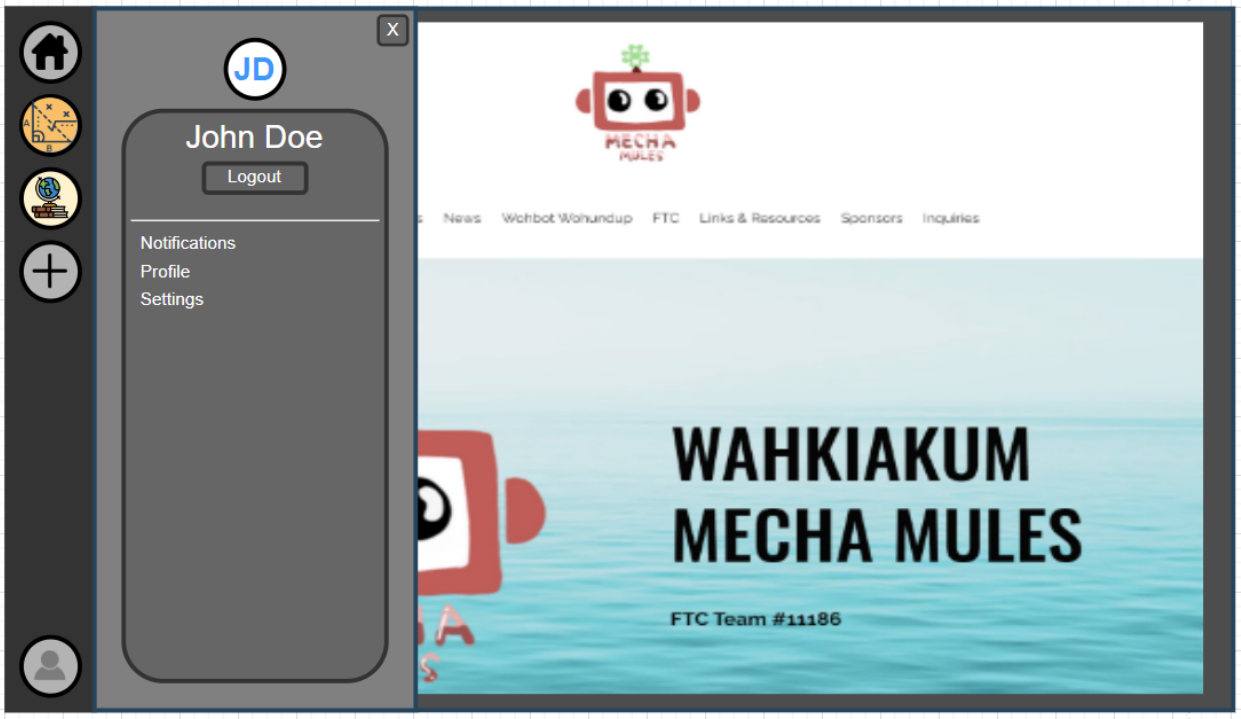
(Image 2)


<Application Name>
Superuser Login
Superuser email
Superuser Key
Login

(Image 3)

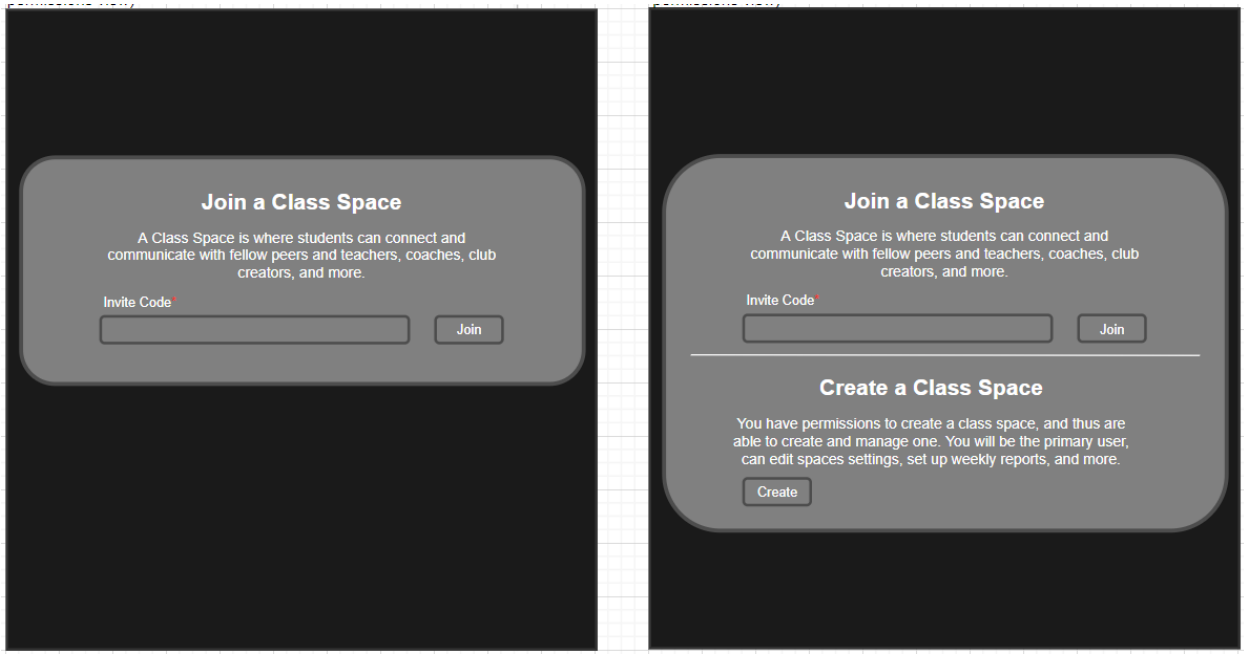

Home
About Us
News
Wohbot Wohundup
FTC
Links & Resources
Sponsors
Inquiries
MECHA
MULES
WAHKIAKUM
MECHA MULES
FTC Team #11186

(Image 4)

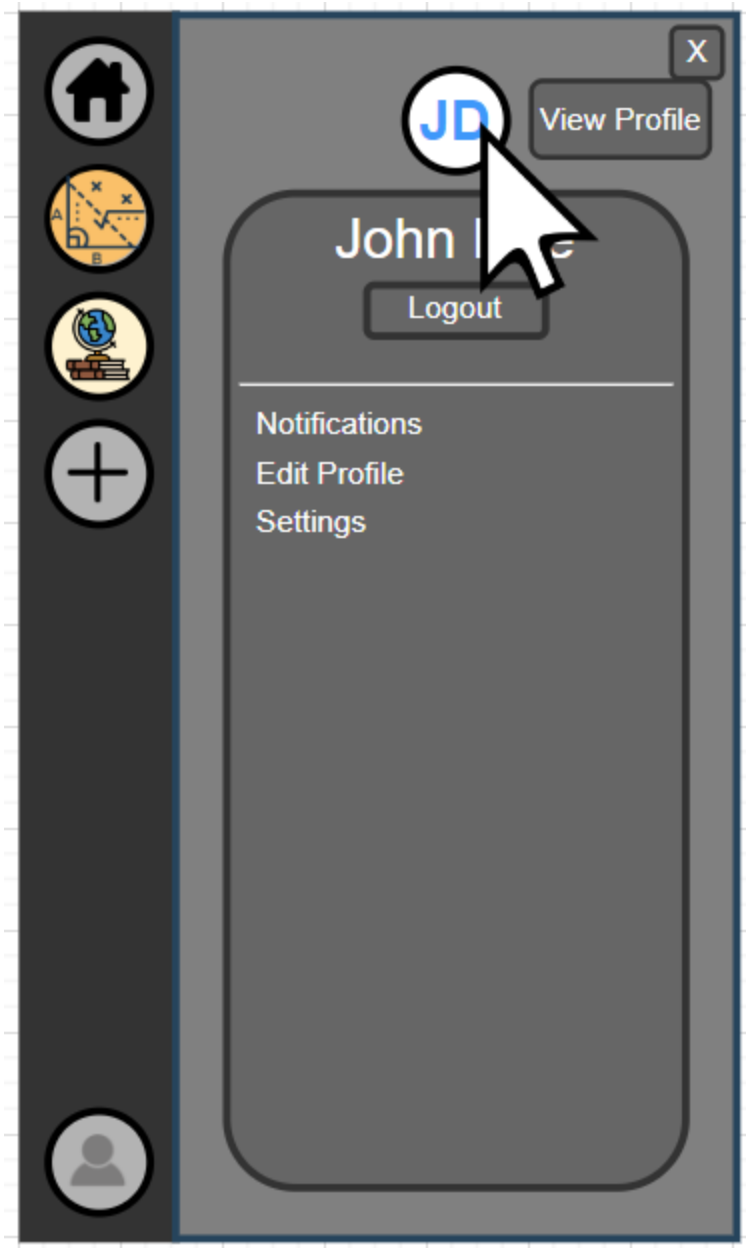


(Image 5)

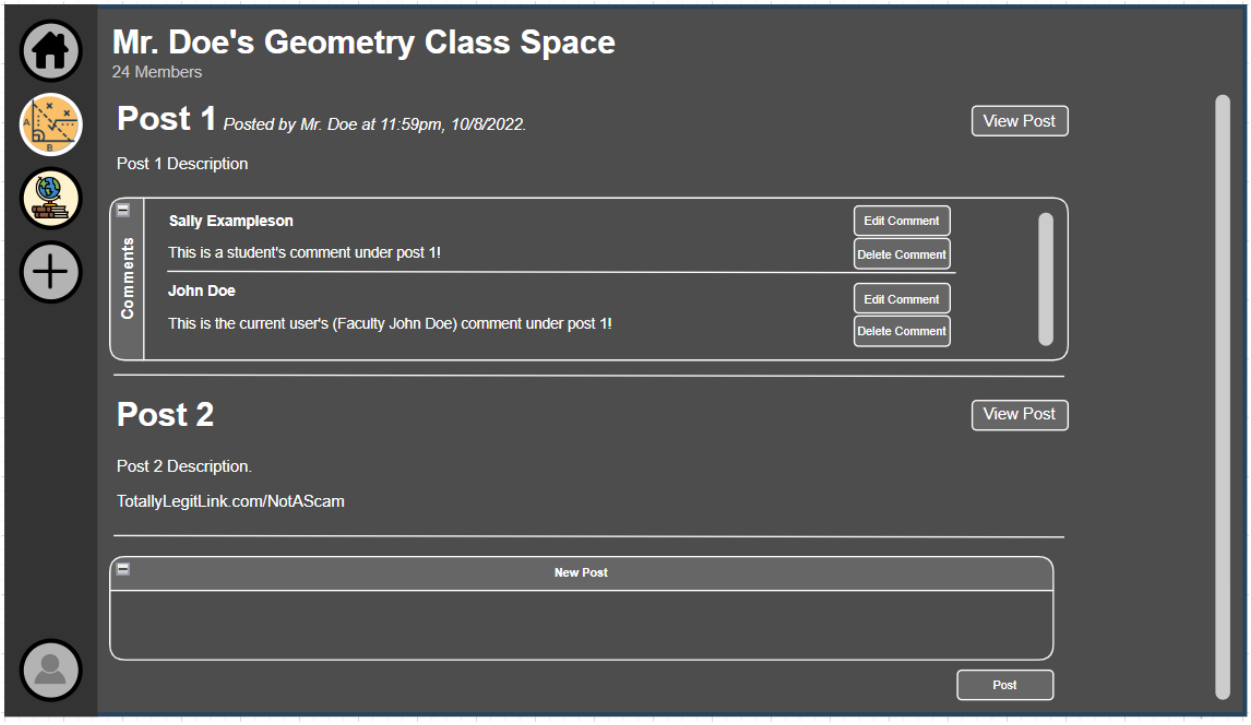
(Image 6)


Join a Class Space
A Class Space is where students can connect and communicate with fellow peers and teachers, coaches, club creators, and more.
Invite Code*
Join
Join a Class Space
A Class Space is where students can connect and communicate with fellow peers and teachers, coaches, club creators, and more.
Invite Code*
Join
Create a Class Space
You have permissions to create a class space, and thus are able to create and manage one. You will be the primary user, can edit spaces settings, set up weekly reports, and more.
Create

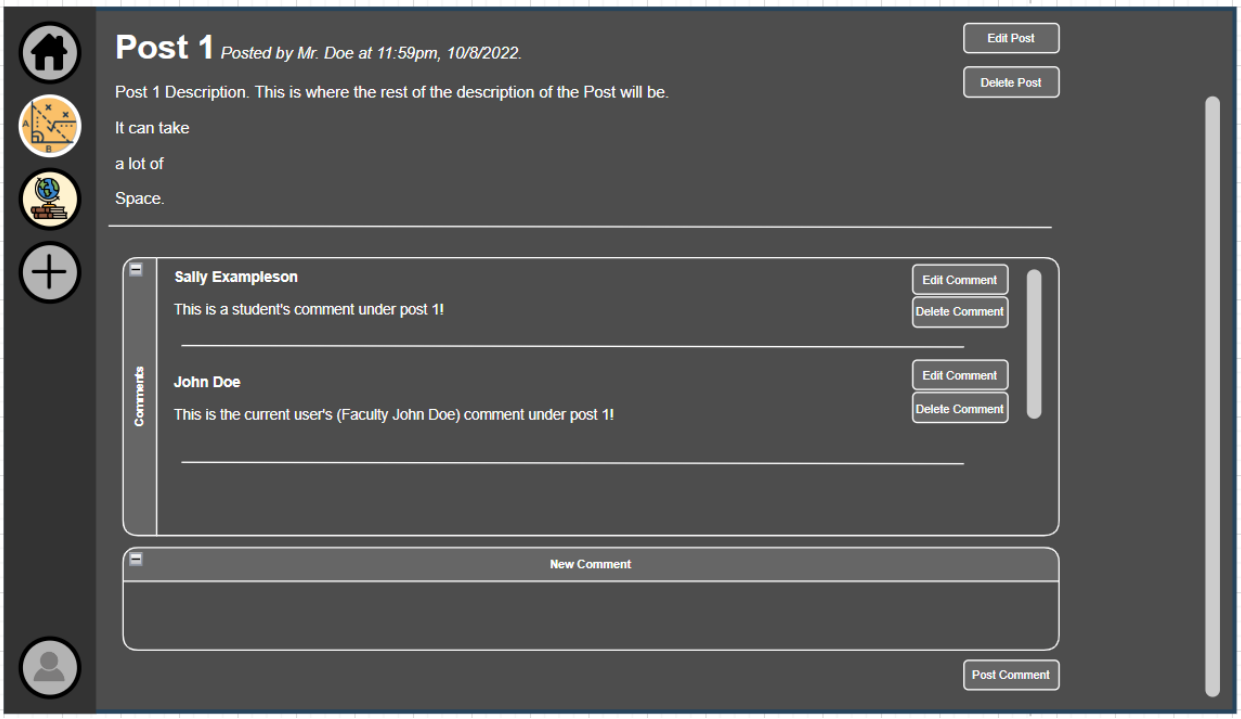
(Image 7)

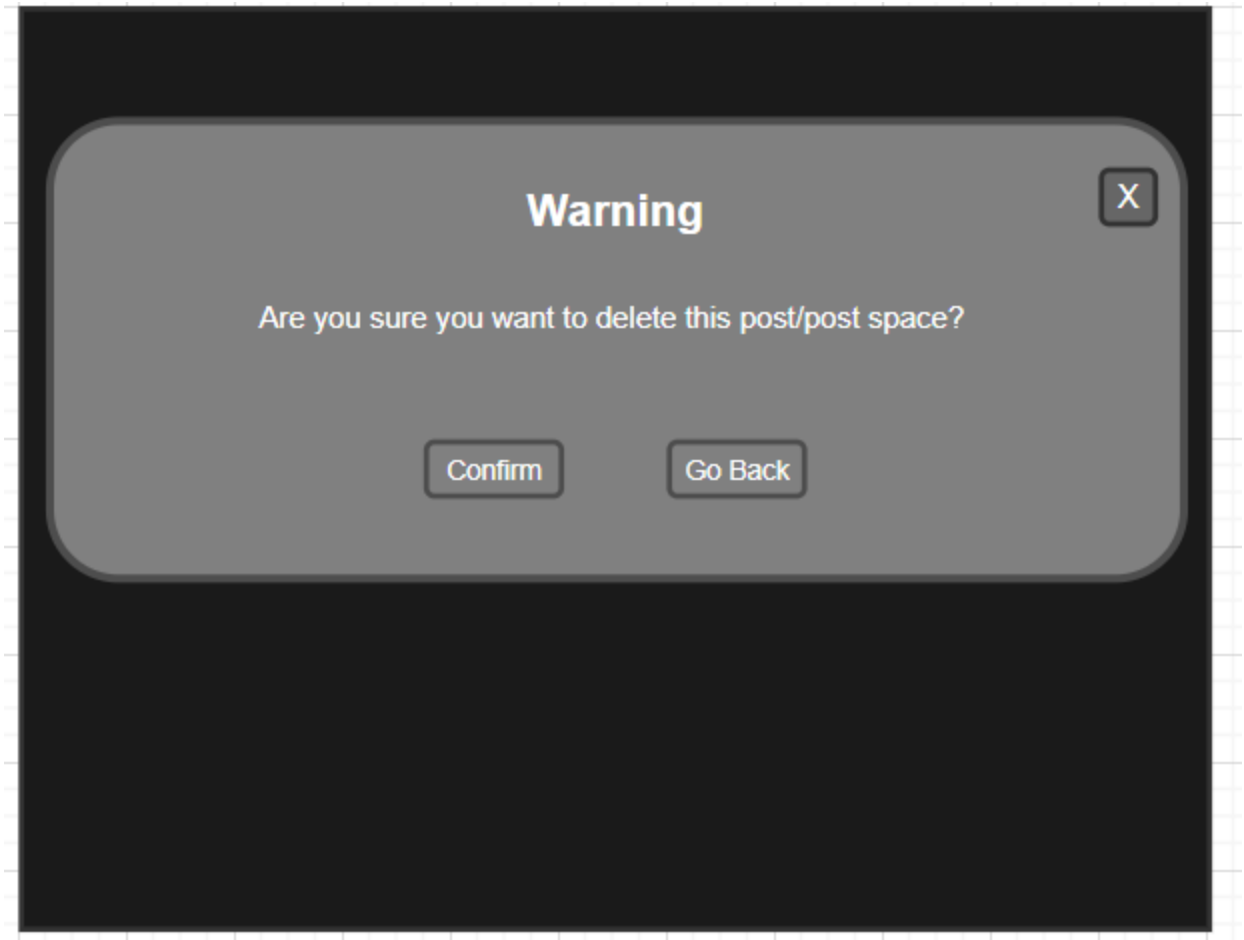

X
JD
View Profile
John
Logout
Notifications
Edit Profile
Settings
(Image 8)

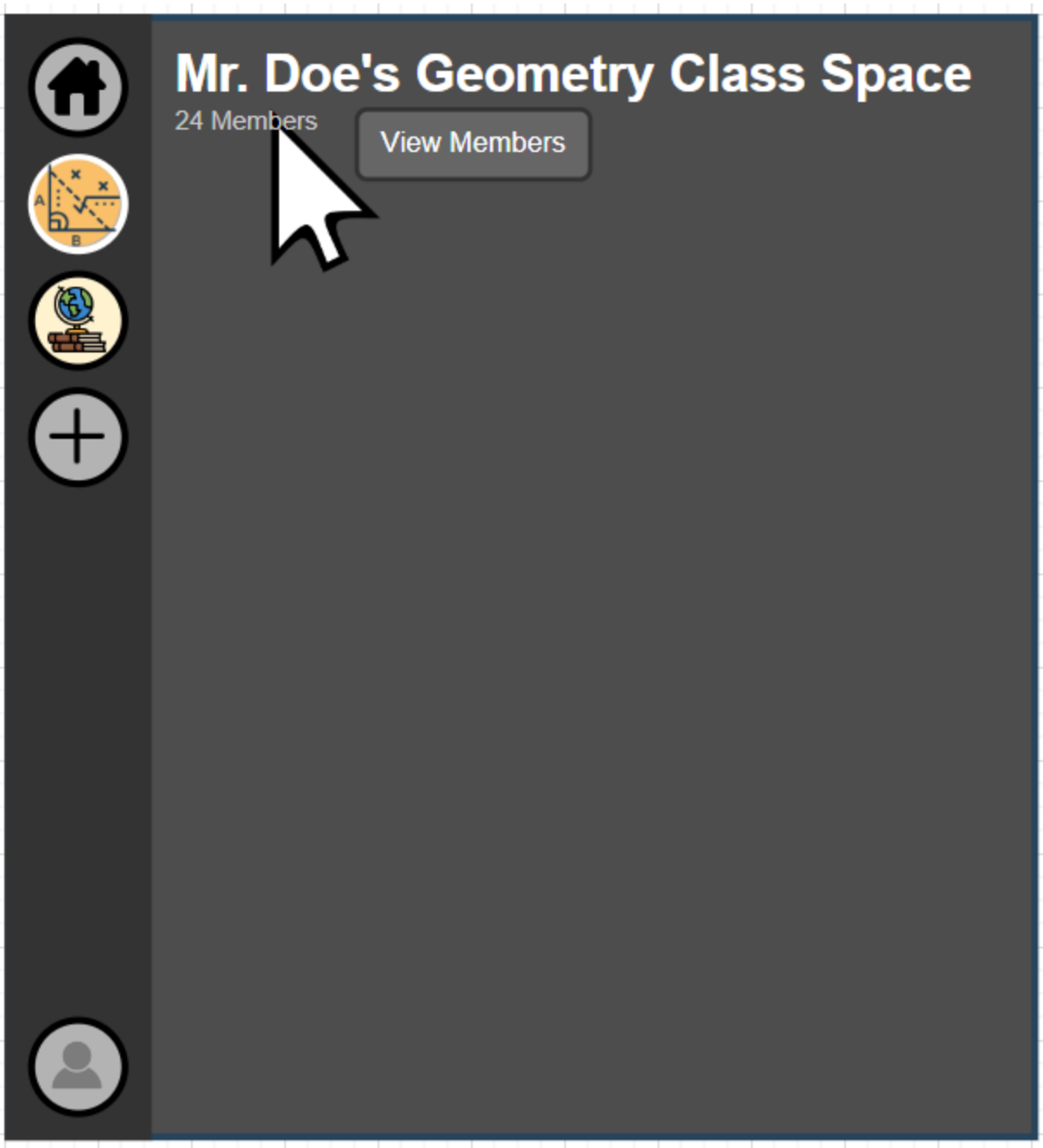
(Image 9)



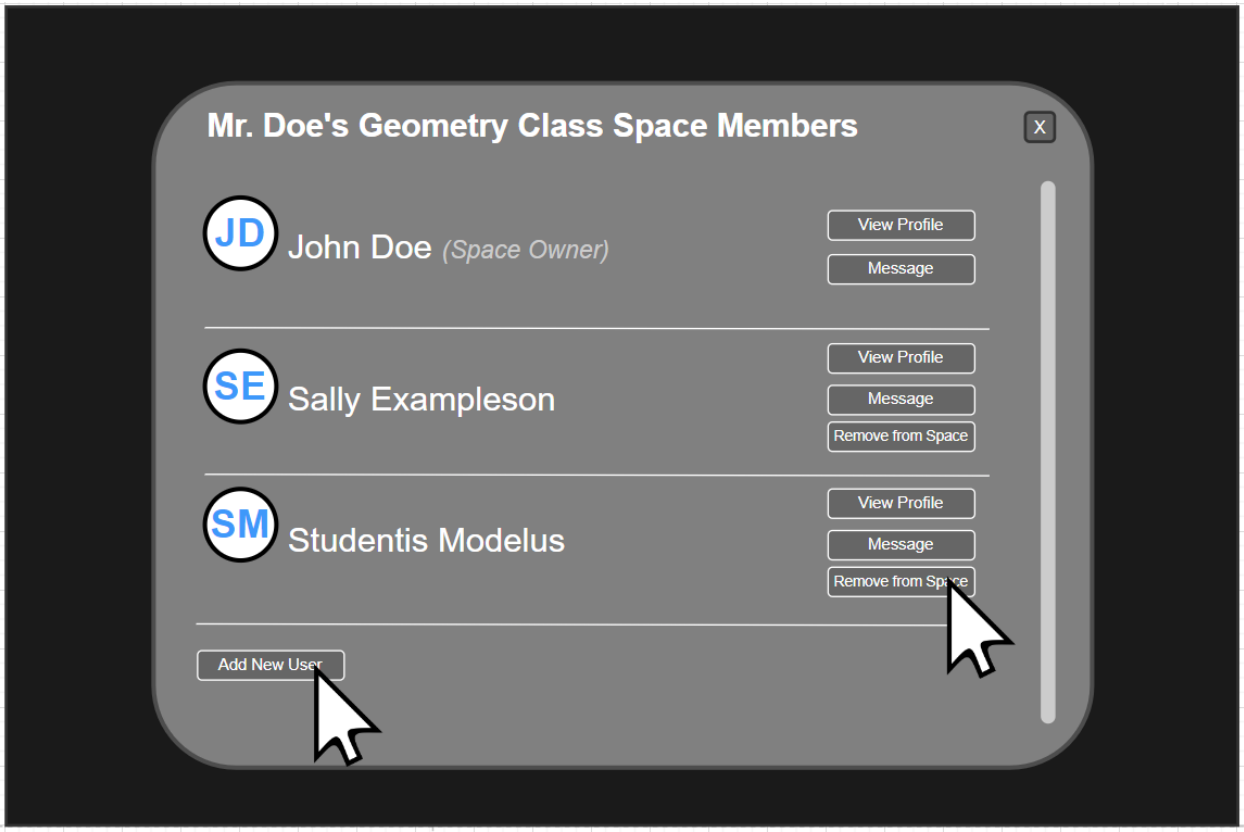
(Image 10)


Post 1
Posted by Mr. Doe at 11:59pm, 10/8/2022.
Edit Post
Delete Post
Post 1 Description. This is where the rest of the description of the Post will be.
It can take
a lot of
Space.
Comments
Sally Exampleson
This is a student's comment under post 1!
Edit Comment
Delete Comment
John Doe
This is the current user's (Faculty John Doe) comment under post 1!
Edit Comment
Delete Comment
New Comment
Post Comment
(Image 11)

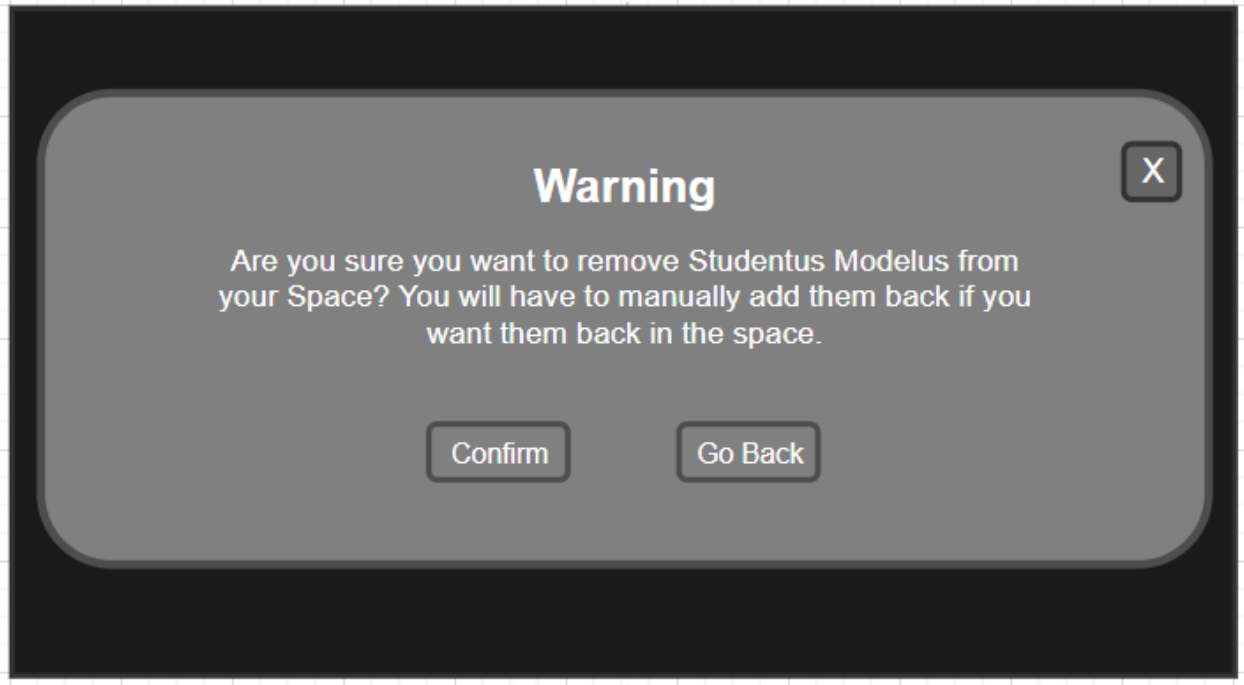

Warning
X
Are you sure you want to delete this post/post space?
Confirm
Go Back
(Image 12)



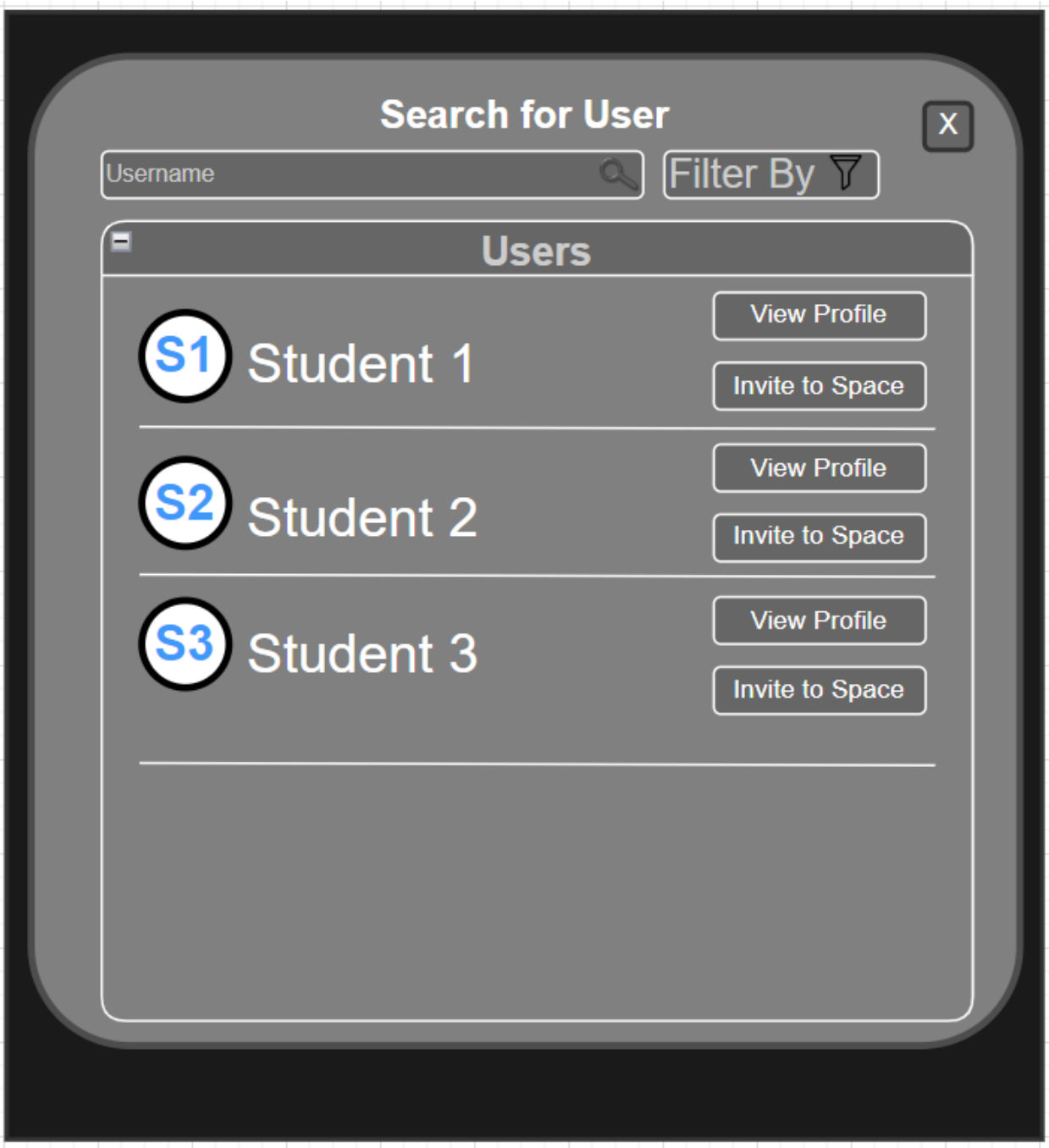
(Image 13)



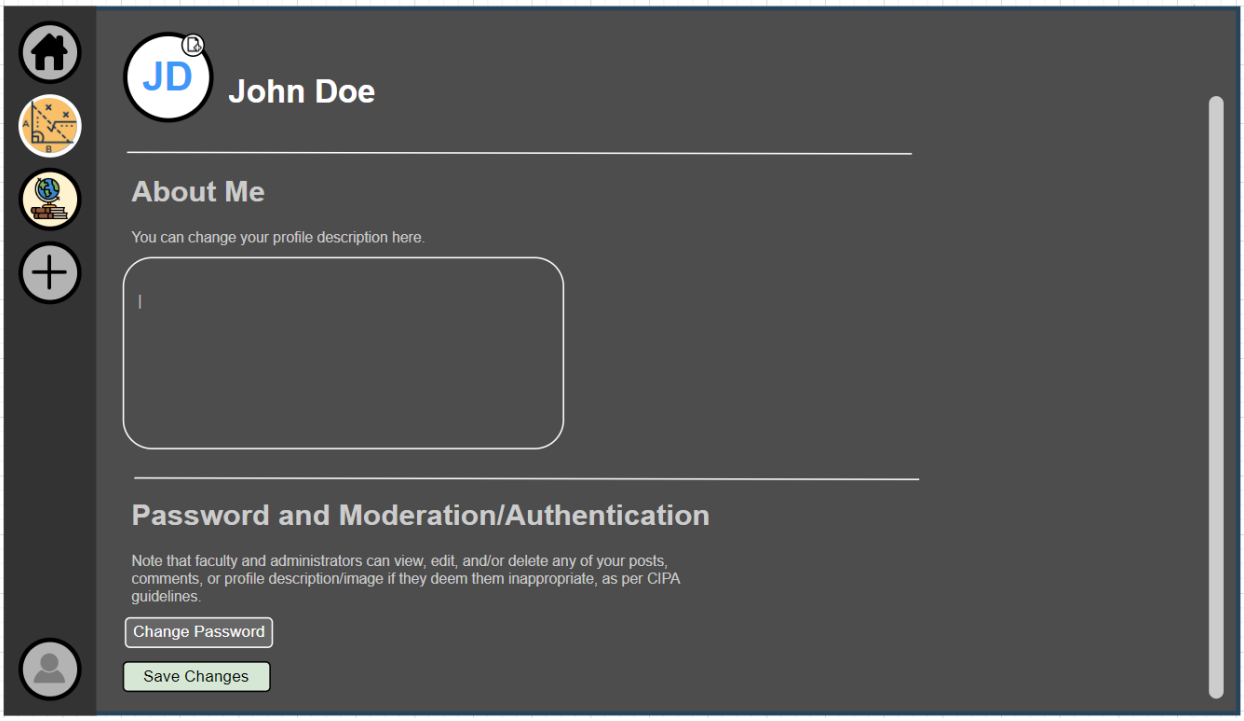
(Image 14)



(Image 15)


Search for User
X
Username
Filter By
Users
S1
Student 1
View Profile
Invite to Space
S2
Student 2
View Profile
Invite to Space
S3
Student 3
View Profile
Invite to Space

(Image 16)


JD
John Doe
About Me
You can change your profile description here.
Password and Moderation/Authentication
Note that faculty and administrators can view, edit, and/or delete any of your posts, comments, or profile description/image if they deem them inappropriate, as per CIPA guidelines.
Change Password
Save Changes

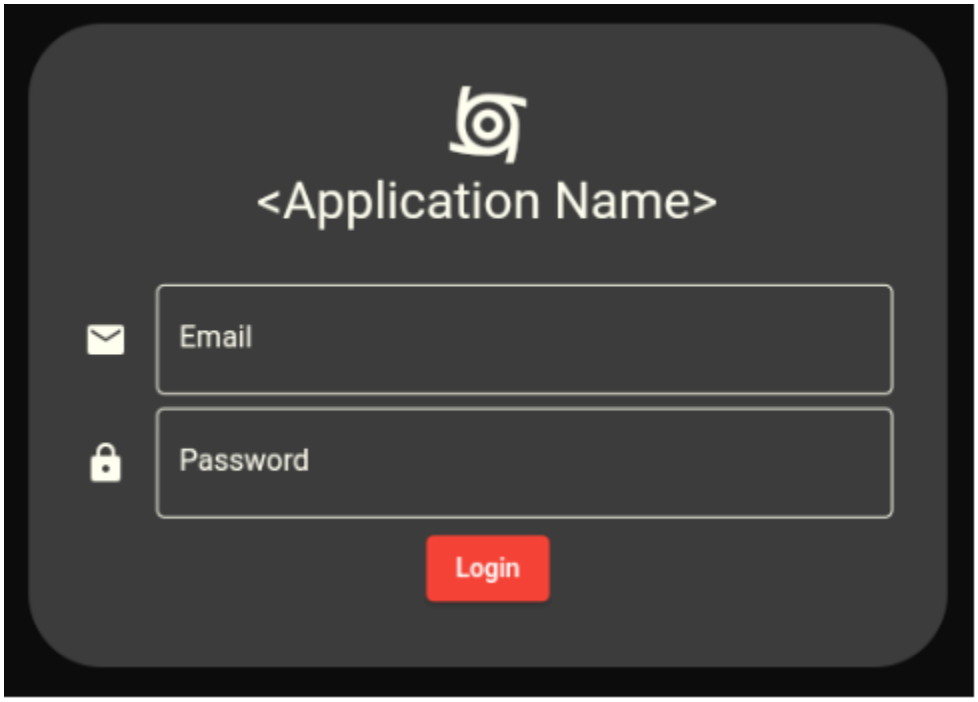
(Image 17)


Change Password
Old Password *
New Password *
<Password Requirements>
Repeat New Password *
Save Changes

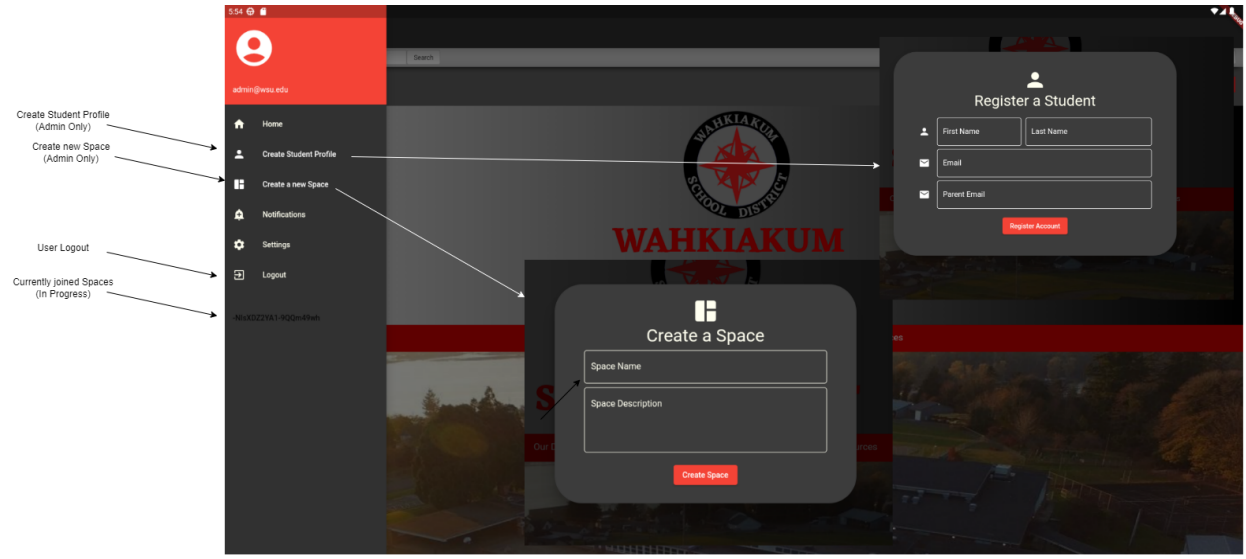
(Image 18)

(Image 19)

X.4. Appendix D - Prototype Project Report



(Image 1)



(Image 2)

(Image 3)

```
UserData currentUser =
    context.read<AuthRepository>().currentUser as UserData;
```

```
if (currentUser.isFaculty)
  ListTile(
      leading: Icon(Icons.pe
```

(Image 4)

X.5. Appendix E - Test Results

Components to testing:

Aspect
Expected results
Observed results
Test results

- Pass or fail

Test Case Requirements

5-6 test cases
1 test case for function and non functional requirement