1.1 Introduction

This document presents the architecture and detailed design for the software for the Squibble project. This project performs as a web application that can be used as a whiteboard to add media that resets and archives after a three day time period.

1.1.1 System Objectives

The objective of this application is to provide an online interface to allow users, with sign in through Firebase authentication, to add all forms of media to an online whiteboard including but not limited to text, images, and drawings. This is done through the use of a taskbar all users should be able to use. This media will be saved on the website interface through Firebase storage for a total of 3 days before being reset. Screenshots of the website will be taken and uploaded to a separate portion of the website where they can be viewed with specific dates associated with them.

1.1.2 Hardware, Software, and Human Interfaces

The following Hardware, Software, and Human Interfaces are used in Squibbles design and development.

1.1.2.1 Hardware

The hardware used for the Squibble project can be summarized as follows. Each member of the Squibble team operates either on a Windows 10, Windows 11, or Mac interface present on either a Dell or Apple laptop. Each of these laptops is linked to either home or LMU servers to allow for development. Each of these laptops also come equipped with a basic keyboard and mouse

pad for use with coding and developing. Even with this, some members of the team use external mice to hook up to their devices for easier mouse movement. Presentation of the project will be done on external devices such as projectors provided by LMU.

1.1.2.2 Software

The software used for the Squibble project can be summarized as follows. External browsers such as Safari, Google Chrome, and Firefox will be used to host the broadcast of the website to users when used. Visual Studio Code is the interface Squibble files are created and developed in. React is used to develop in association with Node.js to allow for the creation of the UI and code backend of Squibble and its applications. APIs such as the Puppeteer and Tenor API are used to aid developing certain aspects of the project such as the screenshotting of the website and finding GIFs for upload to the whiteboard. Firebase is used with both authentication and storage to allow for easy sign in and storage for media with the Squibble project. Firebase is also used to host the final website and provide a link for use.

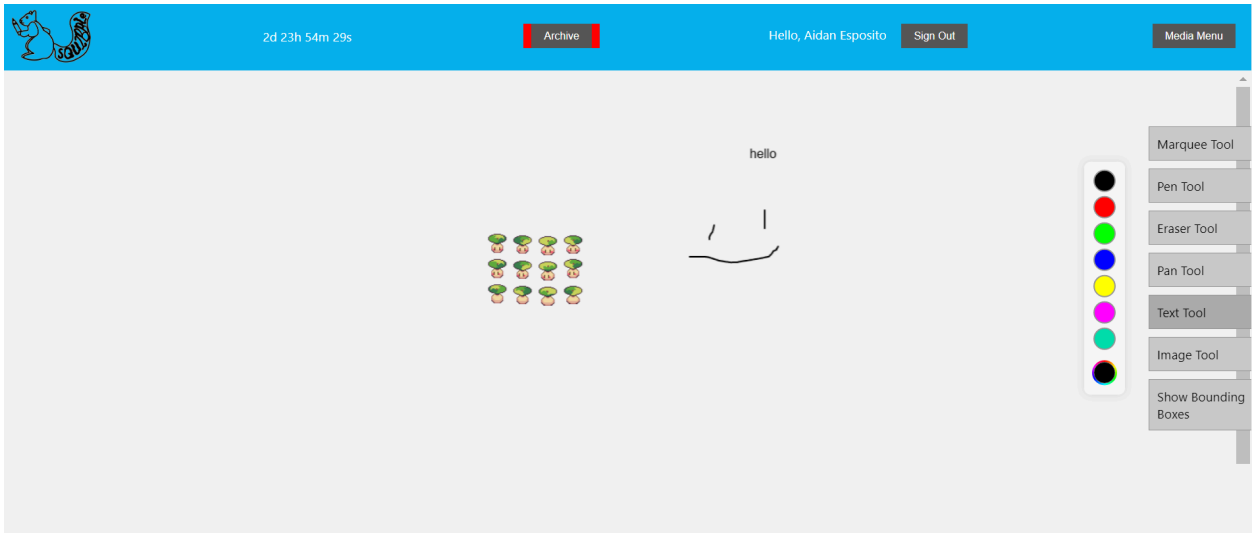
1.1.2.3 User Interface

The UI for the Squibble project can be summarized as follows. The UI of Squibble is split into two main pages: the main homepage and the archive page which will be described in two separate paragraphs.

1.1.2.3.1 Main Homepage

The main homepage of Squibble will be presented as follows when first seen. The website will start with an empty or filled whiteboard depending on the status of the media. The user will be

able to scroll all around the whiteboard and even zoom in on certain areas of it. The user will have a few options presented in the header: click on the logo to reset the website, click on the archive button to move to the archive page, or click on the sign in button to get started. Once signed in through Firebase, the user will get to interact with a new sign in button and media buttons to add to the whiteboard. The user can interact with all forms of media including the text, image, gif, drawing, and post-it tools to add to the whiteboard. Each tool will either open up a submenu for more details or will interact with the whiteboard as is. The user also will gain access to tools such as the bounding box and marquee tool allowing for visual processing in terms of boxes and interactions associated with the media currently present. There will also be a color bar presented to allow users to style their media with any color available.



1.1.2.3.2 Archive Page

The archive page of Squibble will be presented as follows when first seen. There will be a current header with a back button to return to the main page and a header stating “Welcome to

the Archive.” The user will have the ability to scroll up and down the page to see screenshots of past whiteboards ranging from most recent to most past with two images in each row with the time dates under them. Before any screenshots are added, there will be a small message for the user detailing how the page works and what will eventually be added.

1.2 Architectural Design

The Architectural Design of Squibble will be broken down into multiple different subsystems. These include the Media Control, User Interface Control, and Database Control, described as the following:

Media Control:

Squibble’s media management system integrates a user-friendly UI with a robust database to handle images, GIFs, and text content seamlessly. The interface includes a media preview panel, an upload and tagging module, and a detailed view for file properties, enabling efficient management and interaction. On the backend, the database schema organizes media objects,

metadata, and usage analytics, allowing for quick search and filtering, usage tracking, and content recommendations. This setup ensures Squibble can store, categorize, and retrieve diverse media types effectively, supporting dynamic user needs in a collaborative environment.

User Interface Control:

The UI of Squibble is designed based on the main app.js file that splits the media into the header and the whiteboard subclasses. The header subclass controls the media menu, submenus, and timer of the website as well as general display of objects. These are compiled with the css of the website to allow for a nice and presentable design. The whiteboard subclass hosts all of the features of the canvas and tooltab creating subclasses for every menu provided to the user including the color menu and toolbar tab. This, in particular, splits into the tooltabs.js file with classes that allow for specific bars and functions for the user on the website to interact with, allowing for placement of media on the whiteboard. All of these files also interact with the archive.js file that contains the separate webpage for the archive with its own css and UseEffect classes.

Database Control:

The database of Squibble is designed based on the main firebase_config.js file that allows for connection to the firebase database and allows for the set up of initialization, authentication, and storage for the project before exporting them as separate objects. These objects are used in two

different ways. The authentication is used in the auth.js file that allows for sign in and sign out of Google accounts through a SignIn, SignOut, and useAuthentication functions. These link to header.js where they are linked to button objects to allow for their interaction with the main homepage. The storage of the website is currently used with the capture_screenshot.js file which interacts with the puppeteer api to take a screenshot of the website with the takeScreenshot function and const such as data and ref to move the screenshot when taken not only to a screenshots folder but to the storage application of Firebase with the use of the uploadBytes function provided by Firebase. These files are then pulled from Firebase in archive.js and uploaded to the archive page for presentation.

1.2.1 Major Components

Major subsystems present in Squibble that correlate to the functional requirements of the software present in the Requirements Specification document can be described in the following:

- From 1.3.1.1-1.3.1.4, the User Interface subsystem is used to render the main webpage and its submenus present. This is done primarily in the app.js file along with the header.js and whiteboard.js files and their associated css to host the normal website and allow for changes in presentation.
- From 1.3.1.4-1.3.1.8, the Database Control subsystem allows for interaction with the Firebase server through the use of authentication with Google accounts and the use of the sign in and sign out button.
- From 1.3.1.8-1.3.1.22, the Media Control subsystem will allow for interaction with the whiteboard itself and for separate submenu presentation. The Media Control subsystem

allows for interaction with the whiteboard through general scrolling and movement as well as with the use of adding media to it. The Media Control subsystem hosts the text tool, the gif tool, the image tool, the drawing tool, and the post-it tool. The subsystem also has interactions with the whiteboard itself through the use of tools such as the border display tool and the marquee tool.

- From 1.3.1.22-1.3.1.25, the Database Control subsection will handle moderation and general modification of files not only through the use of authentication and storage present in the code but also through the Firebase website itself.
- From 1.3.2.1-1.3.2.9, the User Interface subsystem will handle the header display and timer display of the website with its associated css and will allow for user interaction with the interactable features provided.
- From 1.3.2.9-1.3.2.11, the reset of the website will be handled by the Database Control subsystem on a timer with Firebase that will delete and save a screenshot to its storage through the use of the puppeteer api.
- From 1.3.2.11-1.3.2.19, the Database Control and User Interface subsystems will be used to both render and highlight the archive page and each screenshot and their positions but will also interact with Firebase to collect the most recent screenshot and push it to the archive page. Both of these subsystems interact in this process.

1.2.2 Major Software Interactions

Major software interactions throughout the Squibble application can be described by the following:

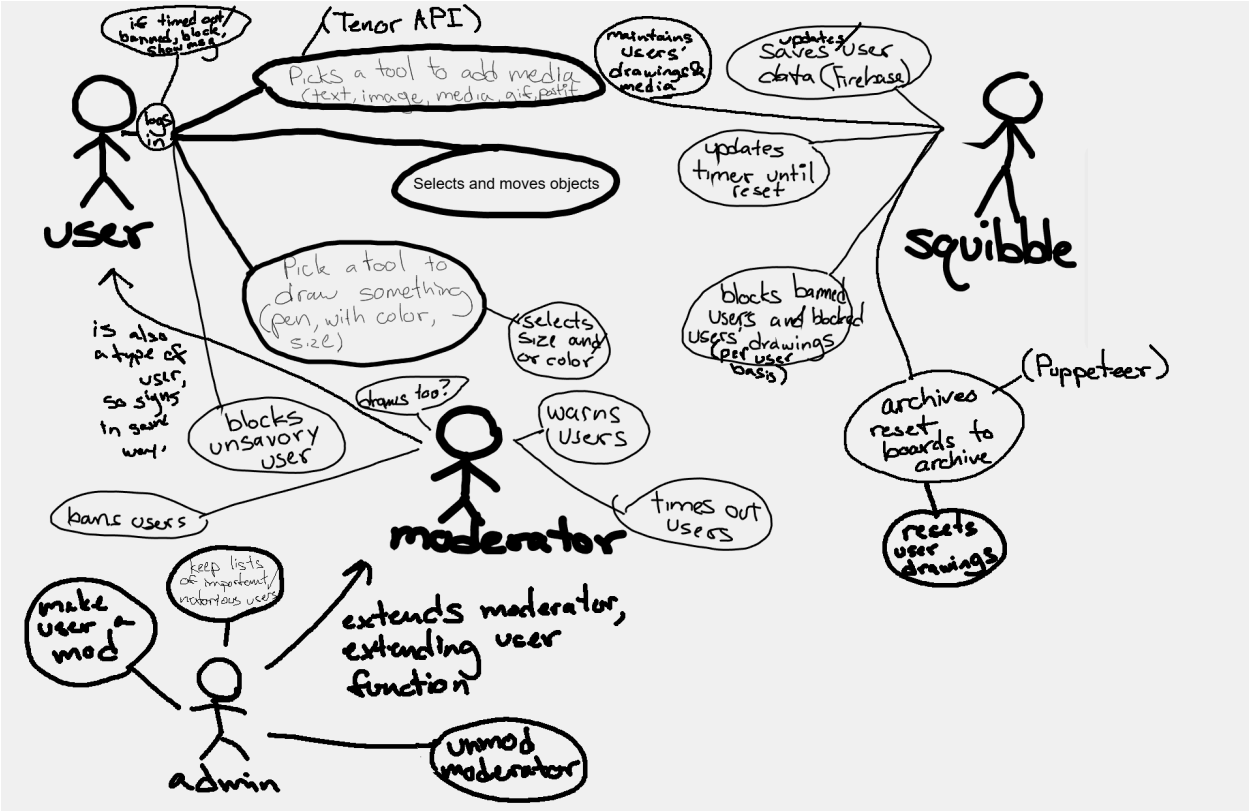
- The main menu for drawing on the canvas will allow users to submit drawings into a database on the firebase server, an array of arrays particular to each user called CanvasDrawings. The list will track the chronological order of each user's drawings, including text, media, gifs, post-its, and drawn lines, each as its own separate list.
- The main menu will allow for user blocking functionality, which will track each user's list of other blocked users on the firebase backend's UserBlockLists list, which contains the user ID of every user that each user has currently blocked, which tracks how each user sees the canvas on their client.
- This extends to another separate list, the UserBanList, which those who have moderator privileges can add to or remove from. In it, each banned user is present. Any users who have been banned will have all of their drawings set inactive and removed to all users. Bannings can be either temporary or permanent, or even pardoned.
- Moderation privileges are given to trusted users, ranging from developers to volunteers. Not anybody who has moderator privileges is able to mod other users, that is kept only by developers. A Moderators list handles this on Firebase's end. Modded users should have moderator only functions like TimeOut or Ban, and a mod message ToolTab which can send messages that go on top of every other user's drawing on the canvas. Admins, additionally, have these privileges as well as the Mod and Unmod tool to mod or unmod a requested user.
- Moderators can also send a message to a user as a warning, WarnUser will allow functionality to add a drawing on the canvas that only a particular user can see. This is handled with a tag given to the drawing of the user's ID, set by a ToolTab that only moderators at minimum can use.

- Banned users, after logging in, will instead be met with a ban message showing possibly a ban reason and ban time, which is overlaid over a dummy version of the canvas.
- Screenshots of the canvas to put into the archive before deletion every three days are handled by Puppeteer, which handles functionality for taking and uploading screenshots to Firebase. These screenshots are sent to the Archive list, which displays when a user checks the archive section of the React app.

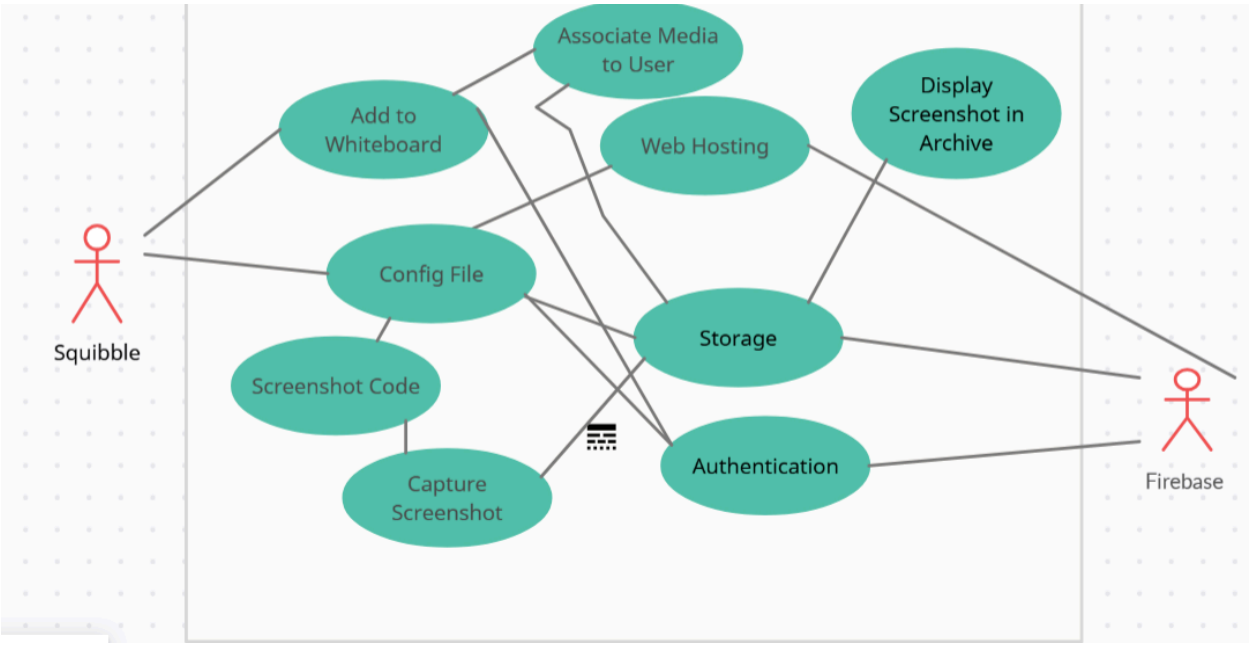
1.2.3 Architectural Design Diagrams

Here are some diagrams for Squibble applications at an architectural level:

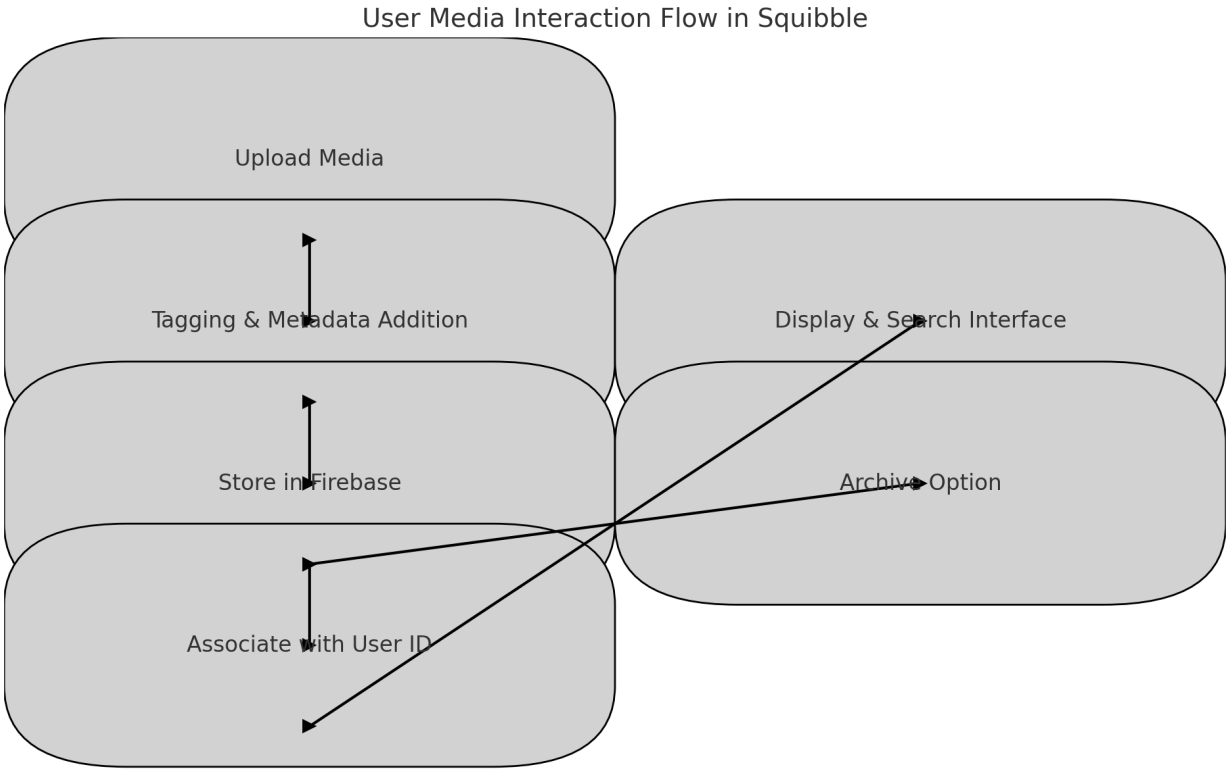
Overall System (made in Squibble!)



Firebase Integration:



Media Interaction Chart:



1.3 CSC and CSU Descriptions

The Detailed Design of Squibble will be broken down into multiple CSC's and CSU's in accordance with the system. These will be highlighted down below:

App and Header Presentator

The CSUs involved in this section of Squibble include:

- App

- Header
- Timer
- Archive (detailed in archive section)
- Media Menu (detailed in media interactions section)
- Submedia Drawing-Menu (detailed in whiteboard interactions section)
- Google Sign-In Container (detailed in Authorization section)

Archive Presentator

The CSUs involved in this section of Squibble include:

- Archive
- Archive Header
- Screenshot-Grid
- Screenshot-Item
- Date

Canvas and Whiteboard Objects and Systems Interaction

- Whiteboard
- Submedia Drawing-Menu / Color Menu

Tool Tab and Media Interaction

- Tool-Tabs
- Tool-Tab
- Tool-Options

- Media Menu

Authorization and Storage useEffects and Providers

- Google Sign-In/Sign-out Container

1.3.1 Class Descriptions

The following sections provide the details of all the classes used in the Squibble application. These will be split into subsections which will be shown here:

1.3.1.1 App

This class interacts with the general main page and archive of the system.This page contains the showArchive constant that allows for the archive button to be present if or if not the user is signed into a Google account. It also works with an onBack if statement to allow a return button from the archive to the whiteboard to be in place. This then splits into the general header section of the class which includes several constants. These include the user, onSignIn, and onSignOut constants that work with an onSignIn and onSignOut function in Auth.js to allow for a sign in and sign out prompt to pop up with the use of Firebase. brushSize, setBrushSize, currentColor, and setCurrentColor allow for interaction of the app with the general whiteboard primarily through the use of the Canvas.js and Whiteboard.js files and their functions. There are also constants for timer and resetTimer that set the timer to 3 days and allow for ticking down of time until eventual whiteboard reset with the resetWhiteboard constant. There is also a whiteboard section of the class that contains the constants for text, setText, brushSize, and currentColor that all allow for whiteboard interaction similar to the constants in the header section. There are also

the consts shouldReset and setShouldReset that allow for resetting of the whiteboard in association with the timer settings in the header section. There is also a small uncommented piece of the class that could be enabled or disabled to enforce additional sign in rules and routes including those of general email other than Google if enabled.

1.3.1.2 Header

This class involves multiple different functions and classes within it. It includes the logo function which uses an aria-label and works the balloon css to create a logo on the top left of the website and also reset the website when it is clicked. There are also uses of the timer, archive, and archive header classes that will be described later. The class then moves to work with the SignIn and SignOut constants to check where the user is in the website and what to show. If the user is signed in, the media menu class will display, specifically displaying the images, gifs, drawings, and post-it notes. This also involves displaying the drawings submenu and adding information associated to it such as its type, min, max, and value of brush size, with each of these being integers and constants. There is also a small label for color allowing for the current color of the tool to be set with the onChange value that works with a function titled setCurrentColor that changes the target value of the tool color. The class also has separate functions if a user is signed out which involve showing the Google Firebase sign in the container to allow for general sign in. The class ends by being exported with the line “export default Header”.

1.3.1.3 Timer

This is a small class used to work with the formatTime function and the const timer to allow for use in the Header class. Timer is a constant that is modified in both a useEffect function and the formatTime function. The useEffect function creates a constant titled interval to set a base time for the timer. The function checks if the timer is greater than zero or not and resets the board if the timer is less than zero before resetting the timer to three days or, more specifically, 259200 seconds. The interval is also cleared and reset in this process. If the timer is greater than zero, then the timer slowly counts down in terms of seconds from the three day starting period. The function that handles consts known as formatTime which creates constants for days, hours, minutes, and seconds to allow for time conversion to format the time shown in the timer as it counts down. This function is used with the timer const to slowly tick down the clock for the website and is presented in the Header class by calling the function with “formatTime(timer)”.

1.3.1.4 Archive

This is a class that can be seen in multiple files across the project. In Header.js, the archive class is used to host a button that, when clicked, will load the archive page both when signed in and signed out. In Archive.js, the class primarily works with a useEffect function that allows screenshots to be fetched from Firebase storage and displayed. It does this in a try catch block by trying to fetch a constant known as result from the screenshots folder in Firebase. It then creates a screenshotData constant with a url and timestamp that are returned and displayed in association with the css on the website. The catch is shown if the screenshots are not returned providing an error message to the developer. This class also works with other classes for display on the archive page including the archive-header, screenshot-grid, and screenshot-item classes which will be described later.

1.3.1.5 Archive Header

This is a small class present in Archive that has one main function which is to host the back button to return to the main site page from the archive page. This is done with an onClick function that works with the constant onBack to return you to the original destination. This class also contains small h1 and paragraph elements that display the text “Welcome to the Archive” and “This is the archive page where you can view the whiteboards saved every 3 days” respectively.

1.3.1.6 Screenshot-Grid

This is a small class that is used in association with css on the Archive.js file to display and showcase the screenshots provided from Firebase in a 2x2 grid fashion. The css involved with this class highlights the information as a grid type and repeats the information every two images also allowing for gaps and margins between them. This class specifically creates an array of constants titled screenshots and index to do this in association with the screenshot-grid class which will be described below.

1.3.1.7 Screenshot-Item

This is a class that works with the Screenshot-Grid class to display specific information on images and allow for the alignment of images that worked in the previous class. This class specifically uses the array created previously to assign keys to the index constants created beforehand. These keys are then associated with an image tag used with the constant “screenshot.url” provided by Firebase to keep count of how many images are on site and adds

one to the index whenever an image is added to prevent overlap. This class is also based with css that displays the screenshots in a flex display and allows for specific alignment, width, and image height. This class also contains the date class within it for linking of images with specific timestamps.

1.3.1.8 Date

This is a small class that works with the established Javascript function of Date in order to pull the associated date of a Firebase screenshot image from storage. It does this by creating a new date constant associated with each timestamp of the screenshot using the line “new Date(screenshot.timestamp)”. This then is used in association with another built-in Javascript function titled toLocaleDateString that converts the date to be associated with the specific timezone or time frame the user is viewing the page in ending with the final line of code being: new Date(screenshot.timestamp).toLocaleDateString”.

1.3.1.9 Whiteboard

This class is mostly a hosting place for many large constants that are used through the general whiteboard interaction process of Squibble. It contains information for the Canvas constant, which specifically contains the original creation of values including currentColor, brushSize, activeTool, lines, setLines, showBoundingBoxes, isTextMenuOpen, and setIsTextMenuOpen to work in combination with the text, drawing, and bounding box tools specifically. The class also contains information for the ToolTabs element that shows the activeTool, temporaryBrushSize, and boundingBoxes with multiple constants in association with specifically the drawing and bounding boxes toolbar icons and tools themselves. The color menu constant is also contained

here with information involving constants with currentColor, handleSampleColorSelect, handleCustomColorSelect, and sampleColors in association with the drawing and color text tool menus respectively. The class then moves into checking specifically if the active tool is a text tool and uses the constant TextOptions to handle information such as text style, color, and location through the use of an array and index. This class is then finally exported with the line “export default Whiteboard”.

1.3.1.10 Submedia Drawing-Menu / Color Menu

This class, present in the ColorMenu.js file, with the parameters currentColor, handleSampleColorSelect, handleCustomColorSelect, and sampleColors, uses an array with the sampleColors parameter to give a tool information on its color, style, and location in the toolbar with the css. This also works with a small onClick function that prevents deselection of a tool when in use and handles color select with the parameter handleSampleColorSelect. The class also allows for the design of the balloon input container for custom colors by giving it a rainbow border, circular shape, and the ability to show a specific color when selected. This class is then returned with the function export default ColorMenu.

1.3.1.10 Tool Tabs

This is a class that primarily hosts the different tool tabs present throughout the tool menu with associated images and text. Each function in here is associated with the Tool Tab class and provides an icon (emoji) and text for the tool box. Each component will be further explained in the Tool Tab class section itself. This class is finally exported with the line “export default ToolTabs”.

1.3.1.11 Tool Tab

This class works in association with the Tool-Tabs class to highlight each individual tool box and selection of the Squibble media editor. Each of these sections is defined by describing this class and checking if the tool is active for use. If the tool is active, the tool will be associated with the activeTool constant at that current moment. If a tool is not active, it can be selected with an onClick function that uses a function titled setActiveTool to set the activeTool constant as the tool selected. The tools included in this class are the marquee tool, pen tool, eraser tool, pan tool, text tool, image tool, gif tool, post-it tool, add image tool, and the bounding boxes toggle tool.

1.3.1.12 Tool-Options

This is a small class used in association with the Tool-Tabs and Tool-Tab classes. This specific class is used in the pen tool to create a small submenu to check the brush size when clicked in the toolbar. This creates a label titled “Brush Size”, with specific values for range, min, and max. This class also works with onChange and onClick functions to set a temporary brush size that the user selected with the submenu and to prevent multiple clicks from bubbling (dirtying) the whiteboard.

1.3.1.13 Media Menu

This is a small class present in the Header.js file to specifically highlight the features of the media menu present that a user can click on when signed in. This works with multiple onClick functions to work with a function titled handleSubmenuToggle to display the submenus for

images, gifs, drawings, and post-it notes. The handleSubmenuToggle function is used to set the active media item and also remove the previously displaying submenu preventing overlap.

1.3.1.14 Google Sign-in/Sign-out Container

This class has the sole function of displaying the Firebase sign in the submenu when loaded with the Sign in and Sign out OnClick functions in Header.js. This function in particular is a try catch handleSignIn constant that awaits the sign in pop up by calling the GoogleAuthProvider function firebase presents. If there is an error, the console will display the message “Error signing in” and alert with an error message to the site itself. This function then returns the button that allows for sign in present on the top right of the header titled “Sign in with Google”. This also works with the similar SignOut function that creates a constant titled handleSignOut which awaits authentication to sign out and, with a try catch, displays an error if failed. This function returns the message “Hello (current username)” on the top of the screen when signed in, and provides the sign out button if the user wants to sign out.

1.3.2 Interface Descriptions

The following sections provide the details of all the interfaces used in the Squibble application. These will be split into subsections which will be shown here:

App and Header Presentator

- **React (UseState):**

Allows for actions to happen across the App and Header files (among others). Is used in instances such as setting timer time, setting tool size and selection, and setting button for accessing archive.

- **Archive:**

The Archive component is used in main app code as a section to work with an onBack() function to allow for a button to lead back to the original whiteboard. Also works with the handleArchiveClick function and the setShowArchive constant to show the archive page or not depending on what button is clicked.

- **Whiteboard:**

The Whiteboard component allows the main app to access the whiteboard based on previous CSU's and CSC's. Works with the function handleBackToWhiteboard to go back to the whiteboard with button in header and the function resetWhiteboard to reset the timer when the timer hits zero. This is also used as a host for information such as text, brushSize, reset time, functionality, and color.

- **Header:**

The header component contains information for the header of the website and its functions. These include checking if a user is signed in and signed out with the SignIn and SignOut interaction, timer information, colors, and text information.

- **React (useEffect):**

The UseEffect component is an Interaction imported from React that allows for the use of effects or interactions throughout the site. This can allow for elements of the project to update over the site's use. This can include resetting the timer and the whiteboard all the way to updating every media piece on the display.

- **SignIn**

The SignIn component is a small interaction used for displaying the sign in container that is provided with google firebase.

- **SignOut**

The SignOut component is a small interaction used for displaying the sign out container that is provided with google firebase.

- **UseAuthentication:**

The useAuthentication component works with Firebase in order to check if a user is “subscribed” to the current website. If a user is subscribed, this will update the user data and display certain parts of the website. It also allows a user to “unsubscribe” and log out of a website.

Archive Presentator

- **Firebase (getStorage, ref, listAll, getDownloadURL)**

The getStorage component is used in association with the const storage in order to gain access to the storage in the Firebase database. This also gives access to the current rules set in the Firebase database which will be applied to the project when used. Ref is a simple reference to the storage screenshots folder allowing it to be modified in the code. listAll is used to list all of the current screenshots in both the project and the Firebase database for comparison. These are used with the const results to check if the data between apps is successfully updated and current. Finally, getDownloadURL is used to make a const in association with puppeteer to clone and screenshot the website before uploading to the online database.

- **React (useEffect, useState)**

Both of these React elements are described earlier in documentation.

Canvas and Whiteboard Objects and Systems Interaction

- **Canvas**

The Canvas component is the main interactive area in the Squibble application, allowing users to draw, erase, and manipulate objects like images and text. It tracks various states for tool selection, drawing, and object management, handling user actions through event listeners. Key functions manage drawing lines, detecting collisions, and implementing undo/redo functionality, providing a responsive and dynamic canvas experience.

- **ColorMenu**

The ColorMenu component provides a palette of color options for users to select the current drawing color on the canvas. It manages the selected color state and updates it when a user chooses a different color. This component allows users to customize their drawings by applying their preferred colors in real-time.

- **ToolTabs**

The ToolTabs component provides a set of tool options for users, such as pen, eraser, marquee, text, and image tools. It manages the active tool state and updates it based on user selection, enabling quick switching between tools. This component allows users to interact with the canvas using different functionalities, enhancing the versatility of the drawing experience.

- **TextOptions**

The TextOptions component offers customization options for adding text to the canvas, such as font, size, and color. It captures user-selected text properties and passes them to the canvas when a new text element is created. This component allows users to personalize text appearance, enhancing the creative flexibility on the canvas.

Tool Tab and Media Interaction

- **React:**

The React component is a general instance that brings in all of the features usable from all of React. These include the useState and useEffect instances as well as the general features react offers such as interaction with css and html as well as providing access to their hosting and modification of website services.

Authorization and Storage useEffects and Providers

- **Firebase (initializeApp, getAuth, getStorage):**

The Firebase setup (using initializeApp, getAuth, and getStorage) configures essential Firebase services for the application. initializeApp initializes the Firebase app with the project's credentials, getAuth enables user authentication, and getStorage provides access to Firebase Storage for handling file uploads. This setup allows secure user management and file storage within the app.

- **App (Firebase Config)**

The App component initializes Firebase with project-specific configuration settings. It uses Firebase Config to connect the app to Firebase services, enabling functionalities like authentication, database, and storage. This setup establishes a foundation for Firebase-powered features throughout the application.

- **Puppeteer**

Puppeteer is a Node.js library that provides a high-level API for controlling headless browsers, primarily Chrome. It allows automated tasks such as web scraping, testing, and rendering page content by programmatically interacting with web pages. Puppeteer enables efficient browser automation for various web-related tasks.

- **Firebase Storage (ref, uploadBytes)**

Firebase Storage, using ref and uploadBytes, enables file storage and management. ref creates a reference to a specific storage path, and uploadBytes uploads files to that path. This setup allows secure, efficient file uploads and organization within Firebase Storage.

1.3.3 Data Structure Descriptions

The following sections provide the details of all the data structures used in the Squibble application. These will be split into subsections which will be shown here:

Arrays: In the Squibble application, arrays play a crucial role in managing and organizing data, especially for storing and handling dynamic content on the canvas. Arrays such as lines, recentLines, and selectedObjects hold different types of drawable objects (like lines, splines, images, and text) and facilitate actions like drawing, selecting, and erasing. For example, lines is an array that maintains the sequence of objects drawn on the canvas, allowing the application to render each item in the correct order and manage user actions like undo and redo. Arrays also help in operations such as selection and collision detection, enabling users to interact seamlessly with multiple objects at once.

Objects: In the Squibble application, objects are a fundamental data structure used to represent and manage the properties of various elements on the canvas. Each drawable item—such as lines, splines, images, and text—is encapsulated within an object, with attributes detailing its type, position, color, and size. This structure enables the application to track and manipulate each element's unique properties efficiently, allowing for actions like drawing, resizing, and positioning. Additionally, objects like marqueeStart and previousPosition help capture specific details of user interactions, making it easier to perform calculations and manage states across different actions on the canvas.

References: In the Squibble application, references (created with the useRef hook) are crucial for managing persistent values that need to remain consistent across renders without causing re-renders. References like isManipulatingRef, previousPositionRef, and selectedObjectsRef store information about the current state of manipulation, the last known position, and selected objects, respectively. These refs are especially useful for complex user interactions, such as dragging or manipulating multiple objects, where immediate updates to the DOM are unnecessary but maintaining state consistency is essential. By using refs, the application achieves smooth performance and avoids unnecessary re-render cycles during intensive actions like drawing or moving items on the canvas.

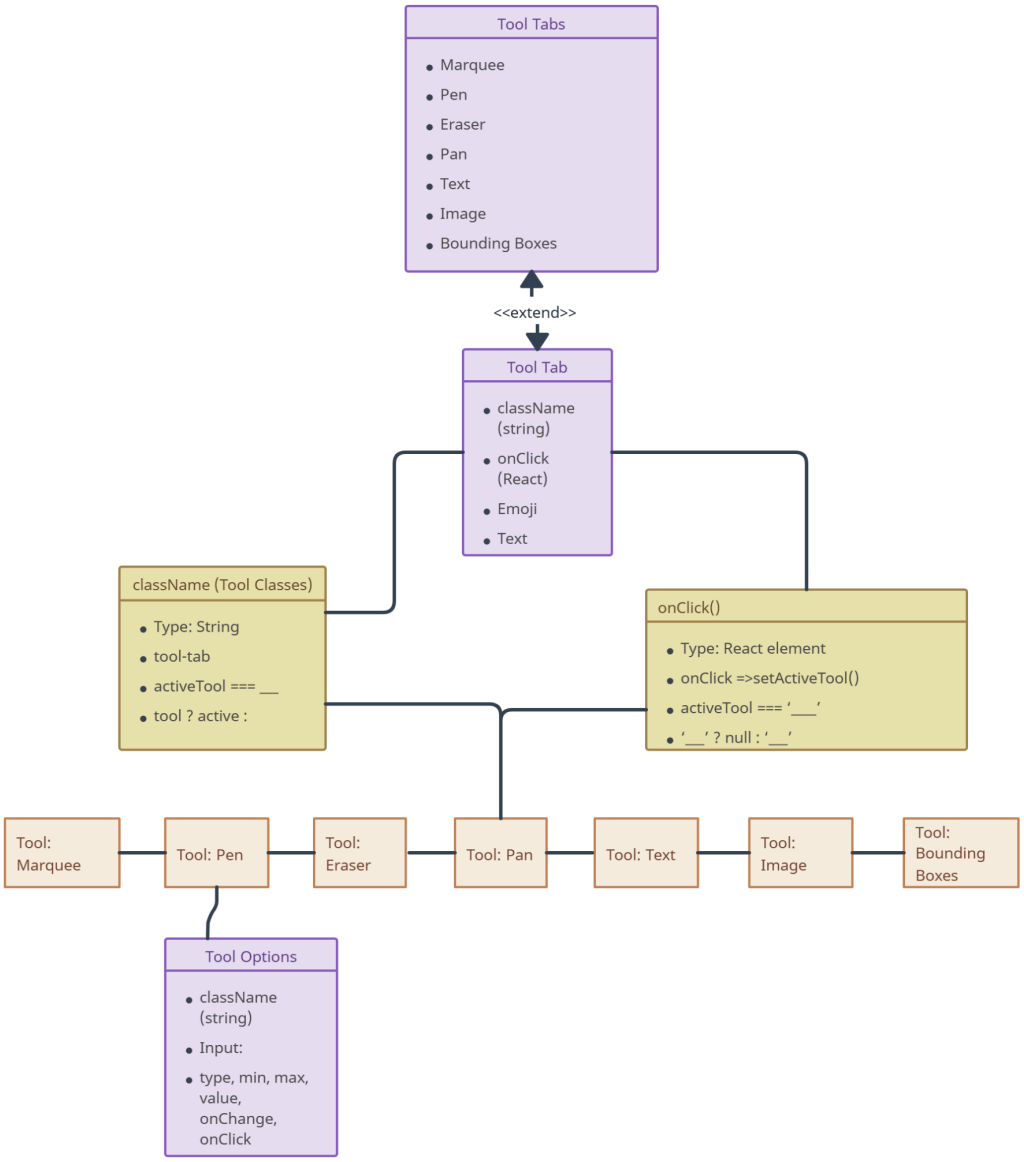
Booleans: In the Squibble application, booleans are essential for managing the active or inactive states of various tools and actions as they are react based.. Boolean variables such as isDrawing, isErasing, and isMarqueeActive control whether certain tools or interactions are enabled at any

given moment, allowing for precise tracking of user actions on the canvas. These booleans make it easy to toggle functionalities, like switching between drawing, erasing, or marquee selection, based on user input. By using simple true or false values, the application ensures efficient state management, enabling smooth transitions and interactions without requiring complex conditional logic.

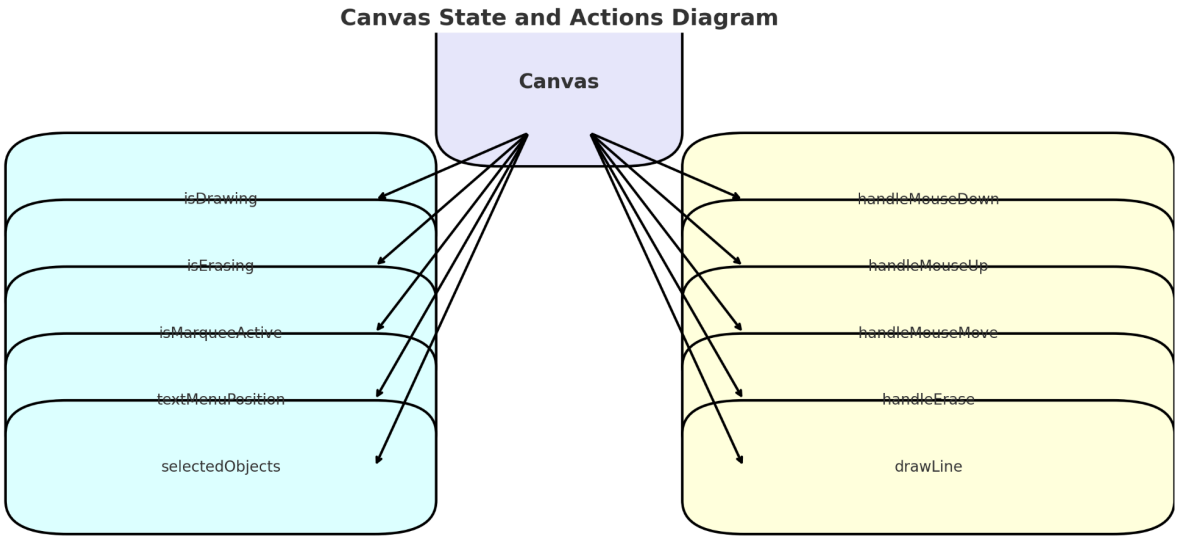
1.3.4 Design Diagrams

Here are some diagrams for Squibble CSCs and CSUs at a design level:

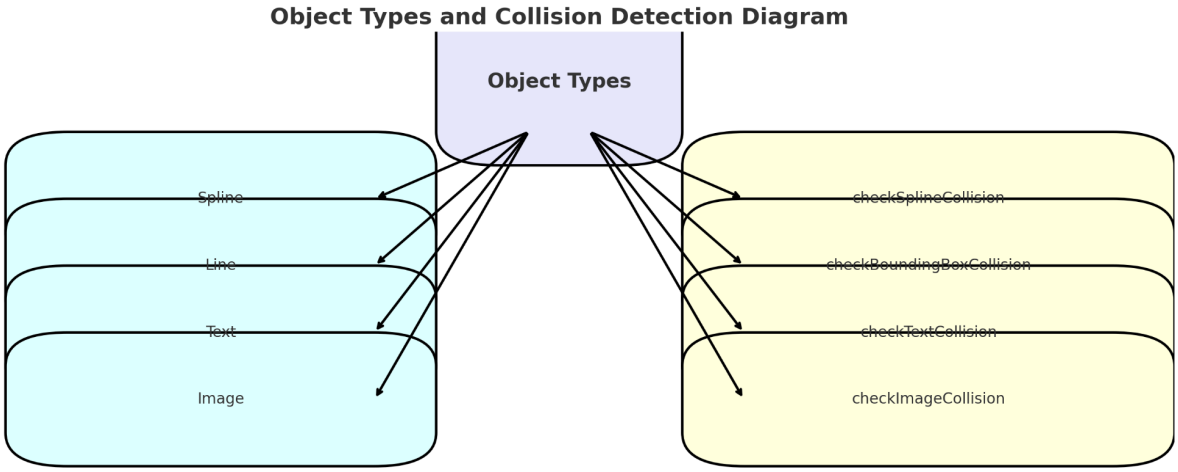
Tool Tabs / Tool Tab / Tool Options


Tool Tabs
Marquee
Pen
Eraser
Pan
Text
Image
Bounding Boxes
<<extend>>
Tool Tab
className (string)
onClick (React)
Emoji
Text
className (Tool Classes)
Type: String
tool-tab
activeTool === __
tool ? active :
onClick()
Type: React element
onClick =>setActiveTool()
activeTool === '___'
'___' ? null : '___'
Tool: Marquee
Tool: Pen
Tool: Eraser
Tool: Pan
Tool: Text
Tool: Image
Tool: Bounding Boxes
Tool Options
className (string)
Input:
type, min, max, value, onChange, onClick

Canvas State



Object Type and Collision Detection (from Canvas.js)



1.4 Database Design and Description

Squibble will use Google Firebase to access database based applications including hosting, authentication, and storage. All of the four team members have access to the Squibble Firebase console. Squibble hosting is done by using the firebase servers to host the website. The terminal is used to run updates and push changes to the online website. Authentication is currently only used with Google accounts that can be logged into the database, giving information on a user's email and sign in date for possible moderation purposes. Storage will be divided into subsections including but not limited to:

Screenshots:

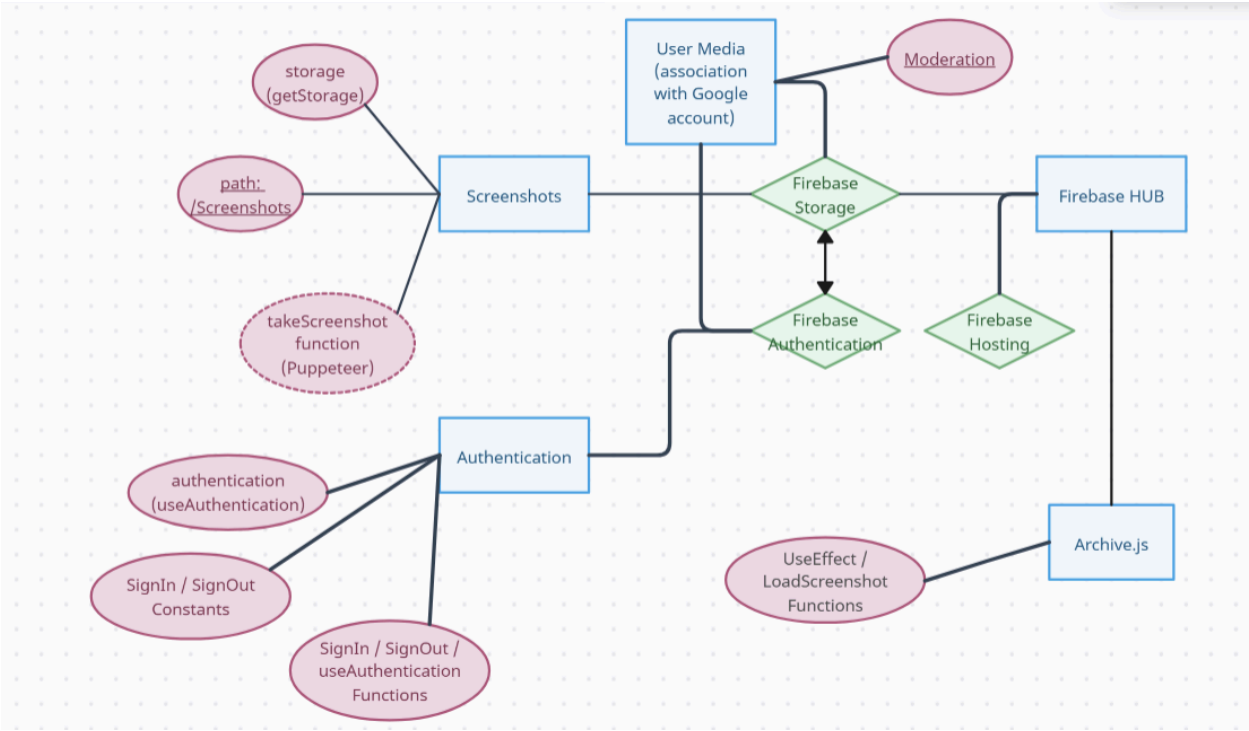
Timed screenshots of the websites that upload to Firebase every three days through the puppeteer api. Uploaded back onto the Archive page for archival purposes.

Media:

User associated media objects added to the whiteboard and stored in the database through the use of Text, Images, Gifs, and Drawings. Can be deleted from storage if violating moderation rules thereby removing it from the whiteboard public website.

1.4.1 Database Design ER Diagram

Here is a diagram describing Squibbles use of Firebase through its systems:



1.4.2 Database Access

The database will be accessed through several avenues throughout the Squibble code including the following:

ConfigFirebase.js:

Pulls from the personal Squibble project website and allows for access of functions such as getAuth, getStorage, and initializeApp before exporting them as objects titled auth, storage, and app.

CaptureScreenshot.js:

Uses “getStorage” from the config file to create a const titled “storage”. This is used in association with the takeScreenshot function based in Puppeteer to create a reference titled “screenshotRef” that creates a folder in the Firebase files titled “Screenshots” to host images with the path “screenshots/${fileName}.” The screenshots are then uploaded with the uploadBytes function provided by Puppeteer to Firebase.

Archive.js:

Uses getStorage from the config file to create a const titled “storage”. Works in tandem with a “useEffect” function to load screenshots from storage by using the previously mentioned “screenshotRef” to list all of the images. The data is then displayed in association with the “screenshotData” function to link dates provided in the Firebase system to images shown. The images are finally displayed in the archive with the function “loadScreenshots” and a HTML class known as “screenshot-grid” working together with css.

Whiteboard.js:

Currently uses the “getStorage” function to link objects to users when posted. Being worked on are the features to upload media to the storage database and fully associated objects to users throughout website constant use.

Header.js

Imports the “useAuthentication” function from ConfigFirebase.js to link to a const titled “user.” This const works with functions built into Firebase that allow for Sign in and Sign out to display and show objects when the user is signed in or not.

App.js

Imports the “useAuthentication” function from ConfigFirebase.js to link to a const titled “user.” Creates references titled “onSignIn” and “onSignOut” that are utilized with the Sign in and Sign out functions and used in Header.js.

1.4.3 Database Security

To ensure security from the Firebase platform, Firebase allows for specific rules based for storage and authentication. For authentication, only Google users are allowed to sign into the website. As mentioned, Firebase gives access to the users account information including email and sign in date. Firebase also gives access to the user UID which can allow for moderation to ban or limit a user from the Squibble website if needed. For storage, Firebase allows for special security rules to be enabled or disabled to prevent or allow user activities to happen. Currently the rules allow for users to interact with the whiteboard sign in with each media piece they add being associated with their account. Users are not allowed to interact with whiteboard storage and other users' media seen on the whiteboard preventing errors. On our end, we are looking to establish rules and checks in the code that will prevent users from messing with settings that Firebase cannot block out.