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. VStudio code is the interface Squibble files are chanted 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.

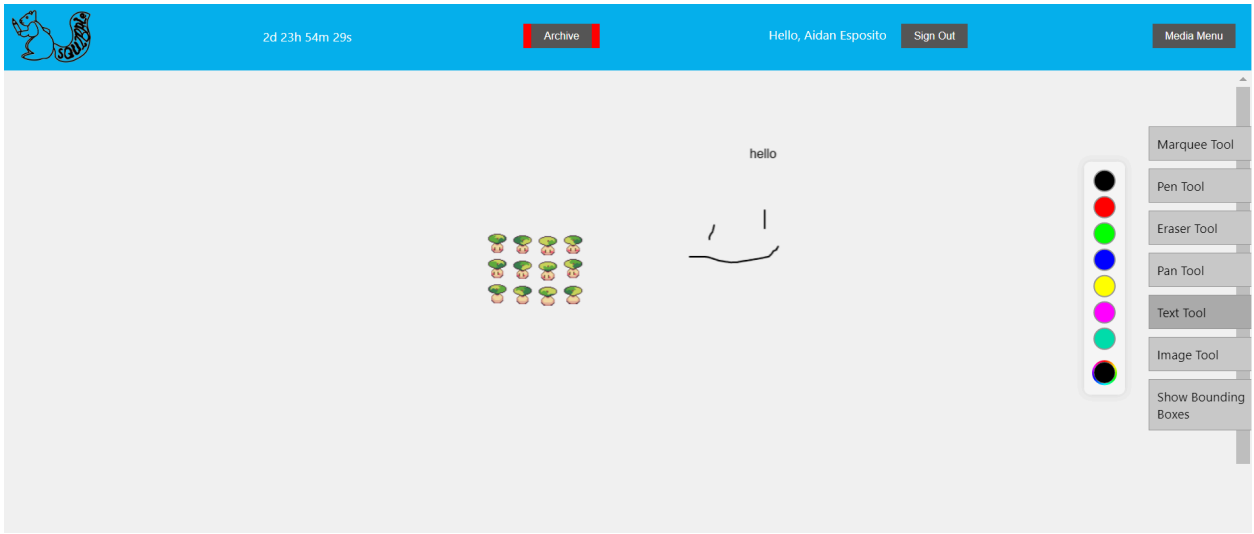
1.1.2.3 User Interface

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:

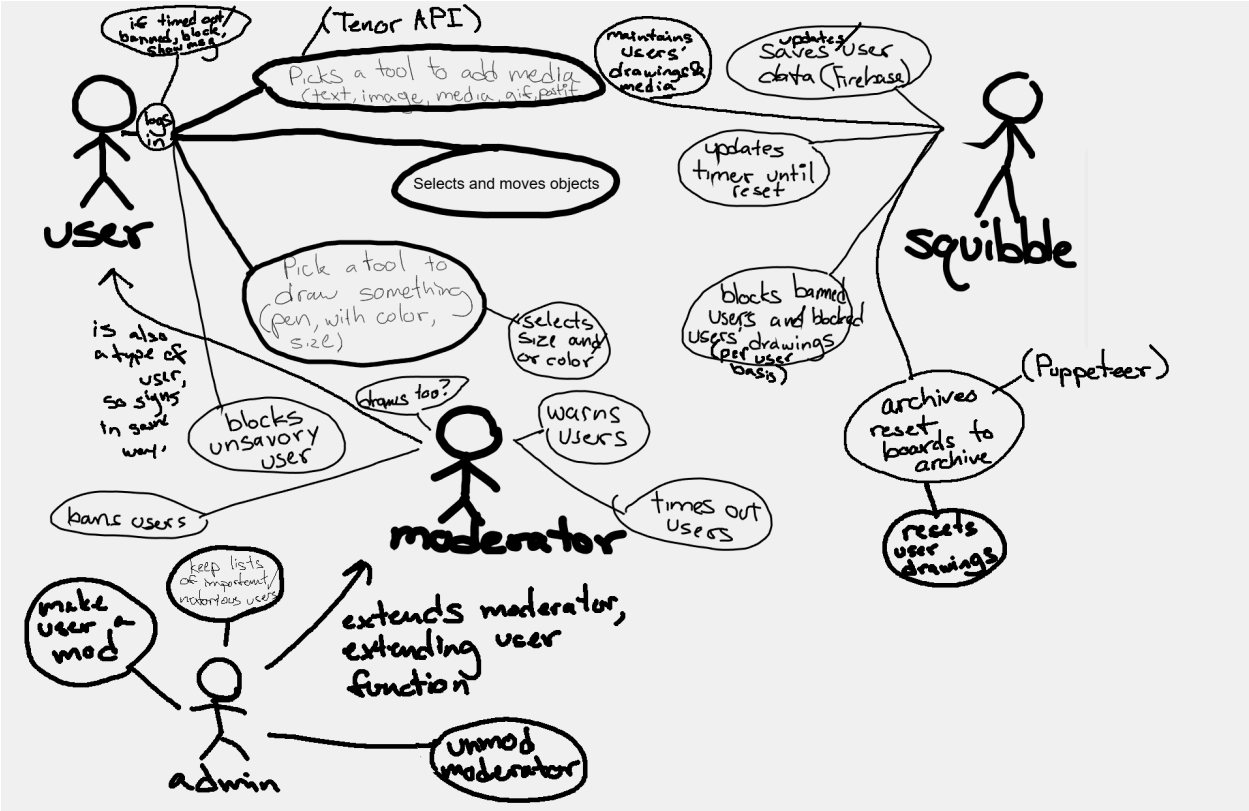
- The main menu for drawing on the canvas will allow users to submit drawings into a database on the firebase server, an array of arrays particular to each user called CanvasDrawings. The list will track the chronological order of each user's drawings, including text, media, gifs, post-its, and drawn 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.

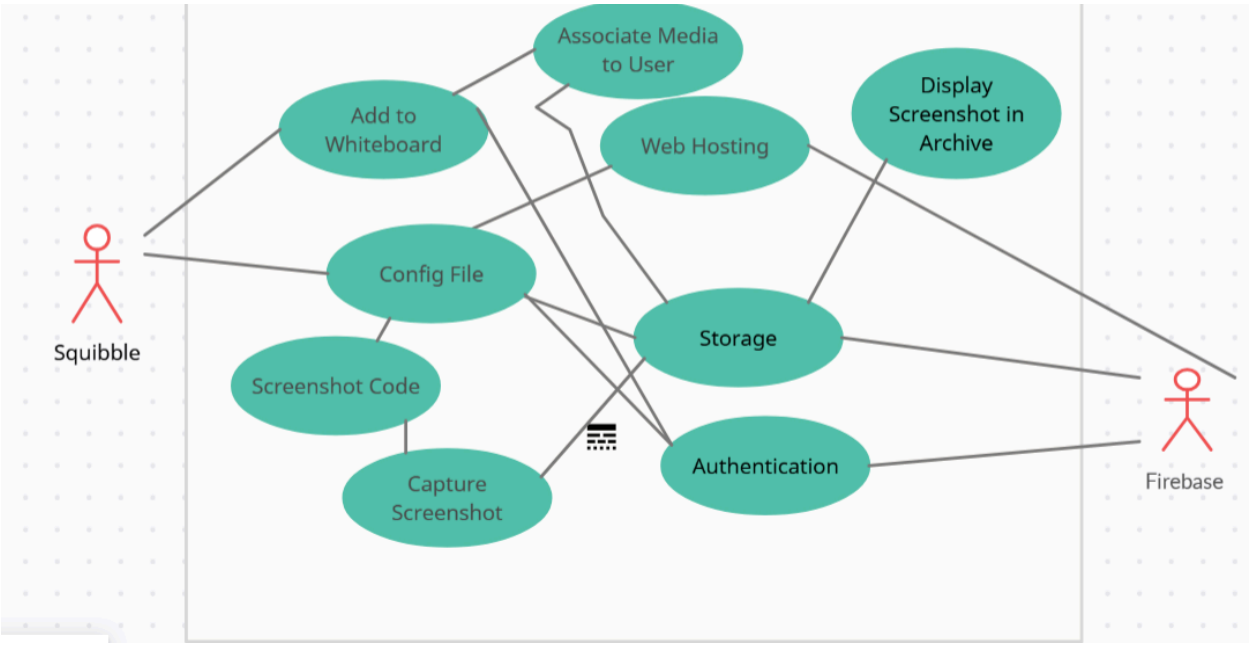
1.2.3 Architectural Design Diagrams

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

Overall System (made in Squibble!)



Firebase Integration:



Media Interaction Chart: