

StudyNest

CS 30700: Design Document

Team 23

Leanne Alsatie

Cadence Chan

Kalea Gin

Advait Panicker

Index

1. [Purpose](#)
 - a. [Functional Requirements](#)
 - b. [Non-Functional Requirements](#)
2. [Design Outline](#)
 - a. [High-Level Overview](#)
 - b. [Sequence of Events](#)
3. [Design Issues](#)
 - a. [Functional Issues](#)
 - b. [Non-Functional Issues](#)
4. [Design Details](#)
 - a. [Client Class Level Design](#)
 - b. [Host Class Level Design](#)
 - c. [Class Diagrams](#)
 - d. [UI Mockups](#)

Purpose

Studying can be more effective and enjoyable when done with others. Having a study space virtually can also be more convenient. We aim to build a study app that helps students study together virtually. Most similar services cater to more community-centered features—working anonymously with other students or joining public study rooms—while ours is more about having a personalized group study experience. As opposed to Discord, our study app is explicitly designed for studying and delivers a distraction-free easy-to-use study platform. Our study app will have more unique features (whiteboard, PDF upload, YouTube watch together bookmarks, etc), which also separates it from any individual group study platform.

The purpose of this project is to build a virtual study room that allows multiple users to get together and study together using different learning tools. This project aims to bring together many digital studying methods, such as online collaborative whiteboards and music-sharing, into one integrated application. Rather than have to go through the hurdles of setting up a voice call, adding friends to the group, and paying for the necessary software to work with others, StudyNest will make the process of setting up a group study room as easy as possible. Simply through the sharing of a room code, students will be able to join an online room already prepared and configured to their studying needs, along with many ways to communicate with their friends throughout their study session. With tools like whiteboard, PDF note-taking and sharing, and YouTube watch-togethers, we hope to make studying easier and enjoyable for all students.

Functional Requirements

Joining / Leaving

- 1) As a user, I would like to join a room with other users.
- 2) As a user, I would like to use a shareable link to join another user's room.
- 3) As a user, I would like to be able to join another user's room with a room code.
- 4) As a user, I would like to login with my Google account.
- 5) As a user, I would like to have the option to join as a guest, not having to make an account.
- 6) As a user, I would like to be able to leave a group.
- 7) As a user, I would like to not need a Google account and be able to log in with an email and password, if time allows.

Account Options

- 8) As a user, I would like to choose a profile icon to represent me from a list of icons.
- 9) As a user, I would like to be able to upload my own custom image to be my profile icon.
- 10) As a user, I would like to be able to choose a username and change it later.
- 11) As a user, I would like to be able to see all the rooms I've joined, and join and leave them as I please.
- 12) As a user, I would like to be able to change notification settings, if time allows.
- 13) As a user, I would like to be able to give friends nicknames, if time allows.
- 14) As a user, I would like to have a profile bio, if time allows.
- 15) As a user, I would like to delete my account, if time allows.
- 16) As a user, I would like to reset my password and or username, if time allows.

Room Options

- 17) As a user (host), I would like to set permissions for my groupmates.
- 18) As a user (host), I would like to customize the appearance of my room.
- 19) As a user, I would like my room to be private. Unauthorized users will be disallowed to access my room.
- 20) As a user, I would like to save my study progress to come back to later.
- 21) As a user, I would like to be able to engage in multiple rooms.
- 22) As a user, I would like to be able to mute a group, if time allows.

Functions

Whiteboard

- 23) As a user, I would like to draw on the shared whiteboard and see other users' drawings.
- 24) As a user, I would like to have different pen colors and writing tools available to interact with the whiteboard.
- 25) As a user, I would like to be able to erase on the whiteboard.
- 26) As a user, I would like to paste images into the whiteboard, if time allows.

Chat

- 27) As a user, I would like to be able to send and receive live messages from other users.
- 28) As a user, I would like to be able to send images, gifs, and stickers.

- 29) As a user, I would like to be able to ping other users and send them a notification with my message content.
- 30) As a user, I would like to be able to edit and delete my messages, if time allows.
- 31) As a user, I would like to be able to download and choose sticker packs, if time allows.
- 32) As a user, I would like to be able to react to another user's message with an emoji, if time allows.
- 33) As a user, I would like to be able to send embeds in the chat, if time allows.

File sharing/links

- 34) As a user, I would like to upload a file into the room.
- 35) As a user, I would like to see the same file as everyone else in the room.
- 36) As a user, I would like to make notes on the file that everyone else in the room can also see.
- 37) As a user, I would like to make highlights on the file that everyone else in the room can also see.
- 38) As a user, I would like to remove notes and highlights from the file.
- 39) As a user, I would like to be able to emphasize a highlight or part of the file, if time allows.
- 40) As a user, I would like to connect to Google Drive, if time allows.
- 41) As a user, I would like to export the annotated file, if time allows.

Video streaming

- 42) As a user, I would like to have a queue of videos that I can watch/listen to.
- 43) As a user, I want to pause, skip, and fast-forward the video, and have the video sync with other users in the room.
- 44) As a user, I would like to be able to make comments and annotations on the video that are persistent and can be viewed by others, if time allows.

Miscellaneous

- 45) As a user, I would like to have a clock to do pomodoro studying.
- 46) As a user, I would like to be able to receive email notifications regarding activity in my account or rooms I am in.
- 47) As a user, I would like to be able to voice chat with my friends, if time allows.
- 48) As a user, I would like to have multiple brain break activities available to play.

Non-Functional Requirements

1. Availability and Performance

- a. The app should have high availability, with an aim for 24-hour availability discounting maintenance periods.
- b. The average latency should be less than 5000 ms for user interactions in the room, including sending messages and making annotations. More real-time features, including video streaming and voice chat, should have lower latency.
- c. The system must be able to handle 10 users per room.

2. Storage

- a. User files should be stored in blob-like file storage, including documents, images, and other static content.
- b. There should be a separate database for storing smaller pieces of data, including text conversations, room metadata, and potentially annotations.
- c. User credentials and other sensitive data should be stored in a relational database.

3. Security

- a. Unauthorized users should not be able to join rooms they are not authorized to join.
- b. Users should only be able to access their static files and chat messages. This should be mitigated by only allowing users to access links to messages/media posted in rooms they are members of.
- c. Users should be notified via email of changes to their account settings, including password resetting.
- d. If a user account gets deleted, data should be saved for a certain period to allow for account recovery.

4. Usability

- a. The web application should be easy to navigate and user friendly.
- b. Users should be able to intuitively find core features. Users should not have to struggle finding important tools like the whiteboard, file sharing, and chat.
- c. The web app will be compatible with different browsers and screen sizes, if time allows.

Design Outline

High-Level Overview

This project is based on the many-to-one client-server architecture to serve a large number of users at the same time. The one server will be able to simultaneously handle access from a large number of users using the Flask and React framework. The server will accept requests from the client, access and store data in the database, and provide responses to the clients accordingly.



1. Client

- The Client provides the user an interface with our system.
- The Client sends input and requests to the server.
- The Client sends requests to validate authentication information to the server and the database to retrieve profile information.
- The Client receives and interprets responses from the server and renders changes in the user interface.

2. Server

- The Server receives and handles requests from the Client.
- The Server validates requests and sends queries to the database to add, modify, or retrieve data on demand.
- The server receives and interprets input from the Client and renders them in its interface.
- The Server generates corresponding responses and sends them to the specific Client.

3. Database

- A Firebase database stores all data in the application, including user information and event information.
- The Database responds to queries from the Server and sends requested data back to the Server.
- The Database also validates user authentication like OAuth.

Sequence of Events

Whether the user is accessing the app for the first time or not, there are multiple interactions made between the client, server, and database before the app's features are presented to them. These are outlined below, as well as in the sequence diagram.

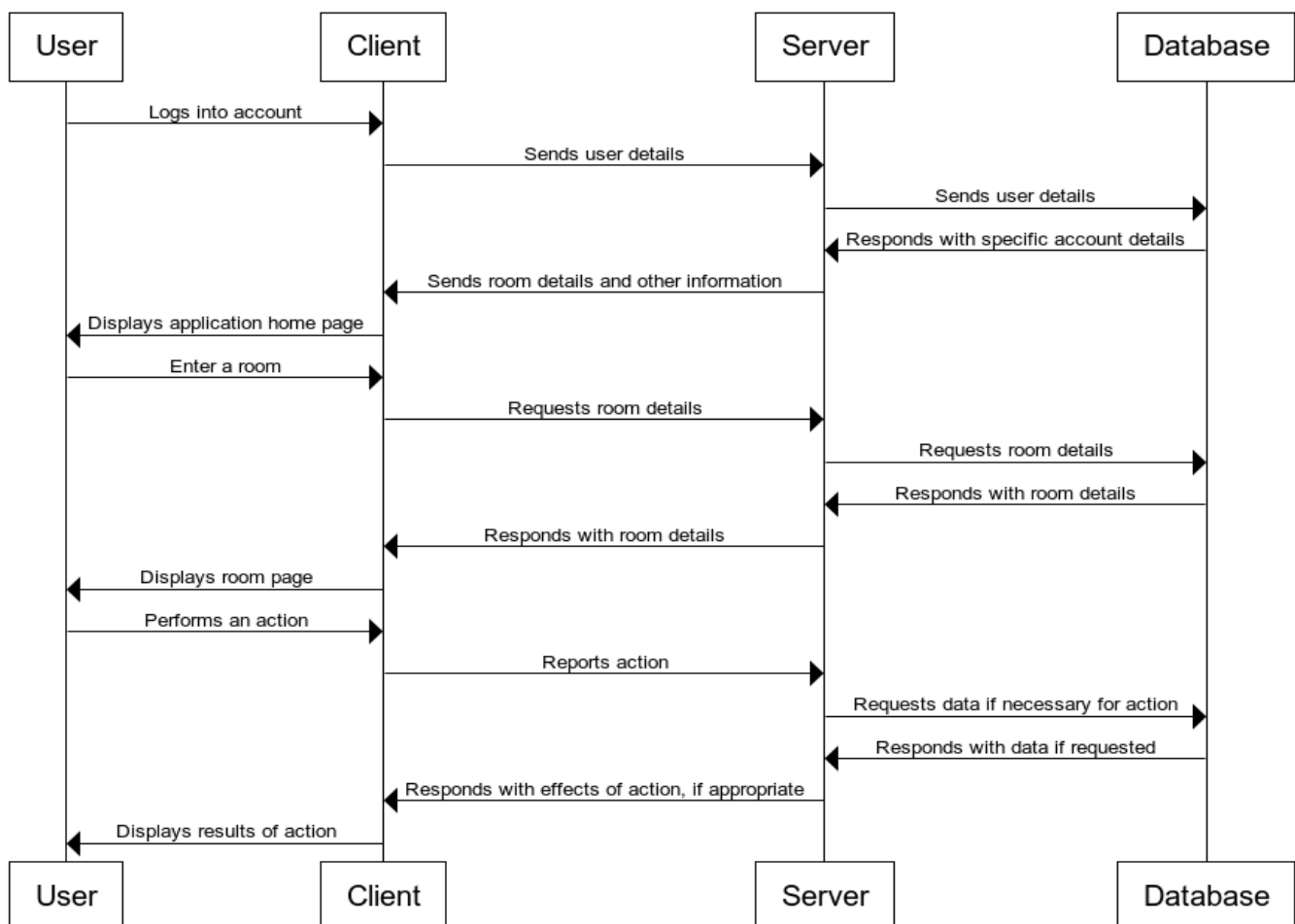
The user first accesses the page from their web browser, which is the Client. If the user is not yet logged in, the Client will display the log in page. Once the user logs in or is already logged in, the Client makes contact with the Server with the user's credentials.

Once the Client has contacted the Server with this information, the Server will use the user's authentication to communicate with the Database. The Database will respond with the appropriate data for that user, including what rooms they are a member of, notifications they have, their profile information, and any other necessary data.

The Server will respond to the Client with this necessary data, after which the Client displays the home page of the application catered to this user, the key being that they can now see which rooms they are a part of, if any, then continue interacting with the application to enter a room and begin working.

After a user has entered a room, the sequence of events will remain generally the same. The user will perform an action, the Client processes that action and sends it the Server, and, if necessary, the Server may interact with some Database before responding to the Client, which gives the user the proper feedback of that action.

Sequence Diagram



Design Issues

Functional

1. Do users need to sign in to use the virtual study rooms?
 - a. Option 1: Users can log in with a username and password.
 - b. Option 2: Users can log in using Google sign-in OAuth.
 - c. Option 3: Users can just continue as a guest.

Choice: All of the above

Justification: We decided that users should have the option to log in through all of the options. With username and password, information can be stored in the database and users can store data of their video queues, whiteboard drawings, and file annotations. Google authentication is also an option, so that users can easily have an account. If people do not want to deal with the hassle of creating an account, they can also sign in as a guest.

2. What will happen if you exit out of the whiteboard?
 - a. Option 1: All drawings will be cleared and the whiteboard is empty.
 - b. Option 2: You can go back and continue from where you left off.

Choice: Option 2

Justification: We want to ensure that studying sessions are uninterrupted, so any progress or work made on the whiteboard will be saved and users can continue from previous sessions. There will be a save function so that users are free to leave the whiteboard and come back to continue working.

3. How can I work on each function/module?
 - a. Option 1: Choose only one function at a time (whiteboard, Youtube queue, file sharing, chat).
 - b. Option 2: Have multiple windows open for each function (have to switch windows).
 - c. Option 3: Split the screen to fit multiple functions on the screen.

Choice: Option 3

Justification: We want users to be able to use any of the study tool functions, so to make it easier, we will split the screen with a dragging bar on the side, allowing users to choose multiple study tool functions that they would want to use. This also optimizes studying as everything is all laid out on the screen.

Non-Functional

1. What frontend language/framework should we use?

- a. Option 1: Raw HTML + JavaScript
- b. Option 2: Angular
- c. Option 3: React

Choice: React

Justification: React is easy to learn, as well as easy to use. It is also used by many companies and organizations, so there are plenty of tutorials and documentation that can be used for reference. React also offers fast rendering and development with virtual DOM (Document Object Model), allowing developers to test changes.

2. What API should we use for the chat interface?

- a. Option 1: Direct Client Socket (no API)
- b. Option 2: Chatscope-Use-Chat
- c. Option 3: Azure Communication Services UI Library
- d. Option 4: Stream

Choice: Chatscope-Use-Chat

Justification: Chatscope provides webhook solutions that will let us have more control versus using a whole SDK like with stream or learning a new language with Azure. Furthermore, chatscope integrates well with its sister library that has a lot of UI features that will make it easy to make visually appealing interfaces. There is also extensive documentation for both of the projects which will make debugging easier.

3. What database should we use for static content?

- a. Option 1: MongoDB
- b. Option 2: Firebase
- c. Option 3: PostgreSQL

Choice: Firebase

Justification: Firebase has a SDK that helps interfacing with the database, and it has authentication to better handle security. Also, for unstructured data like files that might be shared, Firebase will be better because it uses NoSQL.

4. What method should be used for in-room communication, such as sending app events, app updates, and other small-size/high-speed communication?

- a. Option 1: REST API
- b. Option 2: Websockets

Choice: Option 2

Justification: Though REST APIs are a strong, effective communication layer, especially with easy to use libraries like FastAPI, they simply do not meet the requirements of our real-time app the way websockets do. The primary issue is that with REST APIs, server events can only be received by the client via polling, which is not only slower but also adds overhead on either the client and server to distinguish old events from the new. Websockets, on the other hand, can take advantage of event listeners, allowing the client to asynchronously respond to events the way the server can.

5. Through what method should the window resizing feature be implemented?

- a. Option 1: Vanilla Javascript
- b. Option 2: react-split-pane

Choice: Option 2

Justification: react-split-pane has all the features required for having resizable windows or “panes” within each study room, including splitting panes vertically and horizontally. Building the system from the ground up with JS—difficult, though not terribly difficult—offers considerable control, which might be helpful considering the interactions each pane needs to have with p5.js, the common rendering tool for visual annotations in apps like WhiteBoard or FileShare. Despite the tedium that might come from configuring react-split-pane to work with our custom Apps, the time saved from initial setup will be worth it.

6. What should we use to authenticate users?

- a. Option 1: Username & Password
- b. Option 2: Google Authentication
- c. Option 3: Allow users to choose between either

Choice: Option 2

Justification: For a project of this scale, it is more appropriate to use the option that minimizes the amount of sensitive user data that the app itself is storing and handling. By integrating with Google’s Authentication system, we can 1. Make the process of implementing user authentication far simpler by using Google’s API, and 2. Make users feel more trust towards us, knowing that there will not be a flaw in our security that leads to the breach of their personal data. This also makes our application more accessible from the user’s end – rather than need to worry about yet another username and password combination to use our app, users can simply use their existing Gmail account, something that many people, especially students, already have.

7. What database should be used for saving chat data?

- a. Option 1: MongoDB
- b. Option 2: SQLite

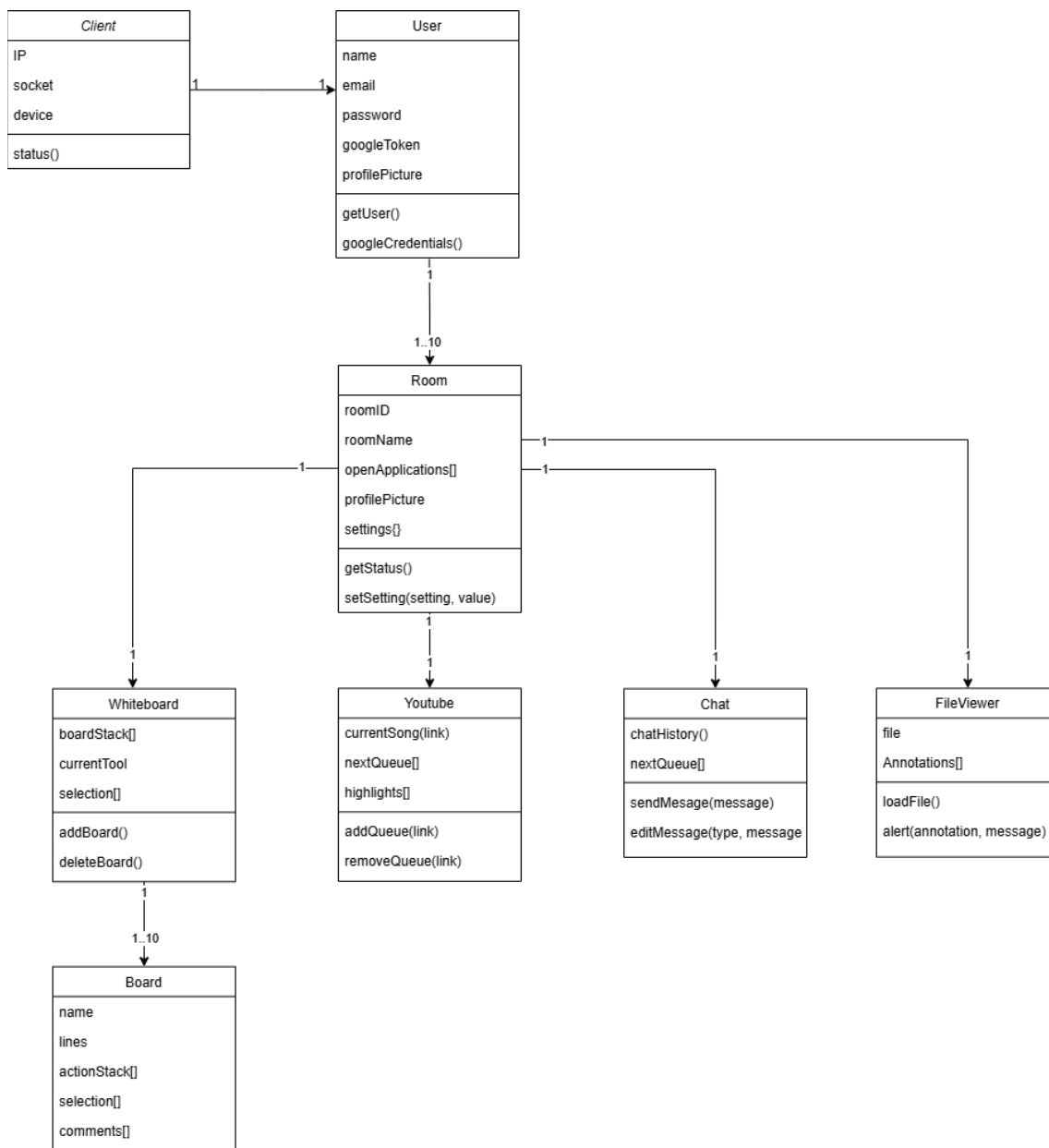
Choice: MongoDB

Justification: While SQLite has its benefits in performance and ease of implementation, MongoDB allows for a more dynamic data model that is more appropriate for storing chat information. When it comes to chat rooms, data that needs to be stored is often dynamic depending on what the users primarily use the chat rooms for. Then, MongoDB’s data structure complexity makes storing chat data easier. MongoDB is also more resilient to concurrent data queries, which allows for more scalability when it comes to the growth of the application and the existence of multiple rooms.

Design Details

Client Class Level Design

The below diagram demonstrates the connection between classes in our project in the context of the connection between the user and a particular room. While the Client class will take care of the network connection and communication with the server, it is the User class that holds all the current user information, and ultimately the User class is what interacts with the information provided by the server regarding a specific room to manage the user's interaction within that room.

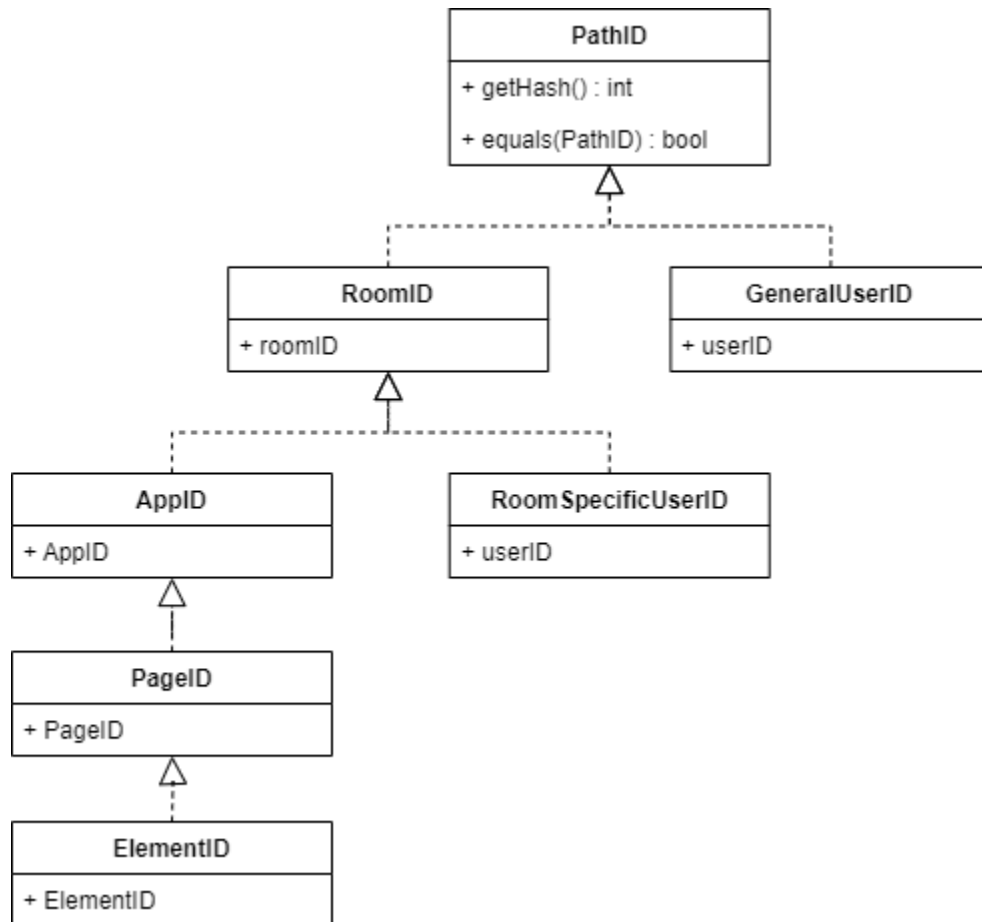


Description of Client Classes and Their Interactions

- The classes are a blueprint of the options and functions of the study room.
- Client
 - Client manages the server client operations
 - Handles the actual client and socket opening
- User
 - User that is logged in or guest
 - Holds user credentials for google functions
- Room
 - Each room object holds references to the states of the other activities
 - Manages the open activities
- Whiteboard
 - Each whiteboard represents one whiteboard app, with a list of boards
- Board
 - Each board represents one drawings
 - Stores the current selection and tools
- Youtube
 - Represents a youtube video
 - Possesses a queue to allow for group playlists
- Chat
 - Represents a group chat for the room
 - Contains a field for sticker packs available to the user
- FileViewer
 - Represents a file that can be shared and annotated with the user's note
 - Can send out an annotation that will send a link in the chat to show a highlight of a file

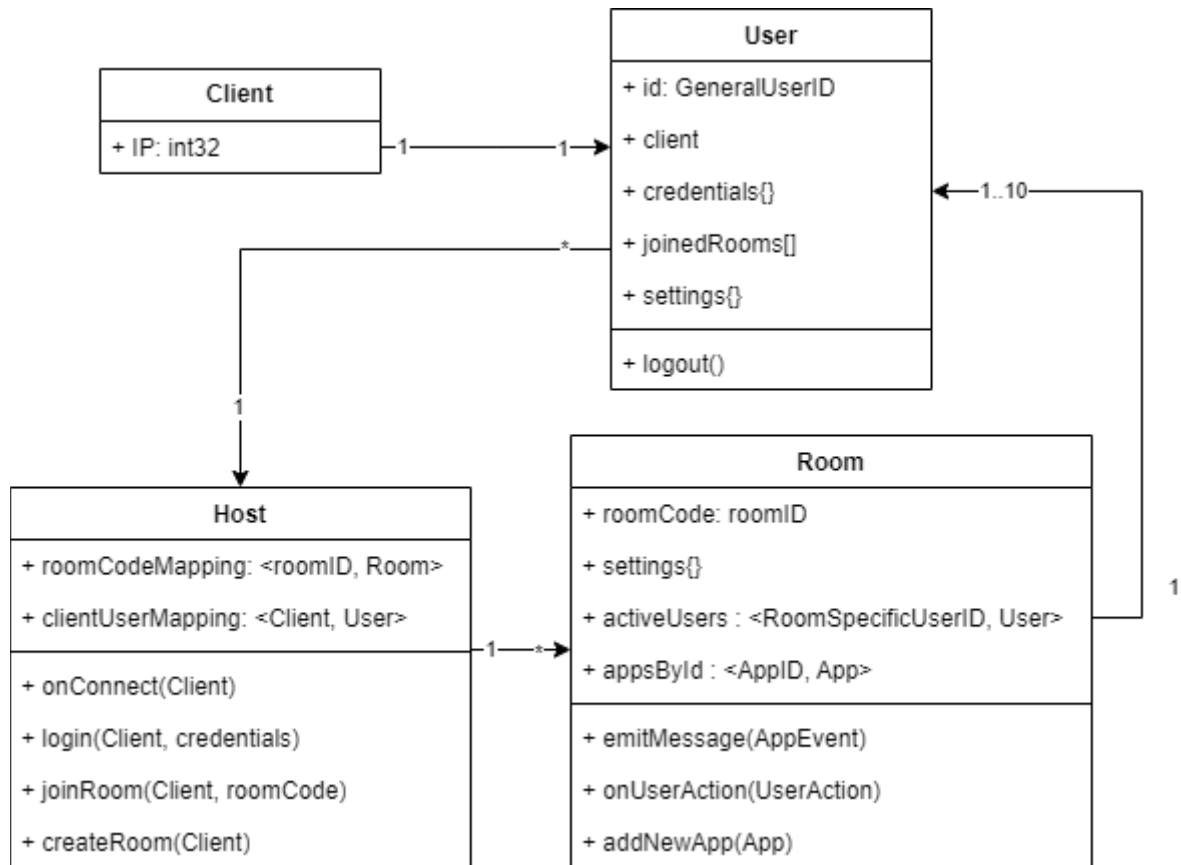
Host Class Level Design

The following diagram maps out the organization of data on the application backend. This includes how the data itself is stored in the database as well as how the Server itself queries the data.



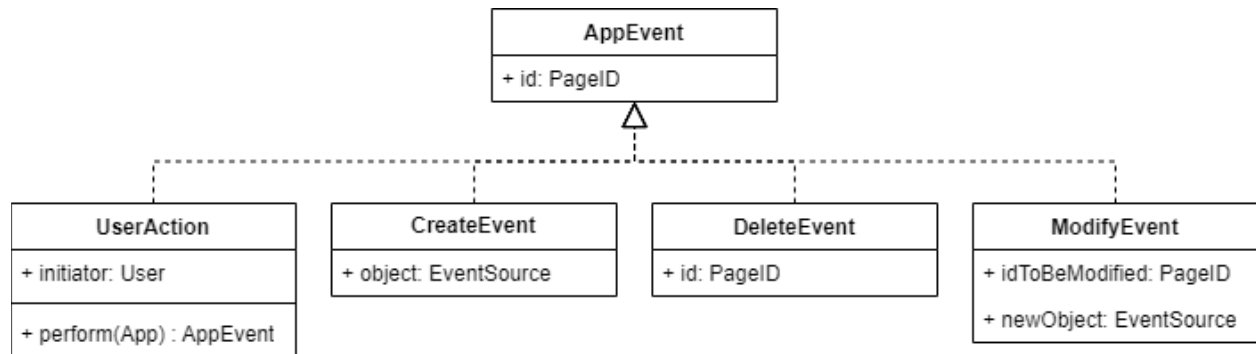
- PathID
 - System for identifying components with both broad and granular specificity, and storing items in map tool like EventSourceManager
 - When events occur in the room,
 - Each subsequent class implementation stores another key for matching, decreasing the chance of collisions

Continuing from the previous diagram, this diagram shows how the system between User(s), a Host, and a Room functions.



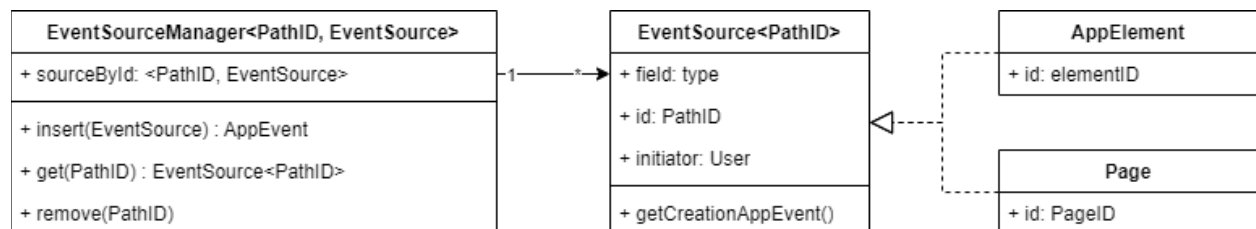
- **User**
 - A wrapper for Client holding additional information, all defaulting to null if the User is a guest
- **Host**
 - Manages connecting Clients (sockets), mapping them to Users, and then mapping join codes to Rooms
 - Handles Room creation and joining
 - Handles User login
- **Room**
 - Contains information relevant to an active room, including currently active Users, Apps, and room settings
 - Emits messages to entire room via a websocket
 - Handles incoming user actions entering via a websocket

This diagram demonstrates how events are triggered by user actions during their use of the application.



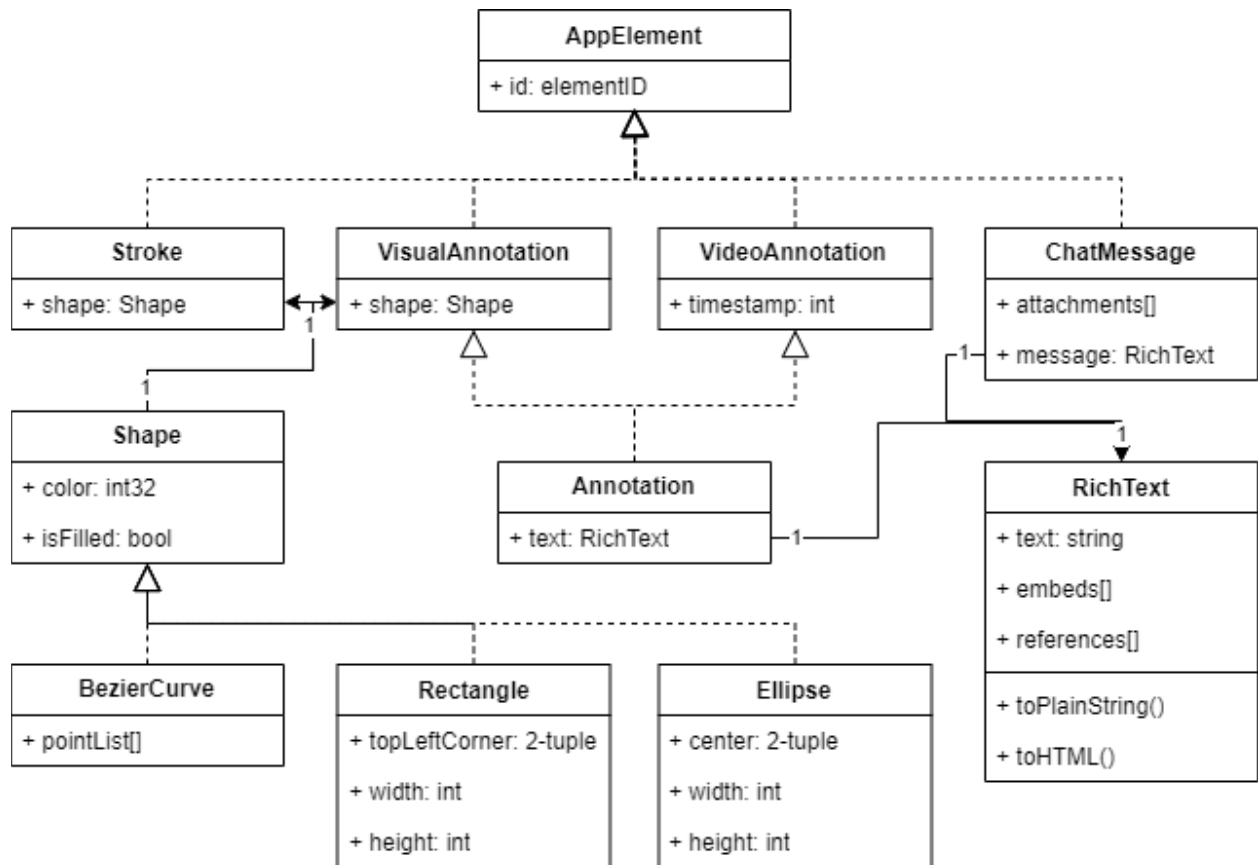
- **AppEvent**
 - The primary interface that connects the User-Host-Room system with each App.
 - Requires PageID level specificity when creating, deleting, or modifying an App.
- **UserAction**
 - The input for the App, which also inherits from AppEvent (since the action itself is often enough to broadcast, without indirect consequences from each App)
 - Contains a method to perform the UserAction on a specified App.

This diagram also relates to the event system, modeling how events are recorded in the server backend.



- **EventSource**
 - Potential source for events, including:
 - **AppElement** – any modifiable, user generated aspect of an App (sketches, annotations, comments)
 - **Page** – a collection of AppElements, again created typically by the user
 - Has `getCreationAppEvent()` for initial load purposes
- **EventSourceManager**
 - Acts as a wrapper for a mapping PathID's to EventSources
 - Generates AppEvent upon inserting any new EventSource

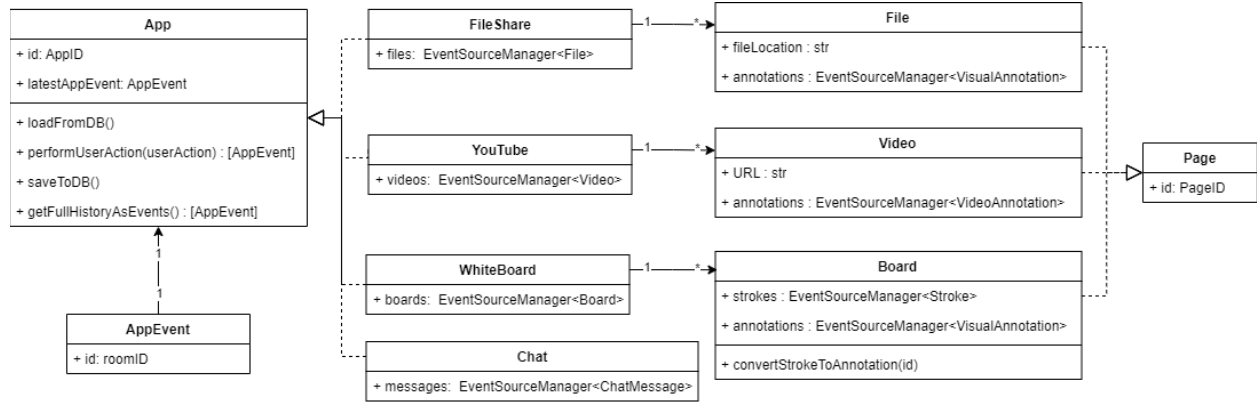
This diagram represents the general structure of the different user interactions with the study tools.



- **AppElement**
 - Any user generated aspect of an App
- **Shape**
 - Abstract class for any kind of shape, includes Rectangles, Ellipses, and BezierCurves, all defined with primitives
- **Stroke**
 - Simple Shape, used for WhiteBoard App
- **RichText**
 - An enhanced text class that allows for embeds and references, with built-in HTML conversion
- **ChatMessage**
 - Contains RichText message and attachments as URLs
- **Annotation**
 - Abstract class for any kind of user generated annotation
 - **VisualAnnotation**
 - Annotation with a Shape, typically filled, with limited palette, for either WhiteBoard or FileShare Apps
 - **VideoAnnotation**

- Annotation with associated timestamp, for YouTube App

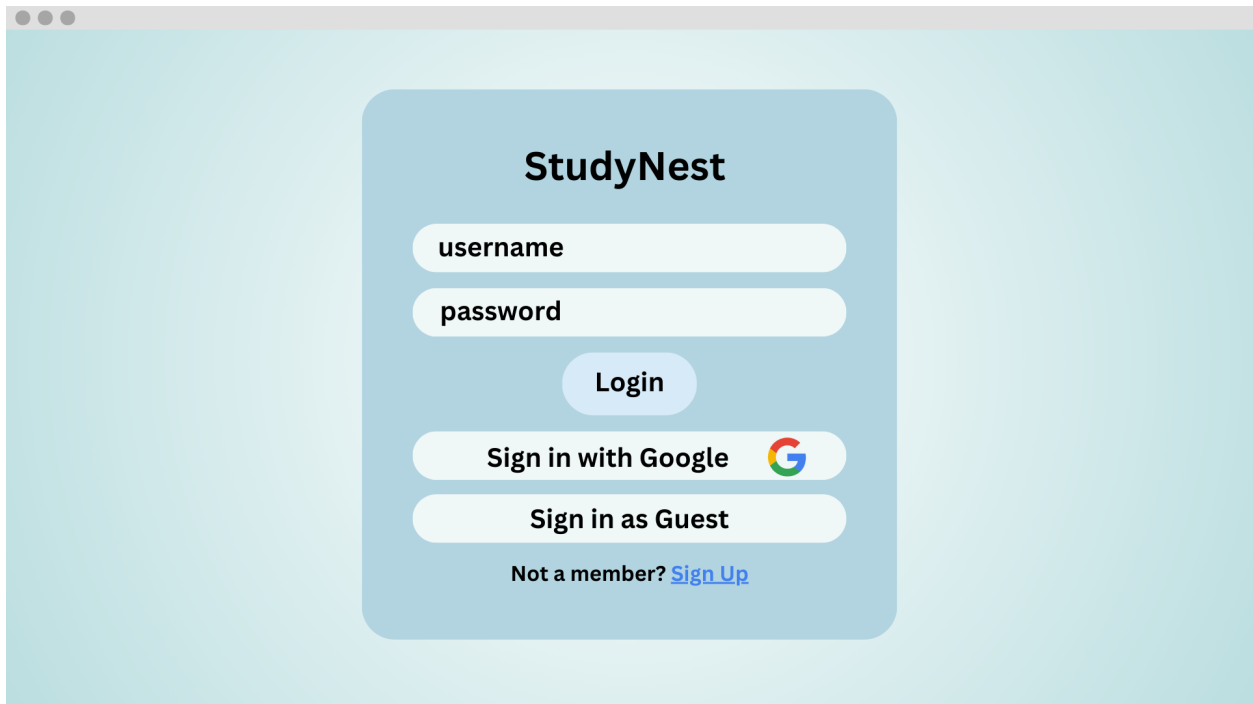
The following diagram relates the main page of the application to the classes corresponding to each unique study tool feature.



- App
 - Abstract class only containing AppID and lastAppEvent, useful for checking for synchronization between server and client
 - Ability to load from and save to the remote database
 - Performs a UserAction, returning a list of AppEvents to push to Users
 - Ability to retrieve current App state as list of CreateEvents, useful for initial User load
- FileShare
 - Contains list of Files (EventSourceManager)
 - File
 - File location as URL
 - VisualAnnotations with EventSourceManager
- YouTube
 - Contains list of Videos (EventSourceManager)
 - Video
 - Video URL
 - VideoAnnotations with EventSourceManager
- WhiteBoard
 - Contains list of Boards (EventSourceManager)
 - Board
 - Strokes and VisualAnnotations with EventSourceManager
 - Converting Stroke to VisualAnnotation

UI Mockup

Login Page




A mockup of a login page for 'StudyNest'. The page has a light blue background. In the center, there is a white rounded rectangle containing the login form. The form includes a title 'StudyNest', two input fields for 'username' and 'password', a 'Login' button, a 'Sign in with Google' button with the Google logo, a 'Sign in as Guest' button, and a link 'Not a member? Sign Up'.

StudyNest

username

password

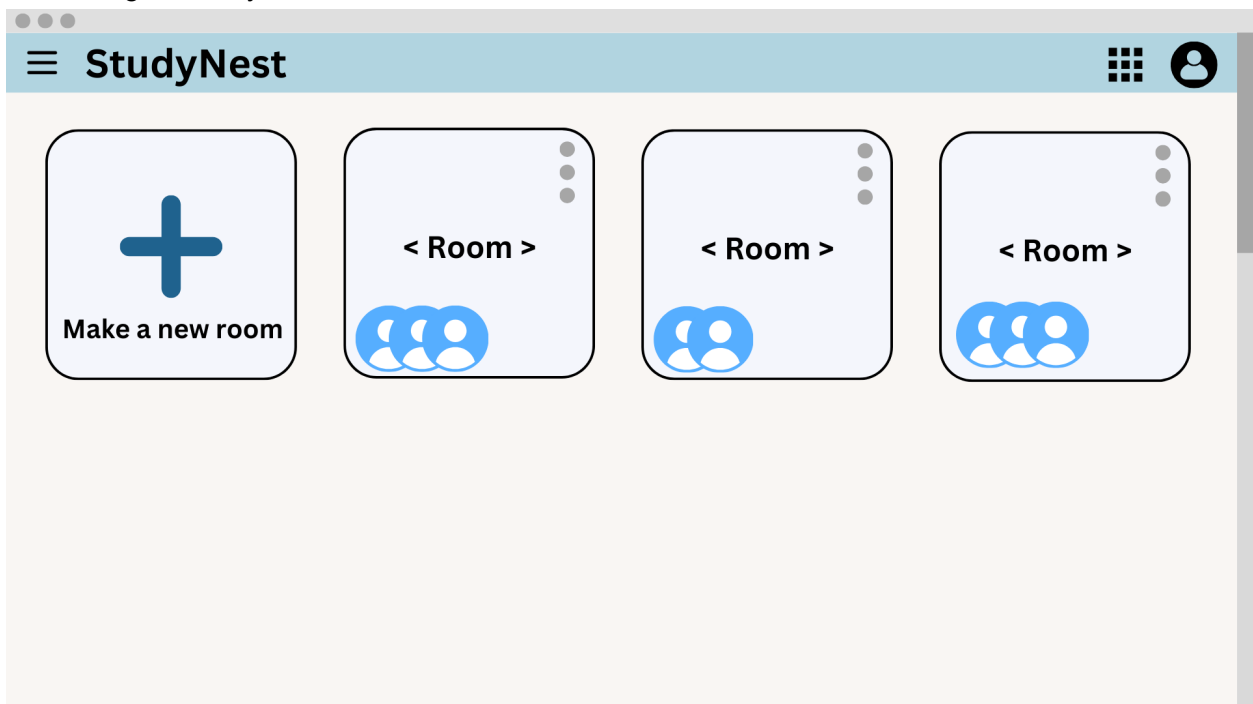
Login

Sign in with Google 

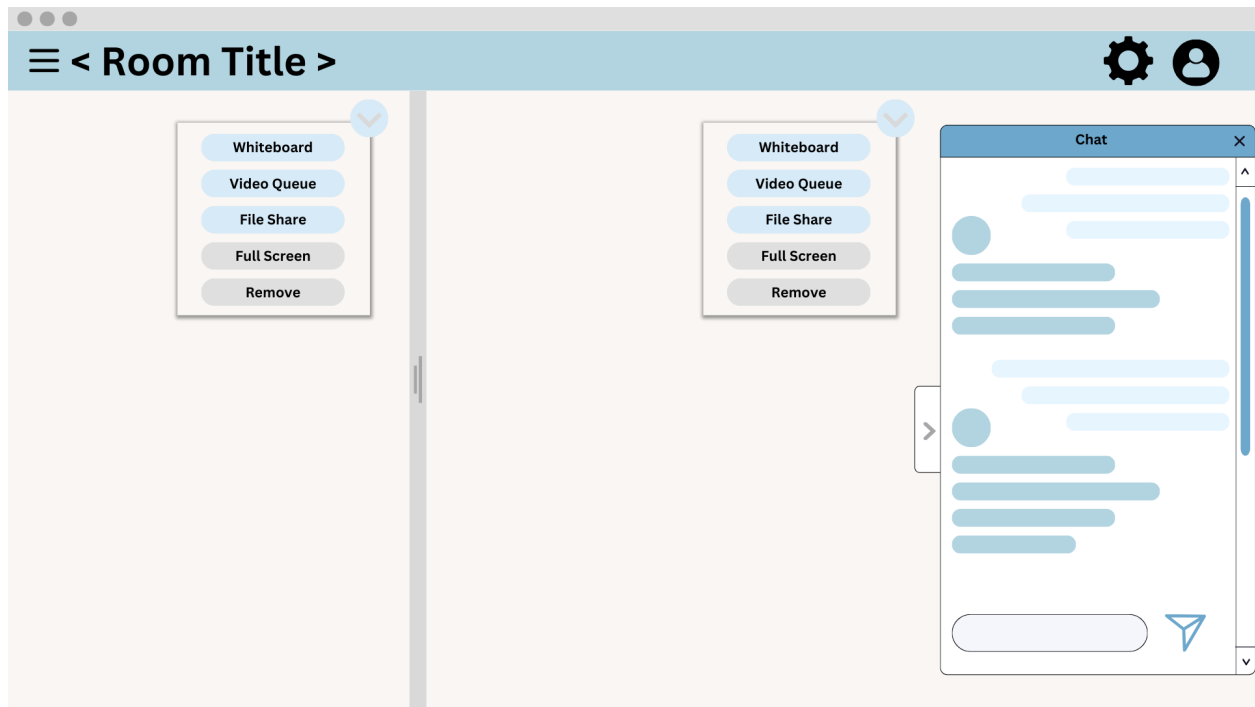
Sign in as Guest

Not a member? [Sign Up](#)

Home Page / Lobby



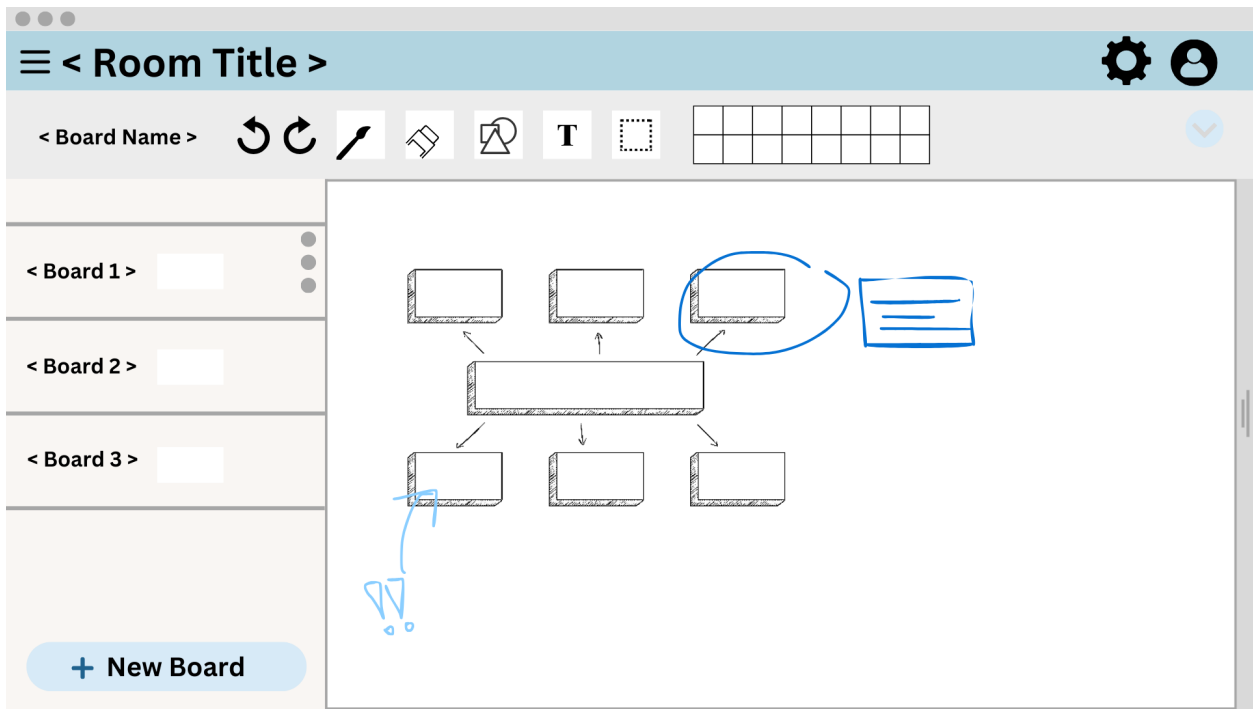
Room Page



YouTube Function



Whiteboard Function



File Sharing Function

