

# Online Shared Whiteboard

Virtuální sdílená tabule

TUAN HUY TRAN

Bachelor

Supervisor: Ing. Svatopluk Štolfa, Ph.D.

Ostrava, 2022

# Bachelor Thesis Assignment

Student:

**Tuan Huy Tran**

Study Programme:

B2647 Information and Communication Technology

Study Branch:

2612R025 Computer Science and Technology

Title:

Online Shared Whiteboard

Virtuální sdílená tabule

The thesis language:

English

Description:

The goal of this work is to develop an online shared whiteboard. Whiteboard will allow to securely share content to other users by link, will be secured by password and encryption of the content. Other functionalities will include: load and save of the content to the participant computer, if allowed, import of images and files, possibility to input text, free drawing etc.

1. Study the similar apps and compare them.
2. Develop and architecture and design of own application.
3. Test the functionality and compare it to the other tools.

References:

- [1] Pfleeger, Shari Lawrence, and Joanne M. Atlee. 2009. Software Engineering: Theory and Practice: Prentice Hall, ISBN 0136061699
- [2] Pressman, Roger S. 2010. Software Engineering : A Practitioner's Approach. 7th ed. New York: McGraw-Hill Higher Education, ISBN 9780073375977
- [3] Sommerville, Ian. 2010. Software Engineering. 9th ed, International Computer Science Series. Harlow: Addison-Wesley, ISBN 978-0137035151

Other literature according to the needs and suggested by the supervisor.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor:

**Ing. Svatopluk Štolfa, Ph.D.**

Date of issue:

01.09.2021

Date of submission:

30.04.2022

---

doc. Ing. Petr Gajdoš, Ph.D.  
*Head of Department*

---

prof. Ing. Jan Platoš, Ph.D.  
*Dean*

## **Abstract**

The goal of this work was to develop the architecture and design a whiteboard for sharing online. Whiteboard will allow content to be shared securely with other users with a link, which will be password protected and encrypted. Other features will include: loading and saving of content to the participant's computer, if allowed; import of images and files; possible input text, hand-drawing, etc., and finally comparing to other existing or similar applications.

## **Keywords**

Whiteboard online sharing; web application; free drawing; JavaScript; Canvas API; Firebase; React;

## **Abstrakt**

Cílem této práce bylo vyvinout a navrhnout vlastní aplikaci pro online virtuální sdílenou tabuli. Tabule umožní bezpečné sdílení obsahu s ostatními uživateli pomocí odkazu, který bude chráněn heslem a zašifrován. Mezi další funkce patří: načítání a ukládání obsahu do počítače účastníka, pokud je to povoleno; import obrázků a souborů; vložení textu text, ruční kreslení atd. a nakonec porovnání podobnými existujícími aplikacemi.

## **Klíčová slova**

Virtuální sdílená tabule; webová aplikace; kreslení zdarma; JavaScript; Canvas API; Firebase; React;

## **Acknowledgements**

I would like to express my gratitude to Ing. Svatopluk Štolfa, Ph. D, for his kind and patient guidance, moral support, as well as valuable advice and comments, which helped me to improve my work.

# Table of Contents

<b>Acknowledgements .....</b>	<b>4</b>
<b>List of symbols and abbreviations used .....</b>	<b>8</b>
<b>List of Table.....</b>	<b>9</b>
<b>List of Figure .....</b>	<b>10</b>
<b>List of code analysis.....</b>	<b>12</b>
<b>Chapter 1: Introduction .....</b>	<b>13</b>
<b>Chapter 2: Comparison and expected functionality.....</b>	<b>14</b>
<b>2.1: Comparison .....</b>	<b>14</b>
2.1.1: Description .....	14
2.1.2: Specific description .....	14
2.1.3: How to create a Whiteboard's room and invite members room? .....	16
2.1.4: Tools .....	16
2.1.5: Upload/Download .....	21
2.1.6: Communicate .....	21
<b>2.2 Expected functionality:.....</b>	<b>22</b>
Instruction: .....	22
2.2.1: Tool.....	23
2.2.2: Working with object .....	24
2.2.3: History of event .....	24
2.2.4: Communicate .....	24
2.2.5: Upload and Download file .....	24
<b>2.3: Summary .....</b>	<b>24</b>
<b>Chapter 3: Diagram .....</b>	<b>25</b>
<b>3.1: Use Case Diagram .....</b>	<b>25</b>
3.1.1: Introduction.....	25
Use Case description .....	26
3.1.2: Summary .....	32
<b>3.2: Class Diagram .....</b>	<b>32</b>
3.2.1: Introduction.....	32
3.2.2: Description Class Diagram .....	33

3.2.3: Summary .....	35
<b>3.3: Component Diagram .....</b>	<b>35</b>
3.3.1: Introduction.....	35
3.3.2: Description Component diagram .....	35
3.3.3: Summary .....	38
<b>3.4: Sequence Diagram .....</b>	<b>38</b>
3.4.1: Introduction.....	38
3.4.2: Summary .....	45
<b>3.5: The State Diagram.....</b>	<b>45</b>
3.5.1: Introduction.....	45
3.5.2: Description State diagram .....	47
3.5.3: Summary .....	47
<b>Chapter 4: Implementation.....</b>	<b>48</b>
<b>4.1: Required programming language and Library:.....</b>	<b>48</b>
<b>4.2: Application Programming Interface (API) and technology.....</b>	<b>48</b>
4.2.1: Canvas .....	48
4.2.2: Roughjs .....	48
4.2.3: Perfect-freehand .....	48
4.2.4: Git .....	49
4.2.5: Firebase .....	49
<b>4.3: Implementation application functionality: .....</b>	<b>49</b>
4.3.1: Introduction:.....	49
4.3.2: The ideas to draw an object into canvas.....	49
4.3.3: Use Tools .....	51
4.3.4: Undo/Redo .....	53
4.3.5: Clear.....	54
4.3.6: Import/Export file.....	54
4.3.7: Adding member .....	57
4.3.8: Giving control .....	62
4.3.9: Accessing via Link .....	65
4.3.10: Login .....	65

4.3.11: Use chat .....	68
4.3.12: Get element at position .....	71
4.3.13: Moving an element: .....	74
4.3.14: Resizing an element: .....	75
<b>Chapter 5: Result.....</b>	<b>79</b>
<b>5.1: Instruction .....</b>	<b>79</b>
<b>5.2: Overall results.....</b>	<b>79</b>
<b>5.3: Compare results with popular whiteboard and method of improvement.....</b>	<b>85</b>
5.3.1: Compare result with popular whiteboard.....	85
5.3.2: Improvement.....	88
<b>Chapter 6: Conclusion.....</b>	<b>90</b>
<b>How to install project for developer? .....</b>	<b>90</b>
<b>References .....</b>	<b>91</b>

## List of symbols and abbreviations used

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – Hyper Text Markup Language

JSON – JavaScript Object Notation

URL – Uniform Resource Locator

PNG – Portable Network Graphics

JPEG – Joint Photographic Experts Group

SGV – Scalable Vector Graphics

PDF – Portable Document Format



## List of Table

Table 1: Compare popular whiteboard function.....	22
Table 2: Comparison project with another whiteboard.....	87

## List of Figure

Figure 2. 1: Describe micro.com's room creation options .....	16
Figure 2. 2: Describe Tutorialspoint.com's room creation options.....	16
Figure 2. 3: describe micro.com's drawing.....	17
Figure 2. 4: describe tutorialspoint's drawing.....	17
Figure 2. 5: Describe whiteboard.fi's shape .....	18
Figure 2. 6: Describe Micro.comi's shape.....	18
Figure 2. 7: Describe tutorialspoint's shape.....	19
Figure 2. 8: : Describe whiteboard.fi's text and image.....	19
Figure 2. 9: Describe Micro.com's text and image .....	20
Figure 2. 10: Describe tutorialspoint's text and image .....	20
Figure 2. 11: Describe Micro.com's Upload and download .....	21
Figure 3. 1: Use Case Diagram.....	25
Figure 3. 2: Class diagram.....	33
Figure 3. 3:Describe data synchronization between client and firebase .....	33
Figure 3. 4: Component Diagram .....	35
Figure 3. 5: User firebase data description .....	36
Figure 3. 6: Rooms firebase data description.....	37
Figure 3. 7: Messages firebase data description .....	37
Figure 3. 8: Boards firebase data description.....	38
Figure 3. 9: Login sequence diagram.....	39
Figure 3. 10: Use tools sequence diagram .....	40
Figure 3. 11: Import and export sequence diagram.....	41
Figure 3. 12: : Give control sequence diagram.....	42
Figure 3. 13: Adding member sequence diagram .....	43
Figure 3. 14: Chat sequence diagram .....	44
Figure 3. 15: State diagram .....	46
Figure 4. 1: Example using Roughjs + canvas to draw a object .....	48
Figure 4. 2: Elements example .....	49

Figure 4. 3: Add members description .....	57
Figure 4. 4: Invite by share link description .....	59
Figure 4. 5: The keywords description .....	59
Figure 4. 6: Invite members with name description .....	60
Figure 4. 7: Give control to members with name description .....	62
Figure 4. 8: Example of messages stored at Firestore .....	69
Figure 4. 9: Chat box result .....	71
Figure 4. 10: For example about resizing element.....	76
Figure 5. 1: Authentication interface .....	79
Figure 5. 2: Viewer dashboard .....	80
Figure 5. 3: Controller dashboard .....	80
Figure 5. 4: Use tools .....	81
Figure 5. 5: example of rectangles Actions and rectangles Options .....	81
Figure 5. 6: Controller other functions.....	81
Figure 5. 7: General functions .....	82
Figure 5. 8: Mobile dashboard version.....	82
Figure 5. 9: User management.....	83
Figure 5. 10: Use Chat box function .....	84
Figure 5. 11: Use tools interface .....	86

## List of code analysis

Code 4. 1: Draw on the canvas example .....	50
Code 4. 2: Draw element function description .....	51
Code 4. 3: Create element function .....	52
Code 4. 4: Update Element function .....	52
Code 4. 5: useHistory function description .....	53
Code 4. 6: Undo/Redo description .....	54
Code 4. 7: Clear function description .....	54
Code 4. 8: Export JSON file description .....	55
Code 4. 9: Import JSON file description .....	56
Code 4. 10: How to encrypt and transfer a URL.....	58
Code 4. 11: List room interface .....	58
Code 4. 12: Authentication layer description.....	66
Code 4. 13: Application Layer Description .....	67
Code 4. 14: add document to Firestore functions .....	68
Code 4. 15: Send messages description .....	69
Code 4. 16: Function useFirestore description .....	70
Code 4. 17: use useFirestore() for update new messages description .....	70
Code 4. 18: Show messages description .....	71
Code 4. 19: Calculate the distance .....	72
Code 4. 20: Check the cursor is inside of line.....	72
Code 4. 21: Check the cursor is inside of rectangle.....	72
Code 4. 22: Check the cursor is inside of circle .....	73
Code 4. 23: Check the cursor is inside of pencil's line.....	73
Code 4. 24: Get element at position description .....	73
Code 4. 25: Check element at position function .....	77
Code 4. 26: resized Coordinates function .....	78
Code 4. 27: Update new size description .....	78

# Chapter 1: Introduction

During the past 20th century, our generation of people has lived in a healthy working and learning environment. Children today grow up in completely different environments than in previous generations - the world has become more global, inclusive, and interactive. Especially the covid-19 pandemic that has taken place in recent years, it has changed the way we view human interaction. We are in the development of digital technology, which makes us motivated to develop a tool to support human interaction without having to worry about space, time or place. That's why we wanted to present our online whiteboard design to serve the current and future needs of us being able to interact in a virtual 3D (metaverse) world. Therefore, the goal of this graduate thesis is to develop a whiteboard to be shared online and allow sharing of content with other users securely.

The main objectives of this project:

- Develop an online shared whiteboard
- Study the similar apps and compare them.
- Develop architecture and design of own application.
- Test the functionality and compare it to the other tools.

## Chapter 2: Comparison and expected functionality

Understanding and comparing popular whiteboards is one of the key steps in figuring out the pros and cons of developing a project. From there, we will focus on giving essential features of an application and try not to focus too much on unnecessary things but instead on simple things that can still solve the problem quickly and simply. After much searching and sifting, I have selected three popular whiteboard projects to research: whiteboard.fi [1], miro.com [2], tutorialspoint [3].

### 2.1: Comparison

#### 2.1.1: Description

In this section, I will describe some important functions of a whiteboard and how they differ from other popular whiteboards. I will consider from the user's perspective, which function will bring the best experience for them, so that I can get ideas to implement my project.

#### 2.1.2: Specific description

- **Whiteboard.fi:** whiteboard.fi is a free online tool which allows you to create a virtual classroom where students each have a digital whiteboard. The students only see their own whiteboards and the teachers, whereas the teacher can see all the students' whiteboards immediately.

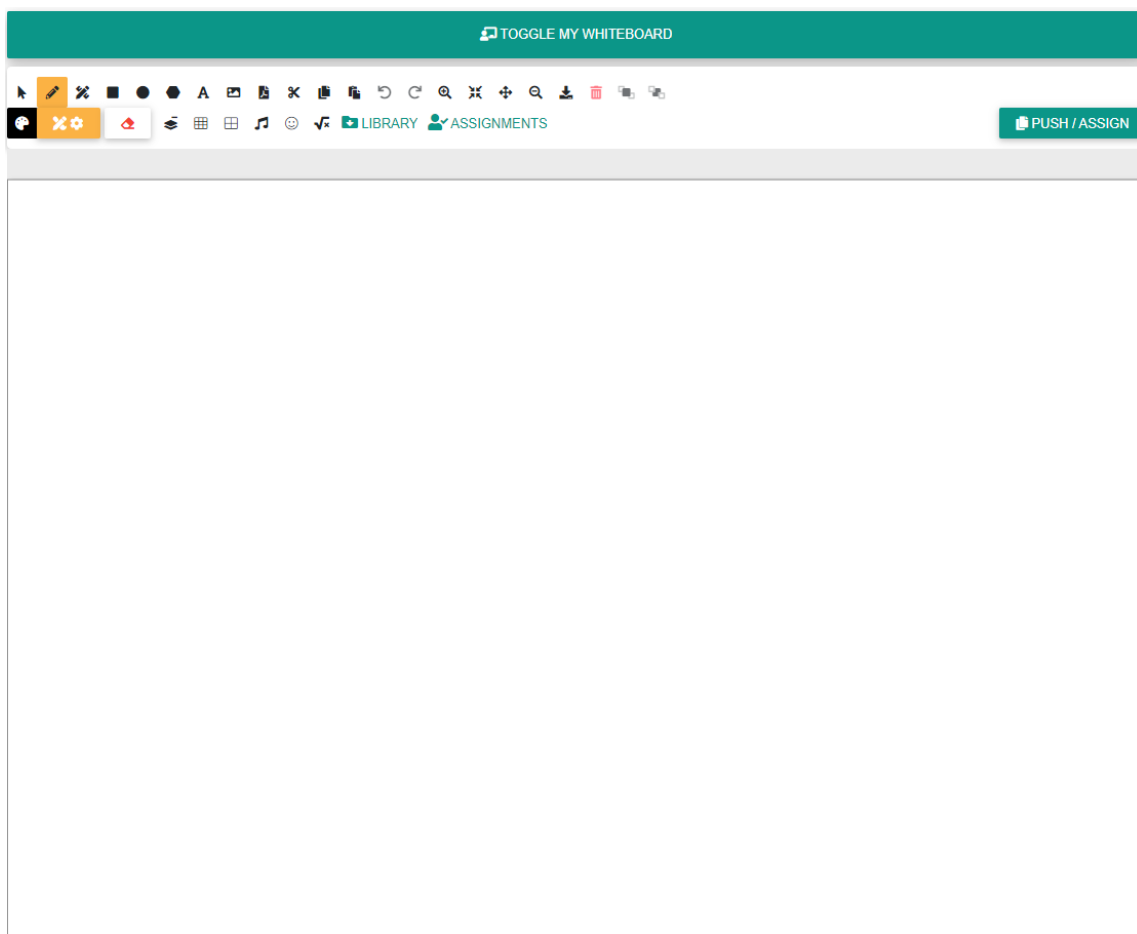


Figure 2.1 : Whitebaord.fi dashboard

- **Miro.com:** Miro is an online collaborative whiteboard platform that allows teams to work together effectively. Miro takes full advantage of collaboration capabilities, making cross-functional teamwork easy and organizing meetings and seminars: use video chat, give presentations, share.

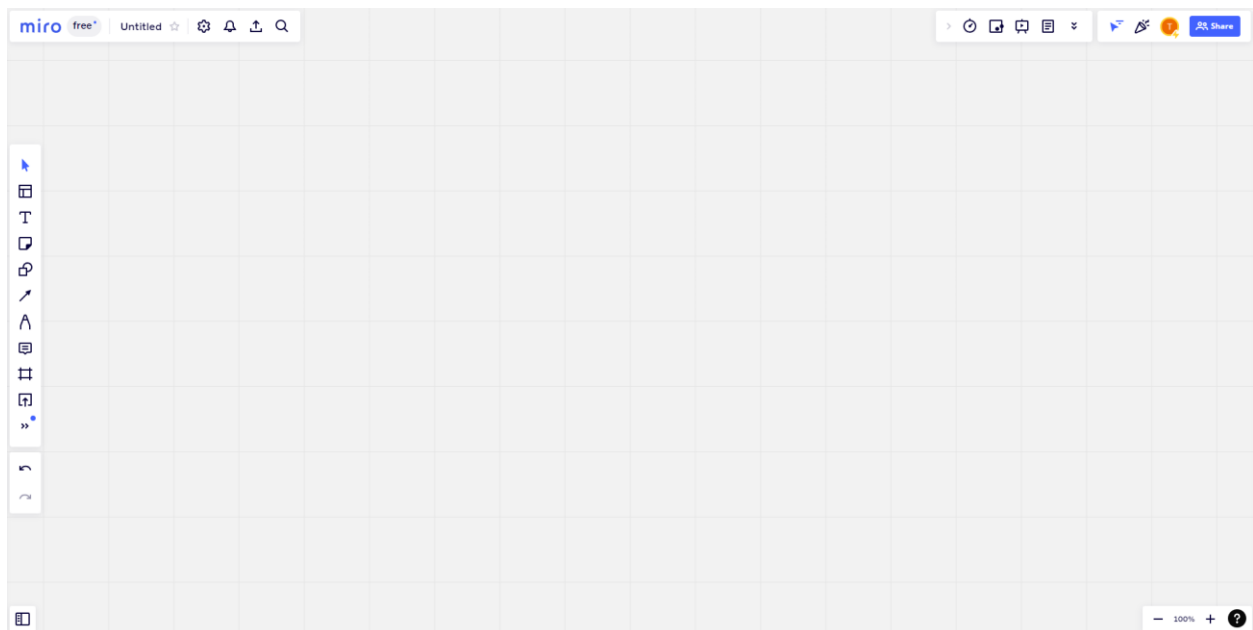


Figure 2.2 : Miro.com dashboard

- **Tutorialspoint:** Tutorialspoint whiteboard will find all the basic functions like adding images, text, drawings, annotations, and it's free.

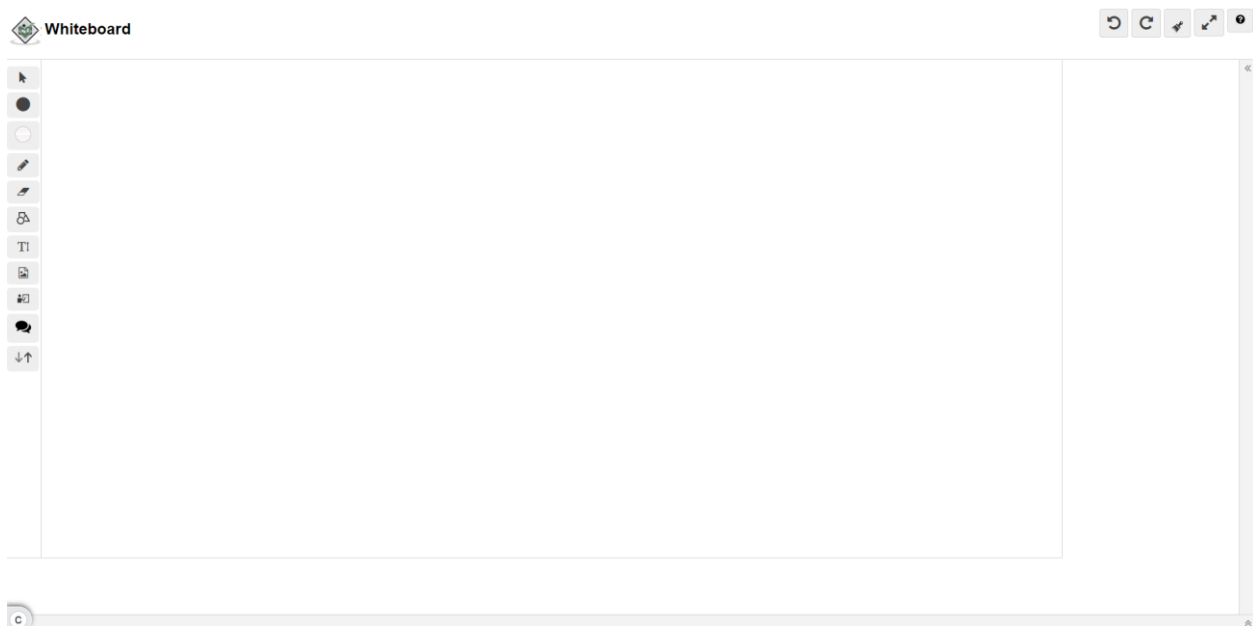


Figure 2.3 : tutorialspoint dashboard

### 2.1.3: How to create a Whiteboard's room and invite members room?

- **Whiteboard.fi:** When 'Create New Class', a unique code and a URL are generated which can be shared with students. After 'close the room', the code is no longer valid and the whiteboards are deleted. In addition, there are related functions such as:
  - Lock room.
  - Enable waiting lobby.
- **Miro.com:** We can invite users into the room using email, Slack addresses, or send unique links so that they can directly access the website.

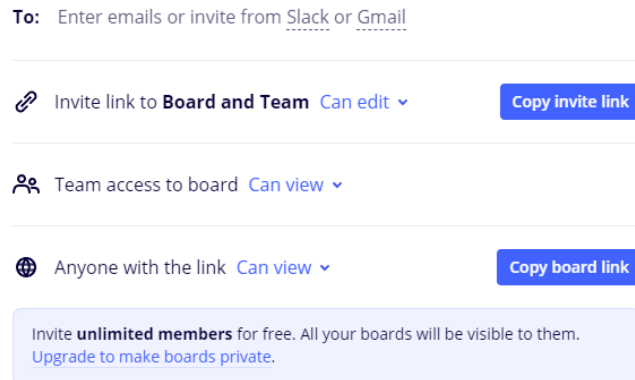


Figure 2. 1: Describe miro.com's room creation options

- **Tutorialspoint:** The initial room will always be single, but to be shared and use multiplayer functions, the room will be initialized with a unique ID and room name.

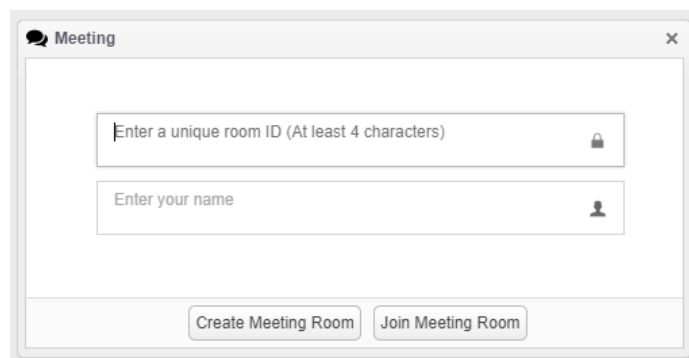


Figure 2. 2: Describe Tutorialspoint.com's room creation options

### 2.1.4: Tools

#### 2.1.4.1: Drawing:

- **Whiteboard.fi:** is the simplest, it consists of drawing and picking color. In addition, you can draw advanced such as drawing lines, drawing arrows, or shape brush. When we use mouse clicks and move coordinates, we will create a line or any pixel we want.
- **Micro.com:** Miro's pencils are designed with more variety, they include: Pencil, highlighter, smart brush, eraser, lasso, color, thickness.



- Smart drawing: is an algorithm built to recognize images that are almost like squares, circles, triangles. When a picture is drawn with a shape that closely resembles the pattern, it will immediately change to that shape
- Lasso: when circling any object, they will be selected and the selected object can change color, thickness.

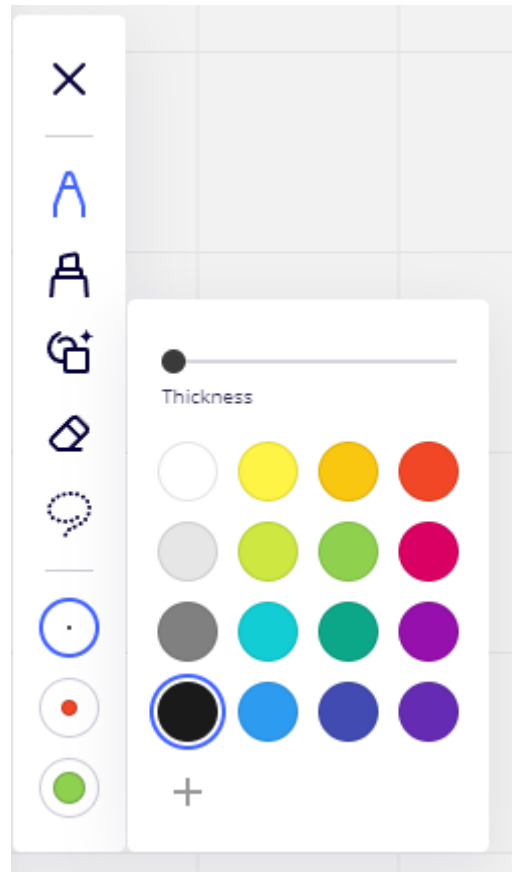


Figure 2. 3: describe micro.com's drawing

- **Tutorialspoint:** Pencil is designed to be simple, including a preset color palette, thickness, and finally a full color palette.



Figure 2. 4: describe tutorialspoint's drawing

### 2.1.4.2: Shape

- **Whiteboard.fi:** Shape is always the simplest, with three basic shapes: circle, square, triangle. Of course, there will be selected colors; a shape is created by dragging and dropping, or using a mouse click to create.

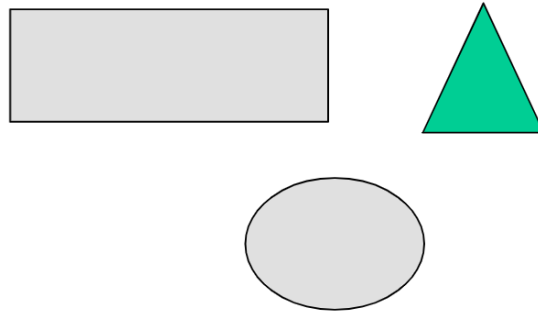


Figure 2. 5: Describe whiteboard.fi's shape

- **Micro.com:** The shapes here are designed in a variety of ways, in addition to simple shapes, we can design shapes and save them for later use. In this case, we need to use lasso to select multiple objects, then save as a template, when we want to use them we can add them just like adding a template.

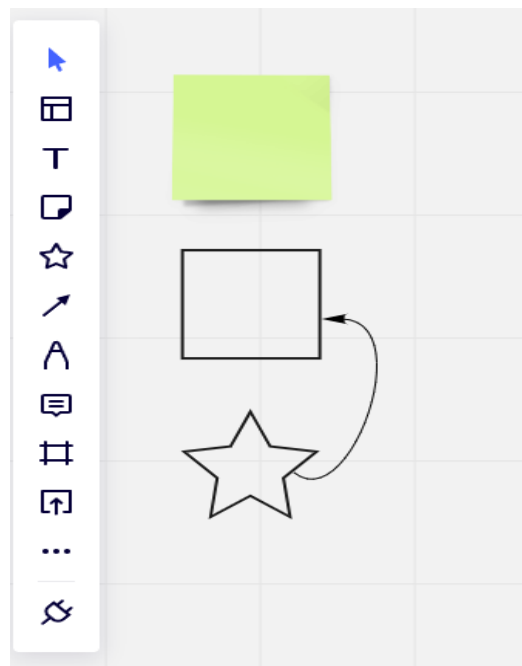


Figure 2. 6: Describe micro.comi's shape

- **Tutorialspoint:** Shapes are designed quite simply, including: circle, square, triangle, rectangle, line, eclipse. When the object is selected, we can use the eraser button to delete the object.

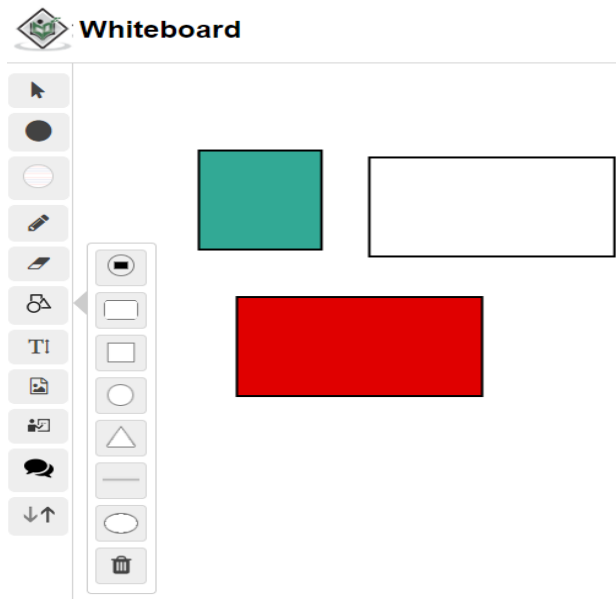


Figure 2. 7: Describe tutorialspoint's shape

#### 2.1.4.3: Text and Image

- **Whiteboard.fi:** A piece of text is written and then we can modify it as we want by selecting the properties that need to be changed.

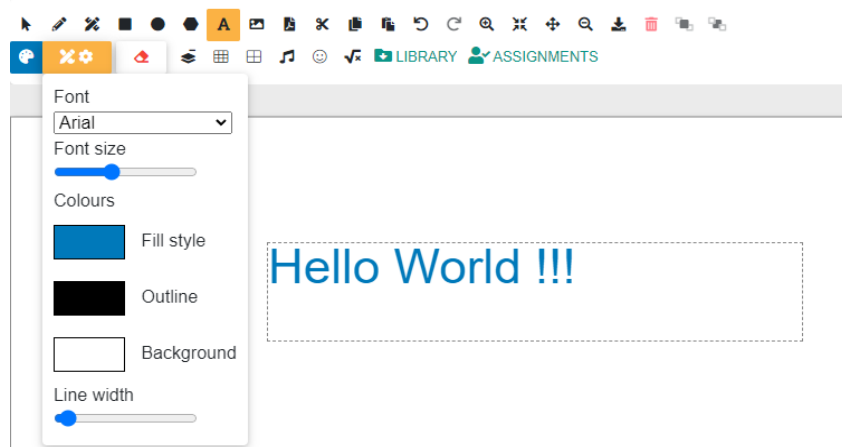


Figure 2. 8: : Describe whiteboard.fi's text and image

- **Miro.com:** In addition to basic text functions like tutorialspoint and whiteboard.fi, they have more special functions like: insert links, highlight, and even more.

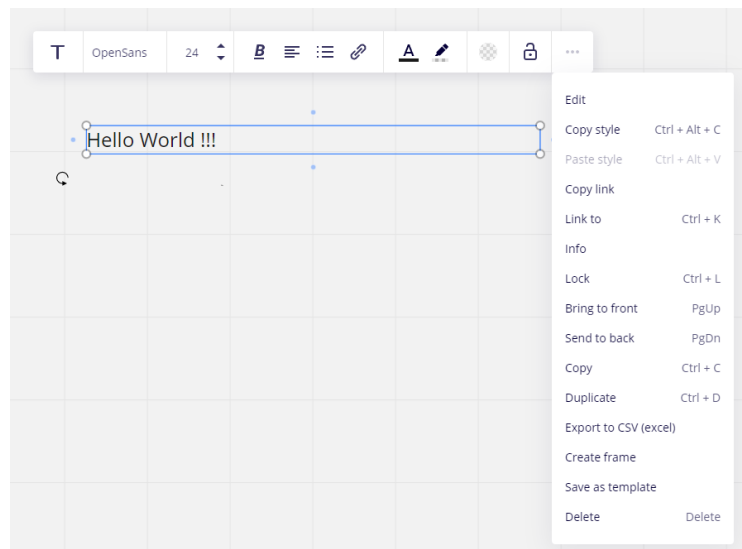


Figure 2. 9: Describe Micro.com's text and image

- **Tutorialspoint:** There are full of basic functions of text such as: font, alignment, size, color, background, border the same with whiteBoard.fi.

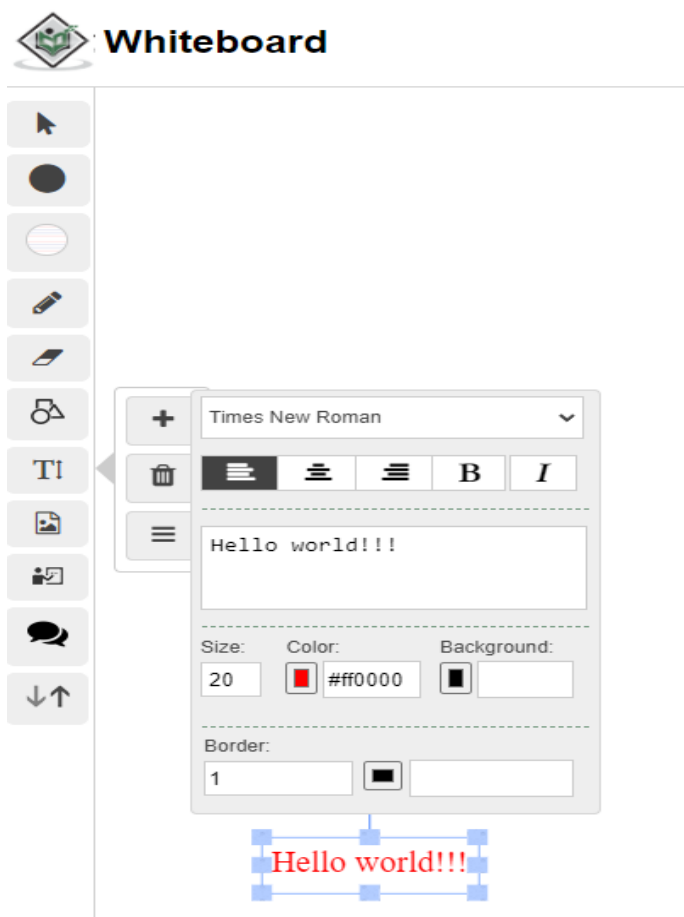


Figure 2. 10: Describe tutorialspoint's text and image

### 2.1.5: Upload/Download

- **Whiteboard.fi:** Using the whiteboard library, it is possible to save, prepare and reuse the whiteboard. We can easily insert our saved whiteboard in any layer at any time. In addition, we can also save the whiteboard in image format: PNG, JPEG, PDF.
- **Miro.com:** Miro has basic functions like uploading documents, video from device, URL. Save files to device and the special thing here is that it can upload / download from third party applications.

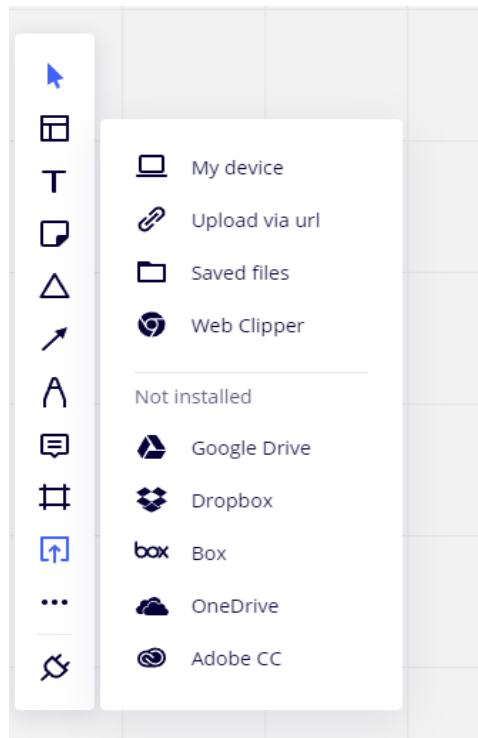


Figure 2. 11: Describe Micro.com's Upload and download

- **Tutorialspoint:** Like whiteboard.fi, tutorialspoint can be downloaded as an image or file for reuse.

### 2.1.6: Communicate

- **Whiteboard.fi:** This app does not support comments or videos.
- **Miro.com:** This app allows to comment and use videos.
- **Tutorialspoint:** The application allows comments, on condition that you are a member of the room.

Features	Miro	Whiteboard.fi	Tutorialspoint
<b>Accessibility</b>			
Registration required	Yes	No	No
Free	No	Yes	Yes
<b>Tools</b>			
Working with objects	Yes	No	Yes
Connecting objects with arrows	Yes	No	No
Writing text	Yes	Yes	Yes
Freehand drawing	Yes	Yes	Yes
Drawing shape	Yes	No	Yes
Insert picture	Yes	No	Yes
Insert table	Yes	No	No
Insert chart	Yes	No	No
<b>Other features</b>			
Templates	Yes	No	Yes
Chat	Yes	No	Yes
Video Chat	Yes	No	Yes
Visible user cursors	Yes	No	No
History of events	Yes	Yes	Yes
Shape recognition	Yes	Yes	No
Import and export as file	Yes	No	No
Export as images	Yes	Yes	Yes

*Table 1: Compare popular whiteboard function*

## 2.2 Expected functionality:

### Instruction:

In this section, I want to describe what functionality I want to build for my application. The following functions will play a very important role in creating a complete whiteboard application, from which we have a more objective view of the application; from a user's perspective we will consider the

functions Is it fit for the intended use? However, for a deeper understanding of the functions, I will implement it in the following sections.

### 2.2.1: Tool

- **Pen:** Of course, on the whiteboard, it is indispensable for the pen; it is the most important tool on the board, to express to the viewer as quickly as possible is the pen. Allows to draw on the PDF (or object whiteboard).
  - How to implement: Click and drag, release when finished
  - Extended Options: Color, Opacity, Stroke width.
- **Text:** Just like the pen, the text is an integral part of the whiteboard, we can add it to any position on the board.
  - How to implement: We can also add text by clicking anywhere with the selection tool.
  - Extended Options: Color, Font Size (S, M, L, XL), Font Family, Text Align, Opacity, Stroke width.
- **Line:** A space made up of an infinite number of pixels, in which a straight line is indispensable.
  - How to implement: Click to start multiple points, drag for single line.
  - Extended Options: Color, Background - when the line is connected to the 2 ends to form a specific shape, there will be background, Fill (solid, cross-hatch, solid), Stroke width/Thickness), Stroke style (Solid, Dashed, Dotted), Edges (Sharp, Round), Opacity.
  - Actions: Delete.
- **Shape:** is a common shape in whiteboards, it can be a main object to work on, it can also be a decorative point for your board.
  - How to implement: Drag and drop or click any point.
  - Extended Options: Color, Background, Fill (solid, cross-hatch, solid), Stroke width/Thickness), Stroke style (Solid, Dashed, Dotted), Opacity.
  - Actions: Duplicate, Delete.
- **Selection:** will have the main function of selecting objects to edit or delete objects.
  - How to implement: Click on object or object zoning
  - Extended Options: None.
  - Actions: extended Options of each object, Duplicate, Delete.
- **Image:** will put specific PNG and jpg images on the whiteboard to make the picture more vivid
  - How to implement: Drag and drop or load from device.

- Extended Options: None.
- Actions: Duplicate, Delete.

### **2.2.2: Working with object**

In the process of using an object, there must be interaction with the user, by the properties after an object is created, the user can customize those properties. For example, change position, change size, color, etc.

### **2.2.3: History of event**

All events after being executed will be saved as history, implemented under undo and redo functions. That makes the operation of the user more convenient and faster.

### **2.2.4: Communicate**

- **Share:** We can invite anyone into the room through a unique link generated.
- **Chat:** Everyone can chat with each other in the chat box

### **2.2.5: Upload and Download file**

Upload and Download files are two necessary functions to save drawings, they will help us to reuse them quickly or even send drawings to anyone.

## **2.3: Summary**

With the goal of designing a full-featured whiteboard for this thesis, I expect my application to satisfy basic user needs and provide a good enough user experience. However, I always keep things simple and always open to expansion and maintenance for the future.



## Chapter 3: Diagram

### 3.1: Use Case Diagram

#### 3.1.1: Introduction

In this diagram, the drawing will show the interaction between the user and the system, the impact of the system on each function that the user uses. When a user performs a function, how will that function affect the system and how will the system handle those cases. Each function will be presented as a specific process, so the user will have a better overview of the system. The whiteboard use case will specify all the features that the user can perform such as: login/logout, use tools, import/export file, give control, add member, undo/redo actions, clear the board, join/leave a room, use chat box. Each feature will have its own implementation conditions. For example, about using tools, the condition is that the user must have permission is true. For a better understanding of each case, see the description below:

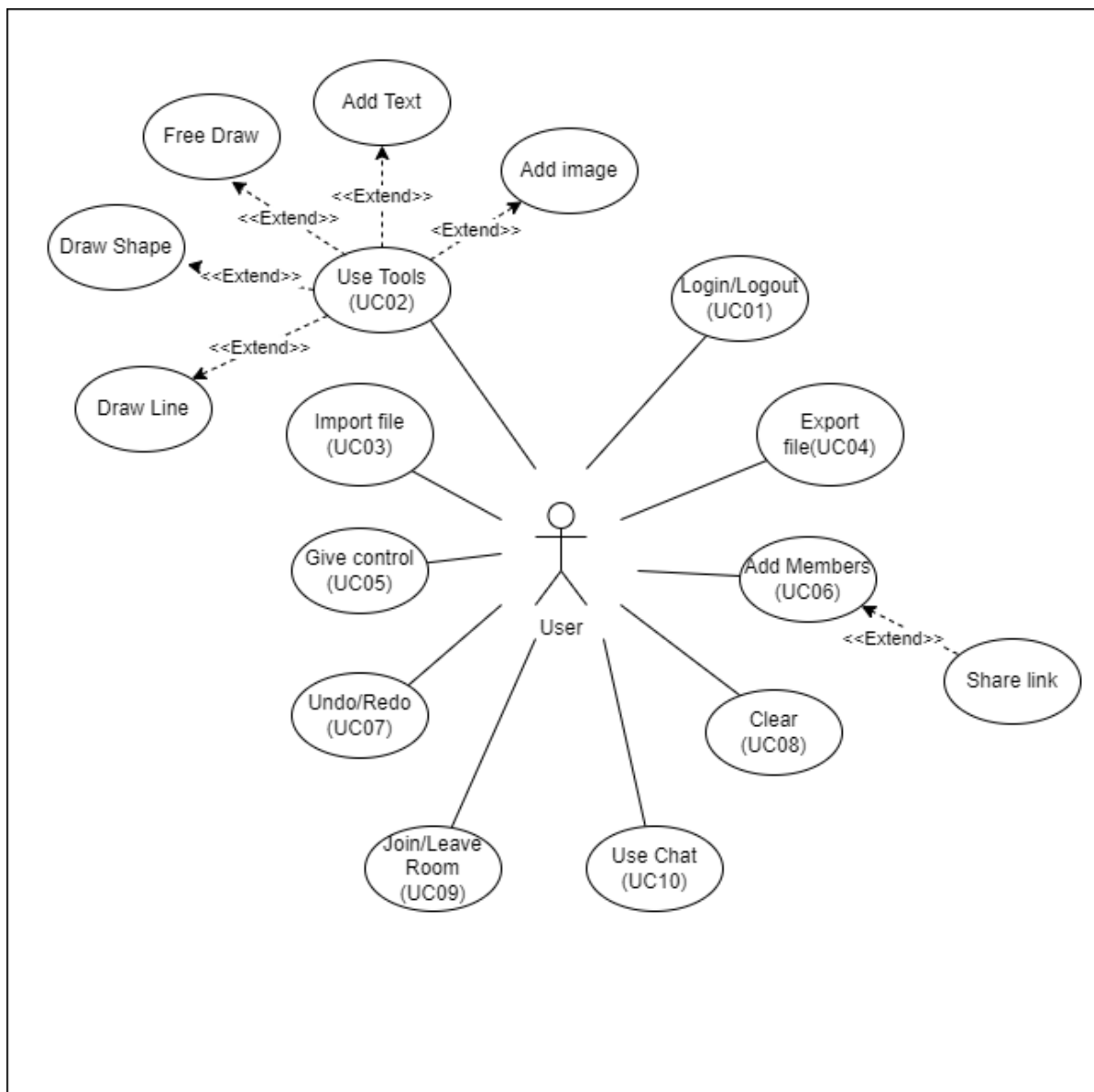


Figure 3. 1: Use Case Diagram

## Use Case description

<b>Use Case ID</b>	01
<b>Use case name</b>	Login/Logout
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	User Login/Logout with Facebook or google account
<b>Preconditions</b>	
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Access the website and click on the button login with Facebook or google</li> <li>2. <b>System</b> – The system opens the login dialog</li> <li>3. <b>User</b> – <b>Enter</b> username and password</li> <li>4. <b>System</b> – Authentication successful.</li> <li>5. <b>System</b> – <b>Go</b> to dashboard</li> <li>6. <b>User</b> – Click on button user profile</li> <li>7. <b>User</b> – Click on the button logout</li> <li>8. <b>System</b> – The system opens the login dialog</li> </ol>
<b>Alternative flow</b>	<ol style="list-style-type: none"> <li>4. <b>System</b> - Authentication failed               <ol style="list-style-type: none"> <li>a. Go back to step 1.</li> </ol> </li> </ol>

<b>Use Case ID</b>	02
<b>Use case name</b>	Use tools
<b>Actors</b>	User
<b>Another Actors</b>	

<b>Description</b>	Use the tools already described
<b>Preconditions</b>	User with permission = true
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the tool you want to use</li> <li>2. <b>User</b> – Click and manipulate on the position you want to use the tool</li> <li>3. <b>System:</b> The system stores the elements in an array, and pushes the data to the firebase database.</li> <li>4. <b>System</b> – The system automatically re-renders every time the elements change.</li> </ol>

<b>Use Case ID</b>	03
<b>Use case name</b>	Import file
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	Import text file
<b>Preconditions</b>	User with permission = true
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the “Import” button</li> <li>2. <b>System</b> – The system opens the window for user select the file</li> <li>3. <b>User</b> – User selects the file and presses open</li> <li>4. <b>System</b> – The system loads the file and checks the file successfully</li> <li>5. <b>System</b> – System render image</li> <li>6. <b>System</b> – Display data that has been saved on the whiteboard</li> </ol>
<b>Alternative flow</b>	<ol style="list-style-type: none"> <li>4. <b>System</b> – The system loads the file and checks the file failed</li> </ol>

	<b>a. System – Alert exception</b>
--	------------------------------------

<b>Use Case ID</b>	04
<b>Use case name</b>	Export file
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	Export json text file
<b>Preconditions</b>	
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the “Export” button</li> <li>2. <b>System</b> – The system opens the window where user can store the file</li> <li>3. <b>User</b> – Choose location, definite the name file and Click on the ‘Save’ button</li> <li>4. <b>System</b> – Alert success</li> </ol>

<b>Use Case ID</b>	05
<b>Use case name</b>	Give Control
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	Give control for members in room
<b>Preconditions</b>	Requires room selection and member must be on the list
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the “Give Control” button</li> <li>2. <b>System</b> – The system requires for a member's name</li> <li>3. <b>User</b> – Find a member’s name on search box</li> </ol>

	<b>4. User</b> – Click on the member’s name to select and Click ‘OK’ button
--	---

<b>Use Case ID</b>	06
<b>Use case name</b>	Add member
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	Add members with search box or share URL
<b>Preconditions</b>	Requires room selection and member must be on the list
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the “Add Members” button</li> <li>2. <b>System</b> – The system open dialog with 2 options to add members <ol style="list-style-type: none"> <li>a. <b>User</b> – choose options share link <ul style="list-style-type: none"> <li>• <b>User</b> – Click on ‘Copy’ button and share link for anyone</li> </ul> </li> <li>b. <b>User</b> – choose options search name <ul style="list-style-type: none"> <li>• <b>User</b> – Find a member’s name on search box</li> <li>• Click on the member’s name to select and Click ‘OK’ button</li> </ul> </li> </ol> </li> </ol>

<b>Use Case ID</b>	07
<b>Use case name</b>	Undo and Redo
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	User action will be saved by the system and can be undo and redo

<b>Preconditions</b>	User must get permission
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – Click on the “Undo” button</li> <li>2. <b>System</b> – The display system reverts to the previously saved operation.</li> <li>3. <b>User</b> - Click on the “Redo” button</li> <li>4. <b>System</b> – The system shows the undo operation again.</li> </ol>

<b>Use Case ID</b>	08
<b>Use case name</b>	Clear
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	Eraser all the content of board
<b>Preconditions</b>	User must get permission
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – the user clicks on “Clear” button</li> <li>2. <b>System</b> – Set elements is empty</li> <li>3. <b>System</b> – The Board automatic re-render image when elements has changed.</li> </ol>

<b>Use Case ID</b>	09
<b>Use case name</b>	Join and leave a room
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	User can join via URL link or be invited by members

<b>Preconditions</b>	
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>Member</b> – Copy and share the URL link to participants</li> <li>2. <b>User</b> – Access the links via browser</li> <li>3. <b>System</b> – The system requires to login</li> <li>4. <b>Participants</b> – Login</li> <li>5. <b>System</b> – Check authorization and add user to room's member on database</li> <li>6. <b>User</b> – Click on the "Leave room" button</li> <li>7. <b>System</b> - Current room is empty</li> </ol>

<b>Use Case ID</b>	10
<b>Use case name</b>	Use chat
<b>Actors</b>	User
<b>Another Actors</b>	
<b>Description</b>	People can chat with each other through the chat frame.
<b>Preconditions</b>	Members of room
<b>Postconditions</b>	
<b>Normal flow</b>	<ol style="list-style-type: none"> <li>1. <b>User</b> – the user clicks on "Chat" button</li> <li>2. <b>System</b> - The system displays the chat frame</li> <li>3. <b>User</b> – The user enters the content in the chat box and the content will be pushed when the enter button is pressed</li> <li>4. <b>System</b> – The system will store content message on database and display the content.</li> </ol>

### 3.1.2: Summary

In summary, we have described specific cases for users to understand what features this product offers and what users can do with it. It will create a complete picture of how users will use specific functions in a system.

## 3.2: Class Diagram

### 3.2.1: Introduction

When a user accesses the whiteboard, the data will be processed at each layer, for example, when a user logs in, the user's data will be processed at the firebase Server layer, then the data continues to be processed and move to the next layers according to the programmer's purpose. The diagram below, gives the programmer a static view of the model, properties, and behavior. So, the Programmer can conceptualize scenarios to implement into a programming language, maintain and upgrade the program systematically.

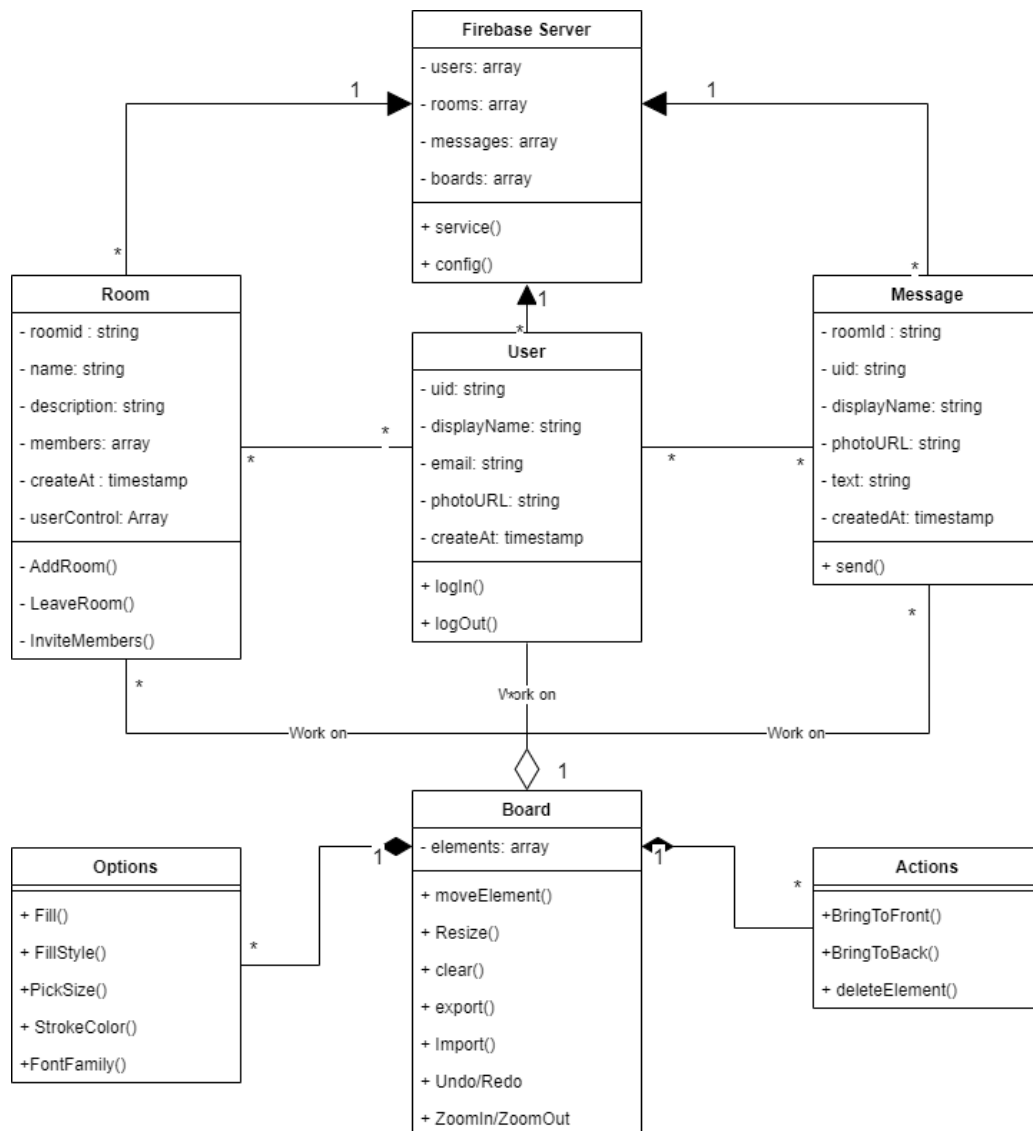




Figure 3. 2: Class diagram

### 3.2.2: Description Class Diagram

We can see the entity class objects as follows:

- **Firestore Server:** Firestore is a platform developed by Google for creating web and mobile applications, this is the place to receive and process requests from users so that the content is consistent. We save a scene by sending it to the server, which gives a shareable URL. The recipient then downloads the data from the server.

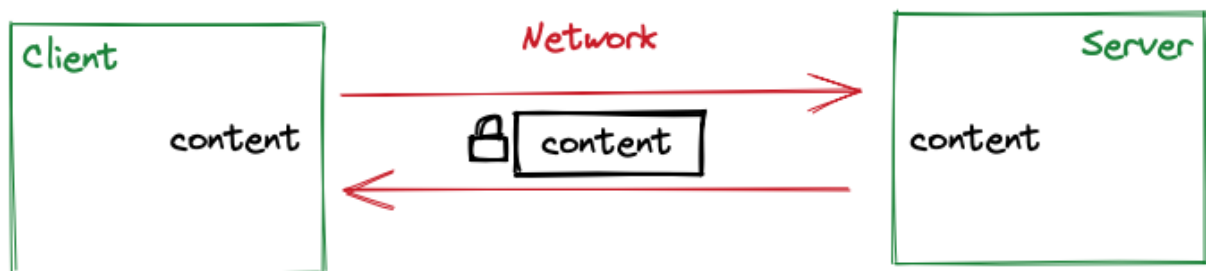


Figure 3. 3: Describe data synchronization between client and firebase

- **Users:** Firestore provides a user authentication mechanism, from here we can use this server as an authentication engine through Facebook or google. After the information is authenticated, the user's information including: uid, displayName, photoURL, email will be saved as an array to firestore's database.
- **Rooms:** After logging in, the user has permission to create rooms, and it will be saved here as:
  - **Name** - this is name of room
  - **Description** - Description of this room
  - **Members** - uid of user will be saved as array.
  - **Controls** - uid of user who has permission to control this room
- **Messages:** This is where the entire message is stored when a message is sent, it will be saved here, when other users connect, the content will be retrieved through the roomid that person joined.
- **Boards:** This is the most important part, when a whiteboard is used, they generate images which I call elements. Elements contain information such as
  - **Id** - is used to distinguish it from other elements.
  - **Position** - is the position of the element, depending on whether the object is a line, shape, free draw or text, there will be different position styles.
  - **Type** - describes the type of an element, for example: line, shape, rectangle, text, draw, and so on.
  - **Options** - are the properties of an element, for example: size, stroke, color, background, and so on.
- **Function:**
  - **Config()** - is the place to configure the connection between Firestore and the application.

- **Service()** - is a place to store pre-initialized functions for easy reuse, such as: add data, load data, update data, delete data, and so on.
- **Room:** This class acts as a receiver and transmits data to the server.
  - **Transmits data** - When a user creates a room, it will post it to the server with the attributes described above.
  - **Receive data** - they will receive a list of rooms, so how do we know the room we are looking for? In each room, we have a list of members' uid, so when a user sends a request to get the data of the rooms, the list will return the rooms that contain the member's uid.
  - **Functions:**
    - **AddRoom()** – After the user enters the information he wants to add a room; this function will save this information on the server.
    - **LeaveRoom()** – This function will update the list of members of the room.
    - **InviteMembers()** – This function will update the list of members of the room.
- **User:** This is a place to store a user's information, each user is distinguished with a userid (uid), from which other components can refer to this uid to request data access.
- **Message:** The message layer will take care of pushing and retrieving data from the server. Based on a firebase-specific support method, when a message data changes, it will automatically return, and our task is to make it re-render again to appear on the screen.
- **Board:** This class will take care of initializing elements and handling them, when an element is processed they will synchronize with the database, and when the element in the database changes, other users will receive the change signal and automatically dynamically updated according to versions of elements. From there, our data will be synchronized with each other. I will define some common functions here:
  - **DrawImageToCanvas()**: function that will draw all the elements that are saved to the canvas on each render
  - **movingElement()**: This function supports updating the position of an element.
  - **Resize()**: This function helps to change the size of an element.
  - **Clear()**: Delete all the board.
  - **Export()**: export the board as an JSON file.
  - **Import()**: Add a JSON saved board.
  - **Undo/Redo:** Undo and redo steps worked on the board.
  - **ZoomIn/ZoomOut:** Support observation.
- **Options:** Since each object will have a different shape, so this is where the properties of each element are handled. Before each object is rendered, the properties are processed and passed to elements. And this class will do it.
- **Actions:** This class collects the actions of an element; the actions will not matter what object it is? What is the attribute? It means these actions will work for all elements.

### 3.2.3: Summary

In this class diagram, we can see the structure of the system, its properties, and how it works. From there, a programmer can observe and develop in a positive direction; it is like a map in the hands of a guide; it tells us where to go and what to do.

### 3.3: Component Diagram

### 3.3.1: Introduction

In this diagram, the figure helps us to show how to divide the system into many components. Each subsystem after construction can be packaged into an independent implementation component; in this case the system is designed to split into three subsystems: Web Board, Client, and Firebase Server.

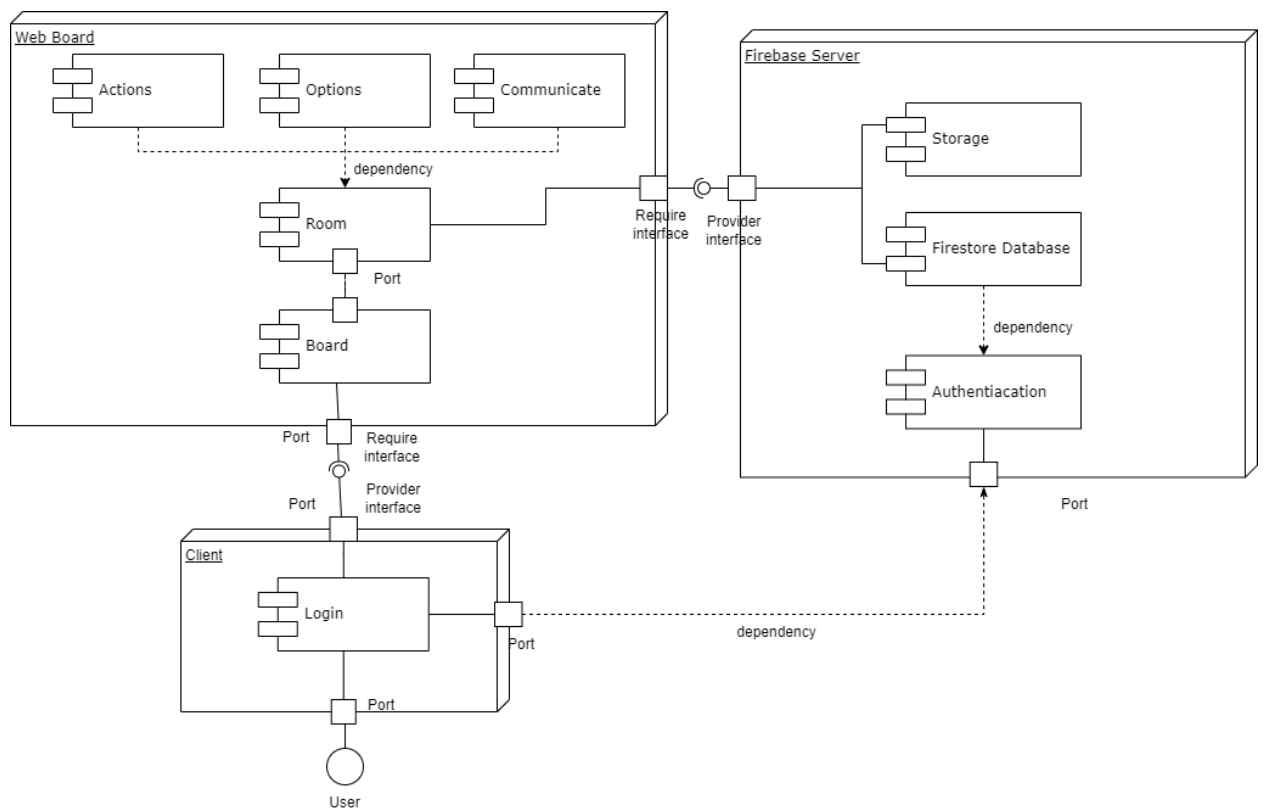


Figure 3. 4: Component Diagram

### 3.3.2: Description Component diagram

**Client:**

This subsystem is where user information is received, which contains the authentication support methods provided by firebase. User information will be authenticated by Firebase Server subsystems, then user information will be saved on Firestore as user: displayName, email, photoURL, uid, providerId, keywords.

## Firestore Server:

This is a platform that supports the development of mobile and web applications. In my project it is considered as a database storage, and it is also an API (Application Programming Interface) that supports authentication users from google and Facebook.

- Authentication: is a simple and secure user management service. Authentication provides many methods of authenticating Google and Facebook email and passwords.
- Firestore: is a global data storage and synchronization service between users and devices. Services using NoSQL are hosted on cloud infrastructure. In my project it will store collections like: users, rooms, messages, boards. I will give some examples of data stored on Firestore cloud and the detailed description of the data I will present in the Implementation (chapter 4):
  - User:

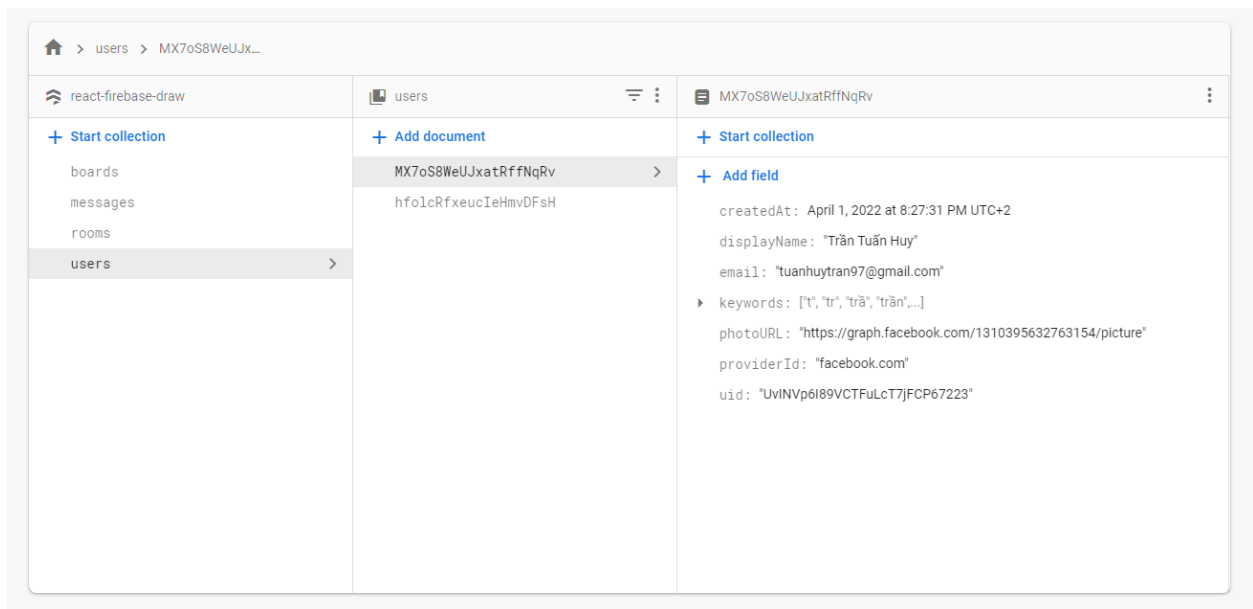


Figure 3. 5: User firebase data description

- Rooms:

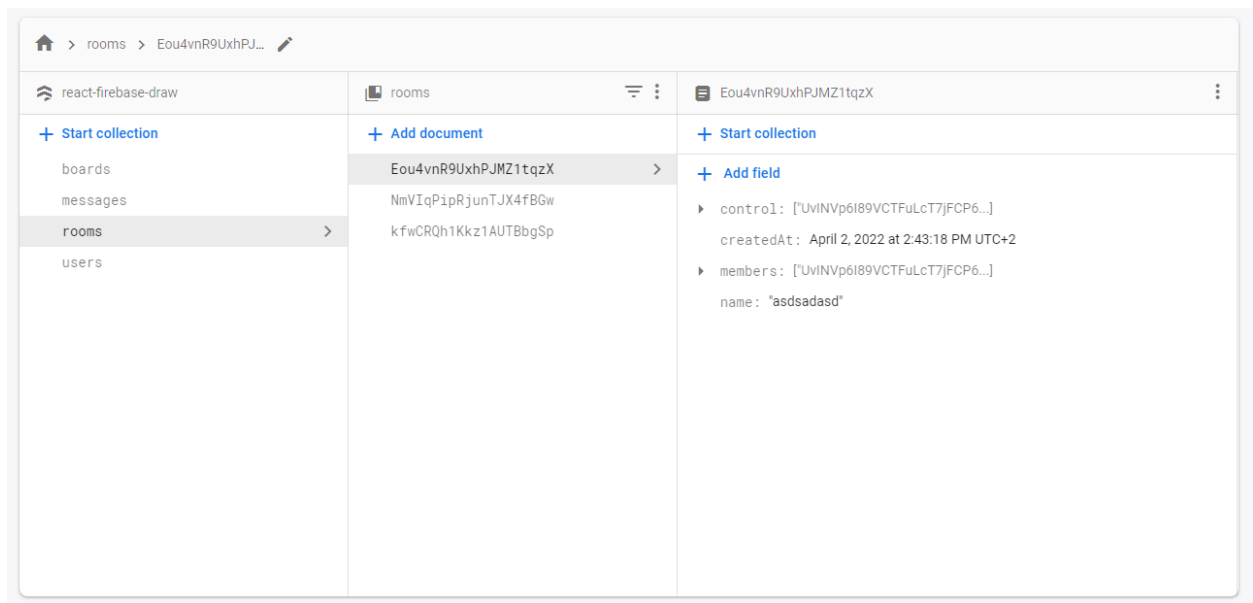


Figure 3. 6: Rooms firebase data description

- Messages:

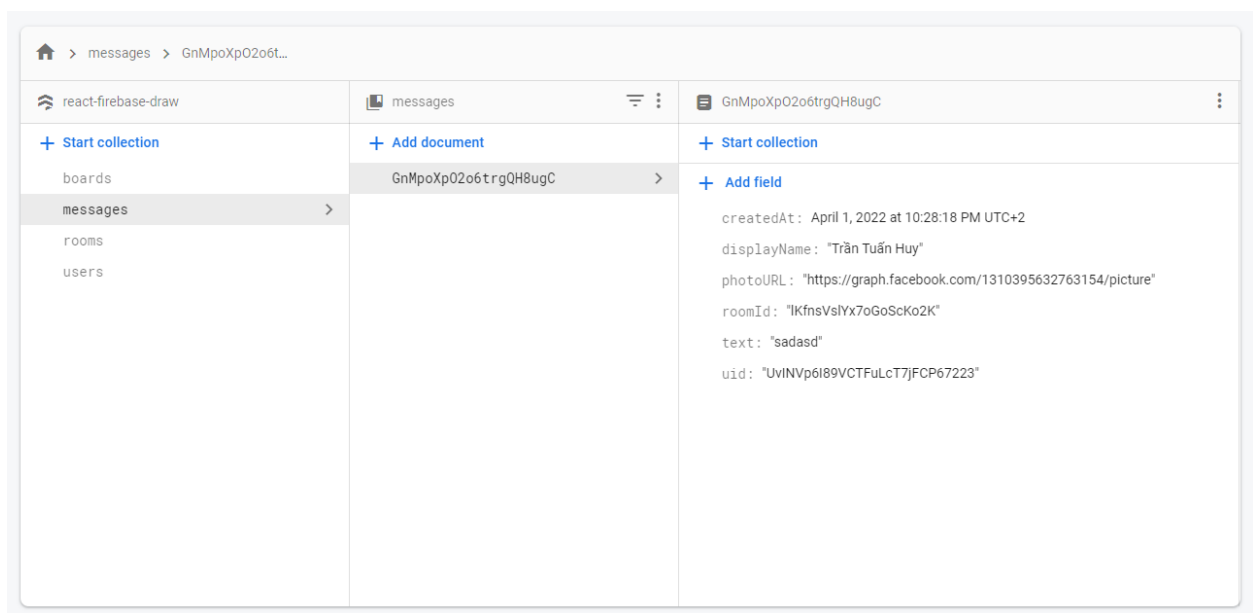


Figure 3. 7: Messages firebase data description

- Boards:

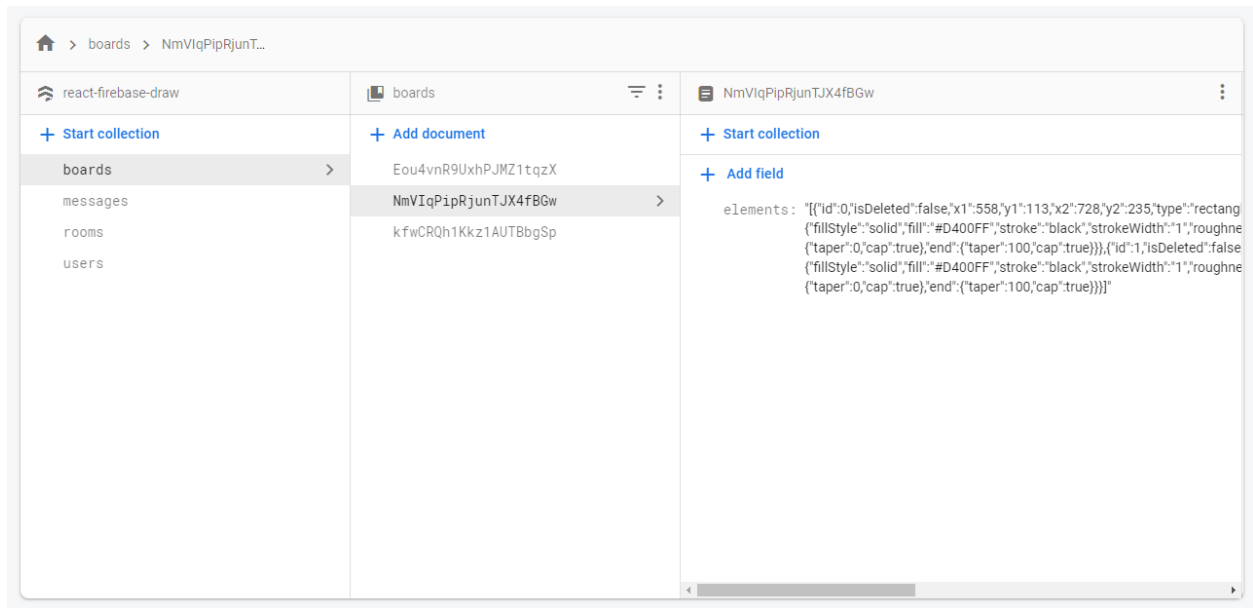


Figure 3. 8: Boards firebase data description

- Storage: is a service that stores and shares user-generated content such as images, audio, and video with powerful, simple, and cost-effective storage built for Google's scale. In my project, it will be used to save images to sync with other users.

### Web Board:

This is a system that is implemented after a user is authenticated; it provides interfaces that role as a front-end, this system ensures users can interact with each other and data is consistent. To better understand how the components in these subsystems work, I will describe them in **Chapter 4: Implementation.**

### 3.3.3: Summary

The above component diagram describes the components and how they work, so programmers can understand the structure of the system and make documentation to implement the system.

## 3.4: Sequence Diagram

### 3.4.1: Introduction

In this section, I will present sequence diagrams of the selected functions to better understand the interaction of objects in the system. From the drawing, it will be easier to see more clearly how the functions work.

#### A. Login

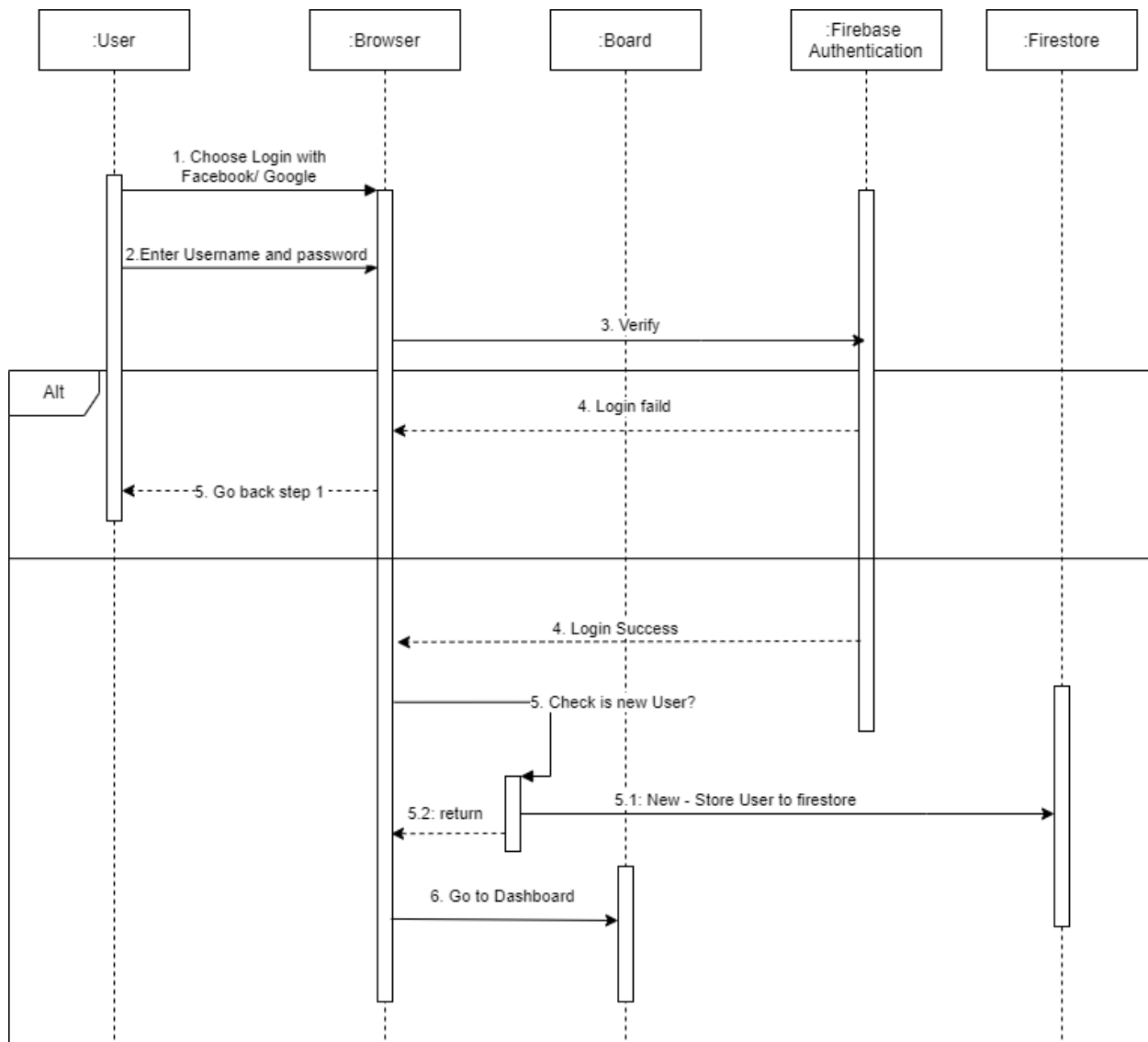


Figure 3. 9: Login sequence diagram

**Description:** When the user accesses the website, the system will ask the user to authenticate through google or Facebook provider, the data will be authenticated on firebase authentication supported by firebase. In case of authentication failure, the system will ask the user to enter the information again, when the login is successful, the system will check the user data on Firestore; if the user is new, the data will be uploaded to Firestore. After successful login, the system will redirect the user to the dashboard.

## B. Use Tools

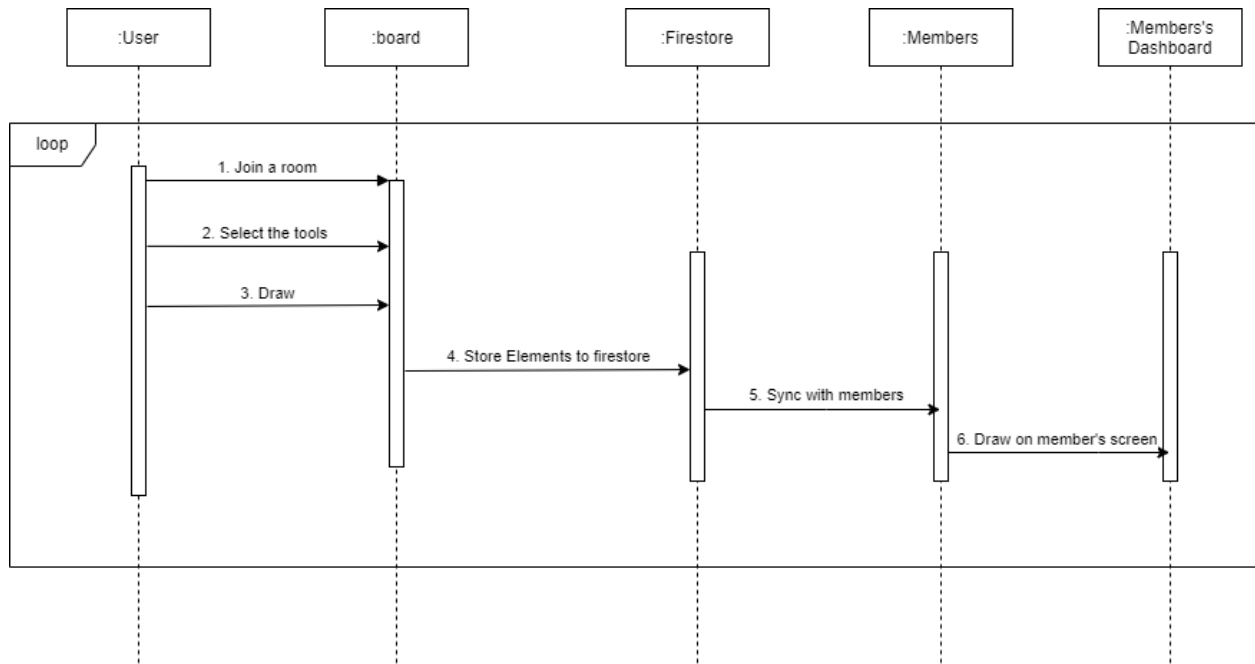


Figure 3. 10: Use tools sequence diagram

**Description:** The function to use the tool will require the user to join any room and must have control permission. In the case of satisfying the above two conditions, the user will choose any tool in the set of tools designed by the programmer and execute them. When a command is executed successfully, they are saved as a JSON array of elements (see **Figure 4. 2: Elements example**), and they are saved in the Firestore for synchronization with other users. When the data is synchronized, each user's screen will be automatically rendered.

### C. Import/Export file



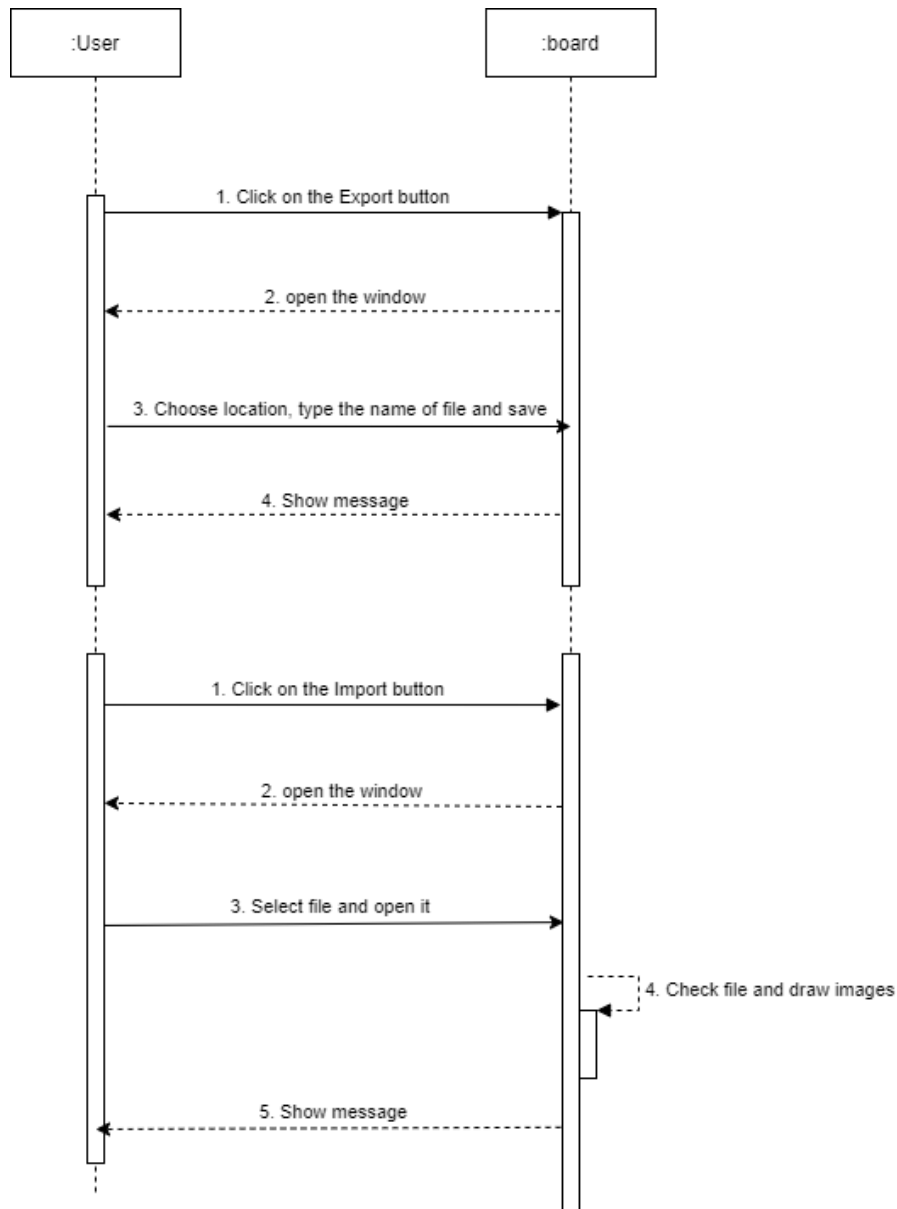


Figure 3. 11: Import and export sequence diagram

**Description:** In order to use the import and export function, the user must also satisfy two conditions: be in a room and have control. The export and import functions will be supported for JSON files, so when a file is executed it will be saved as \*. json, this JSON file will contain all the drawn elements. When a json file is imported from a local computer, the system will check the validity of the elements in the file; if valid they will update with the current elements.

#### D. Give Control

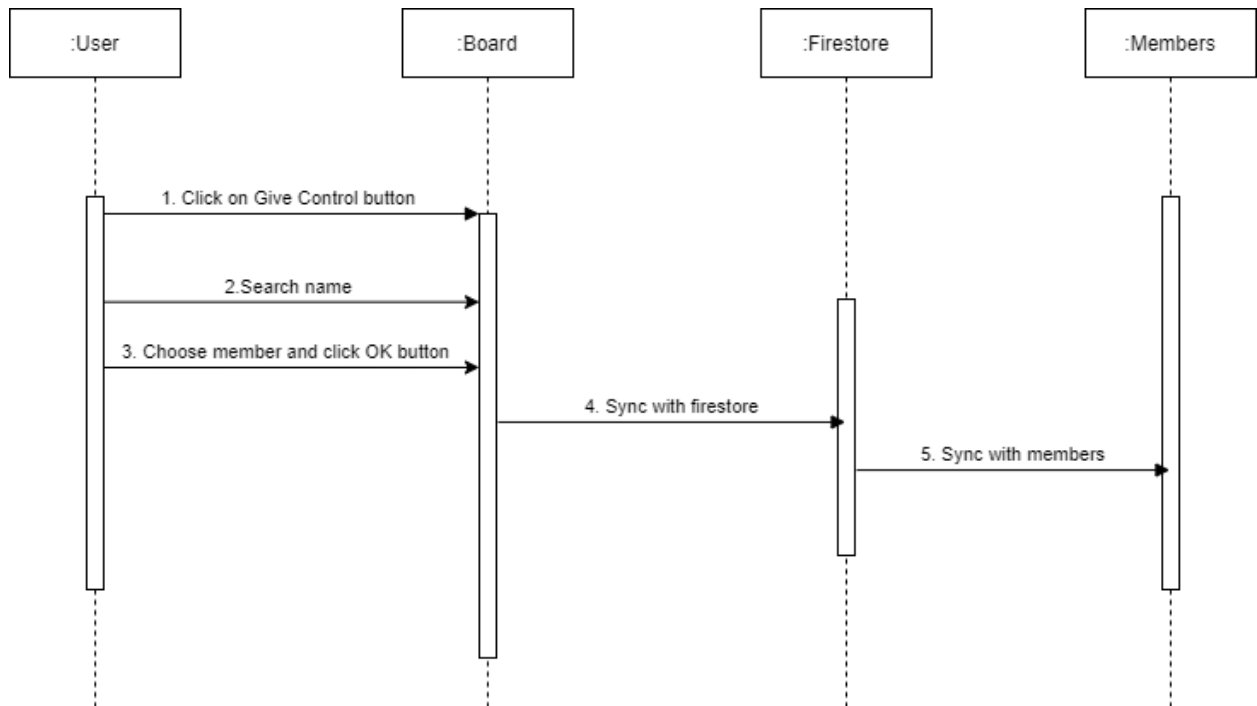


Figure 3. 12: Give control sequence diagram

**Description:** In order to use the give control function, the system will be set up with a function to filter the members' names; when the user starts to enter the name, the function will filter the answers with the closest results and after the user is selected. The system will update the controller result on Firestore and sync with other members. (Look at: **4.3.8: Giving control**)

### E. Add members

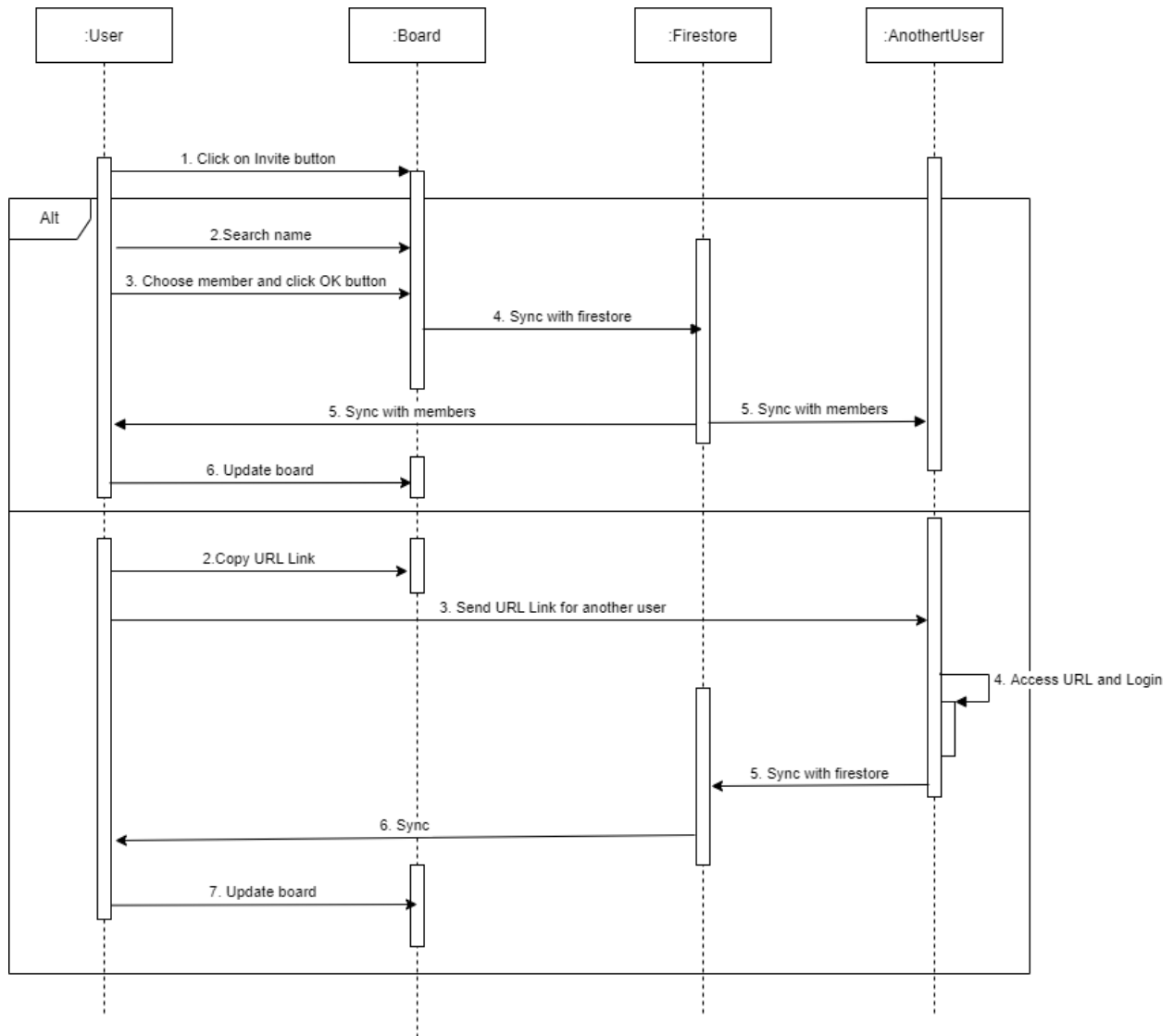


Figure 3. 13: Adding member sequence diagram

**Description:** The function to add members will have 2 options: Add via Link URL and add via search name user.

- Add via username search: this function is similar to **Give Control** function. There will be a filter function username and selected username will update on Firestore and sync with members.
- Add via Link URL: Each room will have a separate encrypted keyword; the user will copy the whole URL with the encrypted keyword, when the recipient gets the URL, they need to encode it to find the room's keyword. When the room keyword is correct, the new user's uid will be updated in the room list. (Look at: **4.3.7.1: Share Link**).

## G. Undo/Redo

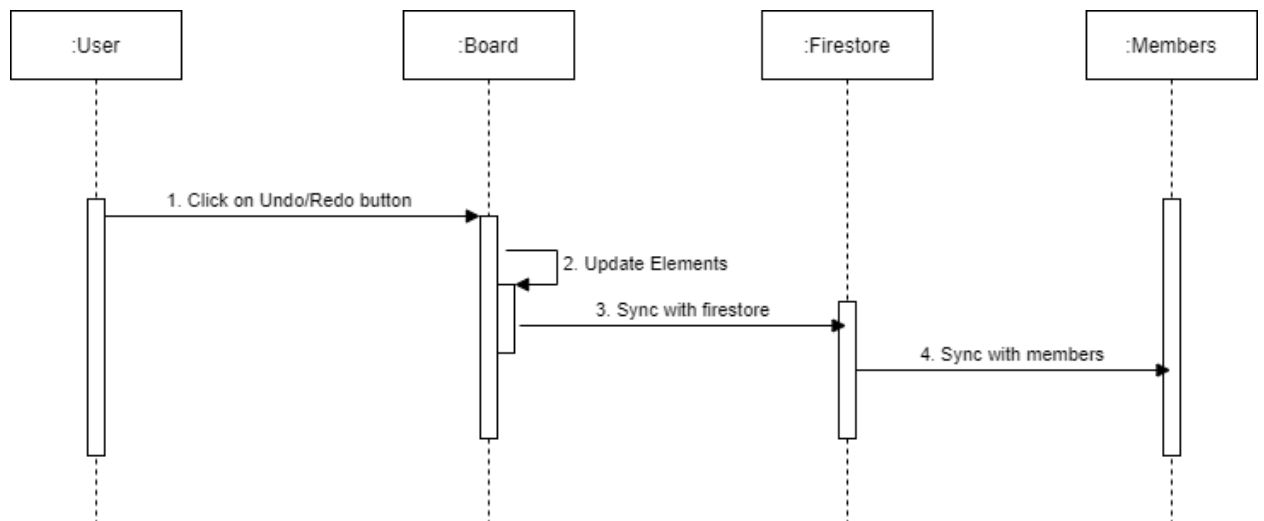


Figure 3. 15: Undo and Redo sequence diagram

**Description:** The programmer will build a function that stores all the changed states of the elements into an array; when an undo or redo command is called, the function will get the element at the position it moved. (Look at: **4.3.4: Undo/Redo**).

#### H. Use Chat

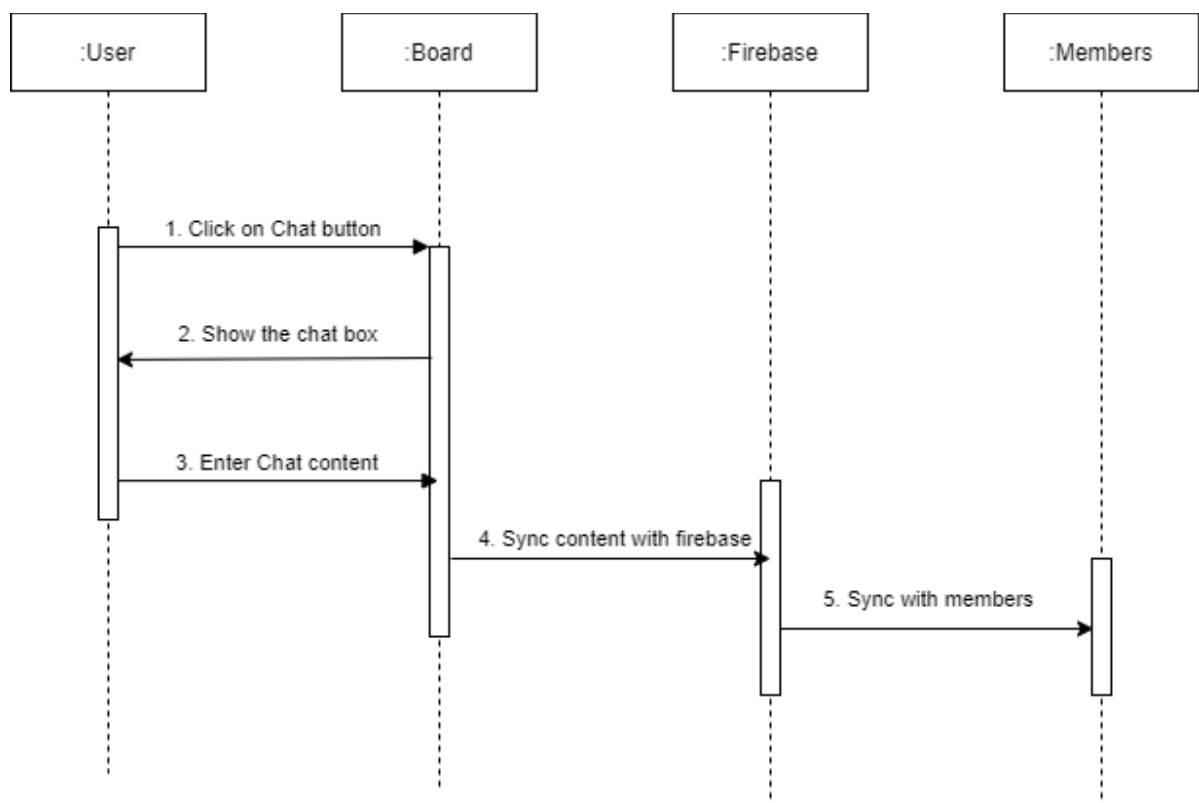


Figure 3. 14: Chat sequence diagram

**Description:** Chat function is used to communicate with other users through synchronizing data with Firestore, by setting up functions to send data and listen to data changes from Firestore, room members can communicate next to each other. (Look at: **4.3.11: Use chat**).

### **3.4.2: Summary**

This section describes the basic sequence of a function; this allows programmers or users to review specific functions quickly. Programmers have a specific view to capture ideas and deploy the system according to a designed route.

## **3.5: The State Diagram**

### **3.5.1: Introduction**

The whiteboard state diagram depicts a process that is started and the events that occur making up the user's lifecycle. It contains most of the ground state to make the elements synchronized with each other. When the elements are synchronized the state is terminated.

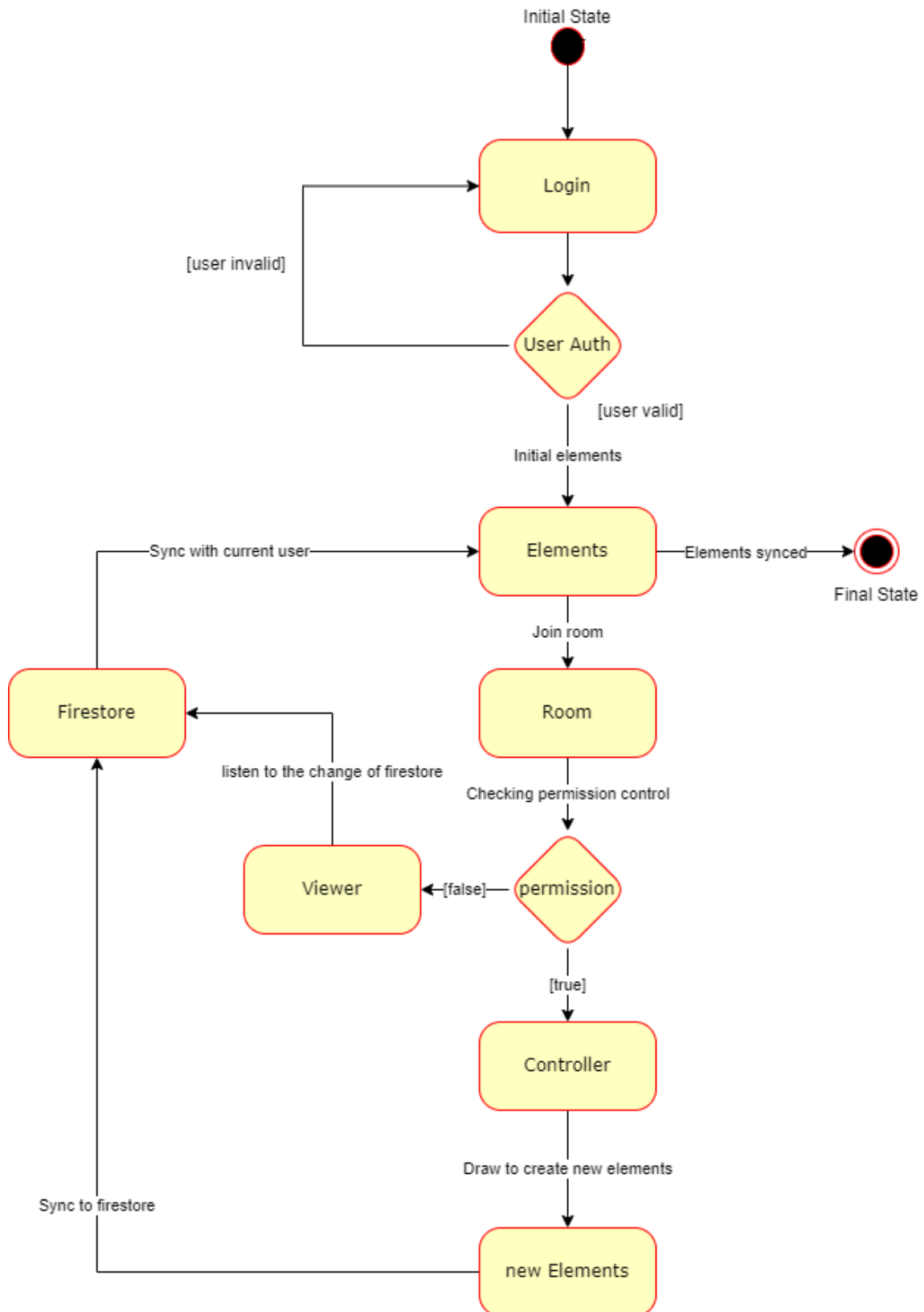


Figure 3. 15: State diagram

### 3.5.2: Description State diagram

- **Login:** When the state is set, the system will ask the user to login and the user will be authenticated by firebase authentication (see Figure 3. 9: Login sequence diagram) if the user is not verified the system will revert back to the logged in state; if allowed verify the system will move to the next state.
- **Elements:** initial elements will be initialized; these elements will always be synced with Firestore if room state is enabled.
- **Room:** When a user enters his room, the permission control room check condition will be activated. This allows us to know if this user is a controller or a viewer.
  - **Controller:** The controller is the person who can use the functions of a whiteboard; when the user draws it, it will create a new element and synchronize it with the Firestore.
  - **Viewer:** The viewer is the only person who has the right to see and listen to changes from the Firestore, when the data from the Firestore changes, the viewer will automatically update their data to synchronize with each other.

When the Controller, viewer, and Firestore Elements are synchronized, they move to the Final State and end a lifecycle.

### 3.5.3: Summary

This section shows how a whiteboard lifecycle will begin and end. When all the data between the server-side and the clients-side is synchronized, the state is terminated. The diagram also provides a summary view from which the programmer can quickly grasp ideas for system implementation and maintenance.

## Chapter 4: Implementation

In this section, some APIs, algorithms and coding ideas will be described and the functions of the tools will be described.

### 4.1: Required programming language and Library:

- JavaScript and CSS (Cascading Style Sheets).
- JSON.
- HTML (Hypertext Markup Language).
- Reacts.

### 4.2: Application Programming Interface (API) and technology

#### 4.2.1: Canvas

Canvas is a powerful whiteboard support tool that will solve most of the basic functional problems of a common whiteboard. In my thesis I will use canvas as the main background along with which I will use other supporting APIs to implement the tools. [4]

#### 4.2.2: Roughjs

Roughjs is a small graphics library that lets you draw in a sketchy, hand-drawn-like, style. The library defines primitives to draw lines, curves, arcs, polygons, circles, and ellipses. In my project, I use roughjs to support drawing lines and shapes. below, is sample code implementing a roughjs. [5]

First, I will connect to the canvas via rough. For example:

```
let roughCanvas = rough.canvas(document.getElementById('myCanvas'), config);
let generator = roughCanvas.generator;
```

Second, I will use the newly instantiated object to call the desired draw method.

```
let rect1 = generator.rectangle(10, 10, 100, 100);
let rect2 = generator.rectangle(10, 120, 100, 100, {fill: 'red'});
roughCanvas.draw(rect1);
roughCanvas.draw(rect2);
```

*Figure 4. 1: Example using Roughjs + canvas to draw a object*

#### 4.2.3: Perfect-freehand

Perfect-freehand is an API I choose to support my drawing tool usage. The way it works is save the pixels from the user's mouse movement, then this API will assist in creating the pixels on my canvas. [6]



#### 4.2.4: Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. [7]

#### 4.2.5: Firebase

Firebase is a platform developed by Google for creating web and mobile applications. My app is using firebase authentication technology and google firebase cloud. [8]

### 4.3: Implementation application functionality:

#### 4.3.1: Introduction:

In this section I will focus on ideas and implementations of the important functions of a whiteboard. All implementations will be based on ideas, and implementations will be somewhat irrational and suboptimal in some cases.

#### 4.3.2: The ideas to draw an object into canvas

The idea is to do draw an object on canvas, I will try to create objects (line, rectangle, text, image, pencil) and save them as an array called Elements. For example:

```
▼ {elements: Array(2)} ⓘ  
  ▼ elements: Array(2)  
    ▼ 0:  
      id: 0  
      isDeleted: false  
      ▶ options: {fillStyle: 'solid', fill: '#ca09f1', stroke: 'black', strokeWidth: '3', roughness: 0, ...}  
        type: "line"  
        x1: 657  
        x2: 849  
        y1: 273  
        y2: 124  
      ▶ [[Prototype]]: Object  
    ▼ 1:  
      id: 1  
      isDeleted: false  
      ▶ options: {fillStyle: 'solid', fill: '#ca09f1', stroke: 'black', strokeWidth: '3', roughness: 0, ...}  
        type: "rectangle"  
        x1: 937  
        x2: 1158  
        y1: 157  
        y2: 304  
      ▶ [[Prototype]]: Object  
      length: 2  
    ▶ [[Prototype]]: Array(0)  
    ▶ [[Prototype]]: Object
```

Figure 4. 2: Elements example

As in the above example, we can see that I have created 2 objects with 2 types, line and rectangle, which are stored in the elements array. Each object will have different properties based on the parameters recorded during the user's use.

**So how do they draw on the canvas?**

Every time an element is created they are rendered on the canvas. Which means they are drawn based on elements. Follow useEffect [9] supported by Reactjs, elements will be listened, when there is a change, it will automatically execute the below code.

```
const context = canvasRef.current.getContext('2d');
context.clearRect(0, 0, canvasRef.current.width, canvasRef.current.height);
const roughCanvas = rough.canvas(canvasRef.current);

elements.forEach(element => {
  if(element.isDeleted === false){
    drawElement(roughCanvas, context, element);
  }
});
```

Clear canvas

Draw element functions

*Code 4. 1: Draw on the canvas example*

```

const drawElement = async(roughCanvas, context, element) => {
  let {x1,y1,x2,y2,options,src} = element;
  switch (element.type) {
    case "line":
      roughCanvas.line(x1,y1,x2,y2,options);
      break;
    case "rectangle":
      roughCanvas.rectangle(x1,y1,x2-x1,y2-y1,options);
      break;
    case "circle":
      const diameter = Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2,2));
      roughCanvas.circle(x1,y1,diameter,options);
      break;
    case "pencil":
      context.beginPath();
      context.fillStyle =options.penFillStyle;
      const stroke = getSvgPathFromStroke(getStroke(element.points,options));
      context.fill(new Path2D(stroke));
      break;
    case "text":
      context.beginPath();
      context.textBaseline = "top";
      context.fillStyle = options.penFillStyle;
      context.font = options.fontSize + ' ' + options.fontFamily ;
      context.fillText(options.text, element.x1, element.y1);
      break;
    case "picture":
      if(src){
        const image = new Image();
        if(localStorage.getItem(`${src.name}`) === null){
          toDataURL(src.address, function(dataUrl) {
            localStorage.setItem(`${src.name}`,dataUrl);
          })
        }
        image.src = localStorage.getItem(`${src.name}`);
        context.drawImage(image, x1,y1,x2-x1,y2-y1);
      }
      default:
  }
};

```

Draw element support by roughjs

Draw element support by canvas

Code 4. 2: Draw element function description

To better understand the created objects, we will look about the tools used bellow.

### 4.3.3: Use Tools

This section will mainly describe the process of creating objects (elements), to see the process of objects (elements) being drawn on the canvas see: **4.3.2: The ideas to draw an object into canvas.**

The tool will include 5 main functions: Draw Line, draw shape, Free draw, add text, add image (See at **Figure 3. 1: Use Case Diagram**). In order for the drawing functions to work, I will create a createElement() function that initializes the object and an updateElement() function that updates the object when a change occurs. These two functions will be described below:

```

const createElement = (id, x1, y1, x2, y2, type, options, src) => {
  switch (type) {
    case "line":
      return { id, isDeleted : false, x1, y1, x2, y2, type, options };
    case "rectangle":
      return { id, isDeleted : false, x1, y1, x2, y2, type, options };
    case "circle":
      return { id, isDeleted : false, x1, y1, x2, y2, type, options };
    case "pencil":
      return { id, isDeleted: false, type, points: [{ x: x1, y: y1 }], options };
    case "text":
      return { id, isDeleted: false, x1, y1, x2, y2, type, options };
    case "picture":
      return { id, isDeleted: false, x1, y1, x2, y2, type, options, src };
    default:
      throw new Error(`Type not recognised: ${type}`);
  }
};

```

*Code 4. 3: Create element function*

```

const updateElement = (id, x1, y1, x2, y2, type, options, src) => {
  const elementsCopy = [...elements];
  switch (type) {
    case "line":
    case "rectangle":
      elementsCopy[id] = createElement(id, x1, y1, x2, y2, type, options);
      break;
    case "circle":
      elementsCopy[id] = createElement(id, x1, y1, x2, y2, type, options);
      break;
    case "pencil":
      elementsCopy[id].points = [...elementsCopy[id].points, { x: x2, y: y2 }];
      break;
    case "text":
      const textWidth = document
        .getElementById("canvas")
        .getContext("2d")
        .measureText(options.text).width;
      const textHeight = 24;
      elementsCopy[id] = {
        ...createElement(id, x1, y1, x1 + textWidth, y1 + textHeight, type, options)
      };
      break;
    case "picture":
      elementsCopy[id] = createElement(id, x1, y1, x2, y2, type, options, src);
      break;
    default:
      throw new Error(`Type not recognised: ${type}`);
  }
  setElements(elementsCopy, true);
};

```

*Code 4. 4: Update Element function*

## How do these functions work?

The ideas: I will use 3 events supported by canvas API which are: **OnMouseDown**, **OnMouseMove**, **OnMouseUp**. See the documentation here: [10]

- **OnMouseDown**: when we first click on canvas background, an object will be created at x, y position with pre-set properties (see at the function: **Code 4. 3: Create element** )
- **OnMouseMove**: when we keep holding the click and start moving, the x, y coordinates will change continuously so i want to use updateElement function (**Code 4. 4: Update Element function**) to update the properties for each case.
- **OnMouseUp**: when we release the click, this event will make sure the action is completed to prepare for the next actions.

We can see an example of an object drawn on canvas after doing the above 3 events here: **4.3.2:** The ideas to draw an object into canvas.

### 4.3.4: Undo/Redo

The idea: objects are stored as elements array, but we want to store all the history that happened to elements array, so I initialize a function useHistory to save all changes of elements, the code that I use below:

```
const useHistory = initialState => {
  const [index, setIndex] = useState(0);
  const [history, setHistory] = useState([initialState]);
  const setState = (action, overwrite = false) => {

    const newState = typeof action === "function" ? action(history[index]) : action;
    if (overwrite) {
      const historyCopy = [...history];
      historyCopy[index] = newState;
      setHistory(historyCopy);
    } else {
      const updatedState = [...history].slice(0, index + 1);
      setHistory([...updatedState, newState]);
      setIndex(prevState => prevState + 1);
    }
  };
  const undo = () => index > 0 && setIndex(prevState => prevState - 1);
  const redo = () => index < history.length - 1 && setIndex(prevState => prevState + 1);
  return [history[index], setState, undo, redo];
};
```

*Code 4. 5: useHistory function description*

and then I just call it again, the elements are already saved as history and contain 2 functions undo and redo.

```
const [elements, setElements, undo, redo] = useHistory(initData);
```

Finally, the undo and redo methods will be fired via the **onClick** event.

```

<div className="Action-Bottom" style={{ position: "absolute", bottom: "15px", left: "10px", padding: 10 }}>

  <div className="undo-redo-clear" >
    <button style={{margin: "5px"}} onClick={undo}>Undo</button>
    <button onClick={redo}>Redo</button>
    <Clear/>
  </div>
</div>

```

Code 4. 6: Undo/Redo description

### 4.3.5: Clear

In order to delete all elements in the whiteboard, I will delete all the content on the Firestore where the room's data is stored. Then I'll set the current elements to an empty array. See the code below:

```

function Clear(props) {

  const {setElements, setSelected} = useContext(BoardContext);
  const {selectedRoomId} = useContext(AppContext);
  const handleReset = () => {
    db.collection('boards').doc(String(selectedRoomId)).delete();
    setElements([]);
    setSelected(null);
  }

  return (

    <button style={{margin:"5px"}} onClick={handleReset}>Clear</button>

  );
}

```

Set empty array

Delete elements on Firestore

Code 4. 7: Clear function description

### 4.3.6: Import/Export file

**Export:** The idea for exporting the file is that I would convert the entire array of elements into a JSON string. Then save everything to local disk.

```

const handleExport = async(e) => {
  setOptions("");
  const content = JSON.stringify(elements);
  const uriContent = "data:application/octet-stream," + encodeURIComponent(content);
  const newWindow = window.open(uriContent, 'Save File');
  setTimeout(function () {
    //Firefox seems to require a setTimeout for this to work.
    var a = document.createElement("a");
    a.href = window.URL.createObjectURL(new Blob([content], { type: "text/json" }));
    a.download = "whiteboard.json";
    newWindow.document.body.appendChild(a);
    a.click();
    newWindow.document.body.removeChild(a);
    setTimeout(function () {
      newWindow.close();
    }, 100);
  }, 0);
}

```

*Code 4. 8: Export JSON file description*

**Import:** Import is the opposite. I will read the file from the local disk, then convert the JSON string to an array. Finally, I will update the newly added elements with the existing elements and finally merge them together.

```

function readFileAsync(file) {
  return new Promise((resolve, reject) => {
    let reader = new FileReader();

    reader.onload = () => {
      resolve(reader.result);
    };

    reader.onerror = reject;
    reader.readAsText(file);
  })
}

const handleLoadFile = async (e) => {
  if(permission === true){
    try {
      let file = e.target.files[0];
      let content = await readFileAsync(file);
      let id = elements.length;
      const elementsRef = JSON.parse(content);
      const elementsCopy = [...elements];
      elementsRef.forEach((element) => {
        const {x1,y1,x2,y2,type,options,src} = element;
        const e = createElement(id,x1,y1,x2, y2, type,options,src);
        elementsCopy[id] = e;
        id = id + 1;
      })
      setElements(elementsCopy);
      setTool("cursor");
      const fileRef = document.getElementById("load");
      fileRef.value = null;
    } catch(err) {
      alert(err);
    }
  }else{
    const fileRef = document.getElementById("load");
    fileRef.value = null;
    alert("No permission!");
  }
}

```

New added element

Existing elements

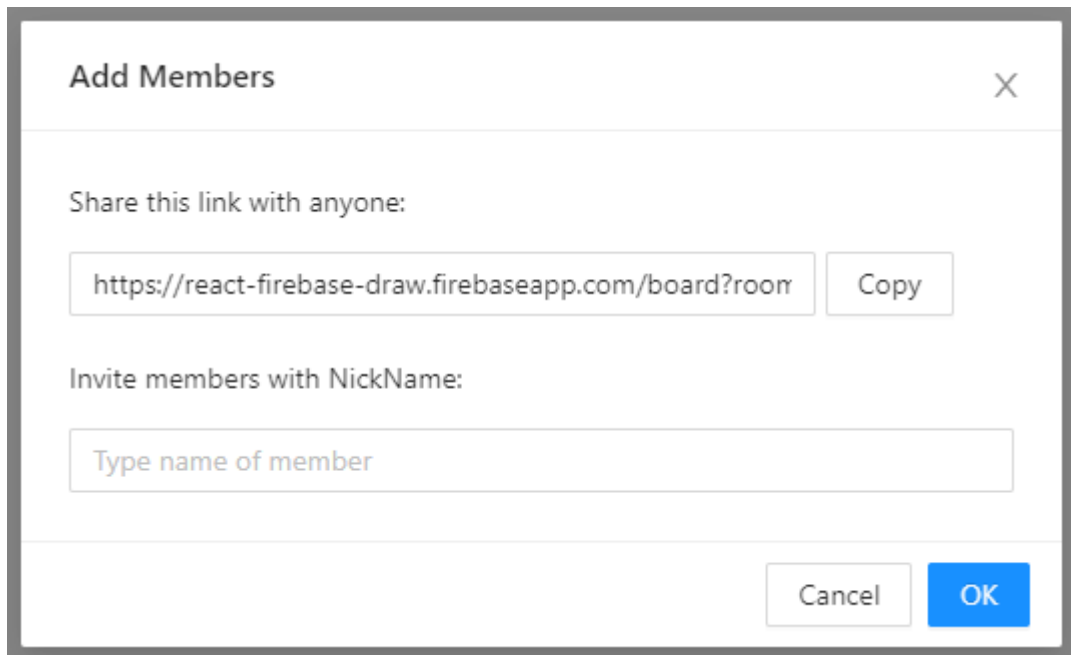
Merge and update elements

Code 4. 9: Import JSON file description



### 4.3.7: Adding member

Look at the figure data: **Figure 3. 6: Rooms firebase data description**, we will see in the room document there will be a member's field, this member field will store all the members uid, so the idea here is that when we add a member, the members field will update the uid of the members into the list. I have designed 2 options for inviting members: share link and invite members with display name. We can see below:

A modal dialog box titled "Add Members" with a close button (X) in the top right corner. The dialog contains two sections. The first section is titled "Share this link with anyone:" and features a text input field containing the URL "https://react-firebase-draw.firebaseio.com/board?room" and a "Copy" button to its right. The second section is titled "Invite members with NickName:" and features a text input field with the placeholder text "Type name of member". At the bottom right of the dialog are two buttons: "Cancel" and "OK".

**Add Members** X

Share this link with anyone:

Invite members with NickName:

*Figure 4. 3: Add members description*

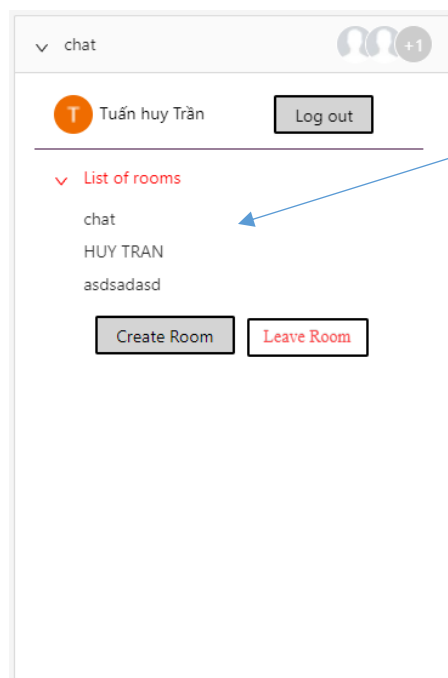
## How to create an encrypted URL?

When a room is selected, the URL is encoded after the room is selected. I will use a special link transfer method called `navigate`, supported by react router [11], below is the code that describes the room list and room selection.



Code 4. 10: How to encrypt and transfer a URL

And here is the interface:



Code 4. 11: List room interface

#### 4.3.7.1: Share Link

After the URL is encoded, my URL will be nested like this:

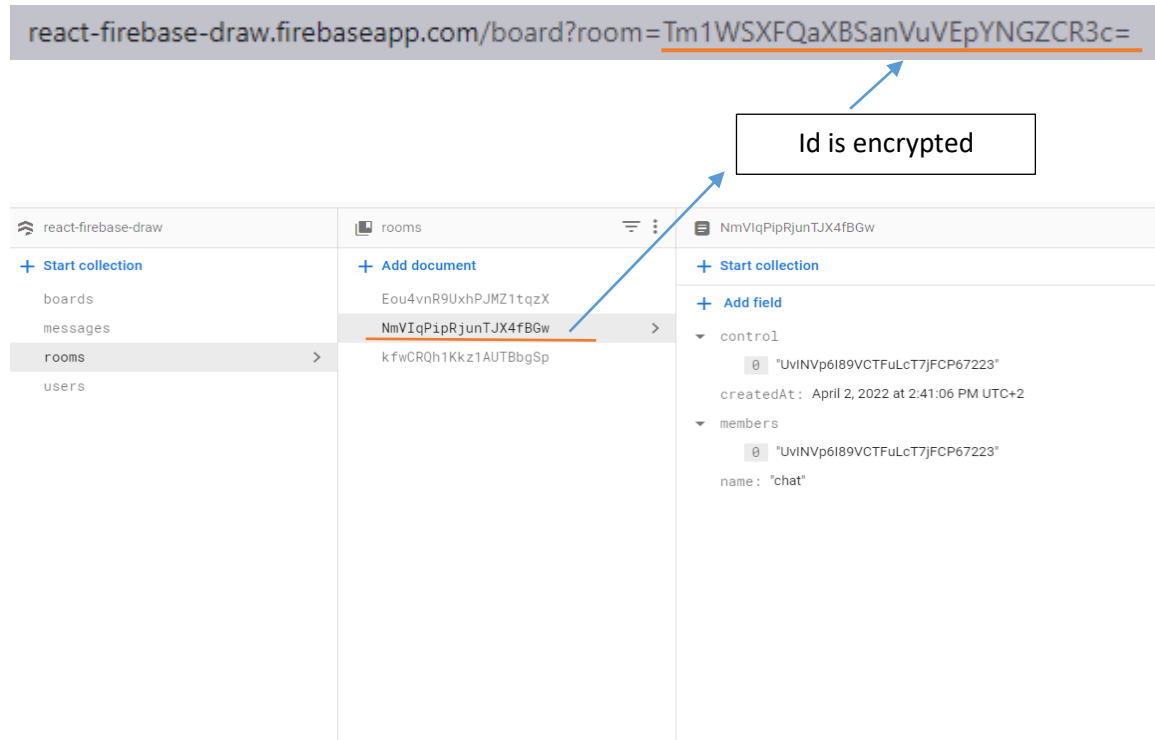


Figure 4. 4: Invite by share link description

#### 4.3.7.2: Invite members with display name

Firstly, we need a function to search for keywords on Firestore and return found objects. Here, I use the debounce library function backed by @mui/material. It helps us perform the search and return the results we want, see below:

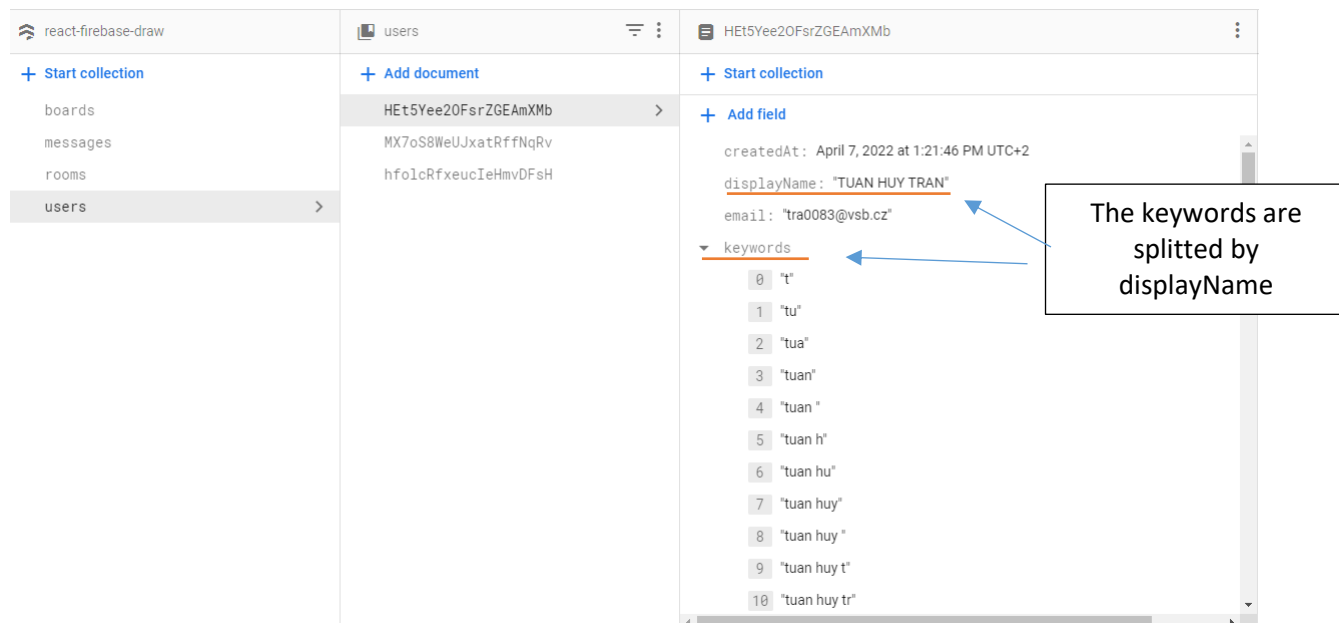


Figure 4. 5: The keywords description

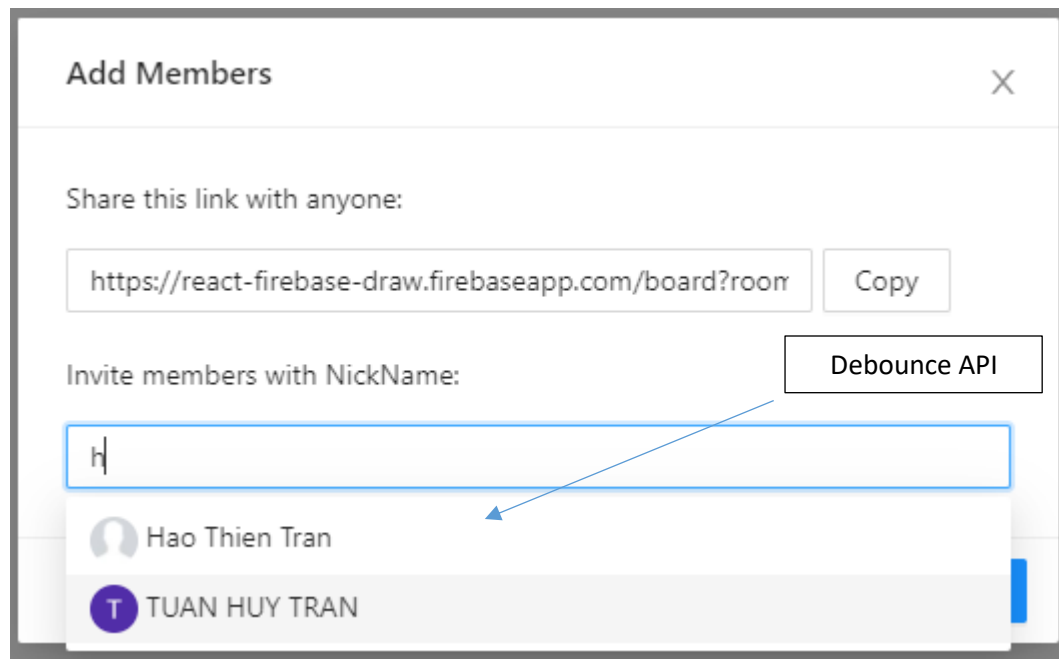


Figure 4. 6: Invite members with name description

```

async function fetchUserList(search, curMembers){
  return db
    .collection('users')
    .where('keywords', 'array-contains', search?.toLowerCase())
    .orderBy('displayName')
    .limit(20)
    .get()
    .then((snapshot) => {
      return snapshot.docs
        .map((doc) => ({
          label: doc.data().displayName,
          value: doc.data().uid,
          photoURL: doc.data().photoURL,
        }))
        .filter((opt) => !curMembers.includes(opt.value));
    });
}

```

Except the uid of the user who logged in

Code 4. 12: Fetch users function from Firestore description

```

function DebounceSelect({fetchOptions, debounceTimeout = 300, ...props}){
  const [fetching, setFetching] = useState(false);
  const [options, setOptions] = useState([]);

  //use for onSearch
  const debounceFetcher = React.useMemo(() => {
    const loadOptions = (value) => {
      setOptions([]);
      setFetching(true);

      fetchOptions(value, props.curMembers).then(newOptions => {
        setOptions(newOptions);
        setFetching(false);
      })
    }
    return debounce(loadOptions,debounceTimeout)
  }, [debounceTimeout,fetchOptions])

  return (
    <Select
      labelInValue
      filterOption={false}
      onSearch={debounceFetcher}
      notFoundContent={fetching ? <Spin size="small"/> : null}
      {...props}
    >
      {options?.map?.((opt) => (
        <Select.Option key={opt.value} value={opt.value} title={opt.label}>
          <Avatar size='small' src={opt.photoURL}>
            {opt.photoURL ? '' : opt.label?.charAt(0)?.toUpperCase()}
          </Avatar>
          {`${opt.label}`}
        </Select.Option>
      ))}
    </Select>
  )
}

```

Debounce

```

<Form form={form} layout="vertical" style={{marginTop:"20px"}}>
  <p>Invite members with Display name: </p>
  <DebounceSelect
    mode="multiple"
    label="Name of members"
    value={value}
    placeholder = "Type name of member"
    fetchOptions={fetchUserList}
    onChange={newValue => setValue(newValue)}
    style={{width: '100%'}}
    curMembers={selectedRoom.members}
  >

  </DebounceSelect>
</Form>

```

Code 4. 12: Fetch users function from  
Firestore description

Code 4. 13: Debounce options description

Secondly, when the form has been filled out we need a function to finish, which will update the selected **uid** to **members** on Firestore.

```

const handleOk = () => {
  try{
    form.resetFields();
    // update members in current room
    const roomRef = db.collection('rooms').doc(selectedRoomId);
    roomRef.update({
      members: [...selectedRoom.members, ...value.map((val) => val.value)],
    });

    setValue([]);
    setIsInviteMemberVisible(false);
  }catch{
    alert("Please join a room!");
  }
}
}

```

Code 4. 14: Update members

### 4.3.8: Giving control

Basically, giving control is quite similar to **4.3.7.2: Invite members with display name**. I completely reused the same functions, but it has a little change accordingly.

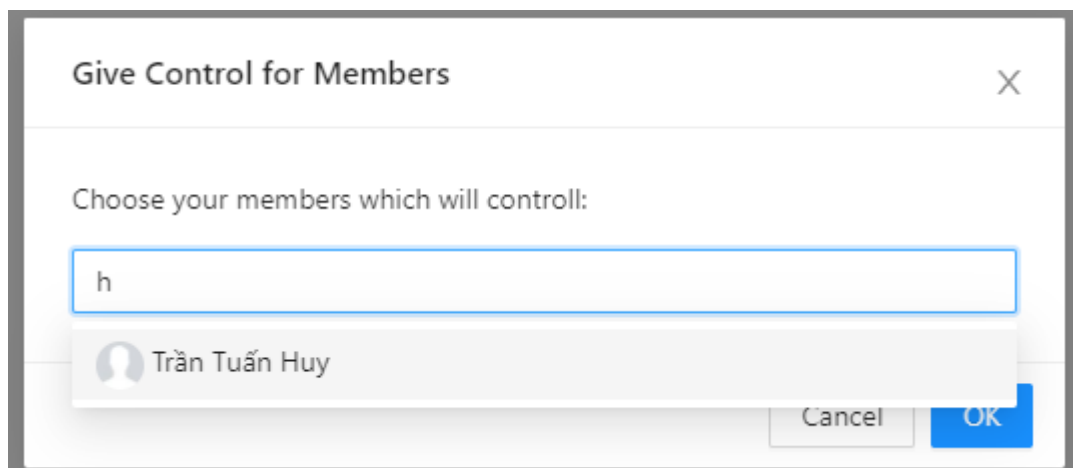


Figure 4. 7: Give control to members with name description

Fetch user function no longer fetches all users on Firestore, this time it only looks for members present in the room. See below:

```

async function fetchUserList(search, curMembers){
  return db
    .collection('users')
    .where('keywords', 'array-contains', search?.toLowerCase())
    .orderBy('displayName')
    .limit(20)
    .get()
    .then((snapshot) => {
      return snapshot.docs
        .map((doc) => ({
          label: doc.data().displayName,
          value: doc.data().uid,
          photoURL: doc.data().photoURL,
        }))
        .filter((opt) => opt.value.includes(curMembers));
    });
}

```

Fetch member in this room

Code 4. 15: Fetch member description

The debounce function is exactly the same but the update function updates the field control.

```

const handleOk = () => {
  try{
    form.resetFields();
    const roomRef = db.collection('rooms').doc(selectedRoomId);
    // update members in current room
    roomRef.update({
      control: [...value.map((val) => val.value)],
    });

    setValue([]);
    setIsGiveControlMemberVisible(false);
  }catch{
    alert("Please join a room!");
  }
}

```

Code 4. 16: update control description

```

function DebounceSelect({fetchOptions, debounceTimeout = 300, ...props}){
  const [fetching,setFetching] = useState(false);
  const [options, setOptions] = useState([]);

  //use for onSearch
  const debounceFetcher = React.useMemo(() => {
    const loadOptions = (value) => {
      setOptions([]);
      setFetching(true);

      fetchOptions(value, props.curMembers).then(newOptions => {
        setOptions(newOptions);
        setFetching(false);
      })
    }
    return debounce(loadOptions,debounceTimeout)
  }, [debounceTimeout,fetchOptions])

  return (
    <Select
      labelInValue
      filterOption={false}
      onSearch={debounceFetcher}
      notFoundContent={fetching ? <Spin size="small"/> : null}
      {...props}
    >
      {options?.map?.((opt) => (
        <Select.Option key={opt.value} value={opt.value} title={opt.label}>
          <Avatar size='small' src={opt.photoURL}>
            {opt.photoURL ? '' : opt.label?.charAt(0)?.toUpperCase()}
          </Avatar>
          {`${opt.label}`}
        </Select.Option>
      ))}
    </Select>
  )
}

```

Debounce

```

<Form form={form} layout="vertical" style={{marginTop:"20px"}}>
  <p>Invite members with Display name: </p>
  <DebounceSelect
    mode="multiple"
    label="Name of members"
    value={value}
    placeholder = "Type name of member"
    fetchOptions={fetchUserList}
    onChange={newValue => setValue(newValue)}
    style={{width: '100%'}}
    curMembers={selectedRoom.members}
  >

  </DebounceSelect>
</Form>

```

Code 4. 15: Fetch member description

Code 4. 17: Giving control description

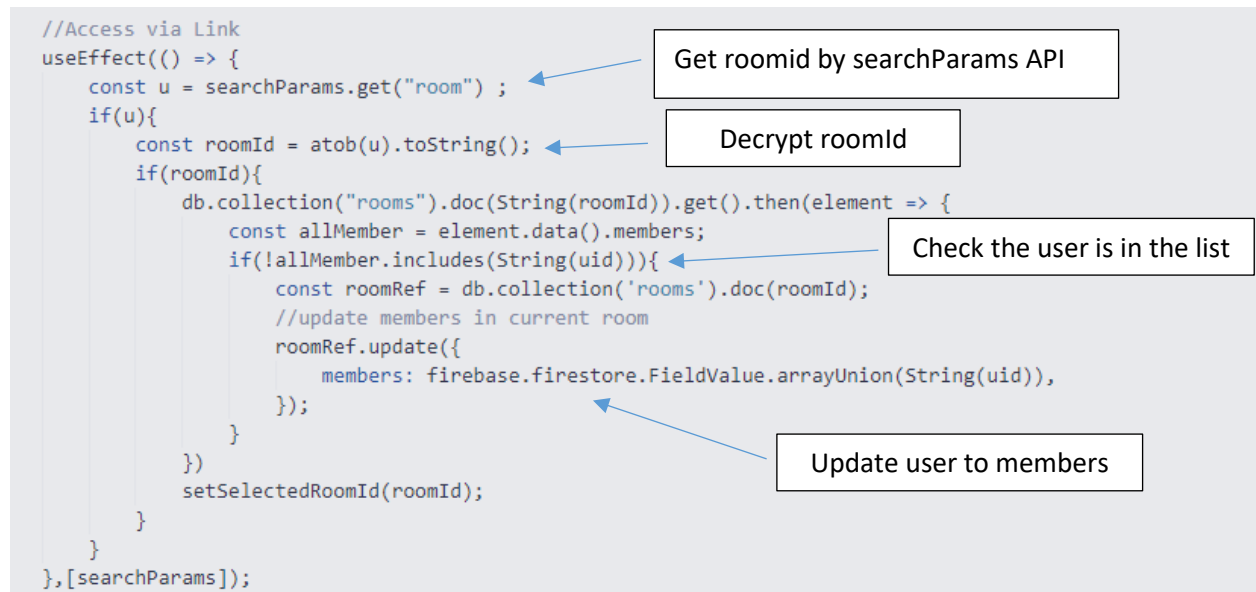


### 4.3.9: Accessing via Link

As we have seen, when a new user accesses through this address, the URL will be split to get the roomId and decrypt it back to the pure roomId. This user will then be updated in the members field if the user is not in this list. In my project, I use atob() to decrypt and btoa() to encrypt my roomId, you can see the documentation here:

Atob: [12]

Btoa: [13]



Code 4. 18: Invite by share link descriptions

### 4.3.10: Login

My idea is to build an authentication layer. If this authentication layer is passed, the session will move to the dash board with navigate by navigate support from react router. [11]

Firstly, I will build an authentication class and the user's data will be saved to the context (document about context here: [14]). if the user does not exist then the page will automatically go to the login page.

```

export const AuthContext = React.createContext();
export default function AuthProvider({children}){
  const [user, setUser] = useState({});
  const navigate = useNavigate();
  const [isLoading, setIsLoading] = useState(true);
  const [searchParams, setSearchParams] = useSearchParams();

  React.useEffect(() => {
    const unsubscribed = auth.onAuthStateChanged((user) => {
      if(user){
        const {displayName, email, uid, photoURL} = user;
        setUser({
          displayName, email, uid, photoURL
        });
        setIsLoading(false);
        if(searchParams.get("room")){
          navigate(`~/board?room=${searchParams.get("room")}`);
        }else{
          navigate('/board');
        }
        return;
      }else{
        setIsLoading(false);
        setUser({});
        navigate('/login');
      }
    });
    return () => {
      unsubscribed();
    }
  },[])

  return (
    <div>
      <AuthContext.Provider value={{user}}>
        {isLoading ? <Spin style={{ position: 'fixed', inset: 0 }} /> : children}
      </AuthContext.Provider>
    </div>
  )
}

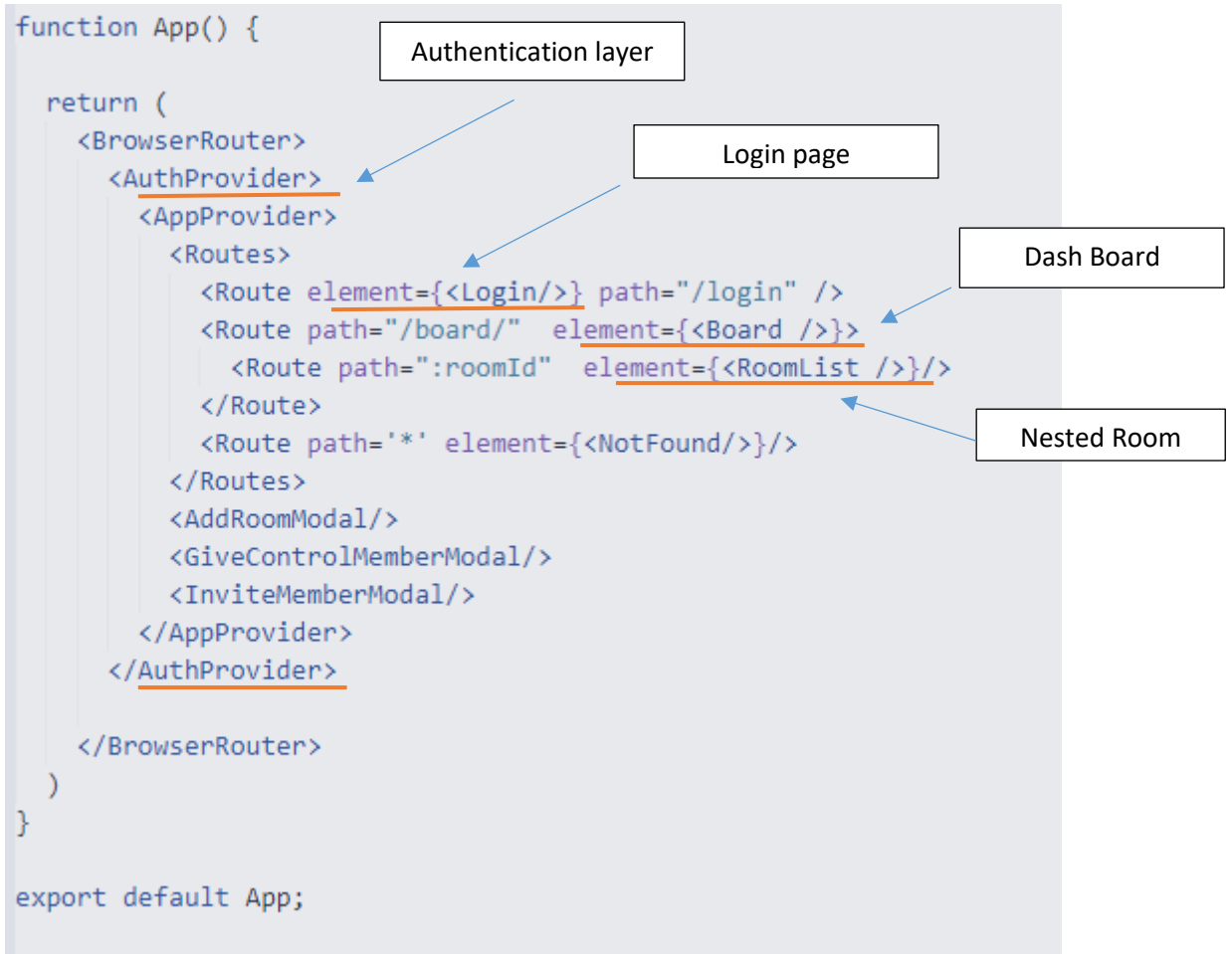
```

Diagram annotations:

- Create a context**: Points to `React.createContext()`
- Check user available**: Points to `if(user){`
- Login success via URL**: Points to `navigate(`~/board?room=${searchParams.get("room")}`);`
- Login successful**: Points to `navigate('/board');`
- Login unsuccessful**: Points to `navigate('/login');`

Code 4. 12: Authentication layer description

Secondly, Authentication layer will wrap my entire application, if it is not passed this application cannot access. I used the react router library to support accessing the website system. You can see the router documentation here: [11].



Code 4. 13: Application Layer Description

The login page will use an authenticator powered by firebase authentication. Here I will use 2 supported validators are Facebook and Google. When the user logs in, it will be checked as a new or old user. If it is new, it will be updated on Firestore. [15]

```
const fbProvider = new firebase.auth.FacebookAuthProvider();
const googleProvider = new firebase.auth.GoogleAuthProvider();
```

Two supported validators

```
export default function Login() {
  const handleLogin = async (provider) => {
    const { additionalUserInfo, user } = await auth.signInWithPopup(provider);
    document.body.style.overflow = "hidden";
    if (additionalUserInfo?.isNewUser) {
      addDocument('users', {
        displayName: user.displayName,
        email: user.email,
        photoURL: user.photoURL,
        uid: user.uid,
        providerId: additionalUserInfo.providerId,
        keywords: generateKeywords(user.displayName?.toLowerCase()),
      });
    }
  };
};
```

```
return (
  <div style={{marginTop: "60px", justifyContent:"center"}}>
    <Row justify='center' style={{ height: 800 }}>
      <Col span={8}>
        <Title style={{ textAlign: 'center' }} level={3}>
          AUTHENTICATION
        </Title>
        <Button
          style={{ width: '100%', marginBottom: 5 }}
          onClick={() => handleLogin(googleProvider)}
        >
          Login with Google
        </Button>
        <Button
          style={{ width: '100%' }}
          onClick={() => handleLogin(fbProvider)}
        >
          Login with Facebook
        </Button>
      </Col>
    </Row>
  </div>
);
}
```

Check new user and update Firestore

### 4.3.11: Use chat

I will divide it into 2 parts: How to send a message and how to receive the message.

#### How to send a message?

Firstly, I will build a function addDocument() to support sending data to Firestore.

```
export const addDocument = (collectionn, data) => {
  db.collection(collectionn).add({
    ...data,
    createdAt: firebase.firestore.FieldValue.serverTimestamp(),
  });
};
```

Code 4. 14: add document to Firestore functions

After that, I will build a function that can send data to Firestore.

The image displays two code snippets with callouts. The top snippet is a React component using `FormStyled` and `Form.Item`. It includes an `Input` field with `onPressEnter={handleOnSubmit}` and a `Button` with `onClick={handleOnSubmit}`. A callout box labeled "Execute functions" has arrows pointing to the `handleOnSubmit` prop in both the `Input` and `Button` elements. The bottom snippet shows the `handleOnSubmit` function implementation, which calls `addDocument('messages', { ... })` and then resets the form fields. A callout box labeled "Code 4. 14: add document to Firestore" has an arrow pointing to the `addDocument` function call.

```
<FormStyled form={form}>
  <Form.Item name={"messages"}>
    <Input
      onChange={handleInputChange}
      onPressEnter={handleOnSubmit}
      placeholder='Input message'
      bordered={false}
      autoComplete="off"
    />
  </Form.Item>
  <Button type="primary" onClick={handleOnSubmit}>Send</Button>
</FormStyled>
```

```
const handleOnSubmit = () =>{
  addDocument('messages', {
    text: inputValue,
    uid,
    photoURL,
    roomId: selectedRoom.id,
    displayName,
  })
  form.resetFields(['messages']);
};
```

Code 4. 15: Send messages description

### How to receive the message?

My idea is to listen for 'message' change from Firestore, if there is change the data will be updated and displayed on client's screen.

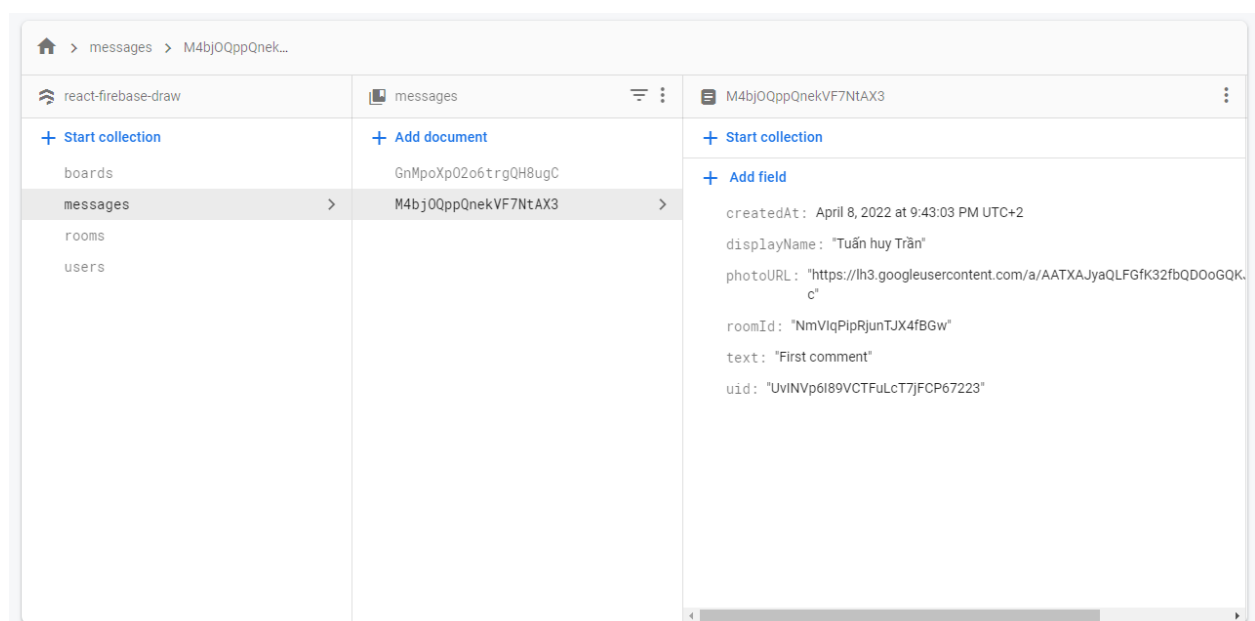


Figure 4. 8: Example of messages stored at Firestore

Firstly, I will initialize a function that listens for changes from Firestore called `useFirestore` which takes an event `onSnapshot()` which is supported by Firestore, it helps us to always listen for changes from the Firestore, for example, if a message is pushed to the Firestore, it will realized the change and clients start receiving new messages from Firestore. [16]

```
const useFirestore = (collection, condition) => {
  const [documents, setDocuments] = useState([]);

  React.useEffect(() => {
    let collectionRef = db.collection(collection).orderBy('createdAt');
    if (condition) {
      if (!condition.compareValue || !condition.compareValue.length) {
        setDocuments([]);
        return;
      }

      collectionRef = collectionRef.where(
        condition.fieldName,
        condition.operator,
        condition.compareValue
      );
    }

    const unsubscribe = collectionRef.onSnapshot((snapshot) => {
      const documents = snapshot.docs.map((doc) => ({
        ...doc.data(),
        id: doc.id,
      }));

      setDocuments(documents);
    });

    return unsubscribe;
  }, [collection, condition]);

  return documents;
};
```

Parameter

Always listen the change

Code 4. 16: Function `useFirestore` description

```
const messCondition = useMemo(() => ({
  fieldName: 'roomId',
  operator: '==',
  compareValue: selectedRoom.id
}), [selectedRoom.id])
const messages = useFirestore('messages', messCondition);
```

Filter data

Code 4. 17: use `useFirestore()` for update new messages description

Second, after the message is updated, I will show it on the client's chat box.

```

<MessageListStyled className='bodyContent'>
  {
    messages.map(mes =>
      <Message
        key={mes.id}
        text={mes.text}
        photoURL={mes.photoURL}
        displayName={mes.displayName}
        createdAt={mes.createdAt}
      />
    )
  }
</MessageListStyled>

```

Code 4. 18: Show messages description

And finally, the result:

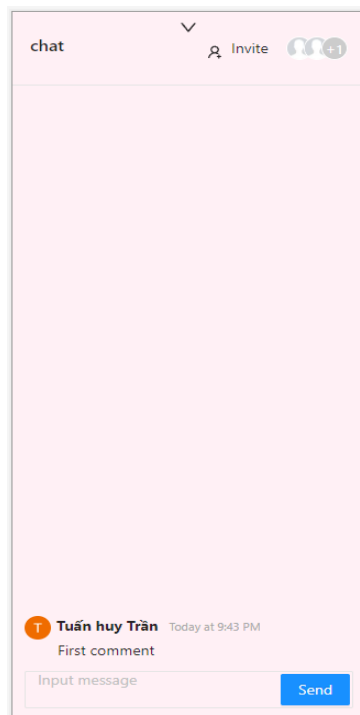


Figure 4. 9: Chat box result

#### 4.3.12: Get element at position

**Description:** This algorithm will tell us if there will be an object at the position of the pointer.

1. **Line:** for the line, how to know that the pointer is the position C is on the line A, B? Suppose pointer C lies on line AB, that means that  $\text{distance}(AB) = \text{distance}(AC) + \text{distance}(CB)$ .

For example:

$$A - - C - - - B$$

We will use the following algorithm:

$$distance(a,b) = (distance(a,c) + distance(b,c))$$

Or:

$$distance(a,b) - (distance(a,c) + distance(b,c)) = 0$$

Distance of  $A(x,y)$  to  $B(x,y)$ :

$$\sqrt{(A.x - B.x)^2 + (A.y - B.y)^2}$$

**Encryption algorithm:**

- Calculate the distance:

```
const distance = (a,b) => Math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y,2));
```

*Code 4. 19: Calculate the distance*

- Check the cursor is inside of line:

```
const onLine = (x1, y1, x2, y2, x, y, maxDistance = 1) => {  
  const a = { x: x1, y: y1 };  
  const b = { x: x2, y: y2 };  
  const c = { x, y };  
  const offset = distance(a, b) - (distance(a, c) + distance(b, c));  
  return Math.abs(offset) < maxDistance ? "inside" : null;  
};
```

*Code 4. 20: Check the cursor is inside of line*

2. **Rectangle:** for the rectangle, To know if the pointer  $C(x,y)$  is in rectangle( $x1,y1,x2,y2$ ) we need to consider:

$$x \geq x1 \ \&\& \ x \leq x2 \ \&\& \ y \geq y1 \ \&\& \ y \leq y2$$

**Encryption algorithm:**

```
if( x >= x1 && x <= x2 && y >= y1 && y <= y2 ){  
  return 'inside';  
}else{  
  return;  
}
```

*Code 4. 21: Check the cursor is inside of rectangle*



3. **Circle:** In order to find out where the cursor is on the circle, I will calculate the position distance from the cursor(x,y) to the center(x1,y1) of the circle and compare it with the radius r of the circle.

$$distance(cursor, center) < r$$

Encryption algorithm:

```
case "circle":
  const r = Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2))/2;
  const insideCircle = Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)) < r ? "inside" : null;
```

*Code 4. 22: Check the cursor is inside of circle*

4. **Picture, text:** For picture and text it is completely similar to rectangle
5. **Pencil (drawn by mouse):** when we draw with the mouse, it's not a straight line, but it has a structure like many adjacent lines. it means, we will use **Code 4. 20: Check the cursor is inside of line** to check all pairs of points on the pencil line.

Encryption algorithm:

```
const onLine = (x1, y1, x2, y2, x, y, maxDistance = 1) => {
  const a = { x: x1, y: y1 };
  const b = { x: x2, y: y2 };
  const c = { x, y };
  const offset = distance(a, b) - (distance(a, c) + distance(b, c));
  return Math.abs(offset) < maxDistance ? "inside" : null;
};

const betweenAnyPoint = element.points.some((point, index) => {
  const nextPoint = element.points[index + 1];
  if (!nextPoint) return false;
  return onLine(point.x, point.y, nextPoint.x, nextPoint.y, x, y, 5) != null;
});

return betweenAnyPoint ? "inside" : null;
```

*Code 4. 23: Check the cursor is inside of pencil's line*

Finally, I will go through all the elements in the array that return 'null' or 'inside' positions and get the element at the nearest position.

```
const getElementAtPosition = (x, y, elements) => {
  return elements.slice().reverse()
    .map(element => ({ ...element, position: positionWithinElement(x, y, element) }))
    .find(element => element.position !== null && element.isDeleted === false);
};
```

*Code 4. 24: Get element at position description*

### 4.3.13: Moving an element:

**Description:** This algorithm supports that I can move an object on the canvas when an object is moved to a new position, the new positions will be updated and continuously re-rendered by the canvas.

- First, when we want to move an object, we need to check that the cursor's pixel is exactly at the position of the object to be moved? (See on: **4.3.12: Get element** at position) Set action is 'moving' and set selected element to update position in next step.

```
if(tool === "selection"){
  const element = getElementAtPosition(clientX, clientY,elements);
  if(element && element.isDeleted === false){

    if (element.type === "pencil") {
      const xOffsets = element.points.map(point => clientX - point.x);
      const yOffsets = element.points.map(point => clientY - point.y);
      setSelectedElement({ ...element, xOffsets, yOffsets });
    } else {
      // if(element.type === "rectangle" || element.type === "line"){
      //   generator.defaultOptions = element.roughElement.options;
      // }
      const offsetX = clientX - element.x1;
      const offsetY = clientY - element.y1;
      setSelectedElement({...element,offsetX,offsetY});
    }
    setSelected(element);
    setElements(prevState => prevState);

    if(element.position === "inside"){
      setAction("moving");
    }else{
      setAction("resizing");
    }
  }
}
```

get element at position

Set selected element

- Second, to move the object's position, we will update the object in the new position by overwriting the object in the new position (see on: **Code 4. 4: Update Element function**).

```

else if (action === "moving") {
  if (selectedElement.type === "pencil") {
    const newPoints = selectedElement.points.map((_, index) => ({
      x: clientX - selectedElement.xOffsets[index],
      y: clientY - selectedElement.yOffsets[index],
    }));
    const elementsCopy = [...elements];
    elementsCopy[selectedElement.id] = {
      ...elementsCopy[selectedElement.id],
      points: newPoints,
    };
    setSelected(elementsCopy[selectedElement.id]);
    setElements(elementsCopy, true);
  }
} else {
  //Line & rec
  const { id, x1, x2, y1, y2, type, offsetX, offsetY, options, src } = selectedElement;
  const width = x2 - x1;
  const height = y2 - y1;
  const newX1 = clientX - offsetX;
  const newY1 = clientY - offsetY;

  updateElement(id, newX1, newY1, newX1 + width, newY1 + height, type, options, src);
  setSelected(elements[id]);
}

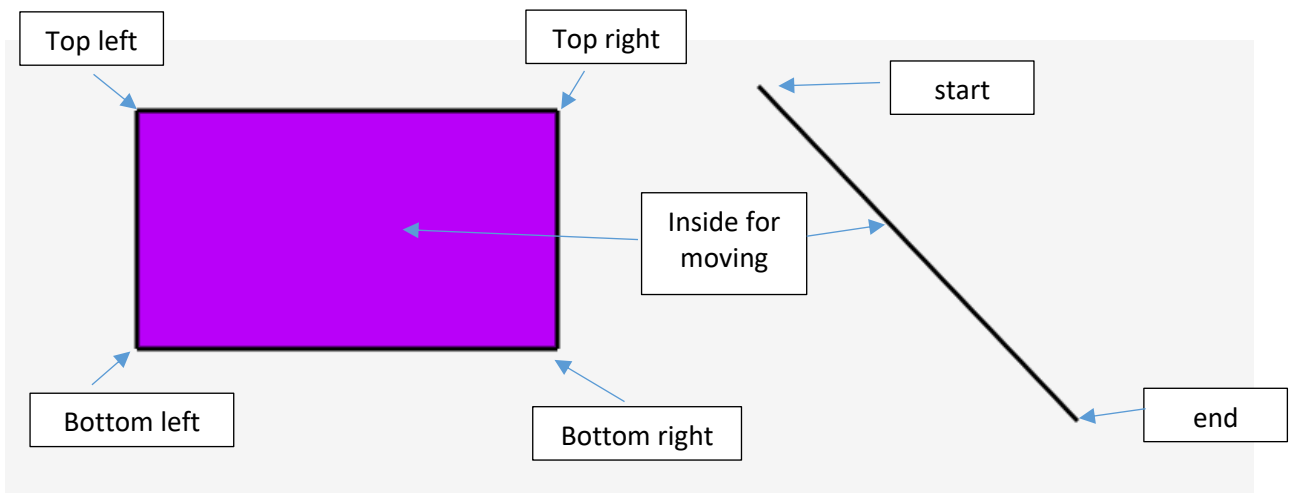
```

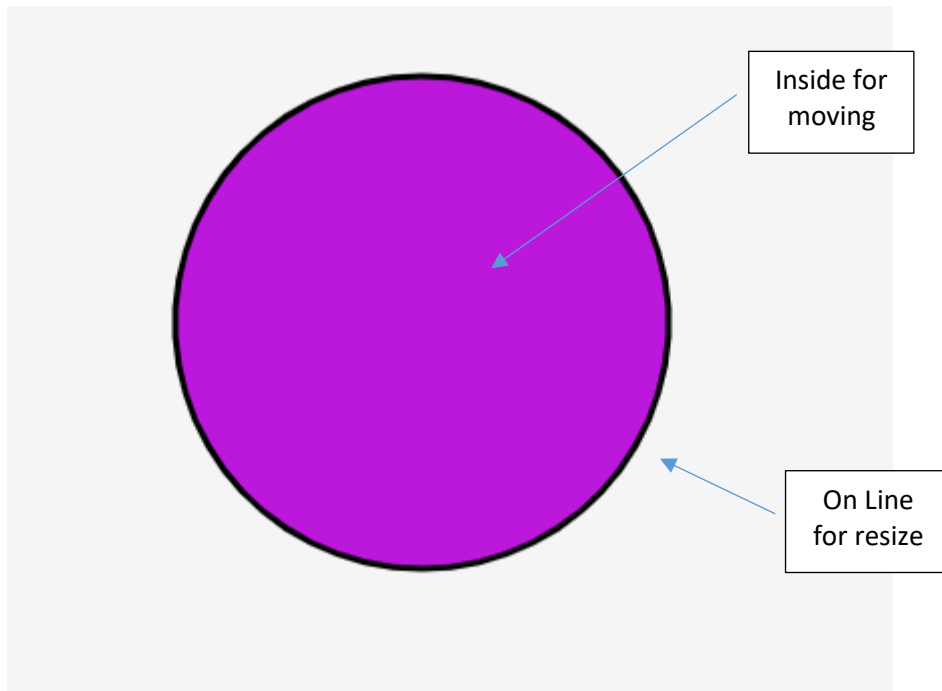
Update element for pencil

Update new position another element

#### 4.3.14: Resizing an element:

To be able to resize an element, I will select some position on the selected element to resize. For example, on the line I will choose the starting and ending position, on the rectangle I choose the top left, top right, bottom left and bottom right positions, on the circle I choose the wallet position on the drawing line. See below for a better understanding.





*Figure 4. 10: For example, about resizing element*

```

const positionWithinElement = (x, y, element) => {
  const { type, x1, x2, y1, y2 } = element;
  switch (type) {
    case "line":
      const on = online(x1, y1, x2, y2, x, y);
      const start = nearPoint(x, y, x1, y1, "start");
      const end = nearPoint(x, y, x2, y2, "end");
      return start || end || on;
    case "rectangle":
      const topLeft = nearPoint(x, y, x1, y1, "tl");
      const topRight = nearPoint(x, y, x2, y1, "tr");
      const bottomLeft = nearPoint(x, y, x1, y2, "bl");
      const bottomRight = nearPoint(x, y, x2, y2, "br");
      const inside = x >= x1 && x <= x2 && y >= y1 && y <= y2 ? "inside" : null;
      return topLeft || topRight || bottomLeft || bottomRight || inside;
    case "circle":
      const r = Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2))/2;
      const insideCircle = Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)) < r ? "inside" : null;
      const onLineC = Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)) >= r
        && Math.sqrt((x-x1)*(x-x1) + (y-y1)*(y-y1)) <= r + 5
        ? "onLine" : null;
      console.log(onLineC);
      return insideCircle || onLineC;
    case "picture":
      const topL = nearPoint(x, y, x1, y1, "tl");
      const topR = nearPoint(x, y, x2, y1, "tr");
      const bottomL = nearPoint(x, y, x1, y2, "bl");
      const bottomR = nearPoint(x, y, x2, y2, "br");
      const insideP = x >= x1 && x <= x2 && y >= y1 && y <= y2 ? "inside" : null;
      return topL || topR || bottomL || bottomR || insideP;
    case "pencil":
      const betweenAnyPoint = element.points.some((point, index) => {
        const nextPoint = element.points[index + 1];
        if (!nextPoint) return false;
        return online(point.x, point.y, nextPoint.x, nextPoint.y, x, y, 5) != null;
      });
      return betweenAnyPoint ? "inside" : null;
    case "text":
      return x >= x1 && x <= x2 && y >= y1 && y <= y2 ? "inside" : null;
    default:
      throw new Error(`Type not recognised: ${type}`);
  }
};

```

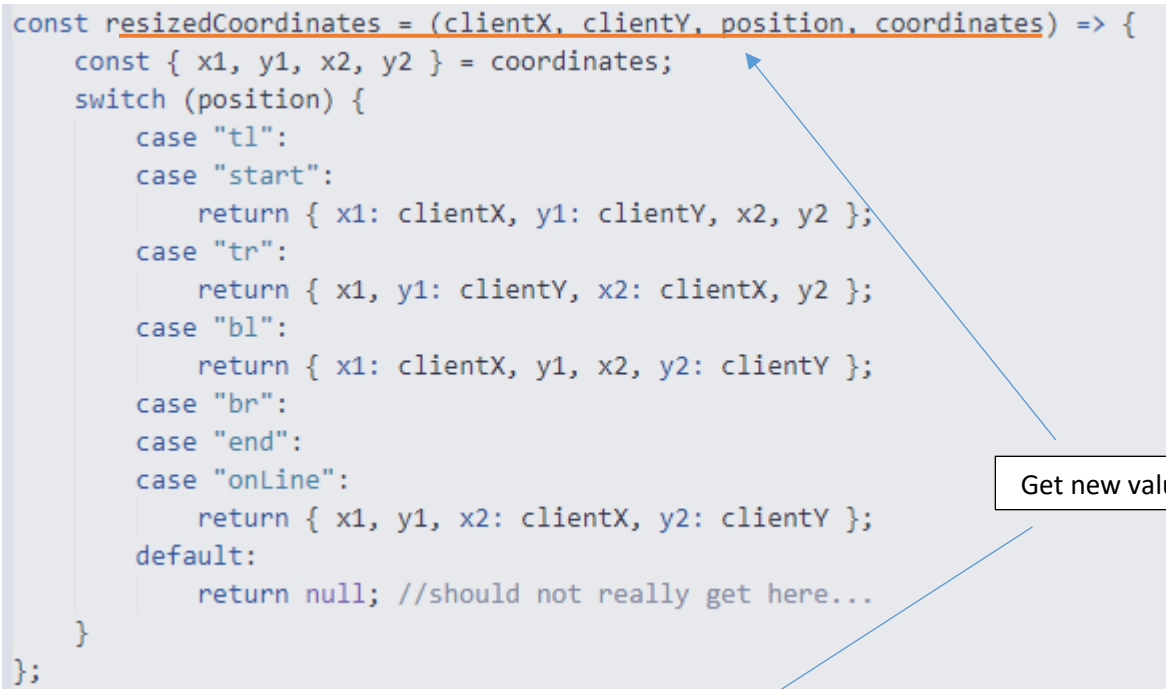
Code 4. 25: Check element at position function

When the cursor is at these positions and click and drag to resize the new size update function named resized Coordinates (**Code 4. 26: resized Coordinates function**) whose function is to return the latest value of the element and the updateElement (**Code 4. 4: Update Element function**) function will update array again.

```

const resizedCoordinates = (clientX, clientY, position, coordinates) => {
  const { x1, y1, x2, y2 } = coordinates;
  switch (position) {
    case "tl":
    case "start":
      return { x1: clientX, y1: clientY, x2, y2 };
    case "tr":
      return { x1, y1: clientY, x2: clientX, y2 };
    case "bl":
      return { x1: clientX, y1, x2, y2: clientY };
    case "br":
    case "end":
    case "onLine":
      return { x1, y1, x2: clientX, y2: clientY };
    default:
      return null; //should not really get here...
  }
};

```



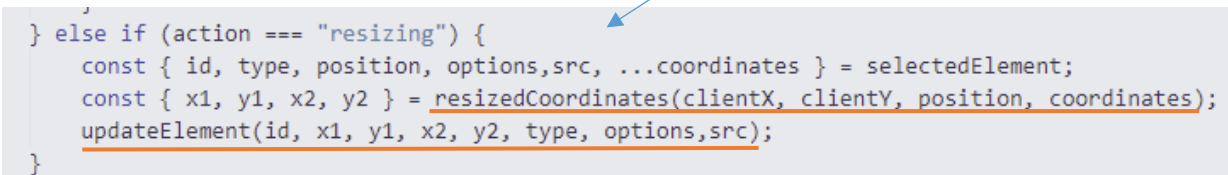
Get new values for update

*Code 4. 26: resized Coordinates function*

```

} else if (action === "resizing") {
  const { id, type, position, options,src, ...coordinates } = selectedElement;
  const { x1, y1, x2, y2 } = resizedCoordinates(clientX, clientY, position, coordinates);
  updateElement(id, x1, y1, x2, y2, type, options,src);
}

```



*Code 4. 27: Update new size description*

## Chapter 5: Result

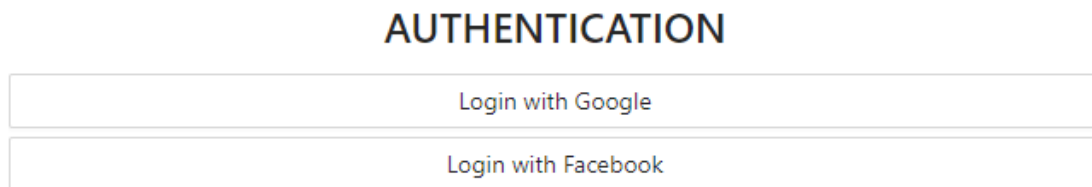
### 5.1: Instruction

This chapter describes and evaluates the results achieved during the project; the results compared with the original plan; the comparison of the results with other popular projects, and finally, the ways to overcome unresolved things.

### 5.2: Overall results

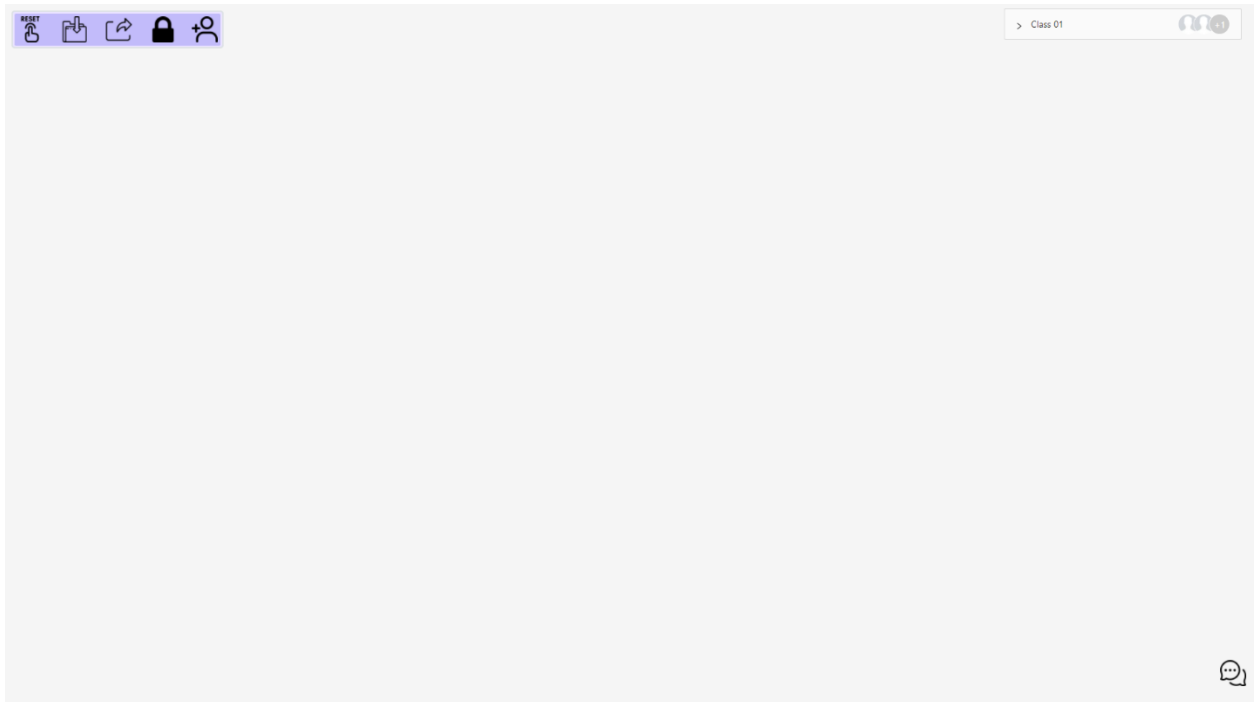
I will describe the state of the whiteboard as follows in **Figure 3. 15: State diagram**, I will present and explain from the user's perspective, if there are mistakes. I hope the teachers will ignore them.

- **Authentication:** The login interface will be designed simply, making it quick and convenient for users to log in. The system requires users to log in to use it via Facebook or Google provider. If the login fails, the page will not transition state.



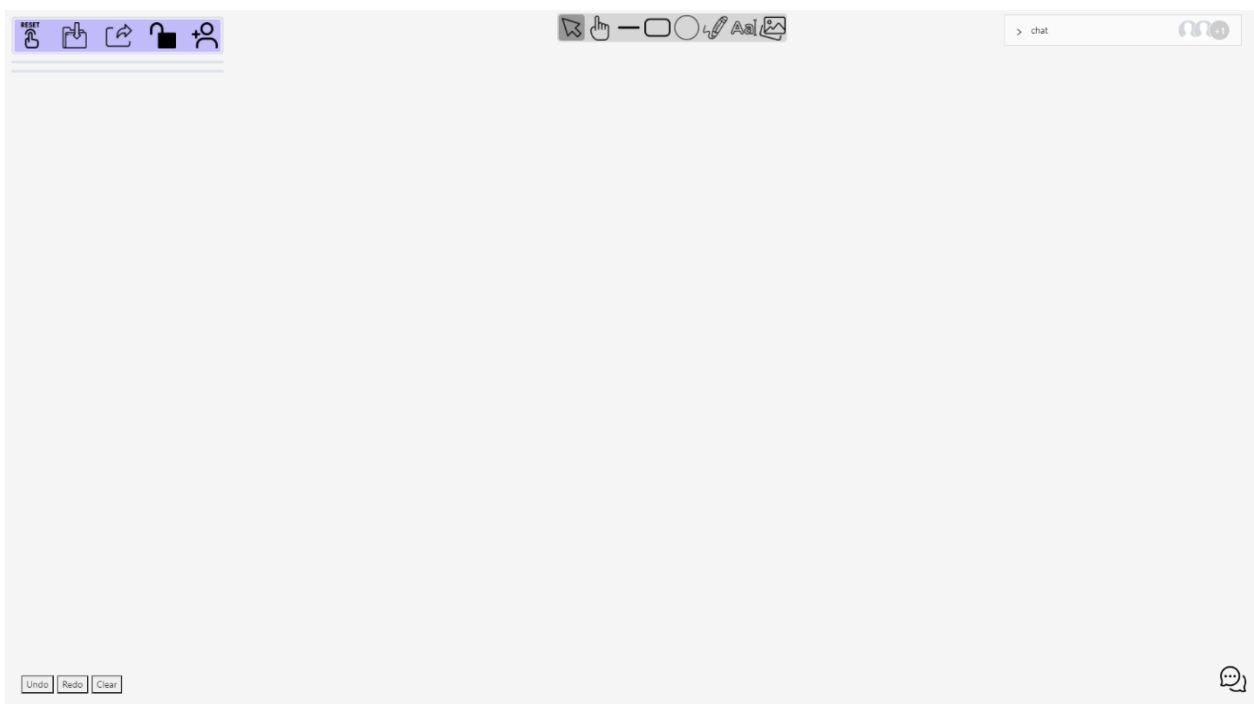
*Figure 5. 1: Authentication interface*

- **Dashboard:** The dashboard will be divided into 2 types: controller and viewer.
  - **Viewer:** Observers will be able to use some limited functionality and use the chat box to chat with other users.



*Figure 5. 2: Viewer dashboard*

- **Controller:** Observers will be able to use some limited functionality and use the chat box to chat with other users.



*Figure 5. 3: Controller dashboard*



Some special features of the controller (**4.3.3: Use Tools**):

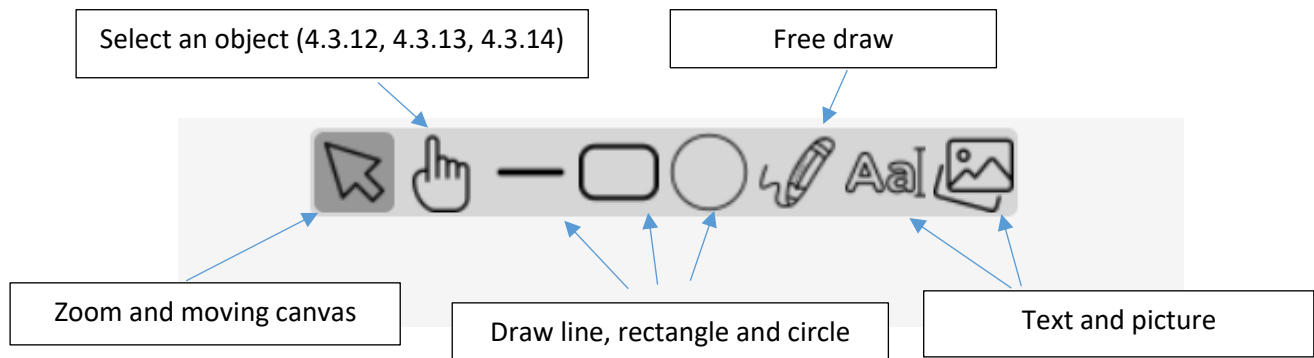


Figure 5. 4: Use tools



Figure 5. 5: example of rectangles Actions and rectangles Options

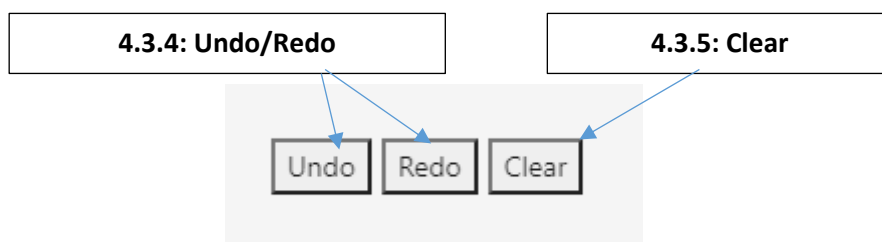


Figure 5. 6: Controller other functions

Some general features:

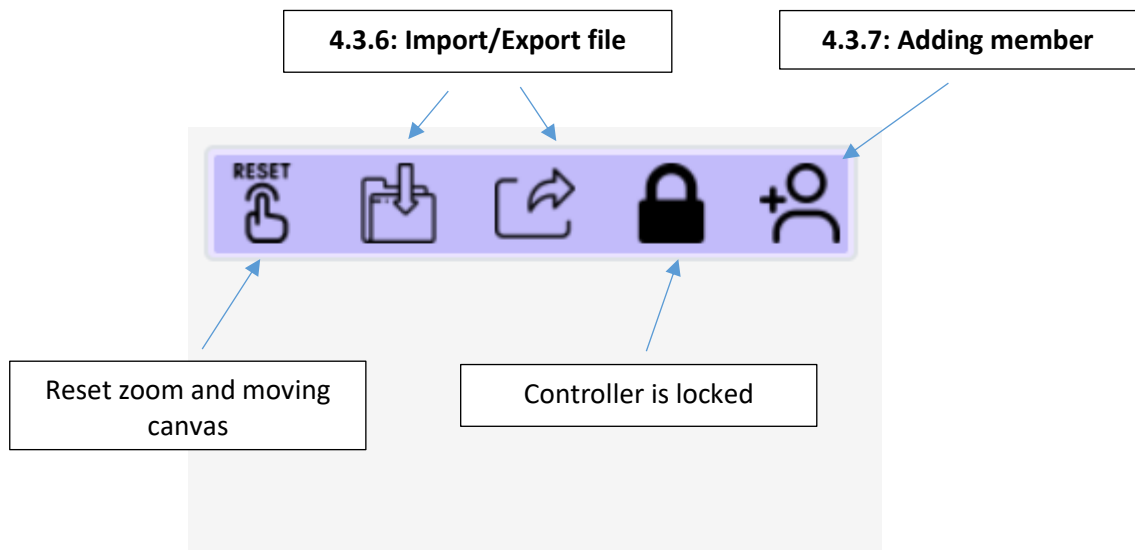


Figure 5. 7: General functions

Dashboard on mobile device:



Figure 5. 8: Mobile dashboard version

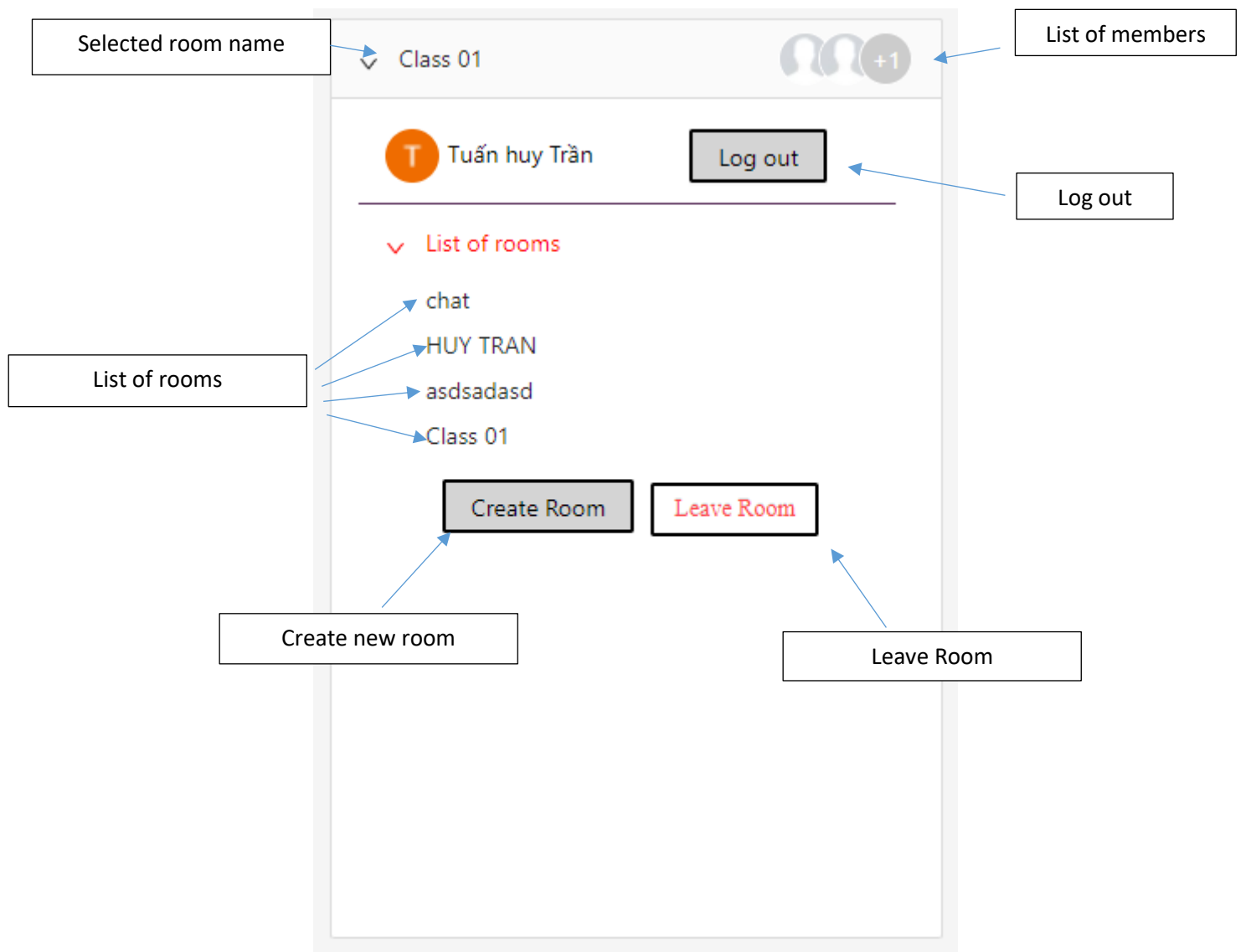


Figure 5. 9: User management

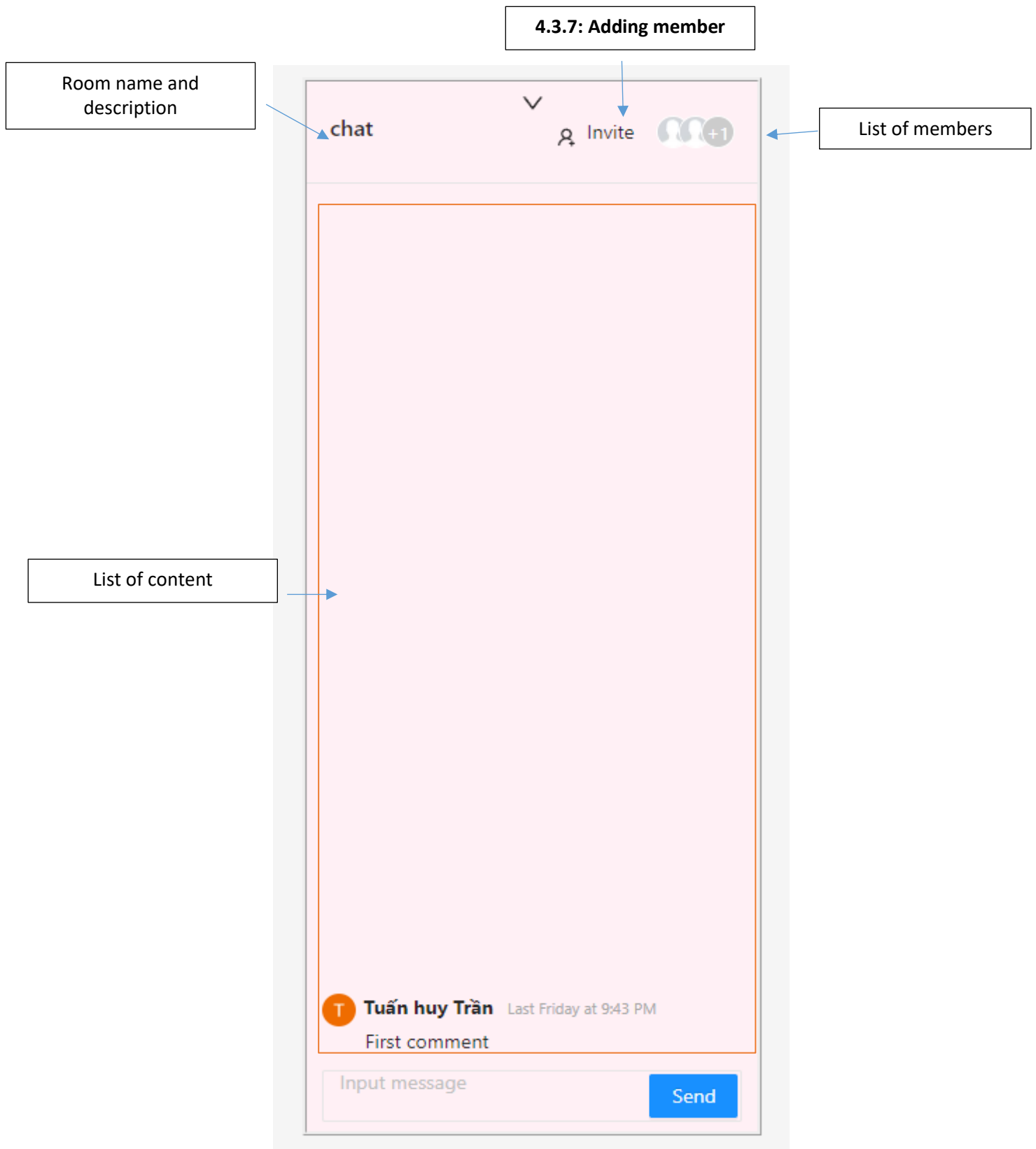


Figure 5. 10: Use Chat box function

In general, the results obtained are positive compared to the initial expected. However, this is the first project that I have done so in the process of making the project I will still have many mistakes and I hope Readers can sympathize and ignore them. In this section I have finished presenting the application from the user's perspective.

## 5.3: Compare results with popular whiteboard and method of improvement

### 5.3.1: Compare result with popular whiteboard

#### *Description:*

In this section, I will once again compare the important functions of a whiteboard against other popular tools, and evaluate the functionality in terms of user experience.

#### *A. How to create a whiteboard's room and members room:*

In this function, after looking through popular whiteboards (micro.com, whiteboard.fi, tutorialspoint) I realized that users can access from a URL which is more convenient and fast compared to other methods. So, I provide this method in my project, and also provide an additional method to add and search for a user by display name in case that person is already logged into the system (**Figure 4. 3: Add members description**). However, if I have more time, I will add more functions such as member management (delete, ban, block, etc.), room management, etc. to bring the best experience to users.

#### *B. Tools*

##### **Interface:**

As mentioned in **Figure 5. 4: Use tools**, with the aim of designing a simple interface that feels modern and accessible to users. I've tried to add the full basic functionality of a composite whiteboard from the three popular sources above. My whiteboard has been designed with basic functions like: observe, select tools, draw lines, draw rectangles, draw circles, draw freely, write text, add images. In addition, the options and actions of each tool will be displayed according to the currently selected tool making the application more interactive (**Figure 5. 5: example of rectangles Actions and rectangles Options**).

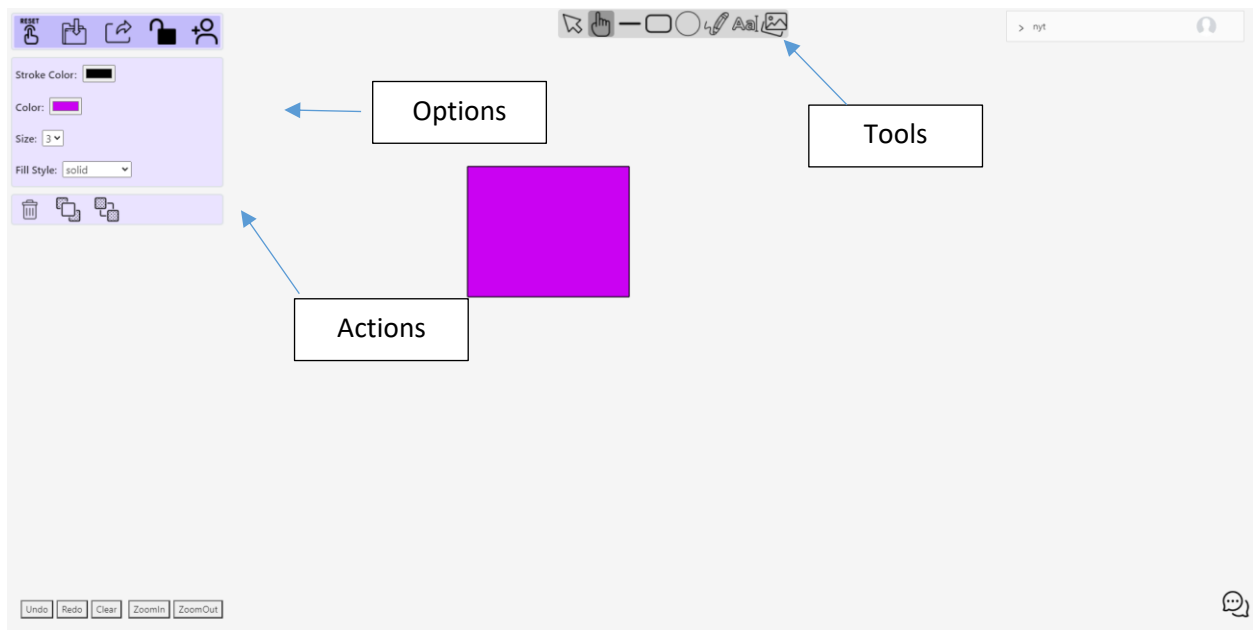


Figure 5. 11: Use tools interface

#### Functions:

The functions of the tools are designed completely similar to the three popular whiteboards above. Most functions have basic options such as color, size. The actions are deleting, bring to front, bring to back. However, compared to other whiteboards the choices are still quite poor, so in the future I hope to be able to add more functions.

#### C. Working with objects

The operations on an object are designed by me such as moving an element, resizing an element, changing properties, deleting objects, bring object to back, bring object to front. Since this is a personal project, I wanted to implement everything I could manually, but to make a project more professional and shorten design and testing time. I suggest using pre-designed and tested technologies. This part I will cover in the next section.

#### D. Communication

As with other whiteboard aids, communication may not be an important feature, but it provides powerful user interaction. That's also why micro or tutorialpoints include this tool in their application. My communication tool is designed to be simple with a chat box interface that can expand and narrow the scope of use, which gives my application freer space. Except for video chat, chat features are quite similar to other apps.

#### E. Upload and download file

In this section, most whiteboards have exported files as images and as files. But in my opinion, exporting the file as an image is not necessary, because most devices from Windows to macOS, or mobile can get the image from the operating system support quickly and conveniently. It is the reason that my project does not have an export as image section. My upload and download is designed around handling all elements generated during drawing as a JSON string (I implemented at:

**4.3.6: Import/Export file** and **Figure 5. 7: General functions**). That makes reuse of drawn elements quick and convenient. Compared to micro.com, upload and download functionality is more widely implemented, with more formats supported. However, it still meets the needs of a simple application.

Features	Miro	Whiteboard.fi	Tutorialspoint	My project
<b>Accessibility</b>				
Registration required	Yes	No	No	Yes
Free	No	Yes	Yes	Yes
<b>Tools</b>				
Working with objects	Yes	No	Yes	Yes
Connecting objects with arrows	Yes	No	No	No
Writing text	Yes	Yes	Yes	Yes
Freehand drawing	Yes	Yes	Yes	Yes
Drawing shape	Yes	No	Yes	Yes
Insert picture	Yes	No	Yes	Yes
Insert table	Yes	No	No	No
Insert chart	Yes	No	No	No
<b>Other features</b>				
Templates	Yes	No	Yes	No
Chat	Yes	No	Yes	Yes
Video Chat	Yes	No	Yes	No
Visible user cursors	Yes	No	No	No
History of events	Yes	Yes	Yes	Yes
Shape recognition	Yes	Yes	No	No
Import and export as file	Yes	No	No	Yes
Export as images	Yes	Yes	Yes	No

*Table 2: Comparison project with another whiteboard*

Overall, looking at the above table, we see some functions that are not yet complete and much need to be brought to the project to be able to compete with other whiteboards. For example, showing all users' cursors will provide a better interactive feel, adding shape recognition features to

help users work faster and so on. However, I hope with the current functions enough to bring a good feeling of experience to the users and complete the thesis well this time.

### 5.3.2: Improvement

I really admit that my project will have a lot of shortcomings. I started my project in react language, because I am new to this language and I admit that I have a hard time trying to reach them in a very short period of time (several months). So, I once again hope that you can forgive my omissions and mistakes. In the process I also know what I am lacking and what to improve. Here is a list of things that need improvement in my project:

- Most of the methods used in my project are **useState** [17]:

This is logically reasonable but not very efficient, for example when the **useState** changes, the application lifecycle repeats causing the program to lead to many unnecessary iterations, which can also lead to delay or even error. So the improvement suggestion here that I would like to present is to use **useReducer** [18]. **useReducer** is often preferable to **useState**, when we have complex state logic involving many sub values, or when the next state depends on the previous state. **useReducer** also allows you to optimize performance for components that trigger deep updates because **useReducer** contains **dispatches** that are used to trigger specific states without having to re-render the entire program. So, I think **useReducer** is perfectly reasonable in this case, in the near future as time allows, I will improve my whole application from **useState** to **useReducer**.

- Some features do not really work effectively:

Most of the features I use in the project are my thinking and documentation on the open document, and then I put everything together manually. Therefore, my application will have some algorithms that are not really optimal or have errors in the execution. I hope that can be accepted and ignored. I know that for an application to be perfect it has to go through a rigorous testing process, and I'm really sorry that my app doesn't go through this process. Most of the testing is done by myself; that is a shortcoming. So, to solve this problem, I hope readers can consider using the built-in APIs on the canvas or library resources that support implementing drawing applications, which of course have been recognized by the community and are undergoing testing. Here are some sample API sources:

- Konva: is an HTML5 Canvas JavaScript framework that extends the 2d context by enabling canvas interactivity for desktop and mobile applications. Konva enables high performance animations, transitions, node nesting, layering, filtering, caching, event handling for desktop and mobile applications, and much more. You can draw things onto the stage, add event listeners to them, move them, scale them, and rotate them independently from other shapes to support high performance animations, even if your application uses thousands of shapes. Documentation here: [19]



- Excalidraw: is a recently developed whiteboard application; it is also an open library to assist programmers that can be directly embedded in their projects. That helps programmers access and deploy projects more quickly and easily. Documentation here: [20]

## Chapter 6: Conclusion

The study covers aspects related to the architectural design and implementation of a whiteboard system; it also includes comparisons with other real-world systems. It gives us a clearer view of how a system is developed and implemented. In addition, we also look at the user side, giving users a convenient and interactive experience that is good enough. As we have seen, current and future educational models will develop towards integrating both online and offline. Especially in 2020 is an unforgettable year when the whole world is affected by the coronavirus pandemic 2019-2020. From there, it is the motivation for us to find new approaches and adapt to life to overcome the current difficult situation.

After a while of research, I have developed a whiteboard to be shared online. Whiteboard allows users to securely share content with other users by means of a link, all content secured with a password and encryption.

In the process of making the project due to time constraints and practical conditions, I know that my project has many shortcomings. But I hope my project will be a small contribution to society; it could be in terms of a reference; it could be an idea, or even someone could learn from my shortcomings.

The main objectives in my project:

- ✓ Develop an online shared whiteboard
- ✓ Study the similar apps and compare them.
- ✓ Develop, architect, and design of own application.
- ✓ Test the functionality and compare it to the other tools.

### How to install project for developer?

1. Download or clone this project from GitHub or provided by the developer to your computer:  
<https://github.com/tuanhuytran97/WhiteBoard>
2. Move into folder and execute the command to install node\_modules:  
`npm install`
3. Move to my-app folder and install node\_modules:  
`cd .\my-app`  
`npm install`
4. Start project  
`npm start`

Try it on: <https://react-firebase-draw.firebaseio.com/board>

## References

- [1] Whiteboard.fi, "Free online whiteboard tool for teachers and classrooms!," [Online]. Available: <https://whiteboard.fi/>.
- [2] Miro, "Online whiteboard for visual collaboration," [Online]. Available: <https://miro.com/app/dashboard/>.
- [3] Tutorialspoint, "Free online whiteboard," [Online]. Available: <https://www.tutorialspoint.com/whiteboard.htm>.
- [4] Mozilla, "Canvas API," [Online]. Available: [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API).
- [5] Roughjs, "Create graphics with a hand-drawn, sketchy, appearance," [Online]. Available: <https://roughjs.com/>.
- [6] Npm, "Perfect-freehand," [Online]. Available: <https://www.npmjs.com/package/perfect-freehand>.
- [7] Git, "Git-scm," [Online]. Available: <https://git-scm.com/>.
- [8] Firebase, "Firebase Documentation," [Online]. Available: <https://firebase.google.com/docs>.
- [9] Reactjs, "Introducing hooks," [Online]. Available: <https://reactjs.org/docs/hooks-intro.html>.
- [10] Mozilla, "MDN Web doc - MouseEvent," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>.
- [11] Reactjs, "Declarative routing for react," [Online]. Available: <https://v5.reactrouter.com/web/guides/quick-start>.
- [12] Mozilla, "MDN web docs - Atob," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/atob>.
- [13] Mozilla, "MDN web docs - btoa," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/btoa>.
- [14] Reactjs, "React context," [Online]. Available: <https://reactjs.org/docs/context.html>.

- [15] Firebase, "Firebase Authentication," [Online]. Available:  
<https://firebase.google.com/docs/auth>.
- [16] Firebase, "Get realtime updates with Cloud Firestore," [Online]. Available:  
<https://firebase.google.com/docs/firestore/query-data/listen>.
- [17] Reactjs, "Hooks API reference - useState," [Online]. Available: <https://reactjs.org/docs/hooks-reference.html#usestate>.
- [18] Reactjs, "Hooks API reference - useReducer," [Online]. Available:  
<https://reactjs.org/docs/hooks-reference.html#usereducer>.
- [19] A. Lavrenov, "Starting with konva. Konva.js - JavaScript 2d Canvas Library," 21 May 2021.  
[Online]. Available: <https://konvajs.org/docs/index.html>.
- [20] Npm, "Excalidraw - Virtual whiteboard," [Online]. Available:  
<https://www.npmjs.com/package/@excalidraw/excalidraw>.