VIETNAM NATIONAL UNIVERSITY – HO CHI MINH
INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

FINAL REPORT

WEB APPLICATION DEVELOPMENT

Course by
Assoc. Prof. Nguyen Van Sinh
MSc. Nguyen Trung Nghia

Topic:

Sketcha - Collaborative Whiteboard Platform for Modern Teams

Member:

Trần Ngọc Đăng Khôi	ITCSIU21197
Trần Phương Quang Huy	ITCSIU21071

Ho Chi Minh, June 16, 2024

TABLE OF CONTENT

Acknowledgements

Full Name	**Student ID**	**Role**	**Contribution**
Trần Ngọc Đăng Khôi	ITCSIU21197	Leader	100%
Trần Phương Quang Huy	ITCSIU21071	Member	100%

Abstract

Sketcha – The Whiteboard project is designed to create a powerful platform for working, brainstorming and visualizing your idea into reality.

This report delves into the development, functionality, and user-centric design of Sketcha, highlighting its unique features and competitive advantages. Sketcha offers a seamless digital canvas for brainstorming, project planning, and remote teamwork, integrating robust tools for drawing, diagramming, and note-taking. The platform emphasizes user-friendly interaction, real-time collaboration, and cross-device compatibility, ensuring accessibility and efficiency for diverse teams. Through comprehensive testing and user feedback, Sketcha has been refined to meet the dynamic needs of modern work environments, promoting productivity and fostering innovation.

This paper provides an in-depth analysis of Sketcha's architecture, technical challenges, and future development prospects, positioning it as a leading tool in the digital collaboration space.

CHAPTER 1: Introduction

1.1. Background:

In the era of digital transformation, the way teams collaborate and brainstorm has significantly evolved. Traditional whiteboard sessions, once confined to physical meeting rooms, have transitioned to virtual environments to accommodate the needs of remote and distributed teams. However, existing digital whiteboard solutions often fall short in addressing specific user needs and preferences comprehensively. Sketcha is designed to fill this gap, providing a versatile and intuitive digital whiteboard platform aimed at fostering creativity, enhancing productvity, and streamlining collaborative processes.

1.2. Problem Statement:

Despite the availability of digital whiteboard solutions, many platforms fail to fully meet the diverse requirements of modern teams. Common issues include complex user interfaces, limited customization options, insufficient real-time collaboration capabilities, and compatibility problems across different devices. These shortcomings can hinder the effectiveness of brainstorming sessions, project planning, and other collaborative activities, leading to reduced productivity and creativity. Therefore, there is a critical need for a more user-friendly, flexible, and efficient whiteboard platform that can seamlessly integrate into various workflows and enhance team collaboration.

1.3. Scope and Objective:

The primary objective of Sketcha is to develop a digital whiteboard platform that addresses the limitations of existing solutions and provides an enhanced user experience. The scope of this project includes:

- Designing an intuitive and easy-to-navigate user interface.
- Implementing robust tools for drawing, diagramming, and note-taking.
- Enabling real-time collaboration with minimal latency.
- Ensuring cross-device compatibility for seamless access on desktops, tablets, and smartphones.
- Incorporating user feedback to continuously improve and refine the platform.

By achieving these objectives, Sketcha aims to become a preferred tool for teams seeking a reliable and efficient solution for their collaborative needs.

1.4. Assumption and Solution:

The development of Sketcha is based on several key assumptions:

- Users require a digital whiteboard that is easy to use and does not require extensive training.
- Effective collaboration depends on real-time interaction and immediate feedback.
- A flexible and customizable platform can better cater to the diverse needs of different teams and projects.
- Cross-device compatibility is essential for ensuring that users can access the platform from various environments.

To address these assumptions, Sketcha offers a comprehensive solution with the following features:

- A clean and intuitive user interface that simplifies navigation and tool usage.
- Advanced real-time collaboration capabilities, allowing multiple users to work simultaneously with minimal lag.
- A wide range of customizable tools and templates to suit different types of projects and workflows.
- Full compatibility with major operating systems and devices, ensuring that users can seamlessly switch between different platforms.

Through these solutions, Sketcha aims to overcome the common challenges faced by users of existing digital whiteboard platforms and provide a superior collaborative experience.

CHAPTER 2: Methodology

2.1. Project scope:

2.1.1. Description:

This project focuses on enhancing the collaborative experience for users by providing powerful, feature-rich digital whiteboard platform. The platform enables users to draw, write and interact in real-time, facilitating seamless collaboration for educational, professional, or personal purposes. By integrating various tools and features, the platform aims to create an intuitive and effective environment for users to express their ideas and work together.

2.1.2. Objectives:

The primary objectives of the Sketcha – collaborative platform are:

- To provide a user-friendly and intuitive interace that allows users to create and share whiteboard sessions easily.
- To enable real-time collaboration with ninimal latency, ensuring a smooth user experience.
- To offer a variety of tools and features, such as drawing, text input, and multimedia integration, enhancing the collaborative process.
- To ensure data persistence and security, allowing users to save and access their sessions securely.
- To support scalability, accommodating a growing number of users and sessions without compromising performance.

2.2. User requirement analysis:

2.2.1. Functional requirements:

- User Registration and Authentication: Users must be able to register and log in securely.
- Session Management: Users should be able to create, join, and manage whiteboard sessions.
- Real-time Collaboration: Multiple users should be able to interact on the same whiteboard in real-time.
- Drawing Tools: Users should have access to various drawing tools, including pens, brushes, shapes, and erasers.
- Text Input: Users should be able to add and edit text on the whiteboard.
- Multimedia Support: The platform should allow the integration of images and other multimedia elements.
- Undo/Redo Functionality: Users should be able to undo and redo their actions.

- Save and Load Sessions: Users should be able to save their whiteboard sessions and load them later.

2.2.2.Non-functional requirements:

- Performance: The platform should have minimal latency to support real-time collaboration.
- Scalability: The system should handle an increasing number of users and sessions efficiently.
- Security: User data and sessions should be secure, with proper authentication and authorization mechanisms.
- Usability: The interface should be intuitive and easy to use for all user types.
- Reliability: The system should be reliable, with minimal downtime and robust error handling.
- Compatibility: The platform should be accessible on various devices and browsers.

2.3. System Design:

2.3.1. System Architecture:

The Sketcha-Whiteboard Platform follows a client-server architecture, where the client handles the user interface and interactions, while the server manages the backend processes, including data storage, user authentication, and real-time collaboration features.

a. **Client:** Developed using React library with Next.js framework, the client application provides the user interface and handles user interaction. Addtionally, Liveblocks framework is implemented to supply the workplace with real-time interaction by user presences and drawing, texting tools.

b. **Server:** Built with Node.js and Express, the server manages API requests, user authentication, session management, and real-time communication using Liveblocks. Moreover, some of server architectures are applied:

❖ **Model-View-Controller (MVC):** is employed to separate the application into three interconnected components

- **Model**: Represents the data and the business logic of the application. In the Sketcha-Whiteboard Platform, the model includes the data structures for users, sessions, whiteboards, and messages.
- **View**: The user interface that displays the data to the user. In this platform, the React components serve as the view, rendering the whiteboard, chat panel, and other UI elements.
- **Controller**: Handles user input and interacts with the model to update the view. In this platform, the controller logic is embedded in the client-side React components and the server-side API routes.

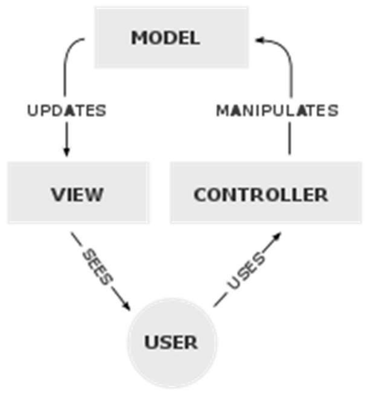


Figure 4.1: MVC Model

❖ **REST** (**representational state transfer or RESTful API**): a software architectural style designed for the World Wide Web, emphasizing uniform interfaces, independent component deployment, scalable interactions, layered architecture for caching, security enforcement, and legacy system encapsulation.

2.3.2.Technologies:

a. Client:

- **Next.js:** is used for server-side rendering and static site generation. It provides a robust structure for developing the client side of the application.
- **Tailwind CSS for Styling:** is used for styling the application. It provides utility-first CSS classes that help in designing responsive and modern UI elements quickly.
- **Axios for HTTP Requests:** is utilized for making HTTP requests to the server. It simplifies the process of sending asynchronous HTTP requests and handling responses.

b. Server:

- **Node.js and Express.js:** is used to build the server-side logic and API endpoints.
- **MongoDB with Mongoose for Database:** is used for data persistence, and Mongoose is used for defining schemas and interacting with the database.
- **Clerk for Authentication and Authorization:** Clerk handles user authentication and authorization, providing secure access control to the application's resources.
- **Liveblocks for Real-time Collaboration:** is integrated to enable real-time updates and collaboration on whiteboards.

These technologies collectively form the foundation of the WhiteBoard platform, enabling a rich, real-time collaborative drawing experience with robust back-end support for data persistence and user management.

2.3.3. User goal use case:

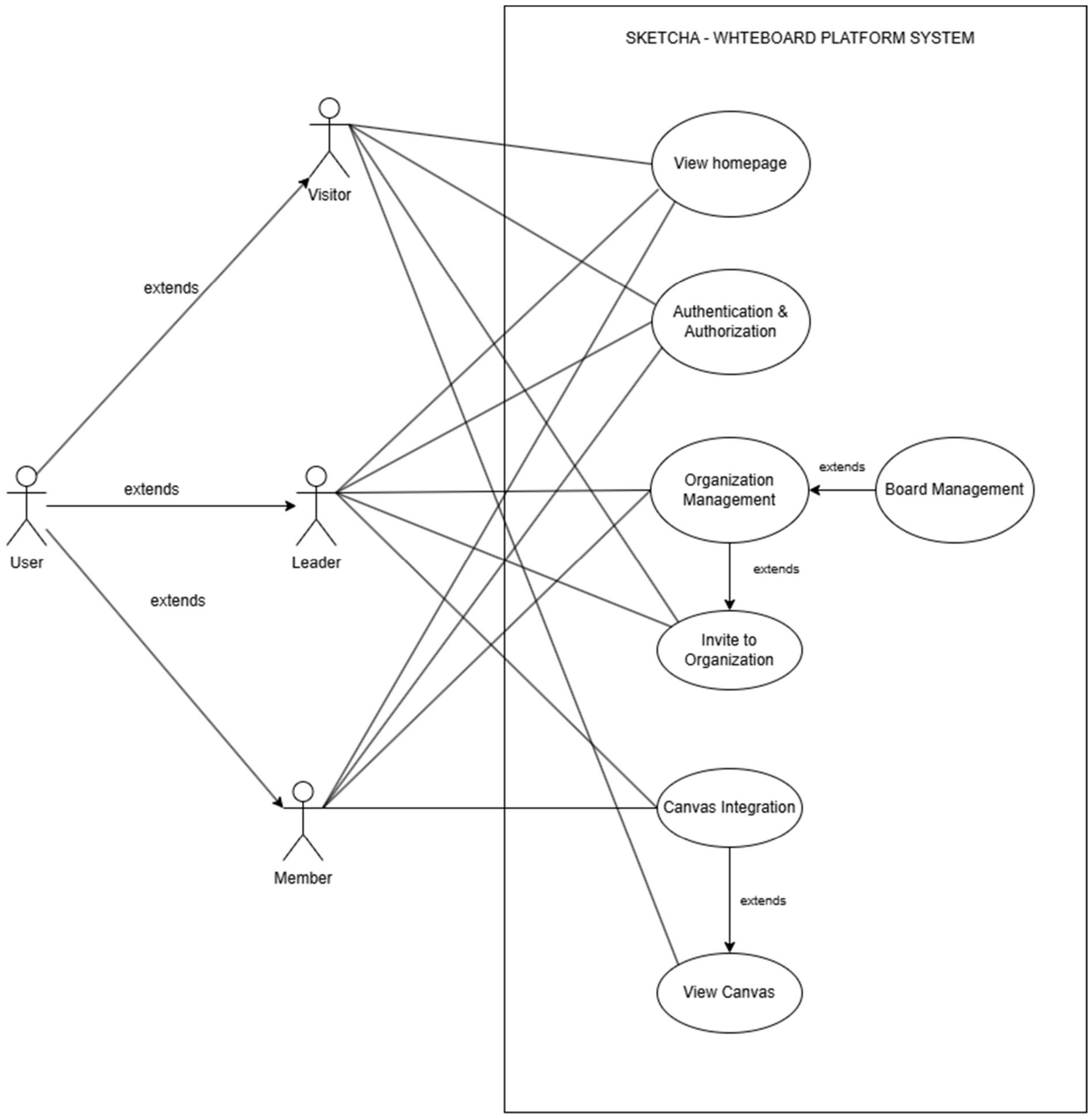


Figure. Use goal use case of ‘Sketcha’ system.

In the Sketcha - Whiteboard Platform System, various actors interact with the system to achieve different goals. The primary actors include Visitors, Users, Leaders, and Members. Each actor has specific use cases that represent their interactions with the system.

❖ **Actors:**

- **User**: A general platform user who can authenticate, access basic functionalities, and potentially become a Member or Leader. Otherwise, just simply as a visitor:

- **Visitor**: Someone who visits the platform but has not logged in. They can view the homepage and initiate the authentication process.
- **Leader**: A user with administrative privileges, capable of managing the organization, inviting members, and overseeing board management.
- **Member**: A user who has been invited to join an organization and can view and interact with canvases.

❖ **Key use cases:**

- **View Homepage**: This use case allows Visitors to access and view the homepage of the platform.
- **Authentication & Authorization**: This essential use case enables Users to log in and gain authorized access to the platform's features.
- **Organization Management**: Leaders manage organizational settings and oversee various administrative tasks.
- **Invite to Organization**: Leaders can invite Users to join the organization, enhancing collaborative efforts.
- **Canvas Integration**: Users and Members interact with integrated canvas features for collaborative whiteboarding.
- **View Canvas**: Members can view and engage with canvases within the platform.
- **Board Management**: An extension of the Organization Management use case, allowing Leaders to manage specific boards within the organization.

2.3.4. Database design:

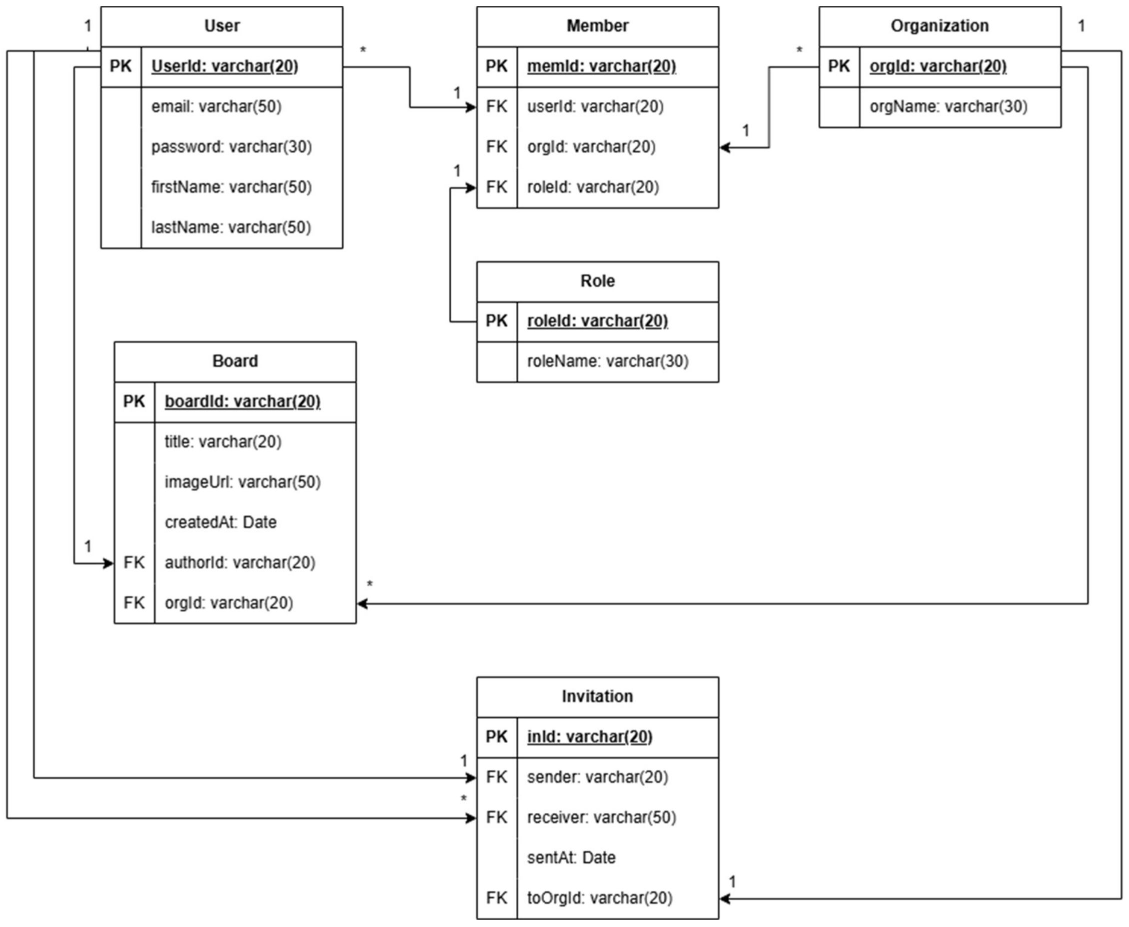


Figure. Entity Relational Diagram of 'Whiteboard Platform'

Table Name	Description
User	Stores information about users, including their login credentials and personal.
Member	Manages the relationship between users and organizations, including the roles users have within those organizations.
Organization	Contains details about each organization, such as its name.
Role	Defines the responsibility of specific user in that organization.
Board	Holds information about boards created by users, including the title, thumbnail URL, creation date, author, and organization to which the board belongs.
Invitation	Manages invitations sent to user to join organizations, including sender, receivers, and the date the invitation was sent.

Figure. Table of Entity Relationship Description.

Overall, the database design is robust, ensuring clear relationships and data integrity across the various entitie, which additionally satisfies:

- Normalization: The database design appears to be normalized in Third-form Normalization, minimizing redundancy and ensuring efficient data management.
- Referential Integrity: Foreign keys are appropriately used to maintain referential integrity between related entities.
- Scalability: The design supports scalability, allowing users to be part of multiple organizations, have multiple roles, and create multiple boards.
- Role Management: The inclusion of a `Role` entity supports flexible role management and access control within organizations.
- Invitation Handling: The `Invitation` entity provides a structured way to manage organizational invitations, enhancing collaboration.

2.3.5.Activity Diagram:

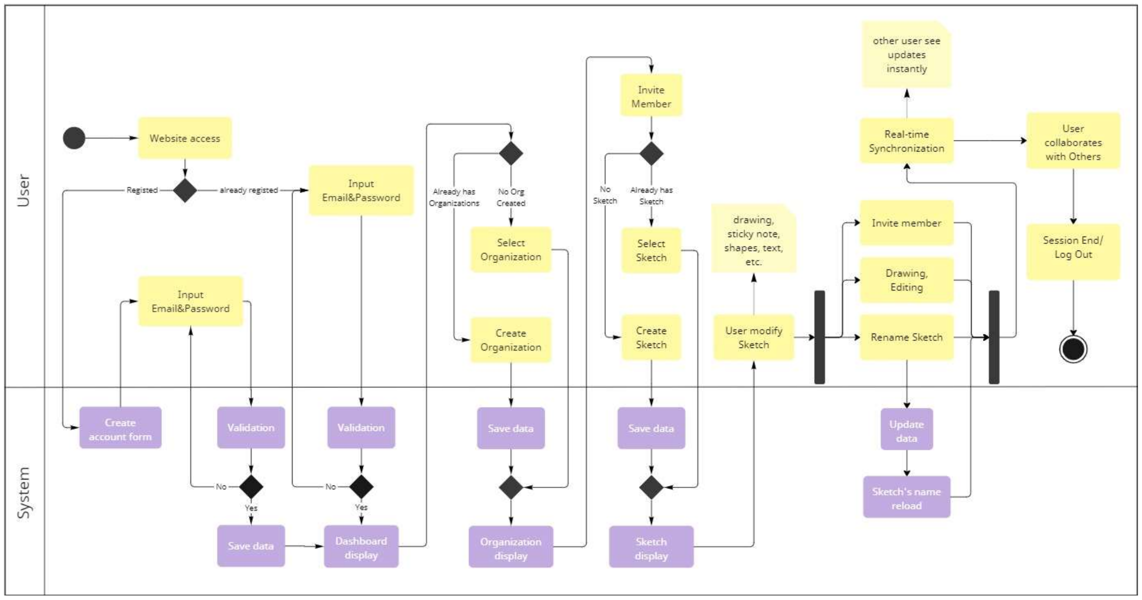


Figure. Activity Diagram of 'Whiteboard-Platform'

The activity diagram outlines the primary workflow for users interacting with a collaborative sketching application. It highlights the processes involved from initial access to ongoing collaboration, which includes:

- **User Registration and Login**: Users can either register for a new account or log in with existing credentials. The system validates inputs and grants access accordingly.
- **Organization Management**: Once logged in, users can create or select an organization to work within. The system manages and displays organizational data.

- **Sketch Creation and Selection**: Users have the option to create new sketches or select existing ones within their organization. The system saves and displays these sketches.
- **Collaboration and Editing**: Users can invite others to collaborate on sketches. Real-time synchronization ensures all collaborators see updates immediately. Users can draw, edit, and modify sketches collectively.
- **Session Management**: Users can end their session or log out when finished, with all data being saved by the system.

The system supports validation, data management, and real-time updates to facilitate a smooth and collaborative user experience.

2.3.6.Class Diagram:

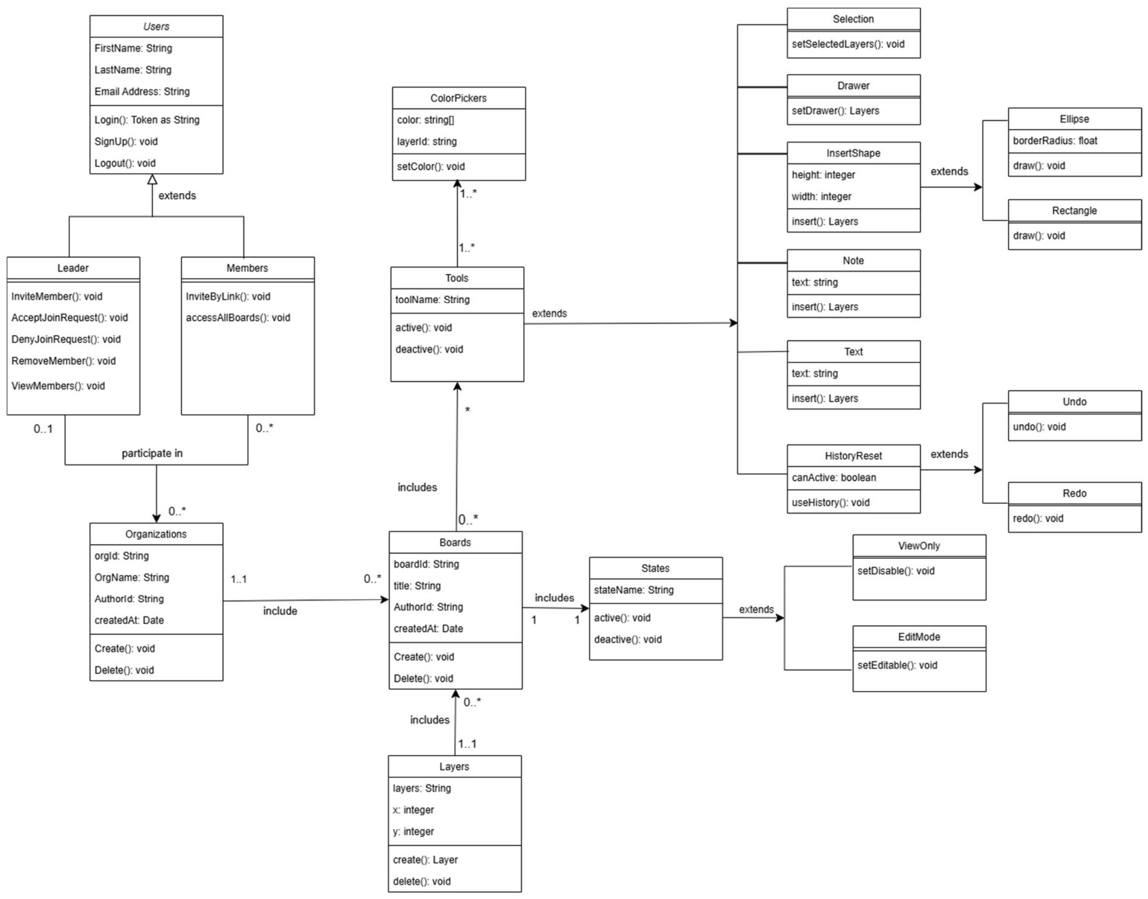


Figure. Class diagram of ‘Sketcha-Whiteboard Platform’ Project

Class Diagram Name	Description
Users	This class represents a user with attributes like first name, last name, and email address. It includes

	methods for login, sign-up, and logout functionalities.
Leader	This class extends the Users class and includes additional methods specific to leaders, such as inviting members, accepting or denying join requests, removing members, and viewing members.
Members	This class extends the Users class and includes methods for inviting by link and accessing all boards.
Organizations	This class represents an organization with attributes like organization ID, name, author ID, and creation date. It includes methods for creating and deleting organizations.
Boards	This class represents a board within an organization, with attributes like board ID, title, author ID, and creation date. It includes methods for creating and deleting boards.
Layers	This class represents layers within a board, with attributes like layer name, x and y coordinates. It includes a method for creating a layer.
ColorPickers	This class represents a color picker tool with attributes for color and layer ID. It includes a method for setting the color.
Tools	This class represents tools that can be used on boards. It includes methods for activating and deactivating tools. This class is extended by several other specific tool classes.
Selection	This class extends Tools and includes a method for setting selected layers.
Drawer	This class extends Tools and includes a method for setting the drawer for layers.
InsertShape	This class extends Tools and includes attributes for height and width, and a

	method for inserting shapes into layers.
Note	This class extends Tools and includes an attribute for text and a method for inserting notes into layers.
Text	This class extends Tools and includes an attribute for text and a method for inserting text into layers.
HistoryReset	This class extends Tools and includes an attribute for activation status and a method for using history. It is further extended by Undo and Redo classes.
Undo	This class extends HistoryReset and includes a method for undoing actions.
Redo	This class extends HistoryReset and includes a method for redoing actions.
States	This class represents the state of a board with an attribute for the state name. It includes methods for activating and deactivating the state. It is extended by ViewOnly and EditMode classes.
ViewOnly	This class extends States and includes a method for setting the state to view-only mode.
EditMode	This class extends States and includes a method for setting the state to editable mode.

Table. Class Diagram's Description Table

2.3.7. User Interface design:

The user interface (UI) design of the Sketcha-Whiteboard Platform is focused on providing an intuitive and seamless user experience. The UI is developed using React, a popular JavaScript library for building user interfaces, which allows for the creation of dynamic and responsive components. Key aspects of the UI design include:

a. Homepage/Introduction/About Us Page:

This is an intial state of my website if there is no token existing, which including all essential information related to the project, such as features, functionalities and a word from developer team.

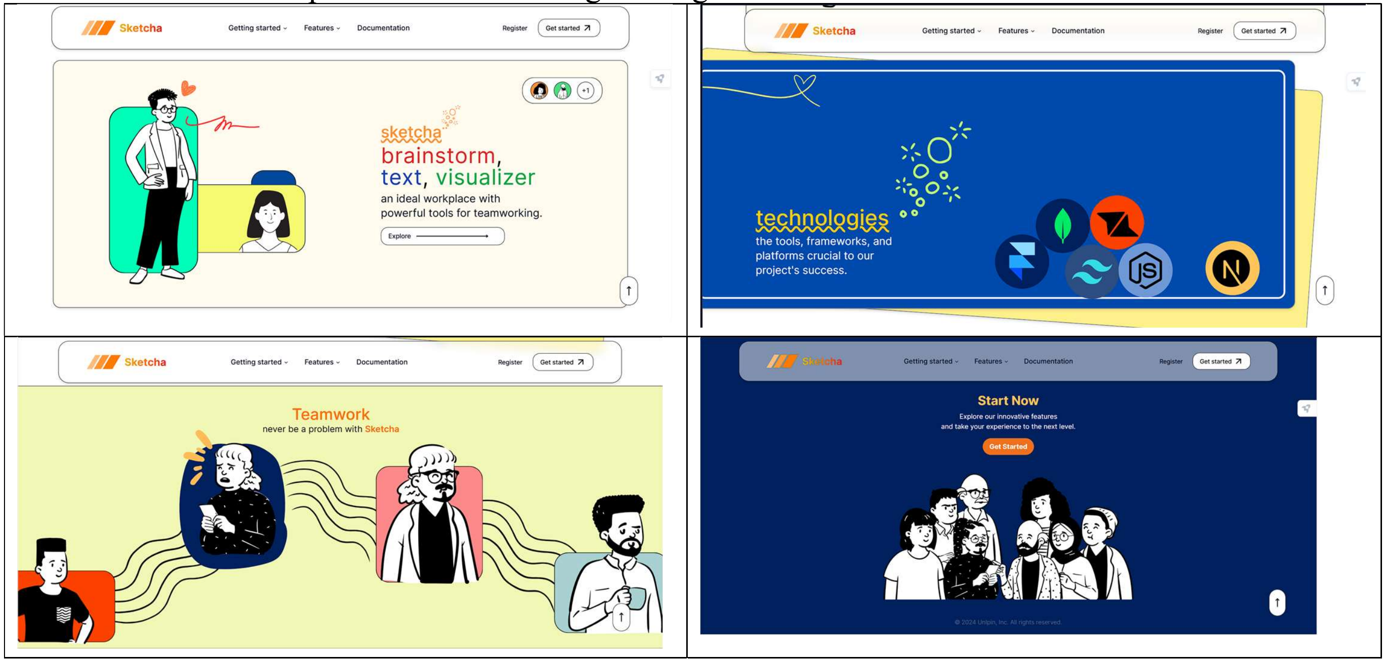
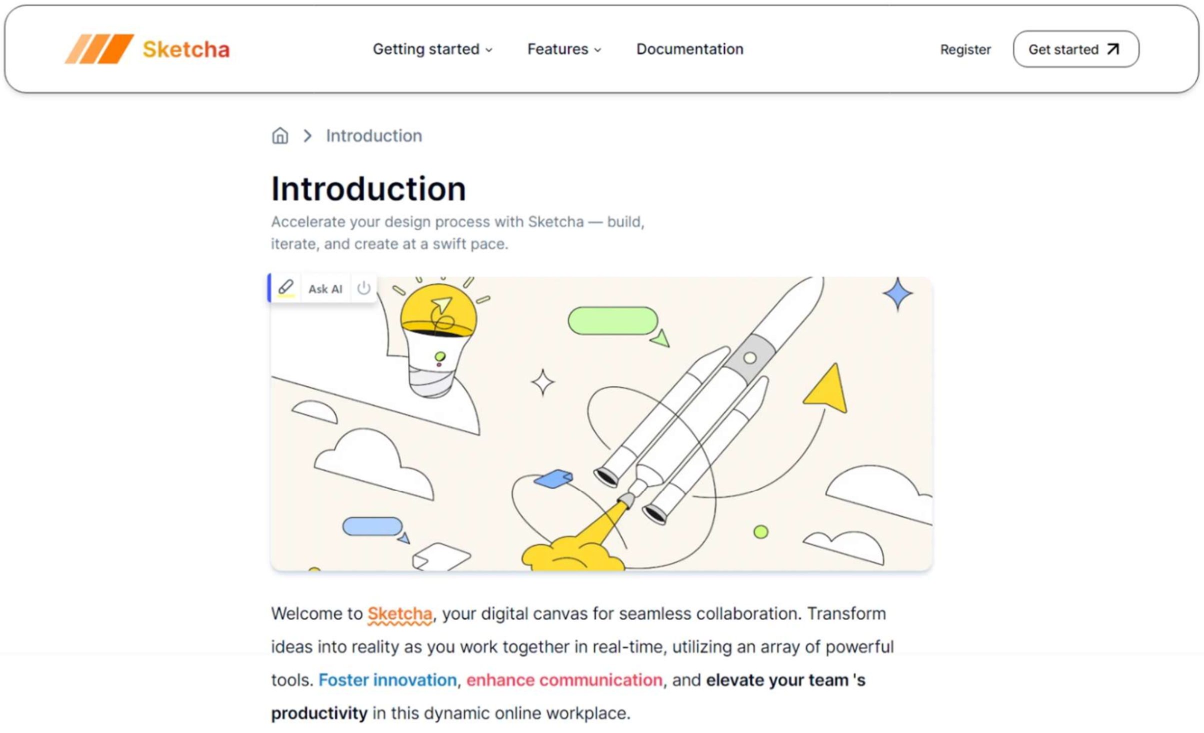

Sketcha
Getting started
Features
Documentation
Register
Get started
sketcha
brainstorm,
text, visualizer
an ideal workplace with powerful tools for teamworking.
Explore
technologies
the tools, frameworks, and platforms crucial to our project's success.
Teamwork
never be a problem with Sketcha
Start Now
Explore our innovative features and take your experience to the next level.
Get Started

Figure. Some highlights of 'Homepage' UI.


Sketcha Getting started Features Documentation Register Get started

Introduction

Introduction

Accelerate your design process with Sketcha — build, iterate, and create at a swift pace.

Ask AI

Welcome to Sketcha, your digital canvas for seamless collaboration. Transform ideas into reality as you work together in real-time, utilizing an array of powerful tools. Foster innovation, enhance communication, and **elevate your team 's productivity** in this dynamic online workplace.

Figure. UI of 'Introduction' page.

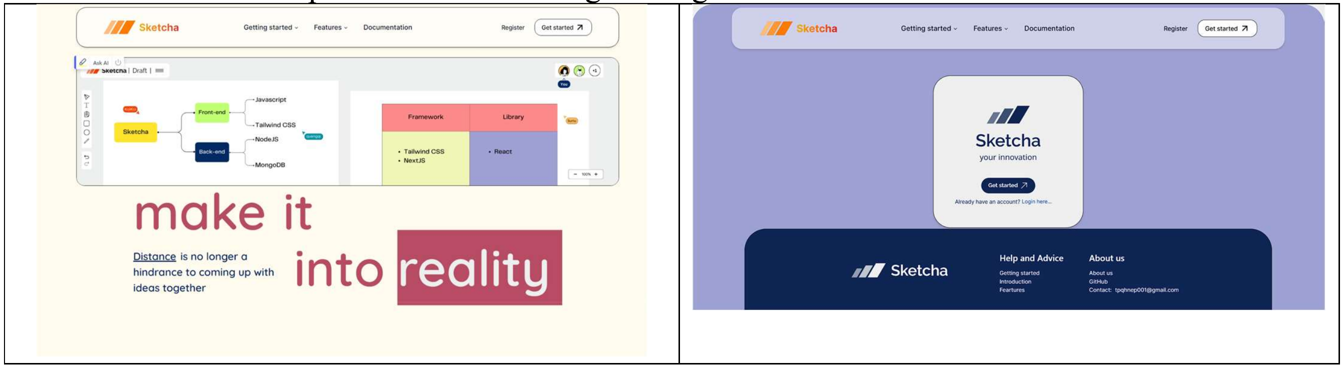
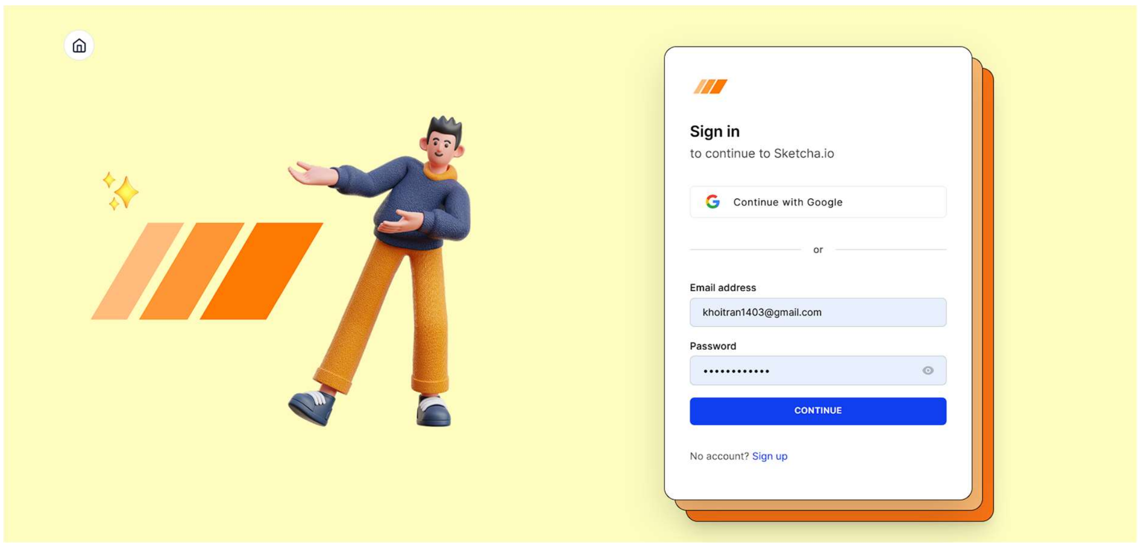


Figure. Highlight of 'About Us' UI.

b. Login/Register Page: in this page, it requires to include:

- **Login Form**: Allows existing users to enter their credentials (username/email and password) to access their accounts.



Sign in
to continue to Sketcha.io

Continue with Google

or

Email address
khoitran1403@gmail.com

Password
••••••••••••

CONTINUE

No account? Sign up

Figure. Login form's UI.

- **Registration Form**: Provides new users with a form to create an account, requiring information such as username, email, and password. Password strength indicators and validation messages enhance usability.

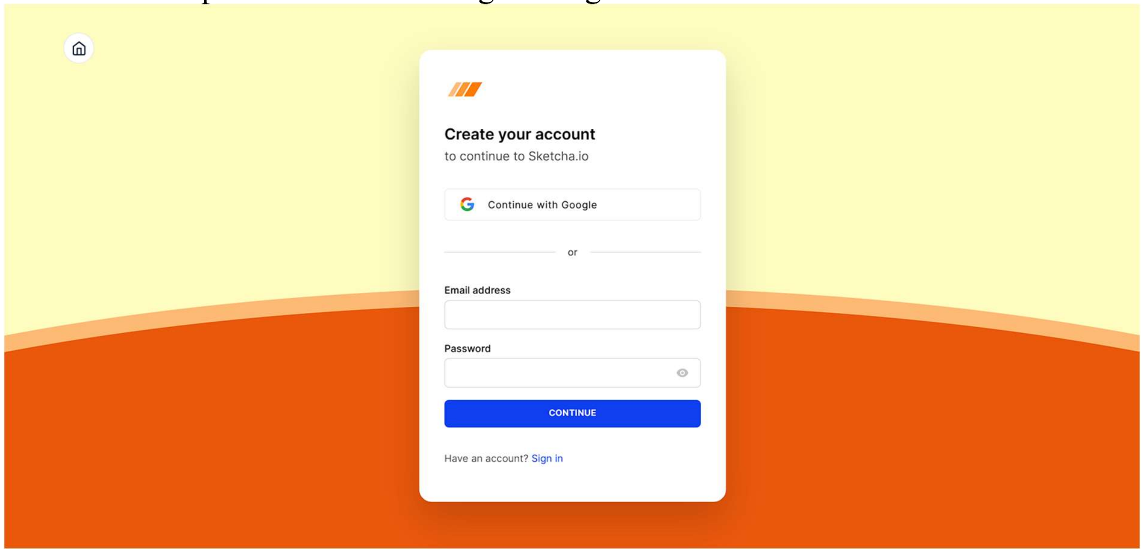


Figure. Registration form's UI.

c. Dashboard:

This section will be accessible only when the user's logged in. Therefore, it will be a main and general workplace for user that contains all the essential information of users, including workplace and list of organizations, all the boards that organization contains and configurations.

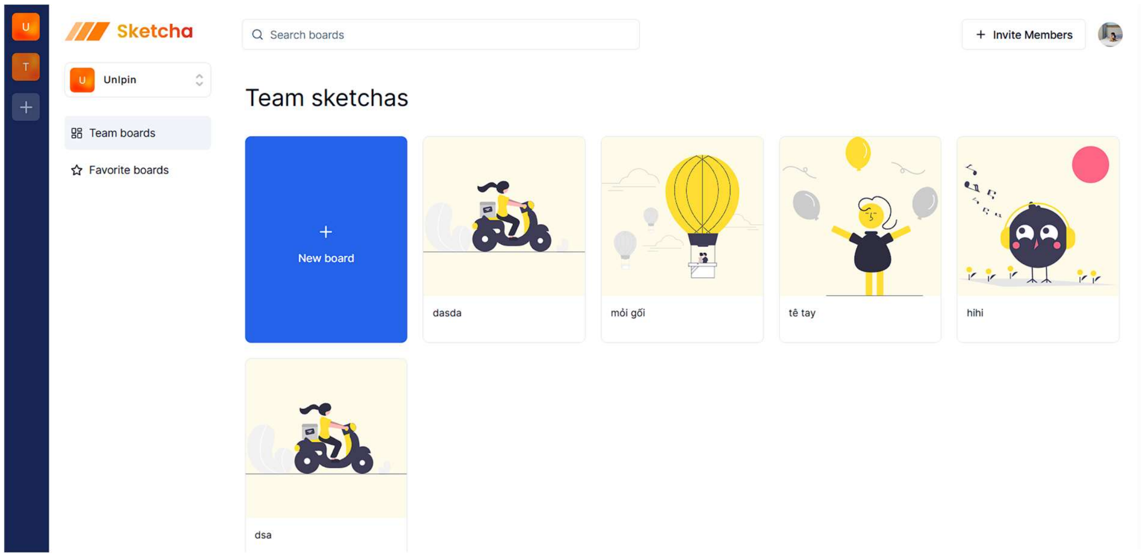
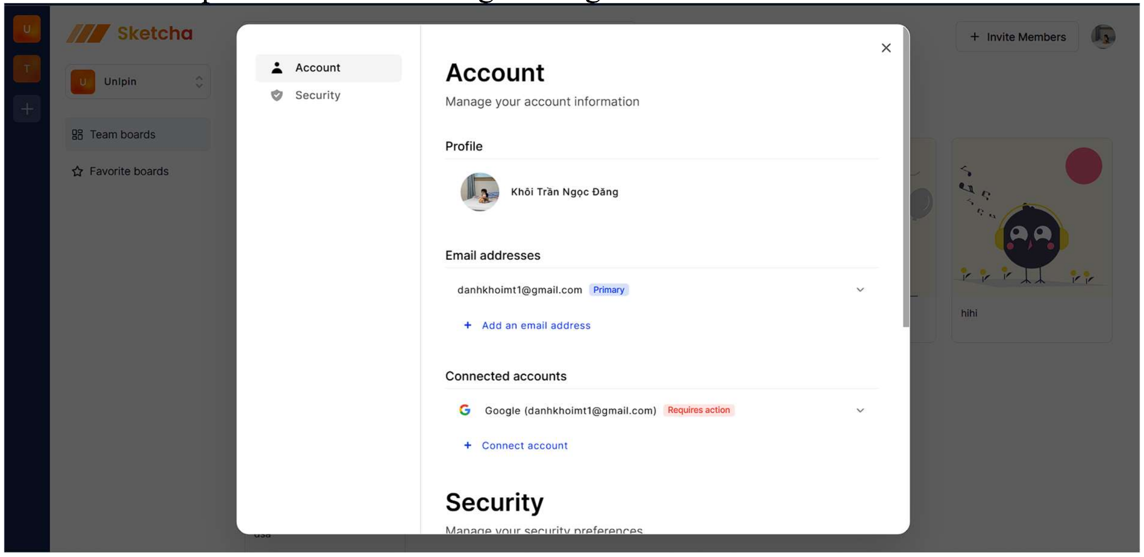


Figure. User's Dashboard

d. Settings and Preferences:



Account
Manage your account information

Profile
Khôi Trần Ngọc Đăng

Email addresses
danhkhoimt1@gmail.com Primary
+ Add an email address

Connected accounts
Google (danhkhoimt1@gmail.com) Requires action
+ Connect account

Security
Manage your security preferences

Figure. User's Settings and Preferences

e. Whiteboard Interface:

This section is the main workplace of this project, which including:

- **Canvas**: The main drawing area where users can create and interact with content. The canvas should support smooth and responsive drawing with real-time updates.
- **Toolbar**: Positioned typically on the left or top of the canvas, providing access to various drawing tools and features:

  - **Selection Tool**: Allows users to select and manipulate objects on the canvas.
  - **Drawing Tools**: Includes pen, brush, shapes (rectangle, circle, line), and eraser. Each tool can have adjustable settings such as color, size, and opacity.
  - **Text Tool**: Enables users to add and edit text on the canvas, with options for font style, size, and color.
  - **Multimedia Integration**: Buttons for uploading images, inserting videos, or other multimedia elements.
  - **Undo/Redo**: Buttons to undo or redo the last actions, providing a way to correct mistakes easily.
  - **Clear Canvas**: A 'Delete' button to clear all content from the canvas, usually with a confirmation prompt to prevent accidental use.
  - **Dropdown Action Menu**: including 'Copy Link' to invite directly into the workplace; 'Rename' the workplace; and 'Delete' button to remove the board.
  - **State of Workplace Setting**: contain two main states for canvas, whose 'View Only' for setting uneditable canvas and disable all tools, which automatically applies for visitor; and 'Edit Mode' for editable state.

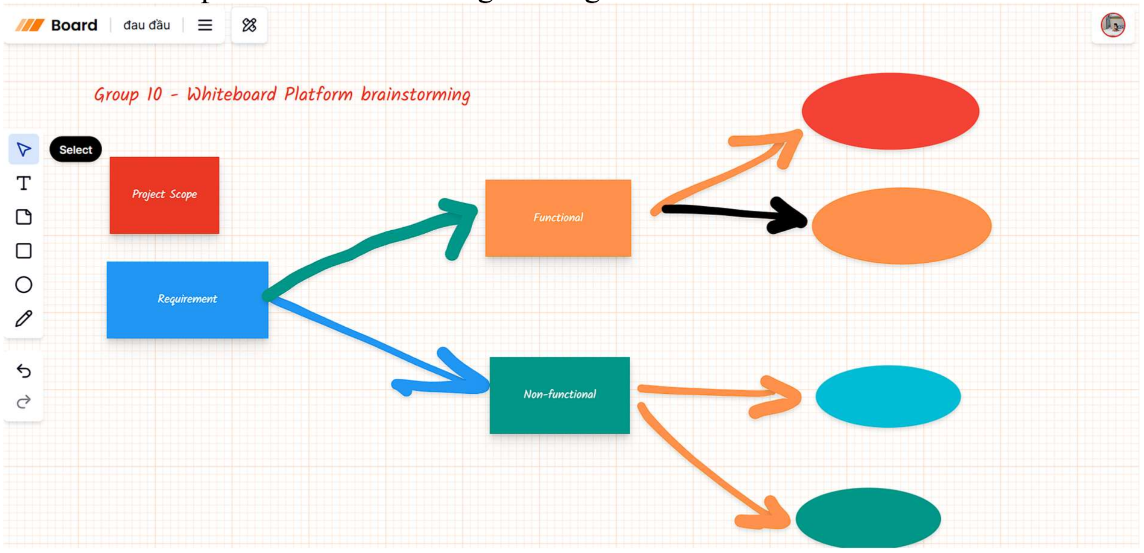


Figure. User's whiteboard workplace

f. **Responsiveness:**

- **Adaptive Layout**: The interface should adapt to different screen sizes and orientations, ensuring usability on desktops, tablets, and mobile devices.
- **Touch Support**: For mobile devices, touch gestures for drawing, panning, zooming, and tool selection should be supported.

g. **Accessibility:** which mainly focus on keyboard navigation, to ensure that all features can be accessed via keyboard shortcuts, enhancing usability for users with disabilities.

2.3.8. Design patterns:

Design patterns are crucial for building a scalable, maintainable, and robust software system. In the Sketcha-Whiteboard Platform, several design patterns are employed to address common problems and enhance the overall architecture. Here are some of the key design patterns used:

a. Singleton Pattern

The singleton pattern ensures that a class has only one instance and provides a global point of access to it. In the Sketcha-Whiteboard Platform, this pattern is used for:

- **Liveblocks Connection**: Ensuring a single Liveblocks connection per client for real-time communication. This avoids multiple connections and ensures efficient use of resources.
- **Database Connection Pool**: Managing a pool of database connections to optimize performance and resource usage.

b. Factory Pattern

The factory pattern provides a way to create objects without specifying the exact class of object that will be created. This pattern is useful for:

- **Drawing Tools**: Creating different types of drawing tools (pen, brush, shapes, eraser) based on user selection. The factory pattern helps in instantiating the appropriate tool class based on user input.
- **UI Components**: Dynamically generating UI components based on the type of content (text, image, video) to be displayed on the whiteboard.

c. Strategy Pattern

The strategy pattern defines a family of algorithms, encapsulates each algorithm, and makes them interchangeable. It allows the algorithm to vary independently from clients that use it:

- **Drawing Strategies**: Different drawing tools (e.g., pen, brush, shape tools) can be implemented as different strategies. Users can switch between tools seamlessly, and each tool uses its own drawing strategy.

d. Command Pattern:

The command pattern encapsulates a request as an object, thereby allowing for parameterization of clients with different requests, queuing of requests, and logging of the requests:

- **Undo/Redo Functionality**: Each user action (draw, erase, move) is encapsulated as a command object. These commands can be stored in a history stack to support undo and redo operations.

CHAPTER 3: Implementation

3.1. Front-end:

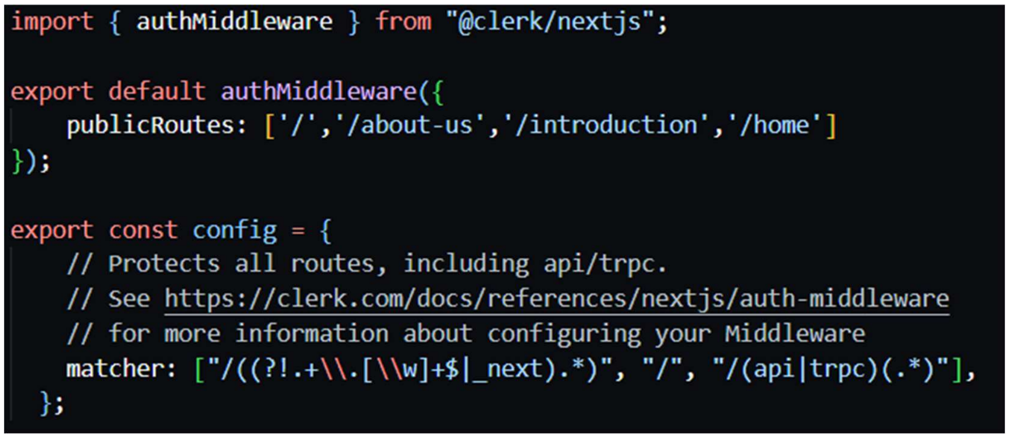
3.1.1. Authentication & Authorization:

a. Core Technology:

Clerk is a modern authentication and user management platform designed to simplify and secure the process of integrating user accounts into web applications. It provides developers with tools to easily implement features such as user sign-up, sign-in, password recovery, and more, all backed by robust security measures and compliance with industry standards. Clerk offers customizable UI components and straightforward APIs, making it suitable for various web frameworks and applications seeking reliable authentication and authorization solutions.

b. Methodology:

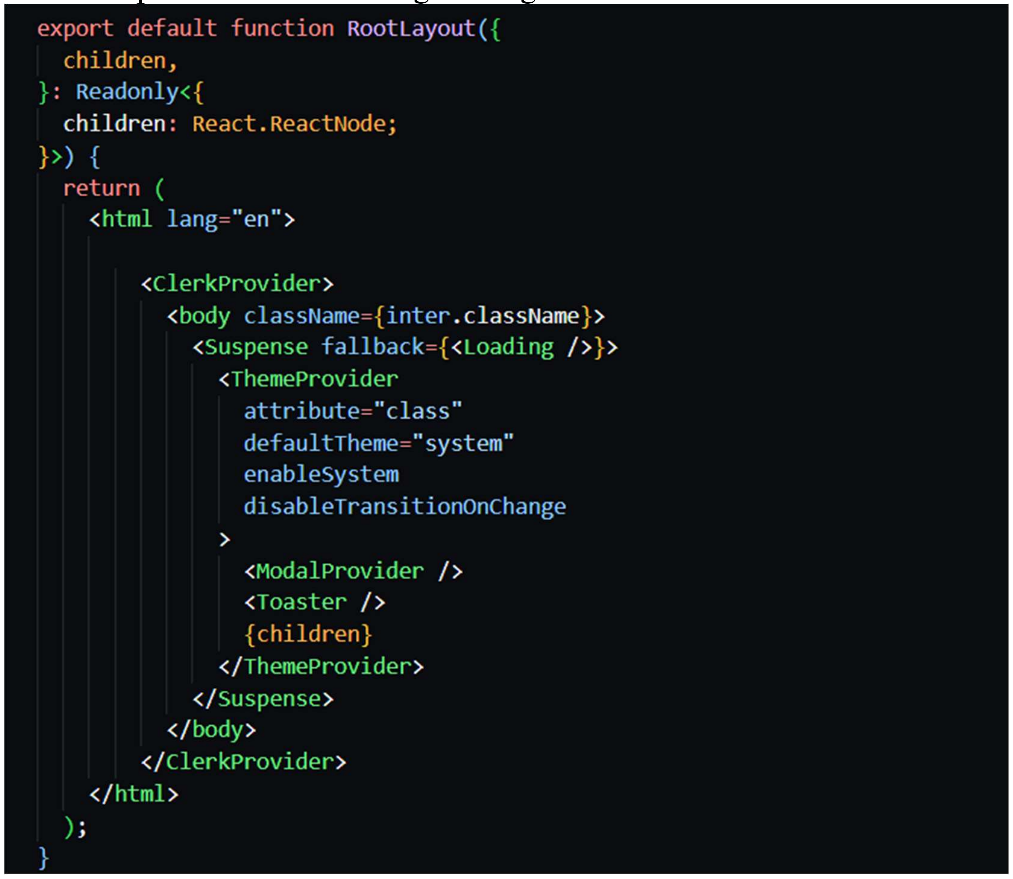
- **Middleware:** Properly configured to protect routes, ensuring only authenticated users can access restricted parts of the application. `authMiddleware` to protect routes defined in `publicRoutes` from unauthorized access. Users must authenticate to access other routes.



```
import { authMiddleware } from "@clerk/nextjs";

export default authMiddleware({
    publicRoutes: ['/','/about-us','/introduction','/home']
});

export const config = {
    // Protects all routes, including api/trpc.
    // See https://clerk.com/docs/references/nextjs/auth-middleware
    // for more information about configuring your Middleware
    matcher: ["/((?!.+\\.[\\w]+$|_next).*)", "/", "/(api|trpc)(.*)"],
  };
```

Figure. Middleware.ts

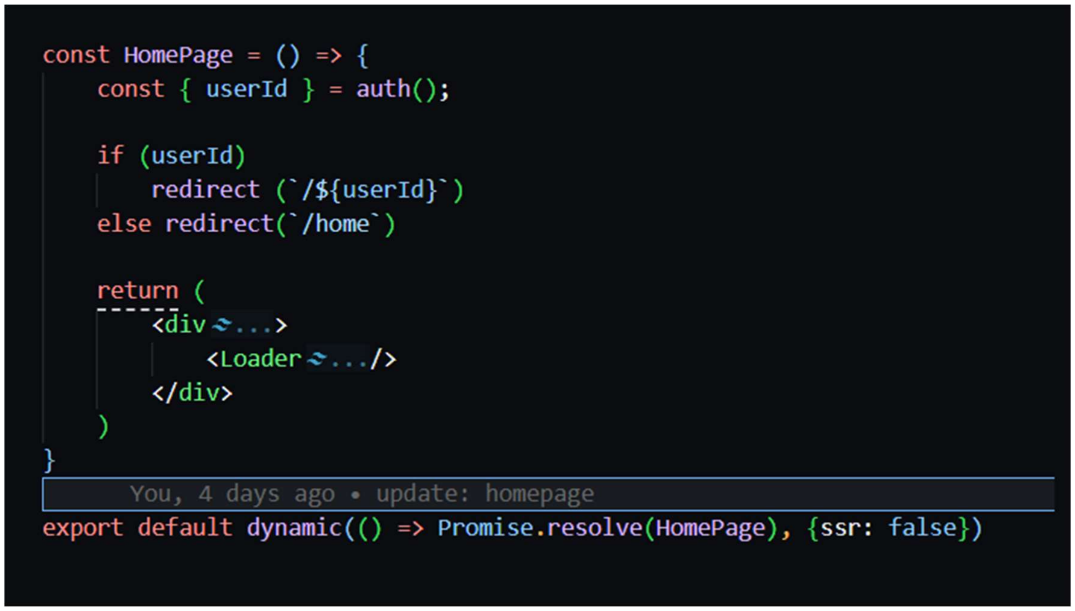
- **ClerkProvider Setup:** Base on Next.js's routing organization, we've wrapped the application with `ClerkProvider` in the global layout component (`RootLayout`), which provides the authentication context to the entire application..



```
export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">

      <ClerkProvider>
        <body className={inter.className}>
          <Suspense fallback={<Loading />}>
            <ThemeProvider
              attribute="class"
              defaultTheme="system"
              enableSystem
              disableTransitionOnChange
            >
              <ModalProvider />
              <Toaster />
              {children}
            </ThemeProvider>
          </Suspense>
        </body>
      </ClerkProvider>
    </html>
  );
}
```

Figure. Wrapping Root Layout with `ClerkProvider` protection.

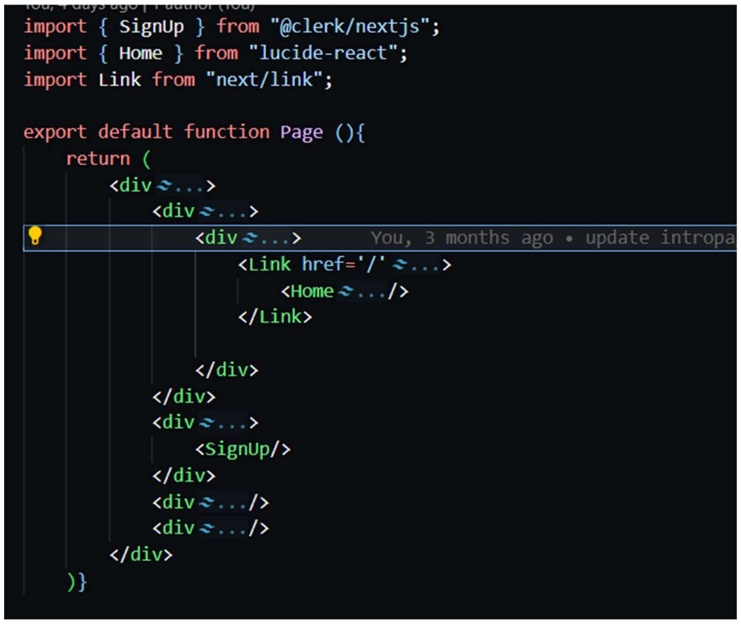
**- Authentication Check (`Homepage):** In your homepage component, you perform an authentication check using `auth()` from `@clerk/nextjs` and redirect users based on their authentication status. `auth()` is called to fetch authentication details. If `userId` exists, it redirects to `/${userId}`, otherwise to `/home`.



```
const HomePage = () => {
    const { userId } = auth();

    if (userId)
        redirect (`/${userId}`)
    else redirect(`/home`)

    return (
        <div ...>
            <Loader .../>
        </div>
    )
}
        You, 4 days ago • update: homepage
export default dynamic(() => Promise.resolve(HomePage), {ssr: false})
```

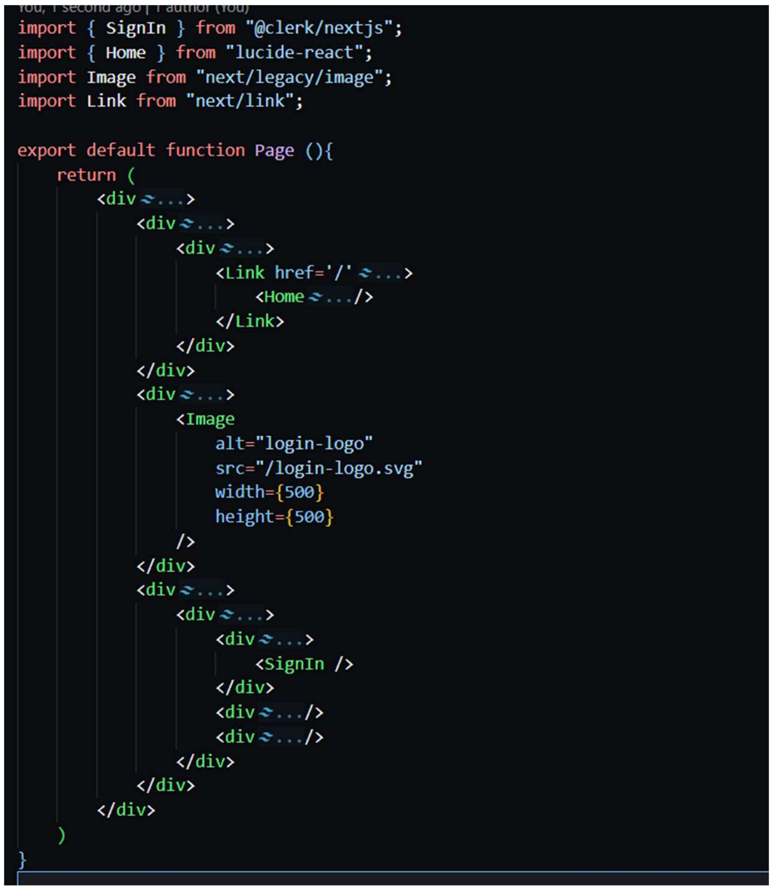
Figure. Authentication Check (`Homepage`)

- **Sign-in & Sign-up Pages:** integrated Clerk's authentication components (`SignUp` and `SignIn`) for user registration and login.



```
You, 4 days ago | 1 author (You)
import { SignUp } from "@clerk/nextjs";
import { Home } from "lucide-react";
import Link from "next/link";

export default function Page (){
    return (
        <div ...>
            <div ...>
                <div ...>        You, 3 months ago • update intropa
                    <Link href='/' ...>
                        <Home .../>
                    </Link>

                </div>
            </div>
            <div ...>
                <SignUp/>
            </div>
            <div .../>
            <div .../>
        </div>
    )}
```

Figure. SignUp Page



```
You, 1 second ago | 1 author (You)
import { SignIn } from "@clerk/nextjs";
import { Home } from "lucide-react";
import Image from "next/legacy/image";
import Link from "next/link";

export default function Page (){
    return (
        <div ...>
            <div ...>
                <div ...>
                    <Link href='/' ...>
                        <Home .../>
                    </Link>
                </div>
            </div>
            <div ...>
                <Image
                    alt="login-logo"
                    src="/login-logo.svg"
                    width={500}
                    height={500}
                />
            </div>
            <div ...>
                <div ...>
                    <div ...>
                        <SignIn />
                    </div>
                    <div .../>
                    <div .../>
                </div>
            </div>
        </div>
    )
}
```

Figure. SignIn Page

3.1.2. Dashboard Management: as a main workplace for management of all boards and relative information as users', organisation's.

a. Dashboard Page: (`DashboardPage`): couple keys of features:

+ **`useOrganization`:** This hook from `@clerk/nextjs` fetches details about the currently authenticated organization.

+ **Conditional Rendering:** If `organization` exists, it renders the `BoardList` component with `organization.id` and `searchParams`. Otherwise, it displays the `EmptyOrg` component indicating no organization data.

+ **Dynamic Component:** The `DashboardPage` component is exported using `dynamic` from Next.js with `{ssr: false }`, ensuring it is rendered client-side only.

b. BoardList Component (`BoardList`):

+ **State Management**: Uses `useState` hooks (`data`, `filteredData`, `loading`) to manage board data and loading state.

+ **Effects:**

- **`useEffect`** fetches boards based on `orgId` and `params.UserID`.
- Another `useEffect` handles filtering boards based on `searchParams`.

+ **Conditional Rendering:**

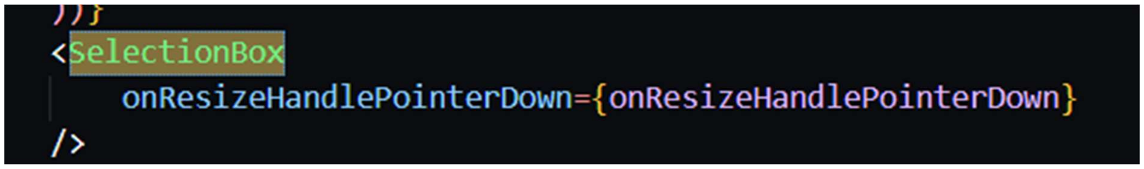
- While loading, displays skeleton loading components (`BoardCard.Skeleton`)
- Handles empty states (`EmptySearch`, `EmptyFavorites`, `EmptyBoards`) based on filtering results and query parameters.

+ **Rendering Boards:** Renders `BoardCard` components based on `filteredData` and `query.favorite` parameter.

3.1.3. Canvas:

a. Selection: enable users to interact with and manipulate specific elements (layers) within the canvas environment.

+ **SelectionBox:** This component provides handles around selected layers, facilitating resizing and transformation operations. It listens for pointer events and adjusts the selected layers accordingly.

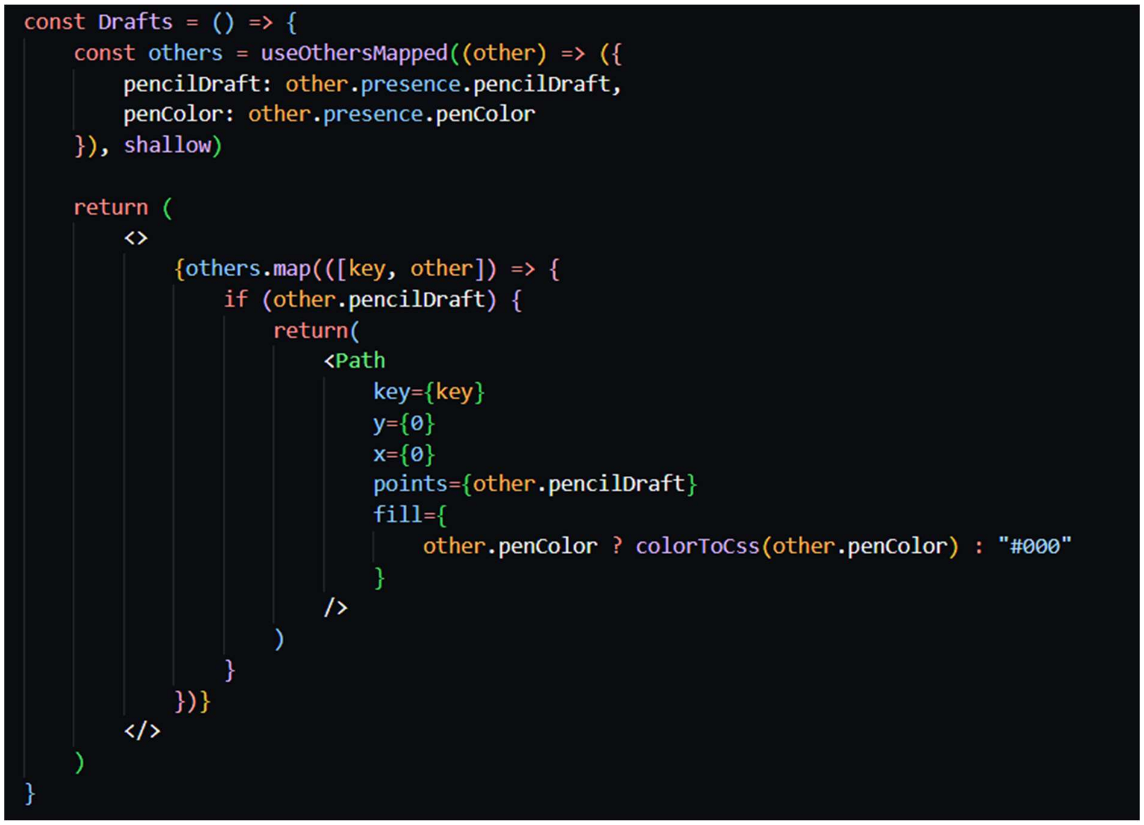


```
))}
<SelectionBox
  onResizeHandlePointerDown={onResizeHandlePointerDown}
/>
```

Figure. `SelectionBox` component.

+ **CursorsPresence:** It shows the cursors of other users who are collaborating on the same canvas. Each user's cursor is displayed with a unique identifier color, enhancing real-time visibility and collaboration. Which mainly defined by two main components:

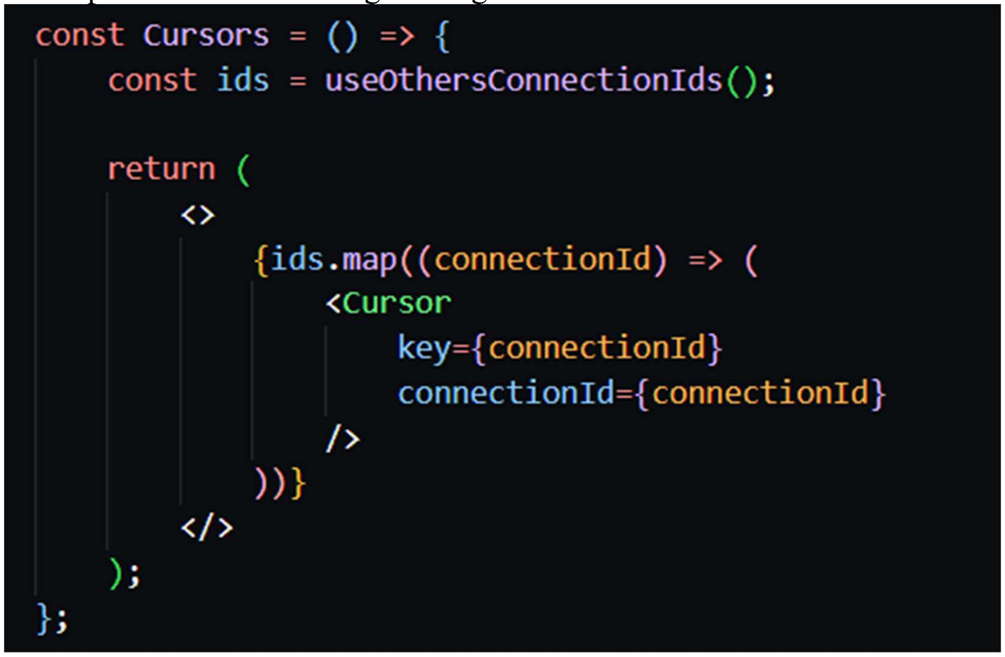
- **Drafts:** components that render pencil drafts made by other users, reflecting their freehand drawings on the canvas. To do that, we mainly use the `useOthersMapped()` to map over other users and extracts specific data (`pencilDraft` and `penColor`) using a shallow comparison for efficiency (`shallow`).



```
const Drafts = () => {
    const others = useOthersMapped((other) => ({
        pencilDraft: other.presence.pencilDraft,
        penColor: other.presence.penColor
    }), shallow)

    return (
        <>
            {others.map(([key, other]) => {
                if (other.pencilDraft) {
                    return(
                        <Path
                            key={key}
                            y={0}
                            x={0}
                            points={other.pencilDraft}
                            fill={
                                other.penColor ? colorToCss(other.penColor) : "#000"
                            }
                        />
                    )
                }
            })}
        </>
    )
}
```

Figure. `Drafts` component.

c. **Cursors:** displays live cursors representing the real-time positions of other users, enhancing visibility and collaboration. Especially, we implemented `useOthersConnectionIds()` to retrieve an array of connection IDs representing other users currently connected.



```
const Cursors = () => {
    const ids = useOthersConnectionIds();

    return (
        <>
            {ids.map((connectionId) => (
                <Cursor
                    key={connectionId}
                    connectionId={connectionId}
                />
            ))}
        </>
    );
};
```

Figure. \`Cursors\` component.

d. **\`LayerPreview\` component:**

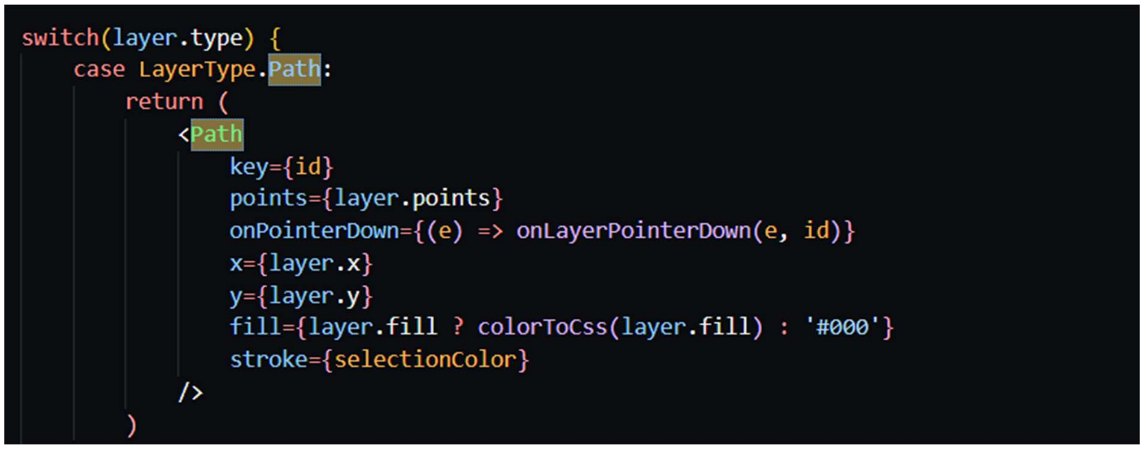
The \`LayerPreview\` component supports freehand drawing using the \`Path\` component. Here's how it integrates with the drawing functionality:

- Drawing: support the freehand drawing with a pencil tool, which mainly using pointer events to capture drawing strokes, then render the path dynamically as the user draws.

+ **Pointer Events**: Captures pointer events (`onPointerDown`) to track and render drawing strokes dynamically.

+ **Path Component**: Utilizes the `Path` component to render the drawn path on the canvas.

+ **Dynamic Rendering**: As the user draws with the pencil tool, the `Path` component dynamically updates to reflect the current drawing state.



```
switch(layer.type) {
    case LayerType.Path:
        return (
            <Path
                key={id}
                points={layer.points}
                onPointerDown={(e) => onLayerPointerDown(e, id)}
                x={layer.x}
                y={layer.y}
                fill={layer.fill ? colorToCss(layer.fill) : '#000'}
                stroke={selectionColor}
            />
        )
```

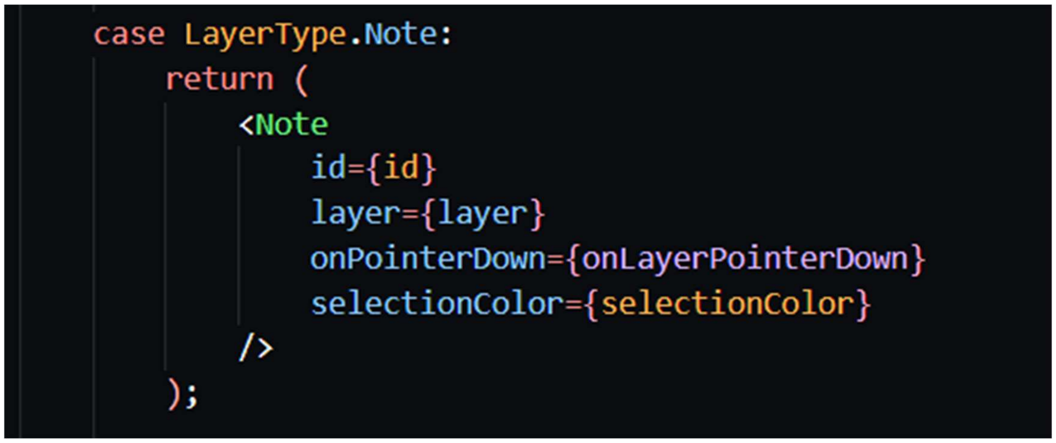
Figure. \`Drawing\` rendering function with \`Path\` component.

- Note-taking: allow adding textual notes on the canvas by integrating with a text tool to input and display text under the supports of editing and positioning functions.

+ **Note Component**: Integrates with the `Note` component to add textual notes.

+ **Editing and Positioning**: Supports editing and positioning functions for the added notes.

+ **Pointer Events**: Handles pointer events (`onPointerDown`) to interact with and manage notes on the canvas.



```
    case LayerType.Note:
        return (
            <Note
                id={id}
                layer={layer}
                onPointerDown={onLayerPointerDown}
                selectionColor={selectionColor}
            />
        );
```

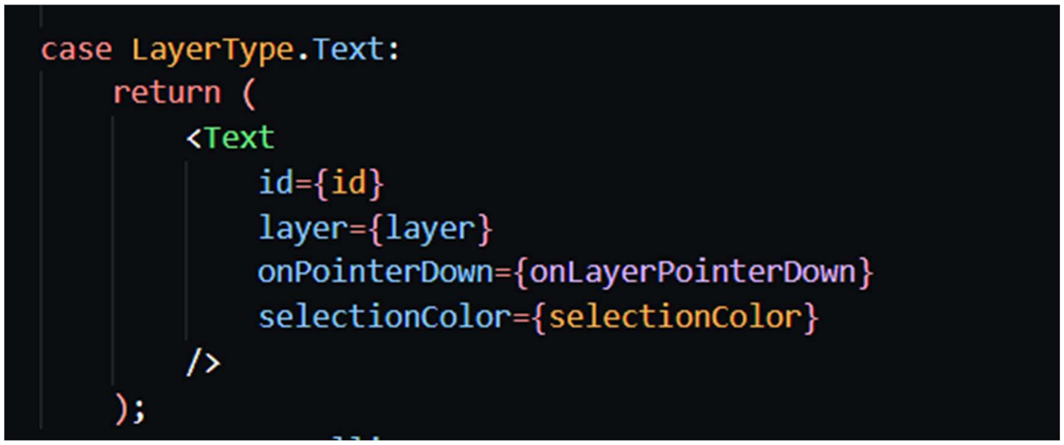
Figure. `Note-taking` rendering component.

- Text: support adding and editing text elements on the canvas by using a text tool to input and render with option for text color formatting.

+ **Text Component**: Integrates with the `Text` component for rendering and editing text.

+ **Text Tool**: Utilizes a text tool to input and display text elements with options for text color formatting.

+ **Interactive Handling**: Manages pointer events (`onPointerDown`) to interactively add and edit text elements on the canvas.

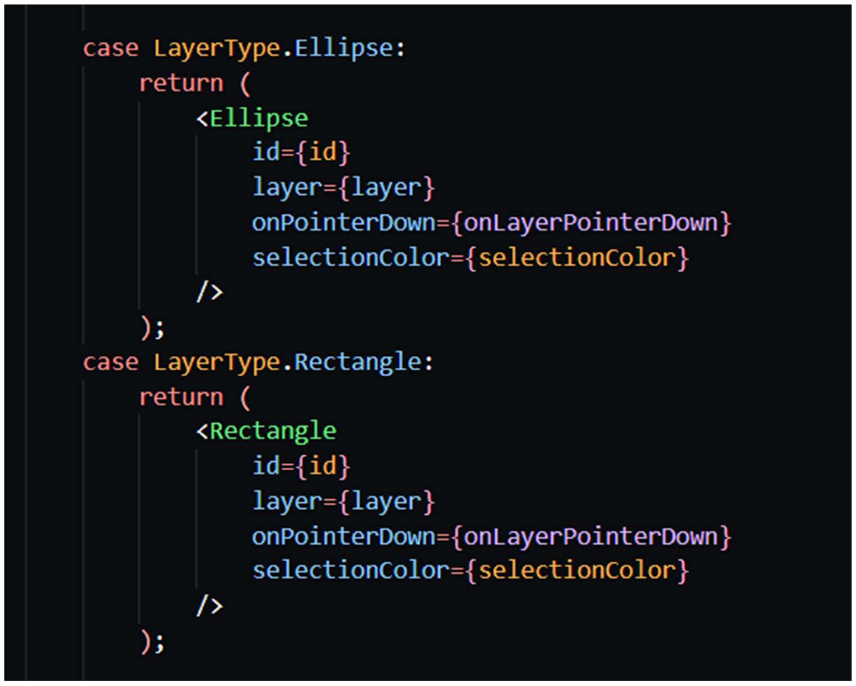


```
case LayerType.Text:
    return (
        <Text
            id={id}
            layer={layer}
            onPointerDown={onLayerPointerDown}
            selectionColor={selectionColor}
        />
    );
```

Figure. `Text ` rendering component.

- Shape Adding: allow adding geometric shapes like rectangles and ellipses, then provides tools for users to insert predefined shapes onto the canvas and enable resizing and editing properties of the shapes.

+ **Shape Components**: Uses `Ellipses` and `Rectangle` components for adding geometric shapes.
+ **Tools for Insertion**: Provides tools for users to insert predefined shapes onto the canvas.
+ **Resizing and Editing**: Supports resizing and editing properties of the shapes once added to the canvas.



```
case LayerType.Ellipse:
    return (
        <Ellipse
            id={id}
            layer={layer}
            onPointerDown={onLayerPointerDown}
            selectionColor={selectionColor}
        />
    );
case LayerType.Rectangle:
    return (
        <Rectangle
            id={id}
            layer={layer}
            onPointerDown={onLayerPointerDown}
            selectionColor={selectionColor}
        />
    );
```

Figure. Shapes components: `Ellipse` & `Rectangle` components.

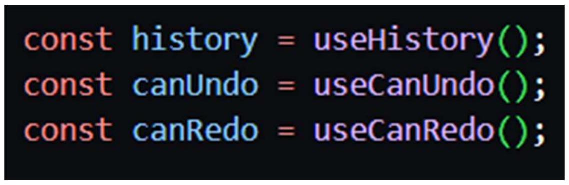
- Layers Formatting: provides UI controls for adjusting properties of selected layers, which mainly allows to resize, move or delete layers.

+ **Resizing**: Allows resizing of layers using interactive controls.
+ **Moving**: Supports moving layers within the canvas area.
+ **Deletion**: Provides options to delete selected layers from the canvas.

- Undo/Redo: Utilizes undo and redo functionality to manage changes on the canvas:

+ **History Management:**

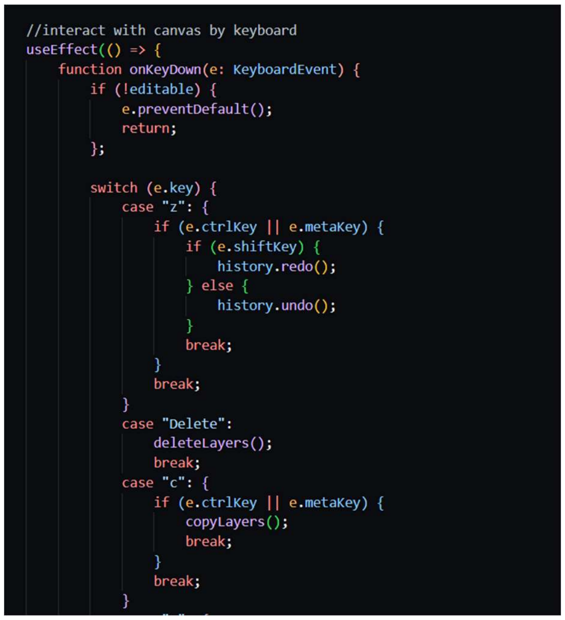
The `Canvas` component utilizes history management from `useHistory`, `useCanUndo`, and `useCanRedo` hooks provided by `@liveblocks/client`. These hooks facilitate tracking and controlling the state of undo and redo actions within the canvas editing session.



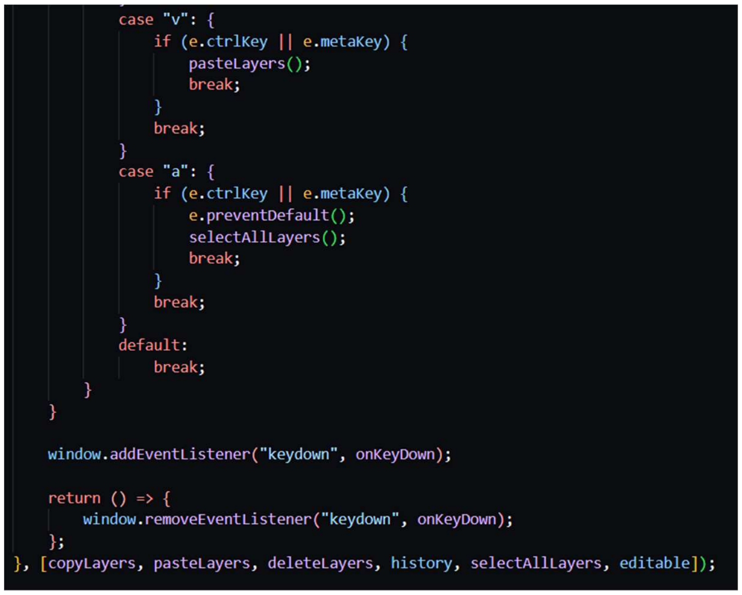
```
const history = useHistory();
const canUndo = useCanUndo();
const canRedo = useCanRedo();
```

Figure. History Management's hooks support for undo & redo.

+ **Keyboard Event Handling:** Keyboard events are used to trigger undo (`Ctrl+Z`) and redo (`Ctrl+Shift+Z`) actions, along with other canvas interactions like deletion, copying, pasting, and selecting all layers.



```
//interact with canvas by keyboard
useEffect(() => {
    function onKeyDown(e: KeyboardEvent) {
        if (!editable) {
            e.preventDefault();
            return;
        };

        switch (e.key) {
            case "z": {
                if (e.ctrlKey || e.metaKey) {
                    if (e.shiftKey) {
                        history.redo();
                    } else {
                        history.undo();
                    }
                    break;
                }
                break;
            }
            case "Delete":
                deleteLayers();
                break;
            case "c": {
                if (e.ctrlKey || e.metaKey) {
                    copyLayers();
                    break;
                }
                break;
            }
```



```
            case "v": {
                if (e.ctrlKey || e.metaKey) {
                    pasteLayers();
                    break;
                }
                break;
            }
            case "a": {
                if (e.ctrlKey || e.metaKey) {
                    e.preventDefault();
                    selectAllLayers();
                    break;
                }
                break;
            }
            default:
                break;
        }
    }

    window.addEventListener("keydown", onKeyDown);

    return () => {
        window.removeEventListener("keydown", onKeyDown);
    };
}, [copyLayers, pasteLayers, deleteLayers, history, selectAllLayers, editable]);
```

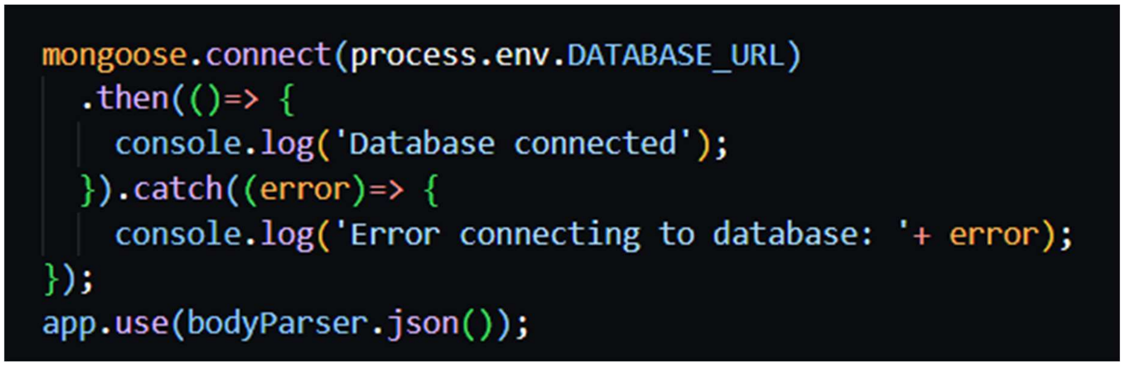
Figure. Keyboard Event Handling by dynamical `useEffect` rendering.

3.2. Backend:

3.2.1. MongoDB & Mongoose Connection Pool:

a. MongoDB Connection:

To ensure efficient database connection handling with MongoDB using Mongoose, it's beneficial to set up a connection pool. This helps manage multiple connections to the database, reducing overhead and improving performance.



```
mongoose.connect(process.env.DATABASE_URL)
  .then(()=> {
    console.log('Database connected');
  }).catch((error)=> {
    console.log('Error connecting to database: '+ error);
});
app.use(bodyParser.json());
```

Figure. MongoDB Connection Setup

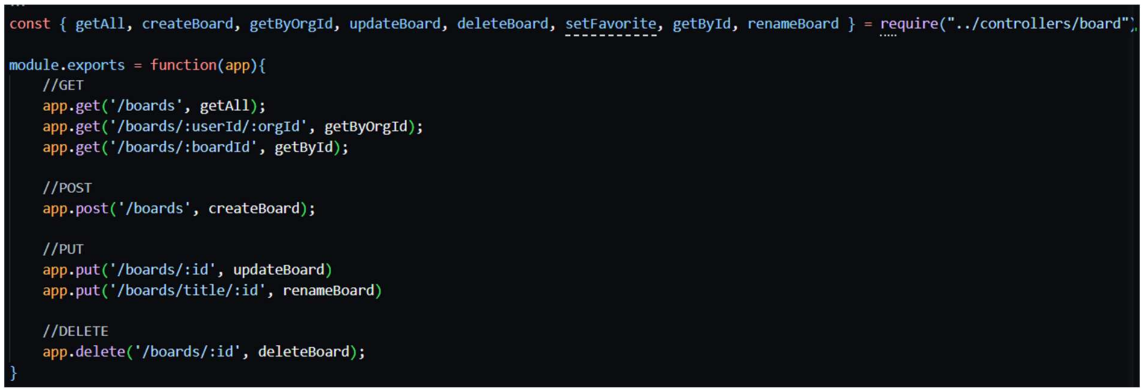
The `connectDB` function ensures that Mongoose connects to MongoDB using the `DATABASE_URL` environment variable. It handles connection errors and ensures essential connection options are set.

b. **Express App Setup**:

Sets up an Express application with middleware (`cors`, `body-parser`) and starts the server on the specified port (`PORT` environment variable or default `4000`). It also connects to MongoDB using the `connectDB` function.

3.2.2. API Endpoints:

Routes for boards and favorites are separated into different modules (`board.routes.js and `favorite.routes.js`). Each module exports an Express `Router` instance that defines specific endpoints (`GET, POST, PUT, DELETE`) for handling board and favorite operations.

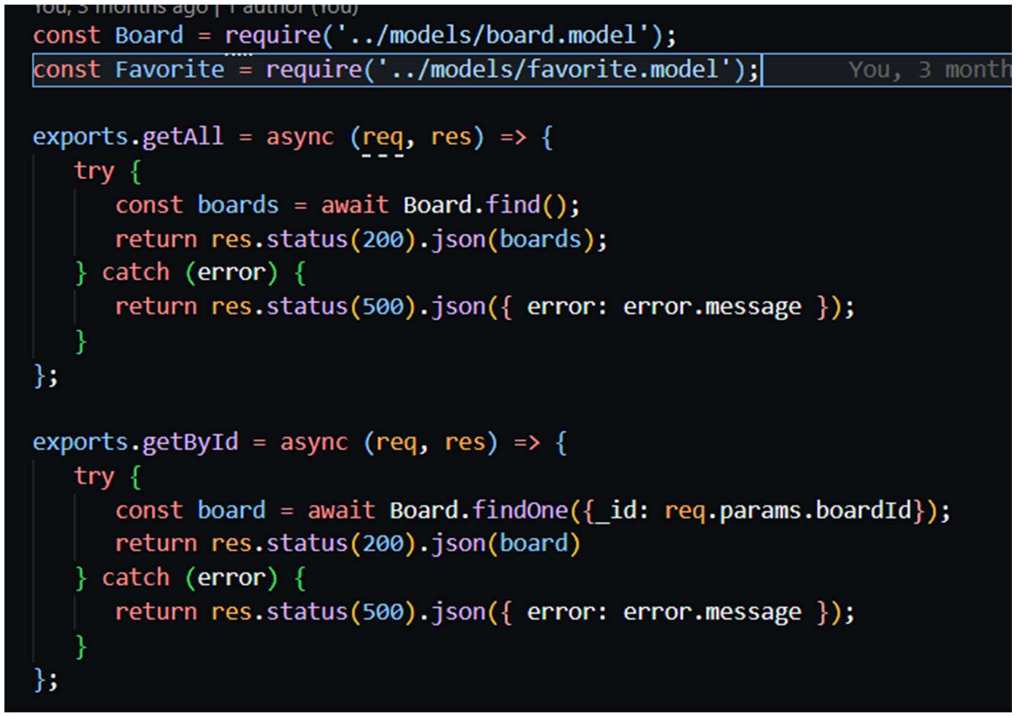


```
const { getAll, createBoard, getByOrgId, updateBoard, deleteBoard, setFavorite, getById, renameBoard } = require("../controllers/board")

module.exports = function(app){
    //GET
    app.get('/boards', getAll);
    app.get('/boards/:userId/:orgId', getByOrgId);
    app.get('/boards/:boardId', getById);

    //POST
    app.post('/boards', createBoard);

    //PUT
    app.put('/boards/:id', updateBoard)
    app.put('/boards/title/:id', renameBoard)

    //DELETE
    app.delete('/boards/:id', deleteBoard);
}
```

Figure. Snippet code of API-endpoint creation.

3.2.3. Controller Integration:

Controllers (`board.controller.js` and `favorite.routes.js`) are assumed to exist and are imported into the route modules (`board.routes.js` and `favorite.routes.js`). These controllers handle business logic for each API endpoint.



```
const Board = require('../models/board.model');
const Favorite = require('../models/favorite.model');        You, 3 month

exports.getAll = async (req, res) => {
    try {
        const boards = await Board.find();
        return res.status(200).json(boards);
    } catch (error) {
        return res.status(500).json({ error: error.message });
    }
};

exports.getById = async (req, res) => {
    try {
        const board = await Board.findOne({_id: req.params.boardId});
        return res.status(200).json(board)
    } catch (error) {
        return res.status(500).json({ error: error.message });
    }
};
```

Figure. Snippet code of controllers creation.

CHAPTER 4: Discussion and Evaluation

4.1. Introduction:

Testing is a crucial phase in web application development, ensuring that the application functions as intended and provides a seamless user experience. This chapter discusses the testing methods employed, the specific test case used, and evaluates the overview effectiveness of project.

4.2. Testing:

4.2.1 Manual:

Manual testing is a software testing approach where testers manually evaluate software or application quality without the help of automated testing tools or test scripts. Tester interact with the system like how an end user would to identify bugs, defects, and issues in the software that create friction in user experience.

When a developer manually runs their web application and tries out the features they have coded, they are doing manually testing. Its simplicity makes manual testing great for small-scale testing of personal projects. Even for large-scale testing where there are thousands and millions of items and features to test, manual testing is still needed to some degree.

Test Method:

4.2.1.1. Functional Testing

Functional Testing is a type of testing where testers check if features of the System Under Test are working as expected according to its specified requirements. These requirements are either collected from the users or provided the stakeholders (development team or product owner).

Test cases:

- Test case 1

Test Case 1: User Authentication	
System: Whiteboard Platform	**Subsystem:**
Designed by: Tran Ngoc Dang Khoi	**Design date:**10/03/2024
Tested by: Tran Phuong Quang Huy	**Execution date:** 13/03/2024
Test case ID: TC_FC_101	**Test case Title:** Sign In
Short description: Ensure that the user can login with valid credentials	
System Testing: Functional Testing	
Pre-conditions: User has a valid credentials	
Description	

Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Navigate to the login page	Displayed login screen with fields for email and button "Continue with Google"	Pass	
2	Enter valid email	Displayed next section screen for entering password	Pass	
3	Enter valid password	The system bring user redirected to the dashboard	Pass	
4	Enter invalid email and password	The system pop up a notification "Couldn't find your account."	Pass	
5	Click "Continue with google" to access by Google	Go to third-party login by gmail of Google and choose which mail to login into application	Pass	
Post-conditions: The user is logged in and redirected to the dashboard screen				

Test case ID: TC_FT_102	**Test case Title: Sign Up**			
Short description: Ensure that the user can sign up with valid credentials				
System Testing: Functional Testing				
Pre-conditions: None				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Navigate to the Sign Up page	Displayed SignUp screen with button "Continue with Google" and fields of email and password	Pass	
2	Enter valid email and compliant password	The system bring user redirected to the dashboard	Pass	
3	Enter invalid/taken email or/and noncompliant password	The system pop up red alert that need user to re-enter valid input	Pass	
4	Click "Continue with Google" to access by Google	Go to third-party login by gmail of Google and choose which mail to login into application	Pass	
Post-condition: Once successful, the user is already logged in and go straight to the dashboard screen				

- Test case 2

Test Case 2: Real-time Collaboration				
System: Whiteboard Platform		**Subsystem:**		
Designed by: Tran Phuong Quang Huy		**Design date:**19/04/2024		
Tested by: Tran Phuong Quang Huy		**Execution date:** 20/04/2024		
Short description: Ensure that updates made by one user are immediately visible to all other user on the same board				
System Testing: Functional Testing				
Pre-conditions: Multiple users are logged in and viewing the same board				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	User A moving their cursor on the board	User B can see User A 's cursor movment in real-time	Pass	
2	User A make change on the sketch (e.g., add a new shape, text)	User B can see the change/update immediately	Pass	
Post-conditions: The delay from point of view of User B less than 0.1 second				

- Test case 3

Test Case 3: Organization/Sketch Management				
System: Whiteboard Platform		**Subsystem:**		
Designed by: Tran Ngoc Dang Khoi		**Design date:**19/03/2024		
Tested by: Tran Phuong Quang Huy		**Execution date:** 25/03/2024		
Short description: Ensure that the user can create a new organization/sketch				
System Testing: Functional Testing				
Pre-conditions: User is logged in				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Navigate to the empty dashboard	Displayed dashboard screen with button "Create your first Organization"	Pass	
2	Click "Create Organiztion" button	The UI pop up a tab that has a field required Organization name	Pass	
3	Input Organization name for	The system move to Organization dashboard interface with "Create	Pass	

	organization creation	sketcha!" button to create first sketcha		
4	Click "Invite Members" button on the top-right of UI	The system pop up a tab that user can enter email's other user to invite them into this organization	Pass	Organization is successfully shared and invited user receive a notification to accept/deny that invitation
5	Click "Create sketcha!" button	The "Create Sketcha" button align at top left of UI and your first sketch is next to it	Pass	
6	Click the sketch's button that latest created	The system move into a sketch page that user can start modifying the sketch from here	Pass	
Post-conditions: The user can modified sketch and interact with elements				

- Test case 4: Drawing tool and interaction sketch

Test Case 4: Drawing tool and interaction on sketch				
System: Whiteboard Platform		**Subsystem:**		
Designed by: Tran Phuong Quang Huy		**Design date:** 01/04/2024		
Tested by: Tran Phuong Quang Huy		**Execution date:** 03/04/2024		
Short description: Ensure that the drawing tool and the entire interaction with elements works as expected				
System Testing: Functional Testing				
Pre-conditions: User is logged in and on a sketcha				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Select the circle/ rectangle/ sticky note/ text on the tool bar	Shape/ Note/ Text is drawn properly	Pass	
2	Select the pencil tool on the tool bar	User can draw a freestyle handwritten having no glitch	Pass	

3	Select element/ group of elements that user want to Copy/ Cut/ Paste/ Delete doing with shortcut on user's keyboard	The action work as expected when doing shortcut of Copy/ Cut/ Paste/ Delete that element/ group of elements	Pass	
4	Select Undo/ Redo when User want to undo or redo what user've done before	The sketch indicated the thing that user want to undo or redo accurately	Pass	
5	Select any element on the sketch	At the top of that element indicating Formation Tool. It concluded color picker, arrange layer and delete layer button	Pass	
Post-conditions: The user is logged in and redirected to the dashboard screen				

4.2.1.2. Usability Testing

- Test case 1

Test Case 1: User Interface (UI) Design				
System: Whiteboard Platform		**Subsystem:**		
Designed by: Tran Ngoc Dang Khoi, Tran Phuong Quang Huy		**Design date:**19/05/2024		
Tested by: Tran Phuong Quang Huy		**Execution date:** 24/05/2024		
Short description: Evaluate the layout of the main dashboard for ease of use				
System Testing: Usability Testing				
Pre-conditions: User is logged in				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Observe the user's interaction with the dashboard, collect feedback on the layout and accessibility of key features	User finds the dashboard layout intuitive and have an user-friendly on key features	Pass	

2	Ask user to do common task on the sketch page (e.g., switching tool, settings) and observe any difficulties or confusion	Users can navigate the board easily without confusion	Pass	
Post-conditions: Users can navigate the board easily without confusion				

- Test case 2

Test Case 2: Content and Readability				
System: Whiteboard Platform			**Subsystem:**	
Designed by: Tran Ngoc Dang Khoi, Tran Phuong Quang Huy			**Design date:** 10/06/2024	
Tested by: Tran Phuong Quang Huy			**Execution date:** 11/06/2024	
Short description: Evaluate the layout and readability of the web application				
System Testing: Usability Testing				
Pre-conditions: User is logged in				
Description				
Step	**Action**	**Expected System Response**	**P/F**	**Comments**
1	Collect opinions's user about layout, size of text of the Sketcha and color palatte	User finds the text is readable and appropriately sized. Color scheme is pleasant and offers good contrast. Content's layout is easy to grasp	Pass	
Post-conditions: Content's layout for users is easy to grasp and have a properly layout				

4.2.2. Automation:

Automation testing is the process of automating the execution of test cases through scripts and/or specialized tools. Instead of having human tester manually executing test case by test case, these scripts and tools directly commnad the Application Under Test (UAT) to perform all of the required actions, freeing up bandwidth for testers to focus more on more strategic activities.

The concept of Automation testing is not strictly limited to the Execution phase, although it is generally understood that and accepted in the testing community that automation testing means "automatically executing a test case".

Automation testing can be an umbrella term referring to the “automation of any testing activity across the testing life cycle”.

<u>Test Method</u>:

Performance Testing:

In order to make sure that Sketcha can support multiple users at once and provide latency-free real-time updates, performance testing is essential. Performance testing's primary goals are to evaluate the system's scalability, stability, and responsiveness to changing loads..

For automating performance testing, we used the following tools:

- JMeter: To simulate multiple users interacting with the application simultaneously.
- Postman: For automated API testing to measure response times and throughput.

<u>Test Scenarios</u>

- Test Scenario 1: Concurrent User Load Testing

Objective: To determine how Sketcha performs when multiple users interact with the application simultaneously.

Results:

- 100 users: Average response time: 200ms, no significant latency observed.
- 400 users: Average response time: 350ms, minor latency observed but within acceptable limits.
- 1000 users: Average response time: 700ms, noticeable latency observed, performance degradation starts.

Conclusion: Sketcha performs well with up to 400 concurrent users, but performance starts to degrade with 1000 users. Further optimization is needed for higher loads.

- Test Scenario 2: API Response Time Testing

Objective: To measure the response time of critical APIs under load.

APIs Tested:

1. Login API
2. Board Creation API
3. Real-time Update API

Results:

- Login API: Average response time: 150ms, 0.5% error rate.
- Sketch Creation API: Average response time: 250ms, 1% error rate.

- Real-time Update API: Average response time: 100ms, 0.2% error rate.

Conclusion: The APIs perform well under load with minimal error rates. Real-time updates are especially efficient, which is critical for a collaborative platform.

4.3. Discussion:

The testing process revealed several critical bugs that were promptly addressed. Manual testing allowed for a thorough exploration of the UI, un, uncovering usability issues that were not initially apparent. Automation testing proved invaluable for regression testing, ensuring that new changes did not break existing functionality

Challenges faced by included the initial time investment for setting up automation test and osscasional discrepancies in manual test results due to human error. Overall, the combination of manual and automation testing significantly enhanced the quality of the web application, leading to a more intuitive and user-friendly product.

Manual and automated testing methodologies provided comprehensive coverage of Sketcha's functionality and usability. Manual testing allowed for nuanced feedback on user experience, while automated testing ensured that key functional aspects were rigorously and repeatedly tested.

Key findings include:

- Functional Integrity: Most features worked as expected
- Performance: The application performed well under load but showed some latency issues that need addressing.
- Usability: User feedback highlighted areas for improvement in navigation and ease of use, user-friendly UI.

They combined approach of manual and automated testing proved effective in identifying a wide range of issues, from minor UI glitches to potential performance bottlenecks. This holistic testing strategy will guide further development and optimization of Sketcha.

4.4. Conclusion:

In conclusion, both manual and automation testing played a crucial role in the successful development of the web application. Manual testing was essential for areas requiring human judment, while automation testing provided efficiency and reliability for repetitive tasks. Future projects should continue to leverage both methods to maintain high-quality standards.

Sketcha through testing phase identified that several future efforts will focus on:

- Bug fixed: Addessing the minor bugs identified during testing phase.
- Usability Improvements: Enhancing the user interface based on feedback and making the mobile experience more seamless.

Overall, the testing phase has been instrumental in refining the Sketcha – Whiteboard Platform and ensuring it meets the user needs and expectations, paving the way for a more polished and reliable release.

CHAPTER 5:
Conclusion and Future Work

5.1. Conclusion:

This project aimed to develop and evaluate Sketcha, a web application designed to for collaborative idea sharing, visualazation and real-time editing on a mutual sketch and mutual organization. Over the course of the project, we undertook a comprehensive implement and testing process, which included manual and automated testing methods to ensure our platform's functionality, performance and usability.

Key accomplishments of the project include:

- Successful Implementation: The core features of Sketcha, including real-time collaboration, user authentication, and sketch management, were successfully implemented.
- Comprehensive Testing: Both manual and automation testing provided thorough coverage of the application, identifying critical issues and validating the stability of the features.

The project demonstrates the feasibility of and potential of Sketcha as a tool for collaborative work regardless of physical location, highlighting the importance of iterative testing and user-centered design in software development.

5.2. Future Work:

While Sketcha has achived its primary objective, several areas offer opportunities for further enhancement and research. Future work will focus on the following aspects:

5.2.1. Feature Enhancements

1. Advanced Collaboration Tools:

- Audio and Video Integration: Adding features for audio and video communication within the board to enhance real-time collaboration.
- Annotation and Commenting: Implementing features for user to leave comments and annotations on the sketch for more interactive feedback.

2. Enhanced User Management

- Export sketcha: Implementing features for user to export the current sketch into PDF, JPG format file and annotaion on the top of the sketch for an ease-of-use interface.
- Share sketcha through URL: Adding more option to share while the platform have inviting member through adding their email.

5.2.2. Performance Optimization

- Database Optimization: Optimization database queries and structures to handle larger datasets and more concurrent users efficiently.
- Load Balancing: Implementing load balancing techniques to distribute the workload evenly across servers and improve performance.

5.2.3. Usability and Accessibility

1. User Interface Refinements:
   - Responsive Design: Enhancing the mobile version of Sketcha to ensure a seamless and intuitive user experience across all devices.
   - User Onboarding: Developing comprehensive onboarding guides and tutorials to help new users quickly understand and use the application
2. Accessibility Enhancements:
   - WCAG Compliance: Ensuring the platform meets Web Content Accessibility Guidelines (WCAG) to make it accessible to users with disabilities.
   - Keyboard Navigation: Improving keyboard navigation to support user who rely on keyboard instead of mice.

5.2.4. Security Enhancement

1. Data Protection:
   - Encryption: Implementing end-to-end encryption for all user data to enhance security and privacy.
   - Regular Security Audits: Conducting frequency security audits and vulnerability assessments to identify and address potential threats.
2. Authentication: Using Multy-factor Authentication to provide an additional layer of security for user.

5.3. Reflection

The development of Sketcha has been a valuable learning experience, highlighting the importance of a structured development process, thorough testing, and user feedback in creating a successful web application. The insights gained from this project will guide future development efforts, ensuring that Sketcha continues to evolve and meet the needs of its users.

In conclusion, Sketcha has proven to be a promising tool for real-time collaborative work, with a strong foundation that can be built upon in future iterations. Continued development and improvement will help realize its full potential, making it an indispensable tool for collaborative idea sharing and creative expression.

References

Clerk. (n.d.). *Clerk*. Retrieved June 17, 2024, from https://clerk.com/

Next.js. (n.d.). *Next.js*. Retrieved June 17, 2024, from https://nextjs.org/

Liveblocks. (n.d.). *Liveblocks*. Retrieved June 17, 2024, from https://liveblocks.io/

Tailwind CSS. (n.d.). *Tailwind CSS*. Retrieved June 17, 2024, from https://tailwindcss.com/

Node.js. (n.d.). *Node.js*. Retrieved June 17, 2024, from https://nodejs.org/fr

MongoDB. (n.d.). *MongoDB*. Retrieved June 17, 2024, from https://www.mongodb.com/

Aggarwal, S., et al. (2018). Modern web-development using ReactJS. *International Journal of Recent Research Aspects*, 5(1), 133-137.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Katalon. (n.d.). *Manual testing: A comprehensive guide*. Retrieved June 17, 2024, from https://katalon.com/resources-center/blog/manual-testing

Katalon. (n.d.). *What is automation testing?*. Retrieved June 17, 2024, from https://katalon.com/resources-center/blog/what-is-automation-testing