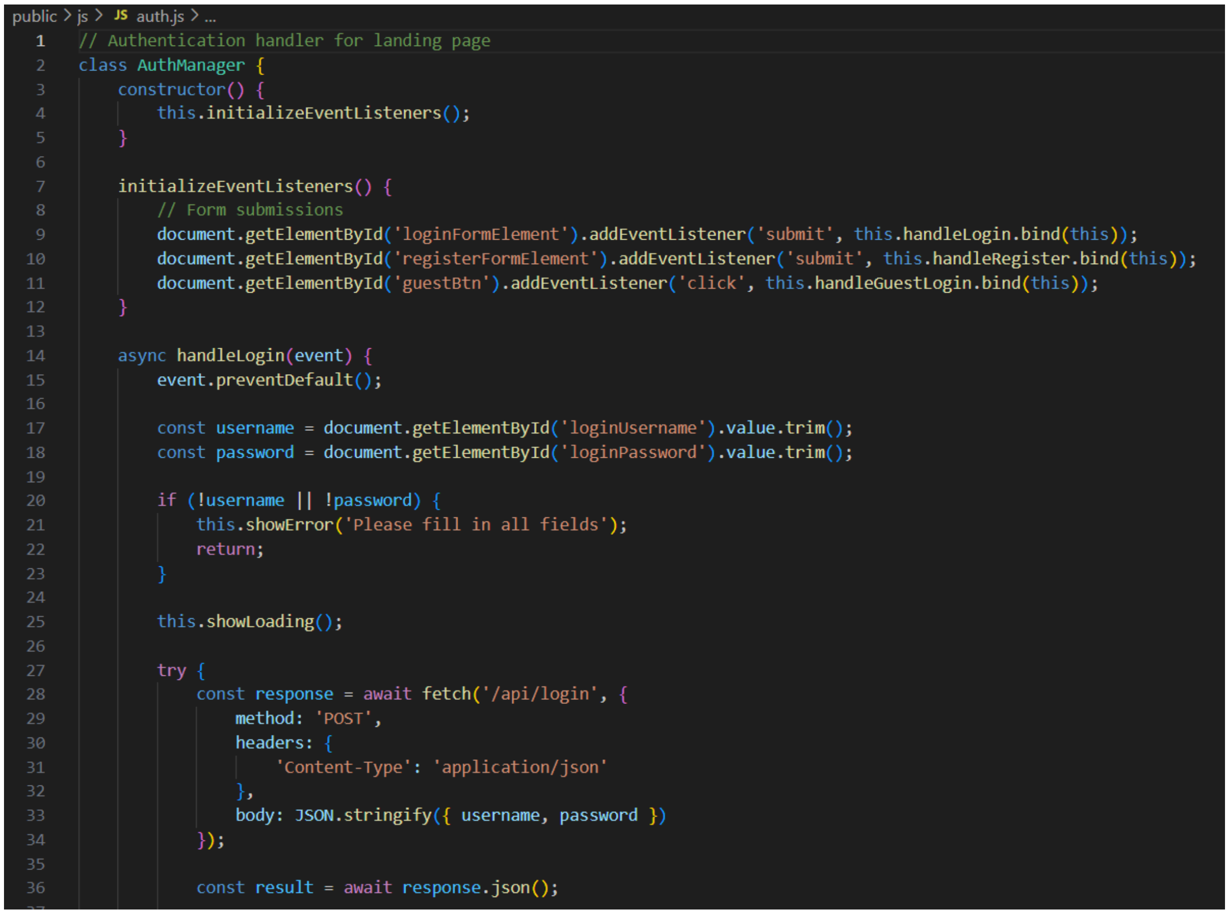
EXPERIMENT 8

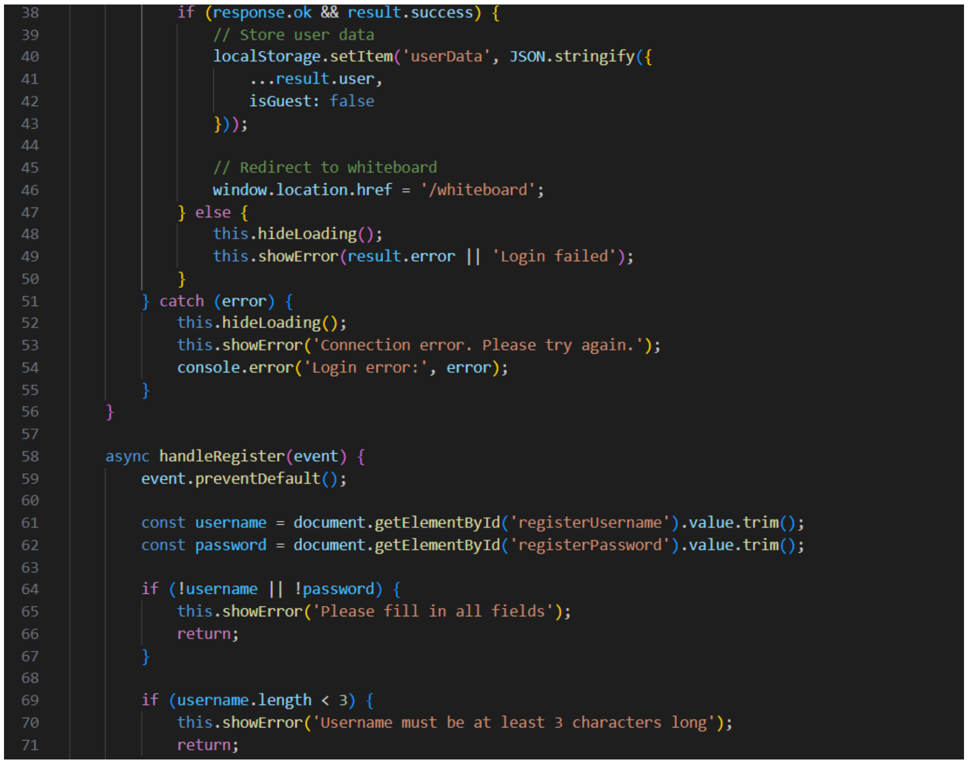
auth.js



public > js > JS auth.js > ...

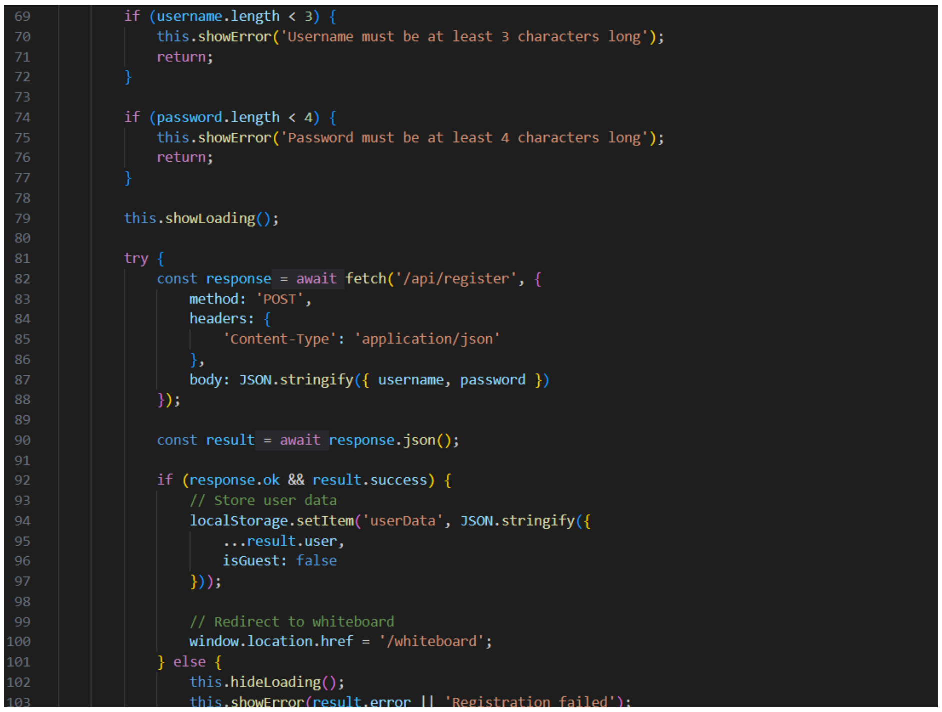
```javascript
// Authentication handler for landing page
class AuthManager {
    constructor() {
        this.initializeEventListeners();
    }

    initializeEventListeners() {
        // Form submissions
        document.getElementById('loginFormElement').addEventListener('submit', this.handleLogin.bind(this));
        document.getElementById('registerFormElement').addEventListener('submit', this.handleRegister.bind(this));
        document.getElementById('guestBtn').addEventListener('click', this.handleGuestLogin.bind(this));
    }

    async handleLogin(event) {
        event.preventDefault();

        const username = document.getElementById('loginUsername').value.trim();
        const password = document.getElementById('loginPassword').value.trim();

        if (!username || !password) {
            this.showError('Please fill in all fields');
            return;
        }

        this.showLoading();

        try {
            const response = await fetch('/api/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ username, password })
            });

            const result = await response.json();
```

Fig 1.1.1



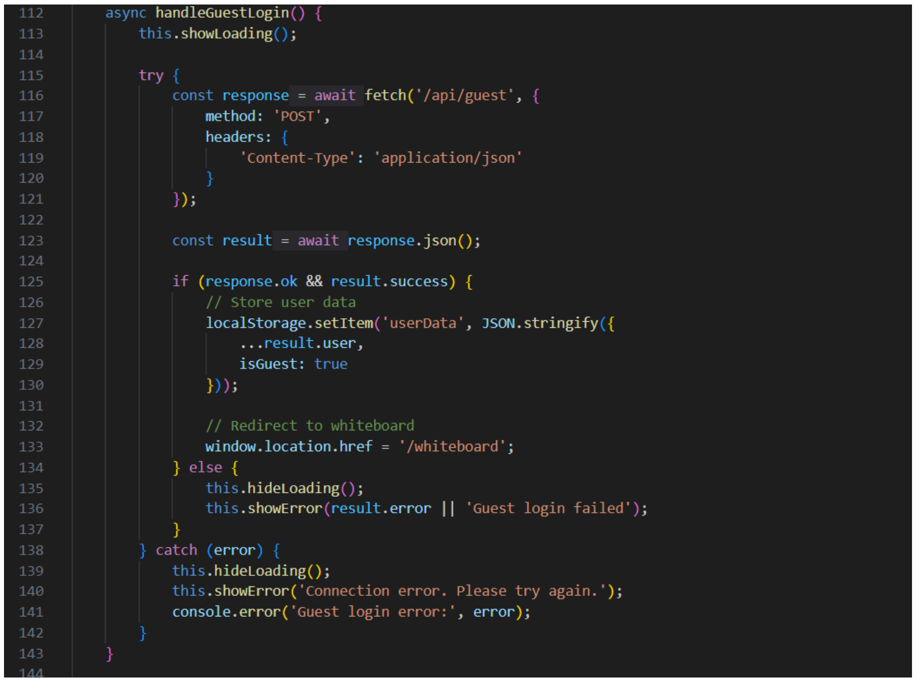
```javascript
        if (response.ok && result.success) {
            // Store user data
            localStorage.setItem('userData', JSON.stringify({
                ...result.user,
                isGuest: false
            }));

            // Redirect to whiteboard
            window.location.href = '/whiteboard';
        } else {
            this.hideLoading();
            this.showError(result.error || 'Login failed');
        }
    } catch (error) {
        this.hideLoading();
        this.showError('Connection error. Please try again.');
        console.error('Login error:', error);
    }
}

async handleRegister(event) {
    event.preventDefault();

    const username = document.getElementById('registerUsername').value.trim();
    const password = document.getElementById('registerPassword').value.trim();

    if (!username || !password) {
        this.showError('Please fill in all fields');
        return;
    }

    if (username.length < 3) {
        this.showError('Username must be at least 3 characters long');
        return;
```

Fig 1.1.2



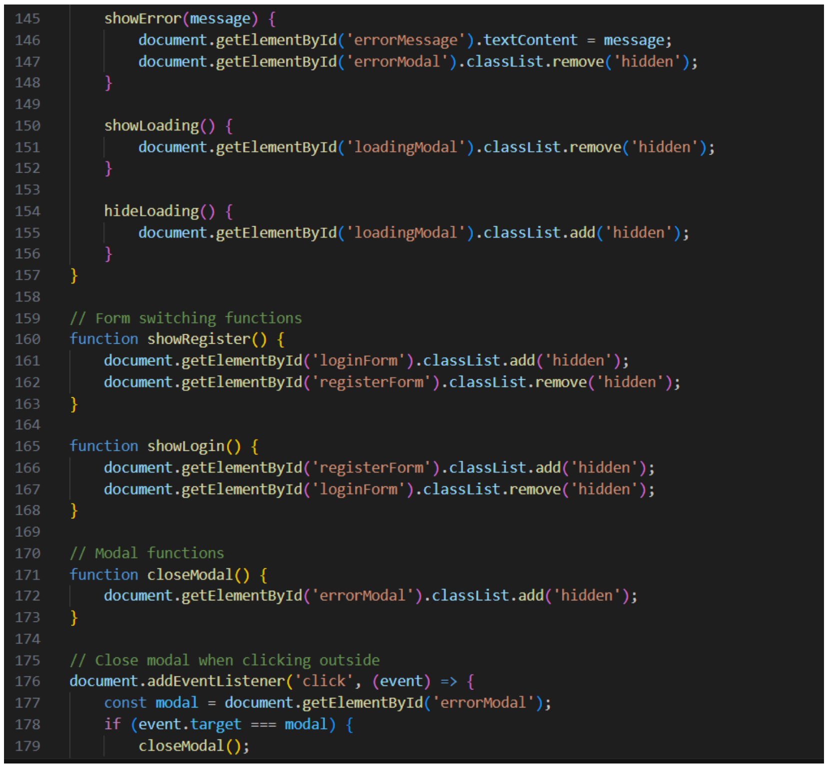
```javascript
    if (username.length < 3) {
        this.showError('Username must be at least 3 characters long');
        return;
    }

    if (password.length < 4) {
        this.showError('Password must be at least 4 characters long');
        return;
    }

    this.showLoading();

    try {
        const response = await fetch('/api/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        });

        const result = await response.json();

        if (response.ok && result.success) {
            // Store user data
            localStorage.setItem('userData', JSON.stringify({
                ...result.user,
                isGuest: false
            }));

            // Redirect to whiteboard
            window.location.href = '/whiteboard';
        } else {
            this.hideLoading();
            this.showError(result.error || 'Registration failed');
```

Fig 1.1.3



```javascript
    async handleGuestLogin() {
        this.showLoading();

        try {
            const response = await fetch('/api/guest', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                }
            });

            const result = await response.json();

            if (response.ok && result.success) {
                // Store user data
                localStorage.setItem('userData', JSON.stringify({
                    ...result.user,
                    isGuest: true
                }));

                // Redirect to whiteboard
                window.location.href = '/whiteboard';
            } else {
                this.hideLoading();
                this.showError(result.error || 'Guest login failed');
            }
        } catch (error) {
            this.hideLoading();
            this.showError('Connection error. Please try again.');
            console.error('Guest login error:', error);
        }
    }
```

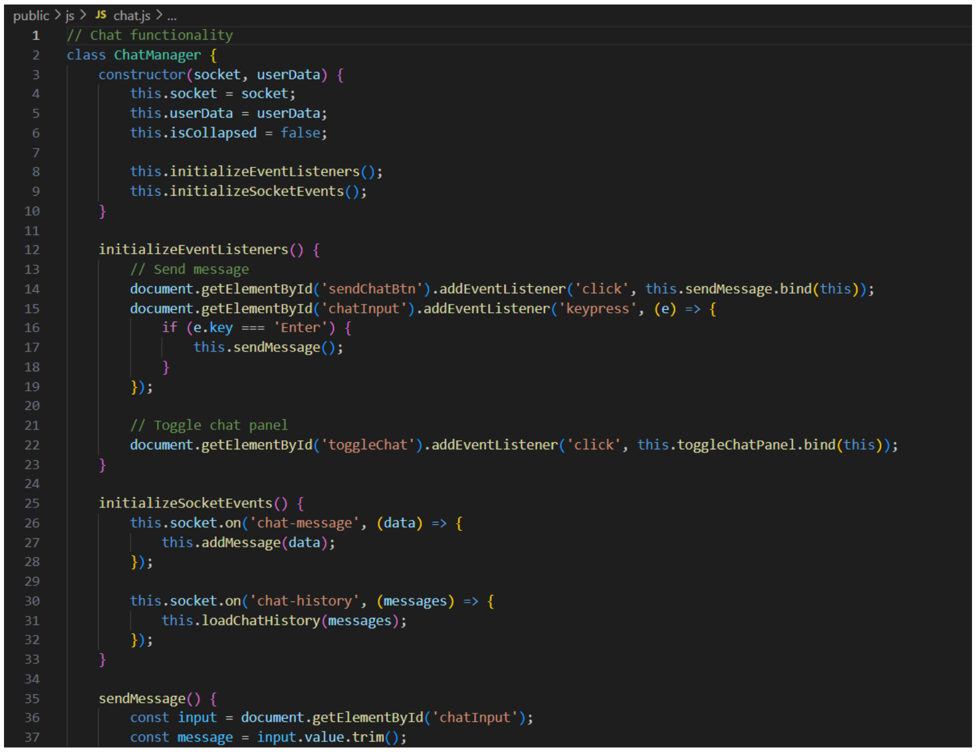
Fig 1.1.4



```javascript
    showError(message) {
        document.getElementById('errorMessage').textContent = message;
        document.getElementById('errorModal').classList.remove('hidden');
    }

    showLoading() {
        document.getElementById('loadingModal').classList.remove('hidden');
    }

    hideLoading() {
        document.getElementById('loadingModal').classList.add('hidden');
    }
}

// Form switching functions
function showRegister() {
    document.getElementById('loginForm').classList.add('hidden');
    document.getElementById('registerForm').classList.remove('hidden');
}

function showLogin() {
    document.getElementById('registerForm').classList.add('hidden');
    document.getElementById('loginForm').classList.remove('hidden');
}

// Modal functions
function closeModal() {
    document.getElementById('errorModal').classList.add('hidden');
}

// Close modal when clicking outside
document.addEventListener('click', (event) => {
    const modal = document.getElementById('errorModal');
    if (event.target === modal) {
        closeModal();
```

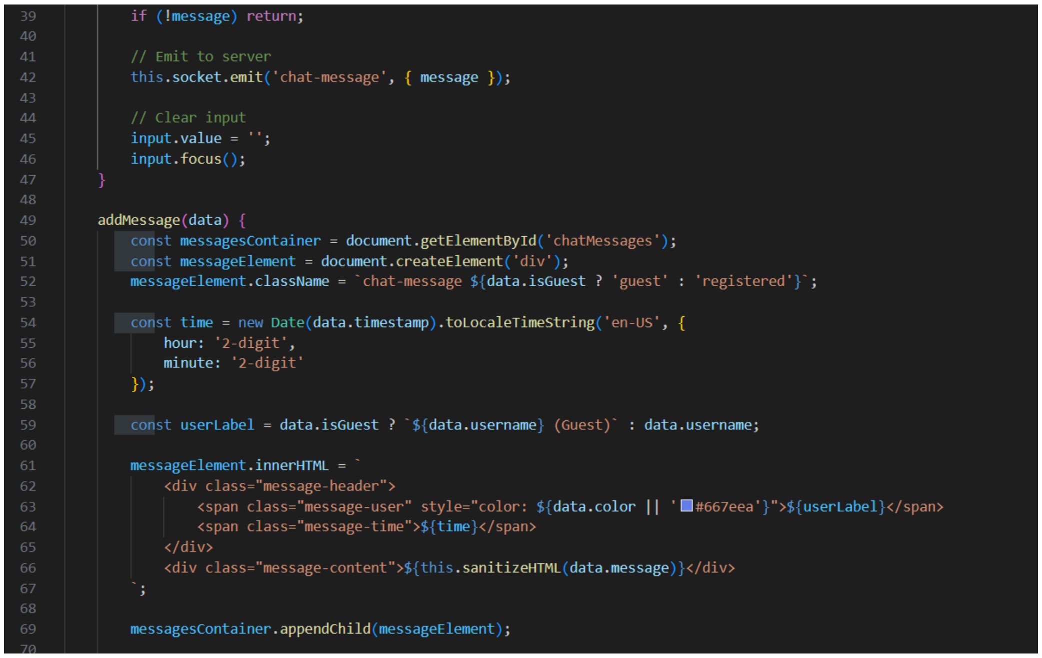
Fig 1.1.5

chat.js



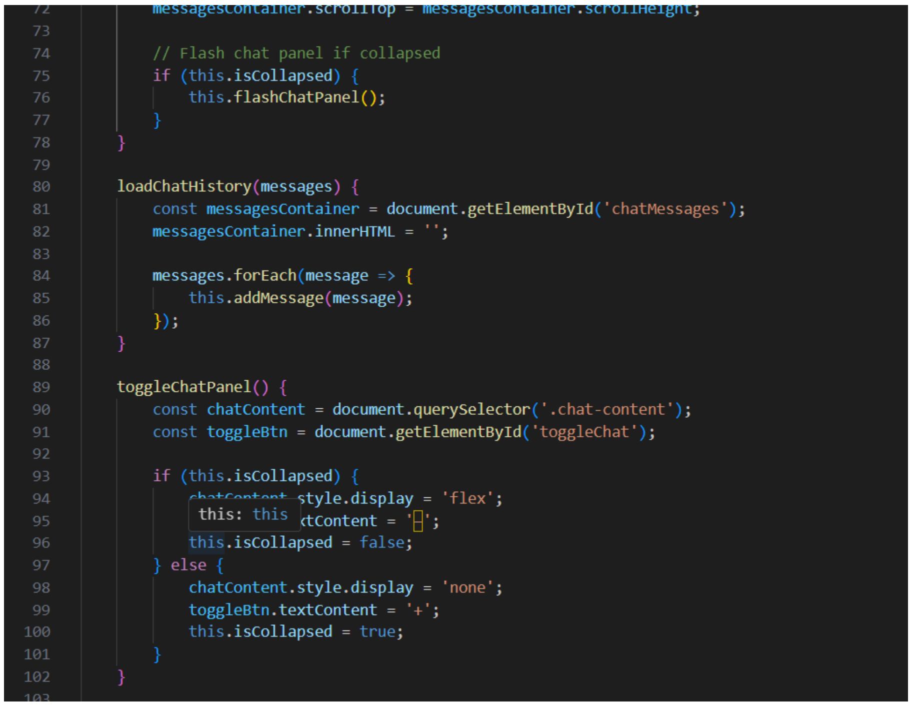
```javascript
// Chat functionality
class ChatManager {
    constructor(socket, userData) {
        this.socket = socket;
        this.userData = userData;
        this.isCollapsed = false;

        this.initializeEventListeners();
        this.initializeSocketEvents();
    }

    initializeEventListeners() {
        // Send message
        document.getElementById('sendChatBtn').addEventListener('click', this.sendMessage.bind(this));
        document.getElementById('chatInput').addEventListener('keypress', (e) => {
            if (e.key === 'Enter') {
                this.sendMessage();
            }
        });

        // Toggle chat panel
        document.getElementById('toggleChat').addEventListener('click', this.toggleChatPanel.bind(this));
    }

    initializeSocketEvents() {
        this.socket.on('chat-message', (data) => {
            this.addMessage(data);
        });

        this.socket.on('chat-history', (messages) => {
            this.loadChatHistory(messages);
        });
    }

    sendMessage() {
        const input = document.getElementById('chatInput');
        const message = input.value.trim();
```

Fig 1.2.1



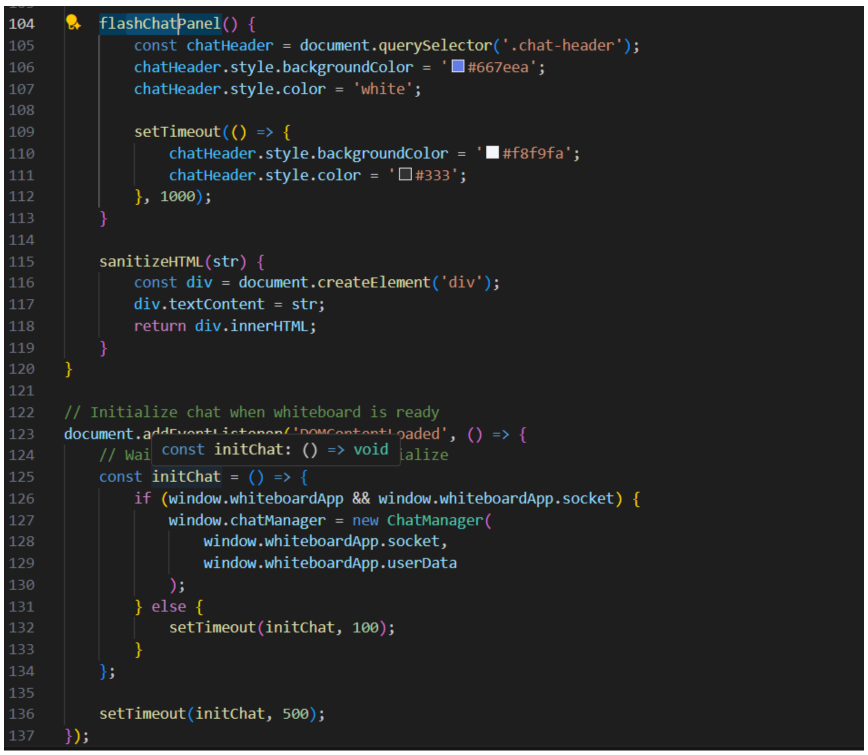
```javascript
        if (!message) return;

        // Emit to server
        this.socket.emit('chat-message', { message });

        // Clear input
        input.value = '';
        input.focus();
    }

    addMessage(data) {
        const messagesContainer = document.getElementById('chatMessages');
        const messageElement = document.createElement('div');
        messageElement.className = `chat-message ${data.isGuest ? 'guest' : 'registered'}`;

        const time = new Date(data.timestamp).toLocaleTimeString('en-US', {
            hour: '2-digit',
            minute: '2-digit'
        });

        const userLabel = data.isGuest ? `${data.username} (Guest)` : data.username;

        messageElement.innerHTML = `
            <div class="message-header">
                <span class="message-user" style="color: ${data.color || '#667eea'}">${userLabel}</span>
                <span class="message-time">${time}</span>
            </div>
            <div class="message-content">${this.sanitizeHTML(data.message)}</div>
        `;

        messagesContainer.appendChild(messageElement);
```

Fig 1.2.2



```
            messagesContainer.scrollTop = messagesContainer.scrollHeight;

            // Flash chat panel if collapsed
            if (this.isCollapsed) {
                this.flashChatPanel();
            }
        }

        loadChatHistory(messages) {
            const messagesContainer = document.getElementById('chatMessages');
            messagesContainer.innerHTML = '';

            messages.forEach(message => {
                this.addMessage(message);
            });
        }

        toggleChatPanel() {
            const chatContent = document.querySelector('.chat-content');
            const toggleBtn = document.getElementById('toggleChat');

            if (this.isCollapsed) {
                chatContent.style.display = 'flex';
                toggleBtn.textContent = '□';
                this.isCollapsed = false;
            } else {
                chatContent.style.display = 'none';
                toggleBtn.textContent = '+';
                this.isCollapsed = true;
            }
        }
```

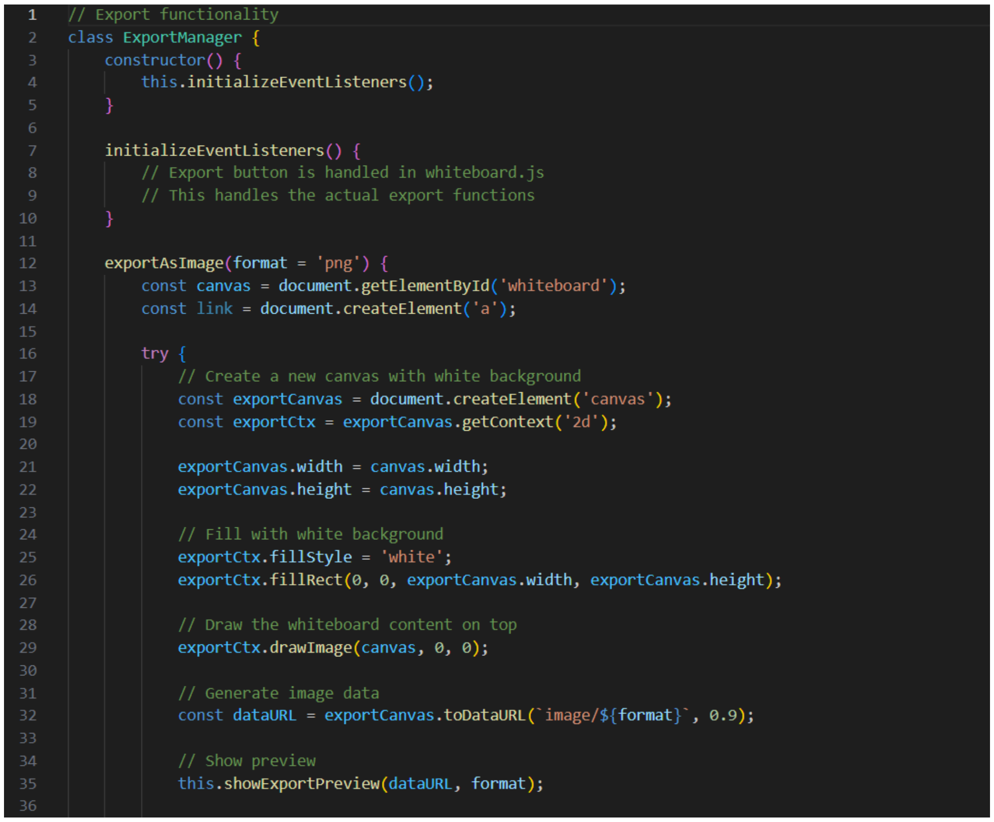
Fig 1.2.3



```
        flashChatPanel() {
            const chatHeader = document.querySelector('.chat-header');
            chatHeader.style.backgroundColor = '#667eea';
            chatHeader.style.color = 'white';

            setTimeout(() => {
                chatHeader.style.backgroundColor = '#f8f9fa';
                chatHeader.style.color = '#333';
            }, 1000);
        }

        sanitizeHTML(str) {
            const div = document.createElement('div');
            div.textContent = str;
            return div.innerHTML;
        }
    }

    // Initialize chat when whiteboard is ready
    document.addEventListener('DOMContentLoaded', () => {
        // Wai[illegible]ialize
        const initChat = () => {
            if (window.whiteboardApp && window.whiteboardApp.socket) {
                window.chatManager = new ChatManager(
                    window.whiteboardApp.socket,
                    window.whiteboardApp.userData
                );
            } else {
                setTimeout(initChat, 100);
            }
        };

        setTimeout(initChat, 500);
    });
```

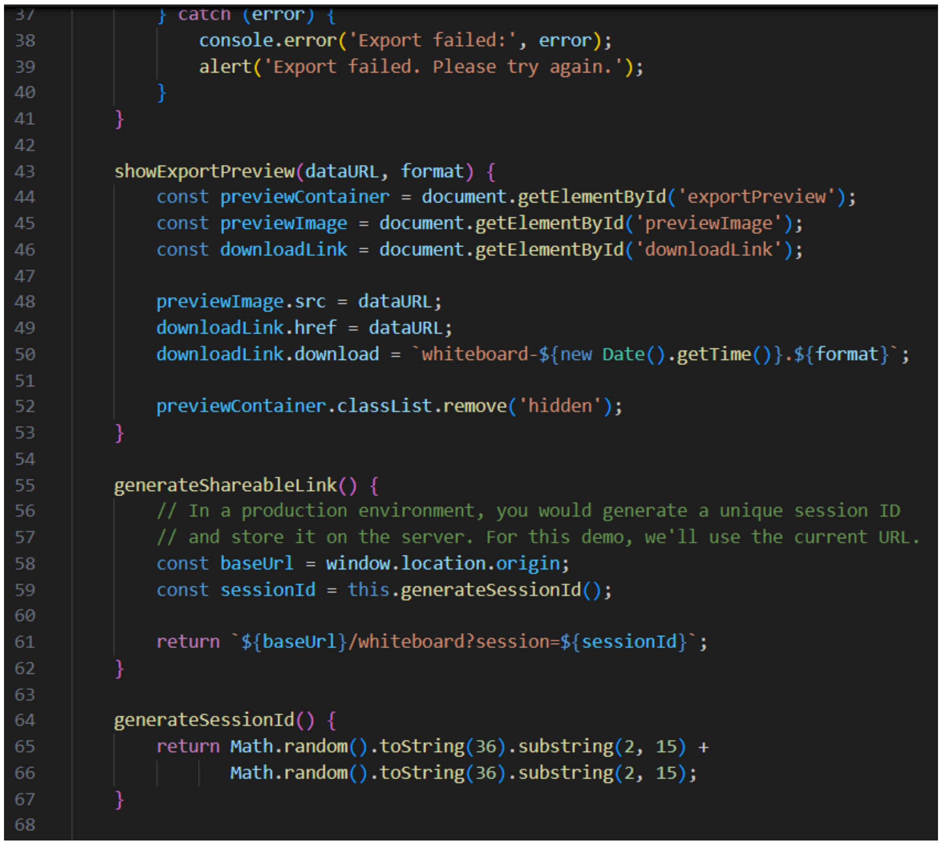
Fig 1.2.4

export.js



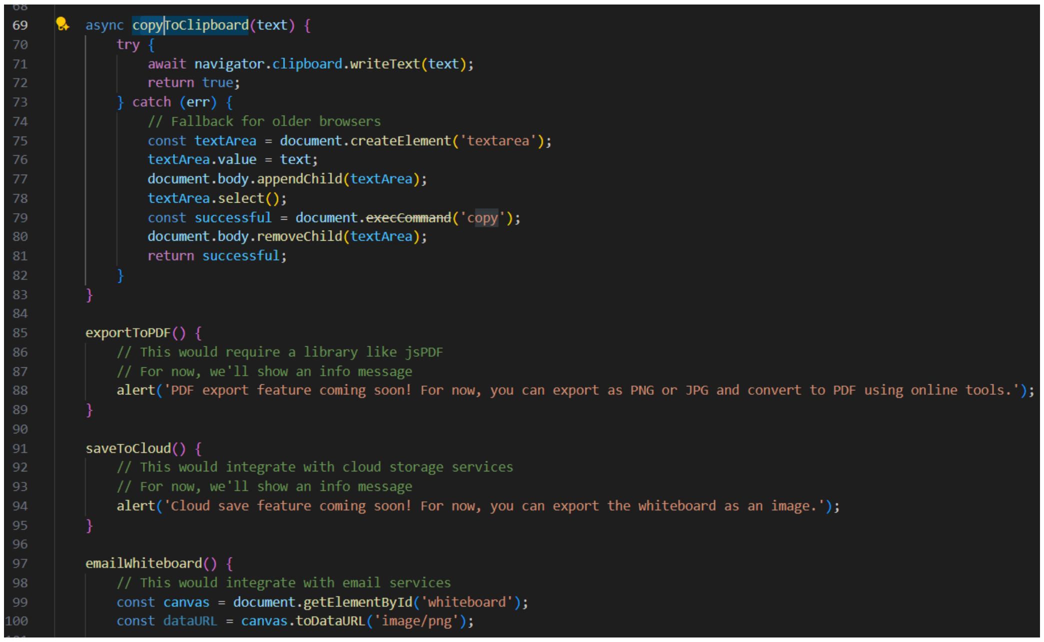
```javascript
// Export functionality
class ExportManager {
    constructor() {
        this.initializeEventListeners();
    }

    initializeEventListeners() {
        // Export button is handled in whiteboard.js
        // This handles the actual export functions
    }

    exportAsImage(format = 'png') {
        const canvas = document.getElementById('whiteboard');
        const link = document.createElement('a');

        try {
            // Create a new canvas with white background
            const exportCanvas = document.createElement('canvas');
            const exportCtx = exportCanvas.getContext('2d');

            exportCanvas.width = canvas.width;
            exportCanvas.height = canvas.height;

            // Fill with white background
            exportCtx.fillStyle = 'white';
            exportCtx.fillRect(0, 0, exportCanvas.width, exportCanvas.height);

            // Draw the whiteboard content on top
            exportCtx.drawImage(canvas, 0, 0);

            // Generate image data
            const dataURL = exportCanvas.toDataURL(`image/${format}`, 0.9);

            // Show preview
            this.showExportPreview(dataURL, format);

```

Fig 1.3.1



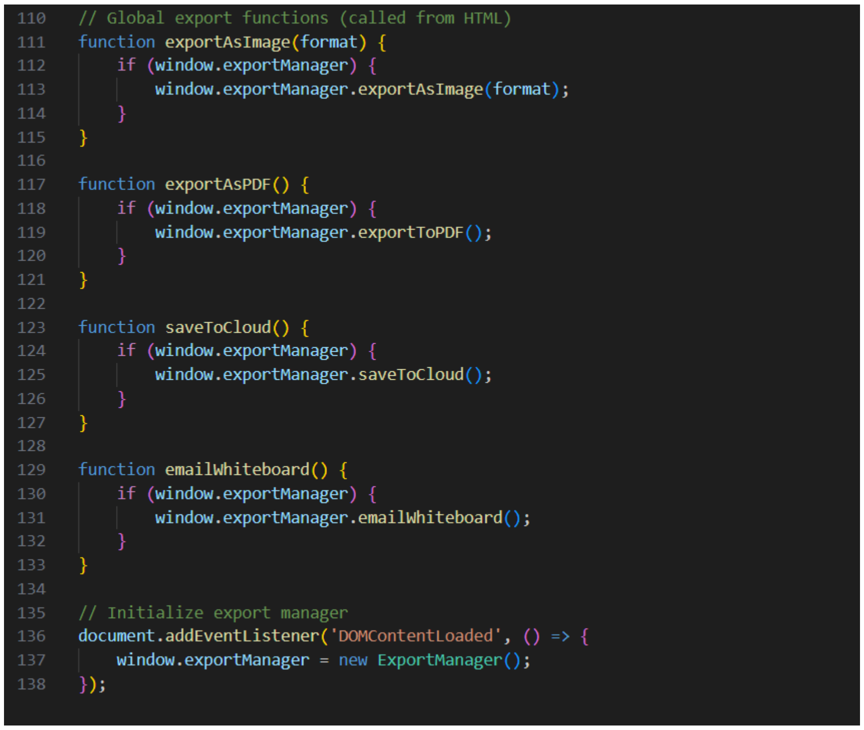
```javascript
        } catch (error) {
            console.error('Export failed:', error);
            alert('Export failed. Please try again.');
        }
    }

    showExportPreview(dataURL, format) {
        const previewContainer = document.getElementById('exportPreview');
        const previewImage = document.getElementById('previewImage');
        const downloadLink = document.getElementById('downloadLink');

        previewImage.src = dataURL;
        downloadLink.href = dataURL;
        downloadLink.download = `whiteboard-${new Date().getTime()}.${format}`;

        previewContainer.classList.remove('hidden');
    }

    generateShareableLink() {
        // In a production environment, you would generate a unique session ID
        // and store it on the server. For this demo, we'll use the current URL.
        const baseUrl = window.location.origin;
        const sessionId = this.generateSessionId();

        return `${baseUrl}/whiteboard?session=${sessionId}`;
    }

    generateSessionId() {
        return Math.random().toString(36).substring(2, 15) +
               Math.random().toString(36).substring(2, 15);
    }

```

FIg 1.3.2



```javascript
    async copyToClipboard(text) {
        try {
            await navigator.clipboard.writeText(text);
            return true;
        } catch (err) {
            // Fallback for older browsers
            const textArea = document.createElement('textarea');
            textArea.value = text;
            document.body.appendChild(textArea);
            textArea.select();
            const successful = document.execCommand('copy');
            document.body.removeChild(textArea);
            return successful;
        }
    }

    exportToPDF() {
        // This would require a library like jsPDF
        // For now, we'll show an info message
        alert('PDF export feature coming soon! For now, you can export as PNG or JPG and convert to PDF using online tools.');
    }

    saveToCloud() {
        // This would integrate with cloud storage services
        // For now, we'll show an info message
        alert('Cloud save feature coming soon! For now, you can export the whiteboard as an image.');
    }

    emailWhiteboard() {
        // This would integrate with email services
        const canvas = document.getElementById('whiteboard');
        const dataURL = canvas.toDataURL('image/png');
```

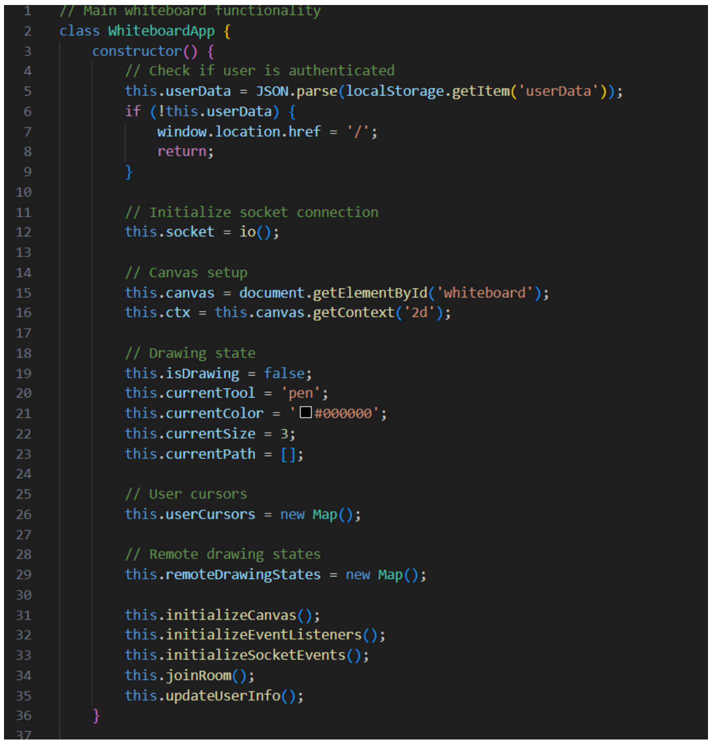
Fig 1.3.4



```javascript
// Global export functions (called from HTML)
function exportAsImage(format) {
    if (window.exportManager) {
        window.exportManager.exportAsImage(format);
    }
}

function exportAsPDF() {
    if (window.exportManager) {
        window.exportManager.exportToPDF();
    }
}

function saveToCloud() {
    if (window.exportManager) {
        window.exportManager.saveToCloud();
    }
}

function emailWhiteboard() {
    if (window.exportManager) {
        window.exportManager.emailWhiteboard();
    }
}

// Initialize export manager
document.addEventListener('DOMContentLoaded', () => {
    window.exportManager = new ExportManager();
});
```

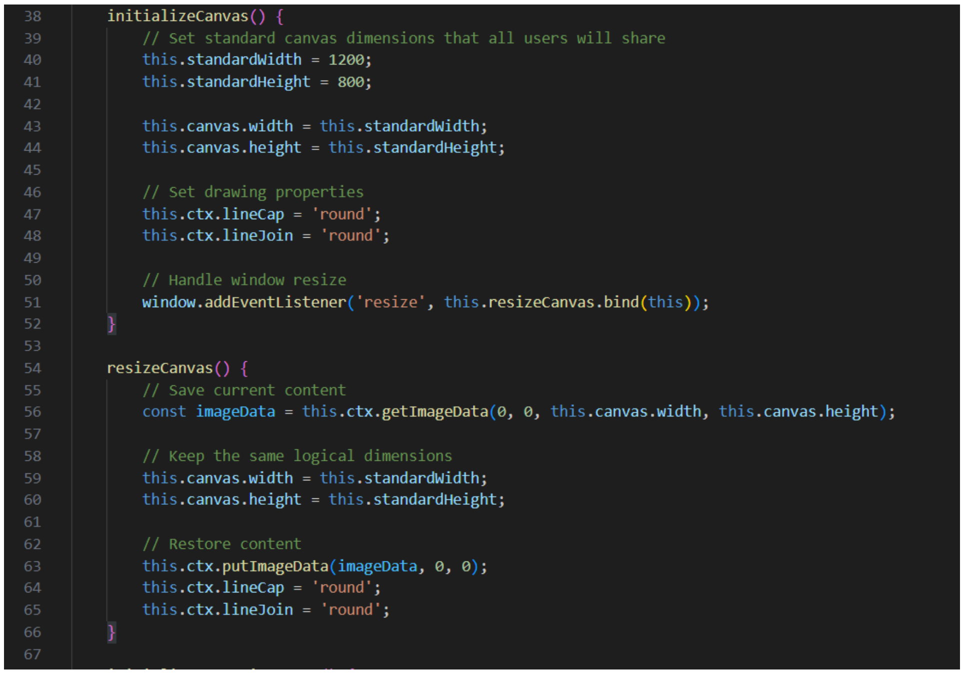
Fig 1.3.5

whiteboard.js



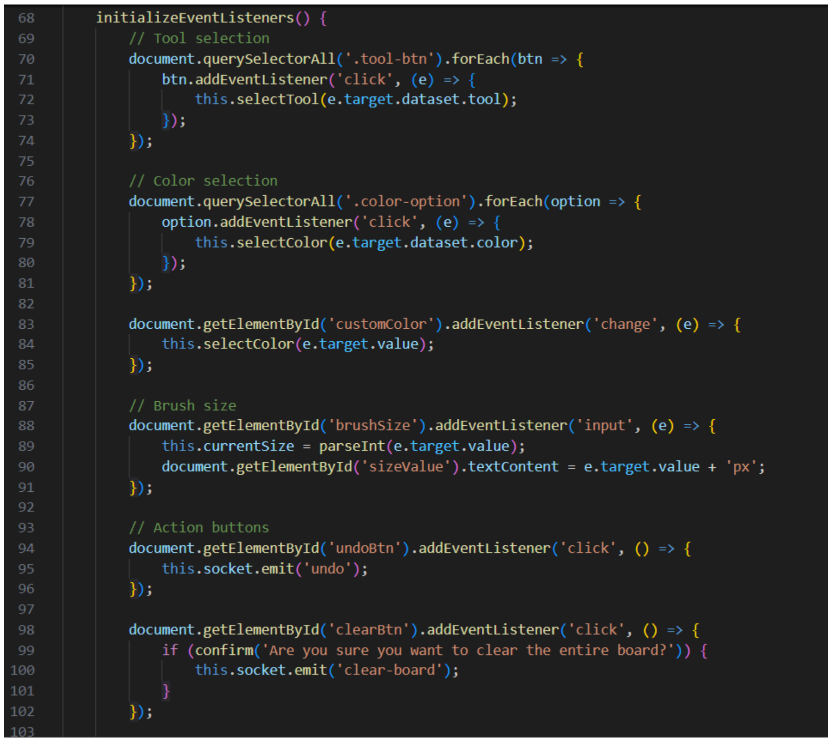
```javascript
// Main whiteboard functionality
class WhiteboardApp {
    constructor() {
        // Check if user is authenticated
        this.userData = JSON.parse(localStorage.getItem('userData'));
        if (!this.userData) {
            window.location.href = '/';
            return;
        }

        // Initialize socket connection
        this.socket = io();

        // Canvas setup
        this.canvas = document.getElementById('whiteboard');
        this.ctx = this.canvas.getContext('2d');

        // Drawing state
        this.isDrawing = false;
        this.currentTool = 'pen';
        this.currentColor = '#000000';
        this.currentSize = 3;
        this.currentPath = [];

        // User cursors
        this.userCursors = new Map();

        // Remote drawing states
        this.remoteDrawingStates = new Map();

        this.initializeCanvas();
        this.initializeEventListeners();
        this.initializeSocketEvents();
        this.joinRoom();
        this.updateUserInfo();
    }

```

Fig 1.4.1



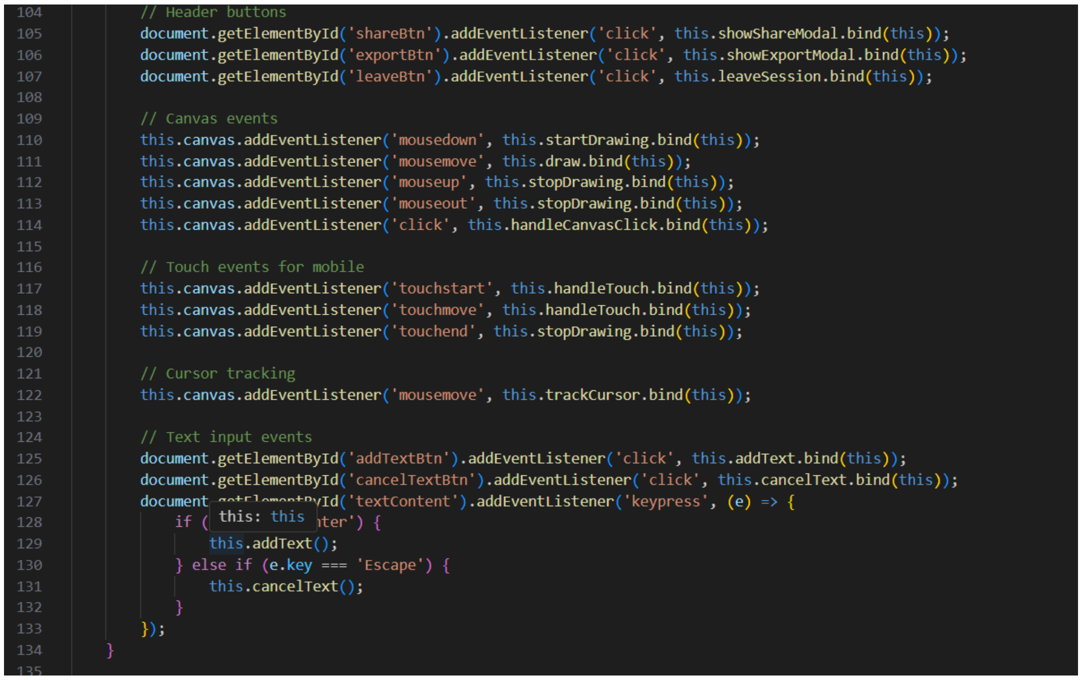
```javascript
    initializeCanvas() {
        // Set standard canvas dimensions that all users will share
        this.standardWidth = 1200;
        this.standardHeight = 800;

        this.canvas.width = this.standardWidth;
        this.canvas.height = this.standardHeight;

        // Set drawing properties
        this.ctx.lineCap = 'round';
        this.ctx.lineJoin = 'round';

        // Handle window resize
        window.addEventListener('resize', this.resizeCanvas.bind(this));
    }

    resizeCanvas() {
        // Save current content
        const imageData = this.ctx.getImageData(0, 0, this.canvas.width, this.canvas.height);

        // Keep the same logical dimensions
        this.canvas.width = this.standardWidth;
        this.canvas.height = this.standardHeight;

        // Restore content
        this.ctx.putImageData(imageData, 0, 0);
        this.ctx.lineCap = 'round';
        this.ctx.lineJoin = 'round';
    }

```

Fig 1.4.2



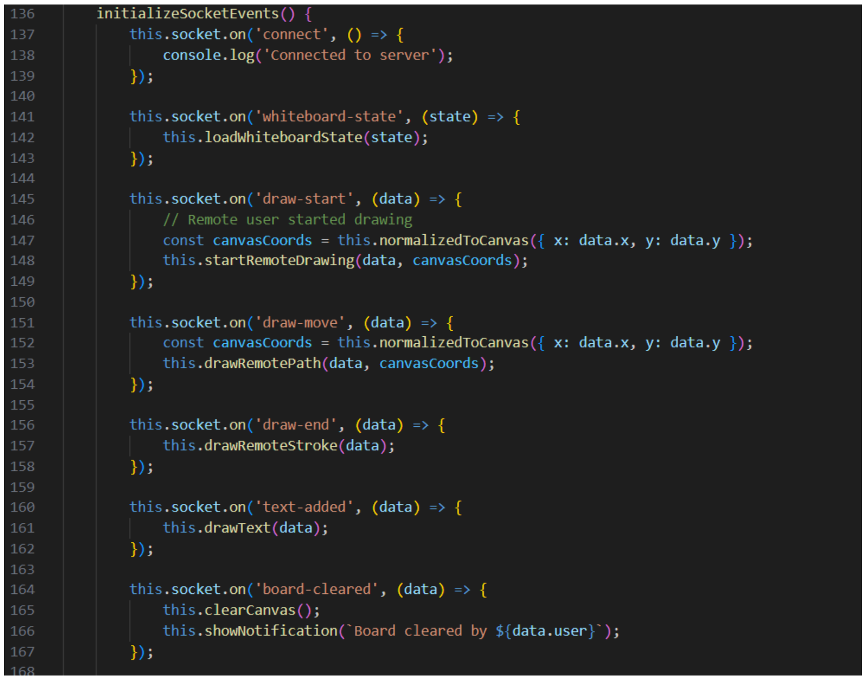
```javascript
    initializeEventListeners() {
        // Tool selection
        document.querySelectorAll('.tool-btn').forEach(btn => {
            btn.addEventListener('click', (e) => {
                this.selectTool(e.target.dataset.tool);
            });
        });

        // Color selection
        document.querySelectorAll('.color-option').forEach(option => {
            option.addEventListener('click', (e) => {
                this.selectColor(e.target.dataset.color);
            });
        });

        document.getElementById('customColor').addEventListener('change', (e) => {
            this.selectColor(e.target.value);
        });

        // Brush size
        document.getElementById('brushSize').addEventListener('input', (e) => {
            this.currentSize = parseInt(e.target.value);
            document.getElementById('sizeValue').textContent = e.target.value + 'px';
        });

        // Action buttons
        document.getElementById('undoBtn').addEventListener('click', () => {
            this.socket.emit('undo');
        });

        document.getElementById('clearBtn').addEventListener('click', () => {
            if (confirm('Are you sure you want to clear the entire board?')) {
                this.socket.emit('clear-board');
            }
        });

```

Fig 1.4.3



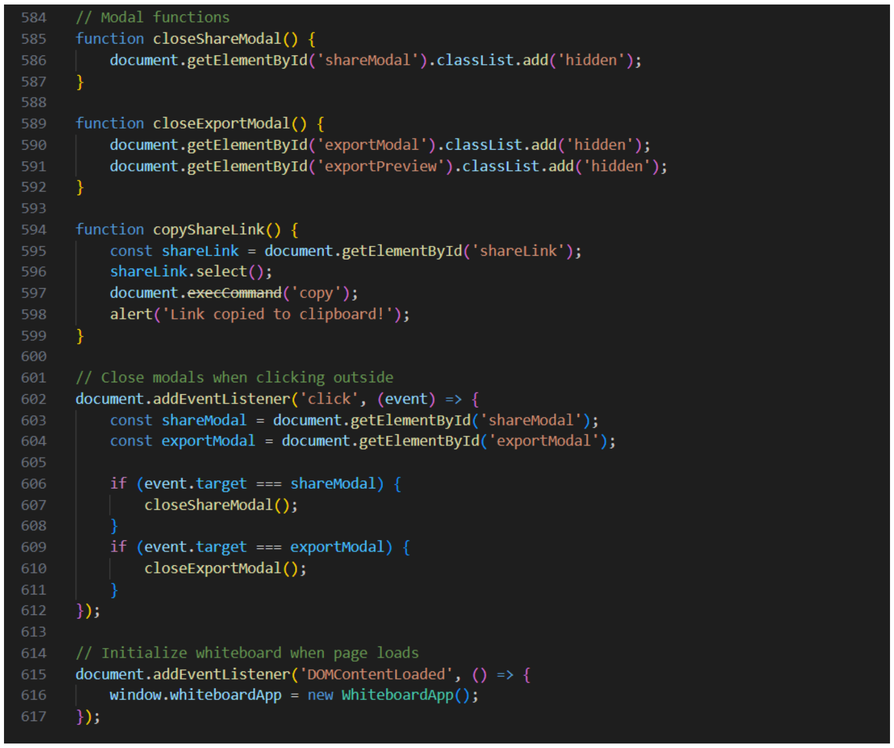
```javascript
        // Header buttons
        document.getElementById('shareBtn').addEventListener('click', this.showShareModal.bind(this));
        document.getElementById('exportBtn').addEventListener('click', this.showExportModal.bind(this));
        document.getElementById('leaveBtn').addEventListener('click', this.leaveSession.bind(this));

        // Canvas events
        this.canvas.addEventListener('mousedown', this.startDrawing.bind(this));
        this.canvas.addEventListener('mousemove', this.draw.bind(this));
        this.canvas.addEventListener('mouseup', this.stopDrawing.bind(this));
        this.canvas.addEventListener('mouseout', this.stopDrawing.bind(this));
        this.canvas.addEventListener('click', this.handleCanvasClick.bind(this));

        // Touch events for mobile
        this.canvas.addEventListener('touchstart', this.handleTouch.bind(this));
        this.canvas.addEventListener('touchmove', this.handleTouch.bind(this));
        this.canvas.addEventListener('touchend', this.stopDrawing.bind(this));

        // Cursor tracking
        this.canvas.addEventListener('mousemove', this.trackCursor.bind(this));

        // Text input events
        document.getElementById('addTextBtn').addEventListener('click', this.addText.bind(this));
        document.getElementById('cancelTextBtn').addEventListener('click', this.cancelText.bind(this));
        document.getElementById('textContent').addEventListener('keypress', (e) => {
            if ( this: this nter') {
                this.addText();
            } else if (e.key === 'Escape') {
                this.cancelText();
            }
        });
    }

```

Fig 1.4.3



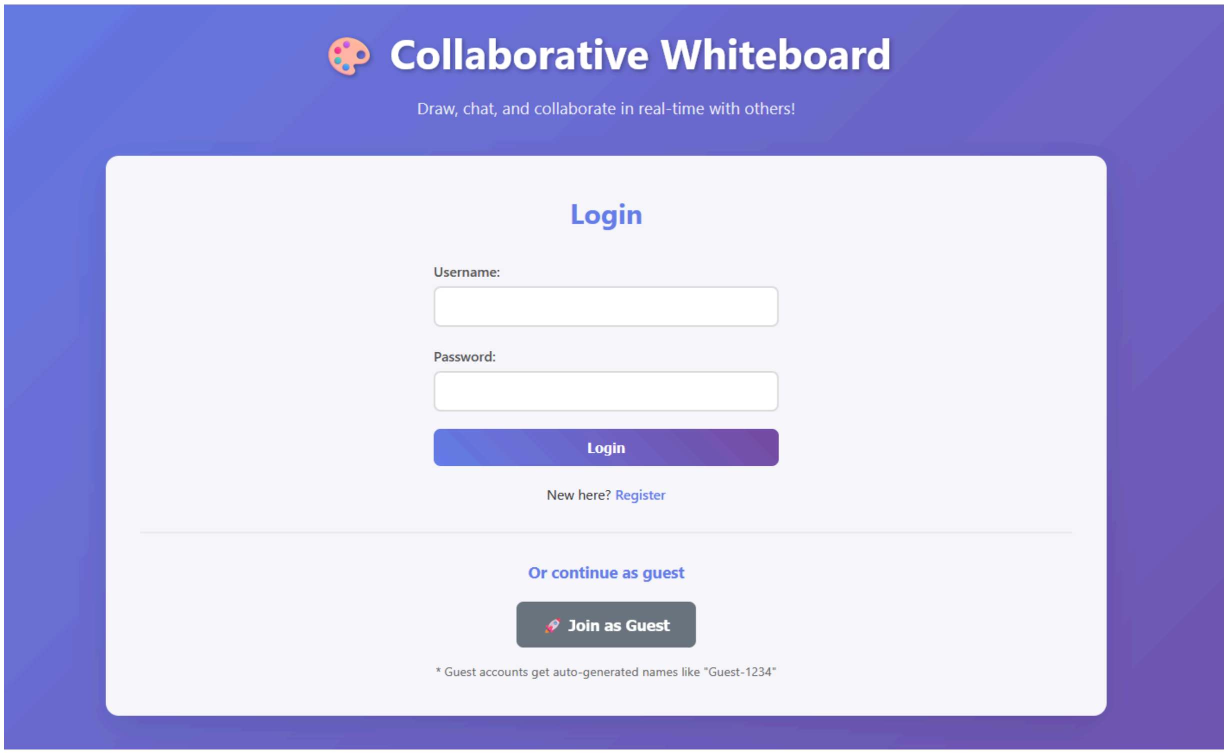
```javascript
    initializeSocketEvents() {
        this.socket.on('connect', () => {
            console.log('Connected to server');
        });

        this.socket.on('whiteboard-state', (state) => {
            this.loadWhiteboardState(state);
        });

        this.socket.on('draw-start', (data) => {
            // Remote user started drawing
            const canvasCoords = this.normalizedToCanvas({ x: data.x, y: data.y });
            this.startRemoteDrawing(data, canvasCoords);
        });

        this.socket.on('draw-move', (data) => {
            const canvasCoords = this.normalizedToCanvas({ x: data.x, y: data.y });
            this.drawRemotePath(data, canvasCoords);
        });

        this.socket.on('draw-end', (data) => {
            this.drawRemoteStroke(data);
        });

        this.socket.on('text-added', (data) => {
            this.drawText(data);
        });

        this.socket.on('board-cleared', (data) => {
            this.clearCanvas();
            this.showNotification(`Board cleared by ${data.user}`);
        });
```

Fig 1.4.4



```javascript
// Modal functions
function closeShareModal() {
    document.getElementById('shareModal').classList.add('hidden');
}

function closeExportModal() {
    document.getElementById('exportModal').classList.add('hidden');
    document.getElementById('exportPreview').classList.add('hidden');
}

function copyShareLink() {
    const shareLink = document.getElementById('shareLink');
    shareLink.select();
    document.~~execCommand~~('copy');
    alert('Link copied to clipboard!');
}

// Close modals when clicking outside
document.addEventListener('click', (event) => {
    const shareModal = document.getElementById('shareModal');
    const exportModal = document.getElementById('exportModal');

    if (event.target === shareModal) {
        closeShareModal();
    }
    if (event.target === exportModal) {
        closeExportModal();
    }
});

// Initialize whiteboard when page loads
document.addEventListener('DOMContentLoaded', () => {
    window.whiteboardApp = new WhiteboardApp();
});
```

Fig 1.4.5


🎨 Collaborative Whiteboard

Draw, chat, and collaborate in real-time with others!

Login

Username:

Password:

Login

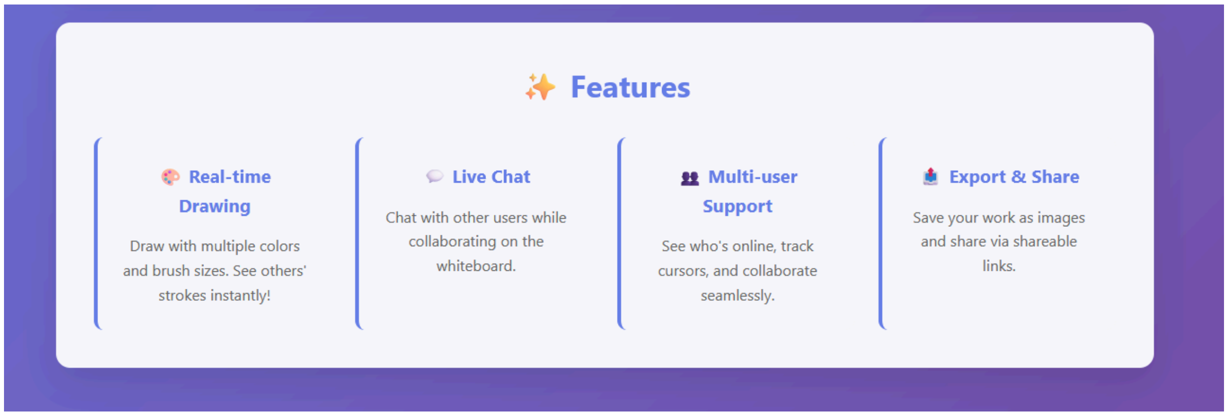
New here? Register

Or continue as guest

🚀 Join as Guest

* Guest accounts get auto-generated names like "Guest-1234"

Fig 2.1


✨ Features

🎨 Real-time Drawing

Draw with multiple colors and brush sizes. See others' strokes instantly!

💬 Live Chat

Chat with other users while collaborating on the whiteboard.

👥 Multi-user Support

See who's online, track cursors, and collaborate seamlessly.

📤 Export & Share

Save your work as images and share via shareable links.

Fig 2.2

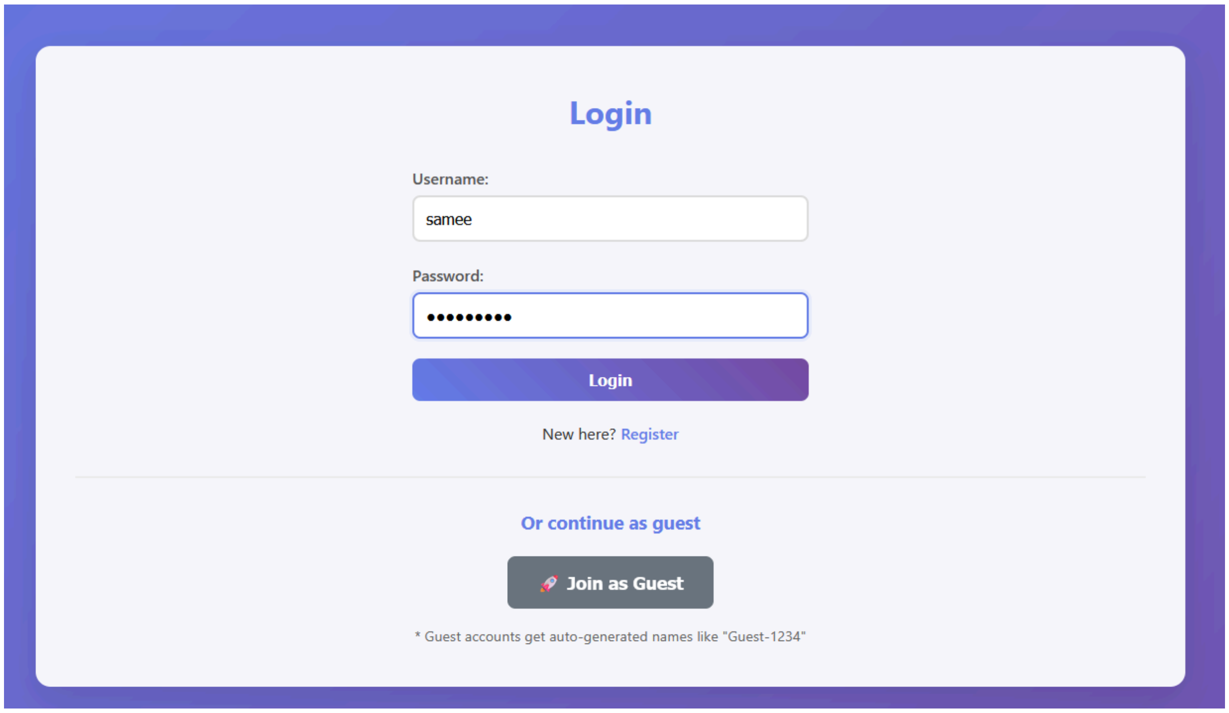

Login
Username:
samee
Password:
•••••••••
Login
New here? Register
Or continue as guest
🚀 Join as Guest
* Guest accounts get auto-generated names like "Guest-1234"

Fig 2.3

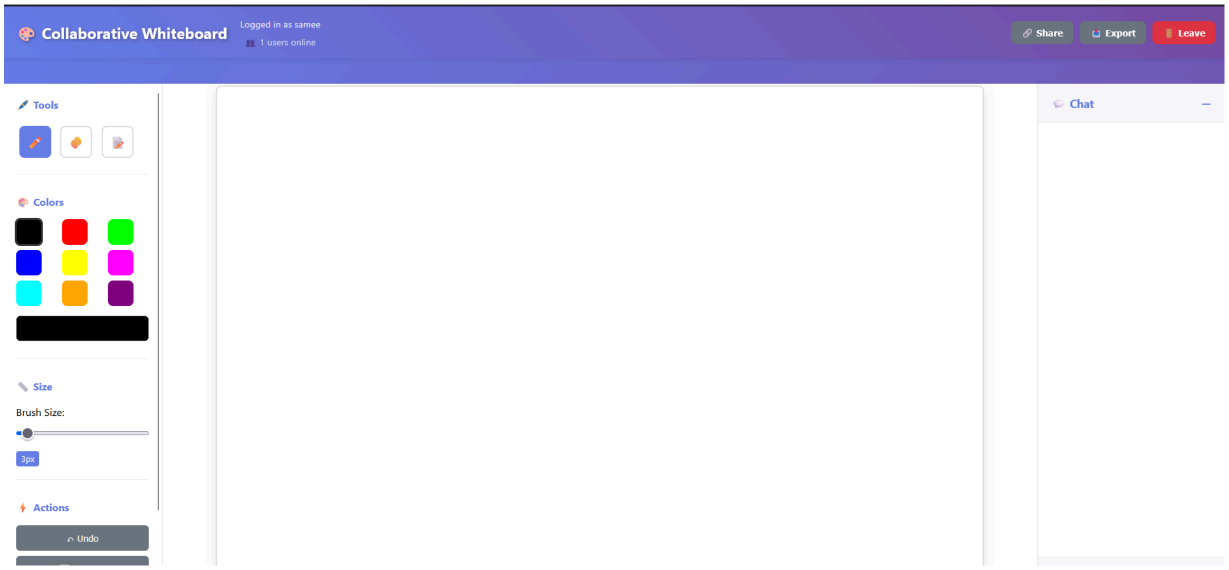

🎨 Collaborative Whiteboard
Logged in as samee
1 users online
Share
Export
Leave
Tools
Colors
Size
Brush Size:
3px
Actions
Undo
Chat

Fig 2.4

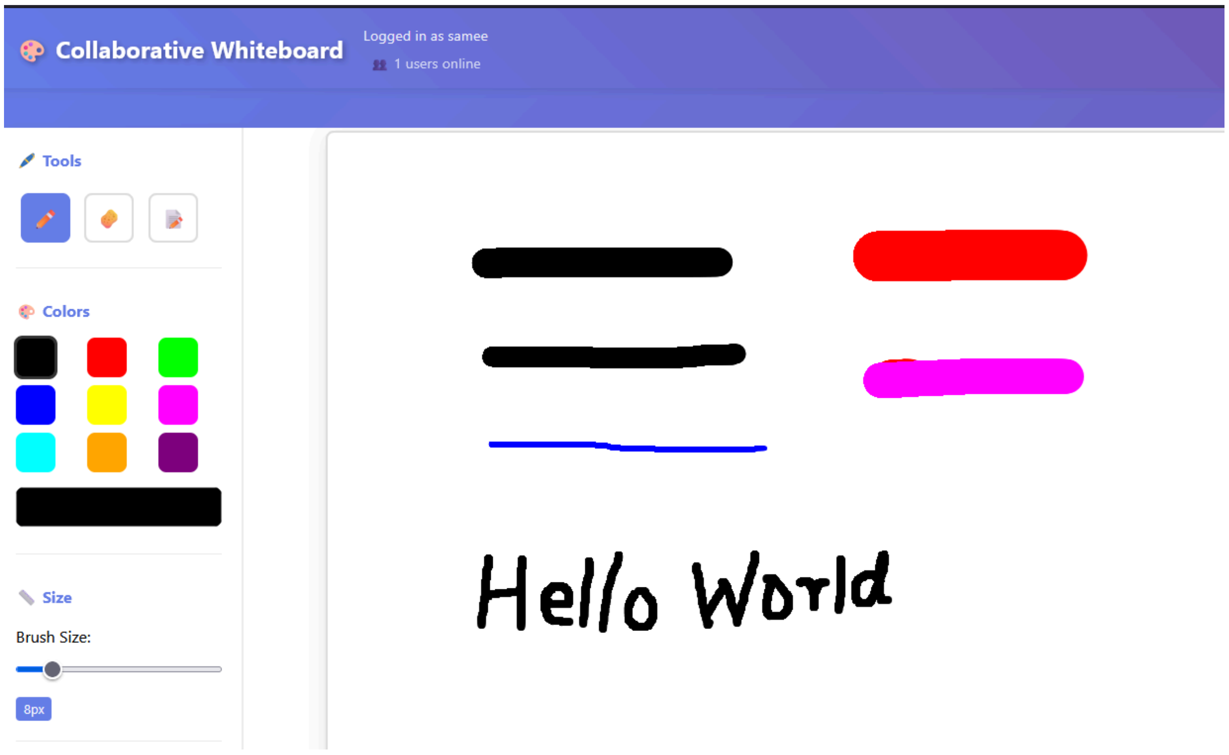

Collaborative Whiteboard
Logged in as samee
1 users online
Tools
Colors
Size
Brush Size:
8px
Hello World

Fig 2.5

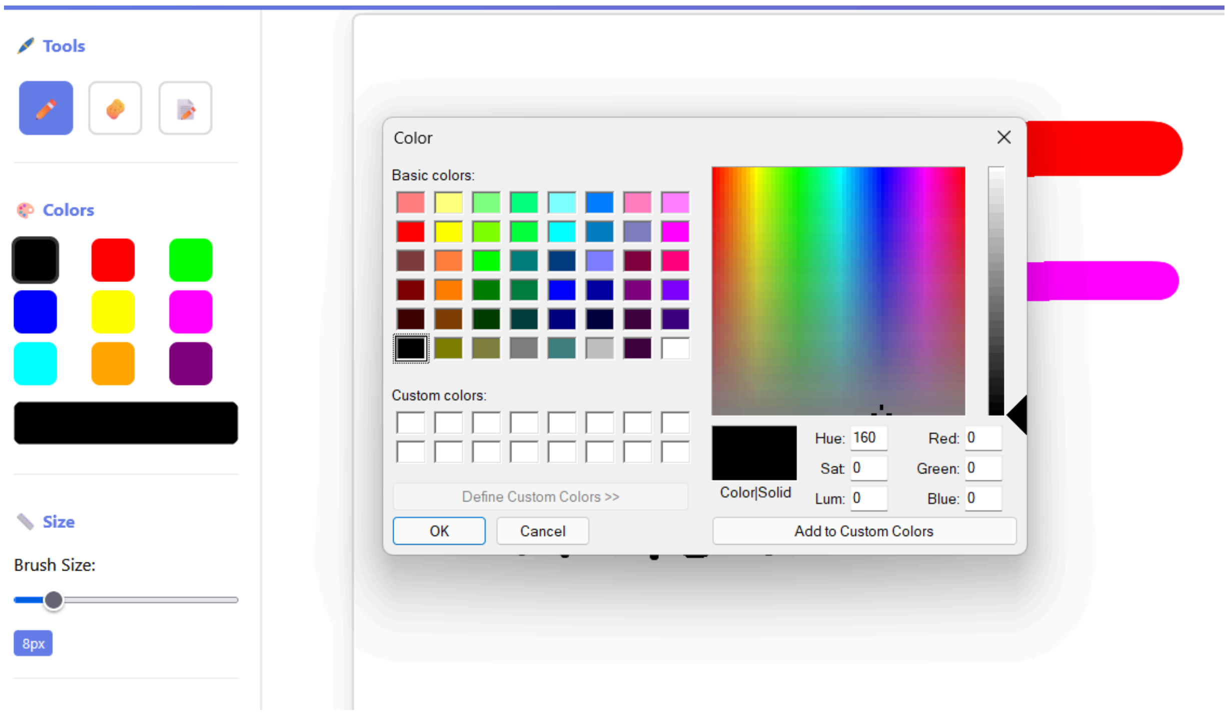
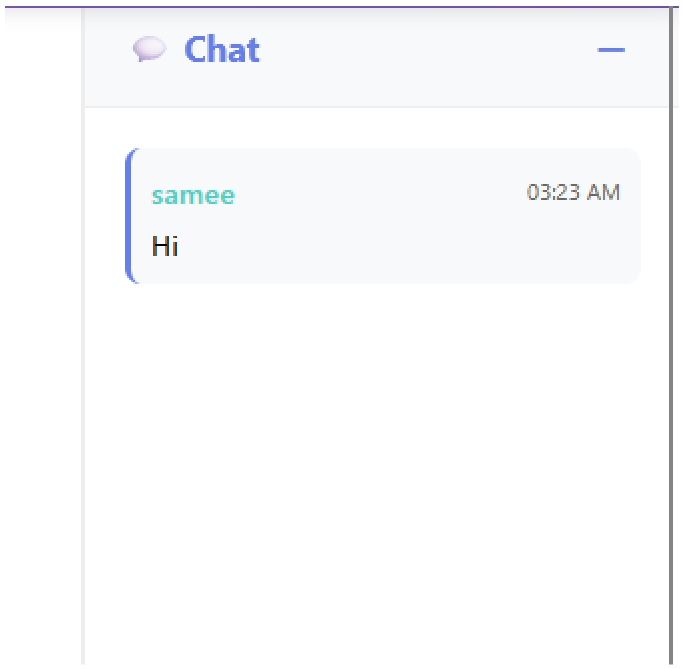
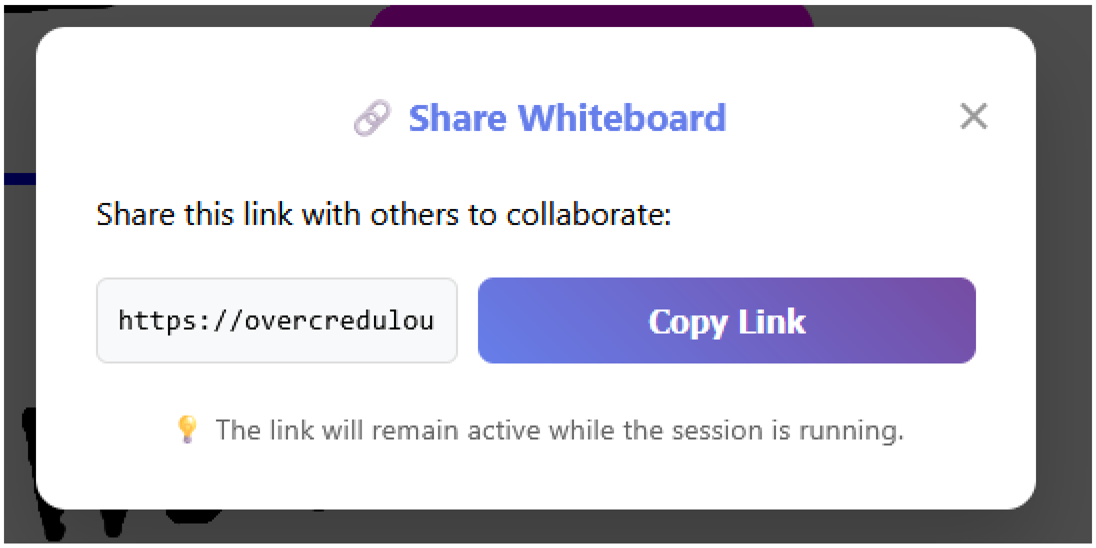

Tools
Colors
Size
Brush Size:
8px
Color
Basic colors:
Custom colors:
Define Custom Colors >>
OK
Cancel
Color|Solid
Hue: 160
Sat: 0
Lum: 0
Red: 0
Green: 0
Blue: 0
Add to Custom Colors

Fig 2.6



Hi

Fig 2.7



Share this link with others to collaborate:

https://overcredulou

The link will remain active while the session is running.

Fig 2.8

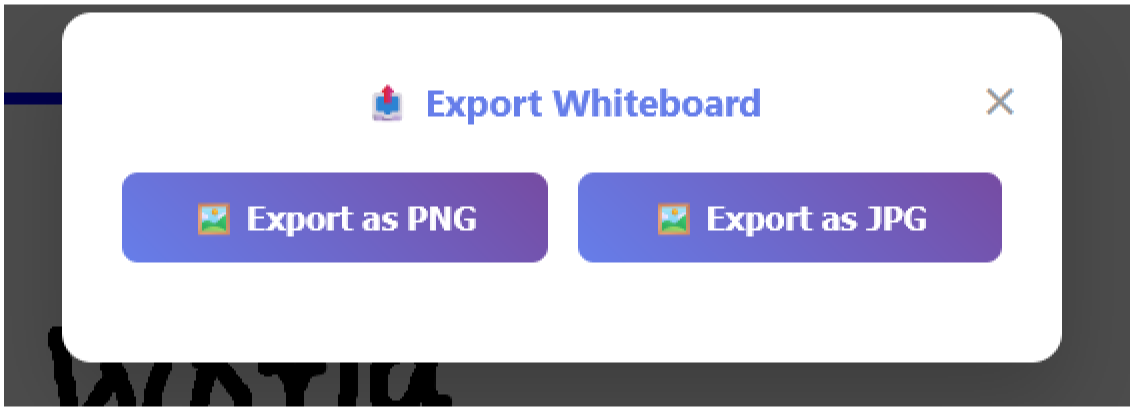


Fig 2.9

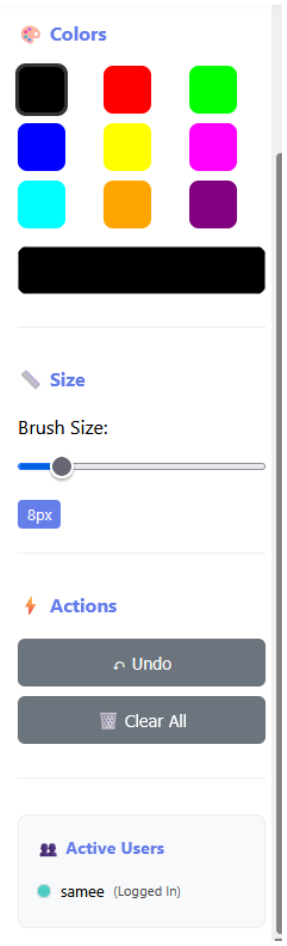


Fig 2.10

Extra Features Implemented (30% Additional Work)

1. Chat Integration

A real-time chat feature was added alongside the whiteboard, allowing participants to communicate while collaborating. Messages are instantly broadcast to all connected users through WebSocket events. This enhances user interactivity and teamwork, transforming the whiteboard from a simple drawing tool into a collaborative workspace.

2. Undo and Clear Controls

Implemented an **Undo** feature that reverses the most recent drawing action without clearing the entire canvas, using an internal action stack. The **Clear** button allows users to wipe the board completely for a fresh start. Both actions are synchronized across all connected clients to maintain shared consistency.

3. Export Whiteboard as Image

Users can save their collaborative work by exporting the current canvas as an image file (PNG format). This allows documentation of brainstorming sessions or design discussions, adding a practical and tangible output to the experiment.

4. Shareable Collaboration Link (via Ngrok)

To make the project globally accessible, **ngrok** was used to tunnel the local WebSocket server to the internet. A live collaboration link was generated, enabling remote users to join the same whiteboard session from different devices or networks.