OpenChat – Real-Time Web-Based Chat Application

1. Executive Summary

OpenChat is a real-time web-based chat application designed to enable seamless, low-latency communication between multiple users through a browser-based interface. The project leverages modern web technologies such as **Node.js**, **Express.js**, and **Socket.IO** to implement a robust client–server architecture capable of handling concurrent users and dynamic chat rooms efficiently.

The application follows an **event-driven WebSocket-based communication model**, allowing instant message delivery, typing indicators, room-based interactions, and real-time user presence updates. Emphasis is placed on **security**, **performance**, and **user experience**, with strong safeguards against common web vulnerabilities such as Cross-Site Scripting (XSS) and spam attacks.

OpenChat supports unique username registration, dynamic room creation, real-time messaging, and responsive UI behavior across desktop and mobile devices. The frontend is deployed as a static application on **Vercel**, while the backend WebSocket server runs on **Render/Railway**, ensuring persistent socket connections and scalability.

From an academic and professional standpoint, OpenChat demonstrates a solid understanding of **real-time systems**, **WebSocket communication**, **state management**, and **secure web application development**. The project highlights practical problem-solving skills, modern development workflows, and deployment strategies suitable for production-ready applications.

2. Technical Architecture Overview

2.1 High-Level Architecture

OpenChat follows a **Client–Server architecture** using persistent WebSocket connections.

Architecture Flow:

Client Browser

↓

Socket.IO Client

↓

WebSocket Connection

↓

Node.js + Express Server

↓

Socket.IO Event Broadcasting

↓

All Connected Clients

2.2 Architectural Justification

- **WebSockets** enable bidirectional, real-time communication
- **Socket.IO** abstracts transport fallback and connection reliability
- **Express.js** provides a lightweight HTTP server for configuration and middleware
- **Client-server separation** enables independent frontend and backend deployment

---

3. Technology Stack Overview

Backend Stack

- Node.js (v14+)
- Express.js (v4.18.2)
- Socket.IO (v4.6.1)
- Validator.js (v13.11.0)
- Nodemon & LiveReload (Development)

Frontend Stack

- HTML5 (Semantic Markup)
- CSS3 (Flexbox, Grid, Animations, Glassmorphism)
- JavaScript (ES6+)
- Socket.IO Client

---

4. Implementation Details

4.1 User Authentication System

- Enforces **unique usernames** (2–20 characters)
- Regex-based validation for allowed characters
- Duplicate prevention using Set
- Sanitization using Validator.js
- User session tied to Socket.IO connection ID

Reasoning:
Avoids password complexity while focusing on real-time identity management.

4.2 Real-Time Messaging System

- Messages transmitted via Socket.IO events
- Broadcasts scoped to rooms
- Timestamped message rendering
- Auto-scroll ensures latest messages are visible
- Rate limiting prevents spam (10 messages / 5 seconds)

4.3 Room Management System

- Dynamic room creation and deletion
- Default **“General”** room always active
- Empty rooms auto-cleaned
- Live room list synchronization
- User count tracked per room

Data Structure Used:

- Map → Room metadata
- Set → Room participants

4.4 User Experience Enhancements

- Typing indicators with timeout-based reset
- Join/leave system messages
- Desktop notifications for background messages
- Mobile-responsive sidebar with toggle button
- Real-time user list per room

---

5. Security Analysis

Security Measures Implemented

- HTML escaping for all user-generated content
- Script tag removal to prevent XSS
- Input validation via Validator.js
- Rate limiting to prevent spam and abuse
- CORS configuration for controlled access

- No client-side trust assumptions

Security Rationale:
Real-time apps are highly vulnerable to abuse; strict validation ensures stability and trustworthiness.

---

6. Challenges Faced and Solutions

Challenge	Solution
Handling duplicate usernames	Set-based uniqueness tracking
Preventing message spam	Rate limiting with timestamps
Managing room lifecycle	Auto-create and auto-delete logic
Avoiding XSS attacks	Validator.js + HTML escaping
Mobile responsiveness	CSS media queries + sidebar toggle

---

7. Testing Strategy and Quality Assurance

- Manual testing for real-time message flow
- Edge case testing (disconnects, invalid inputs)
- Stress testing with multiple browser tabs
- Cross-browser compatibility checks
- UI responsiveness testing on different screen sizes

---

8. Deployment and Production Considerations

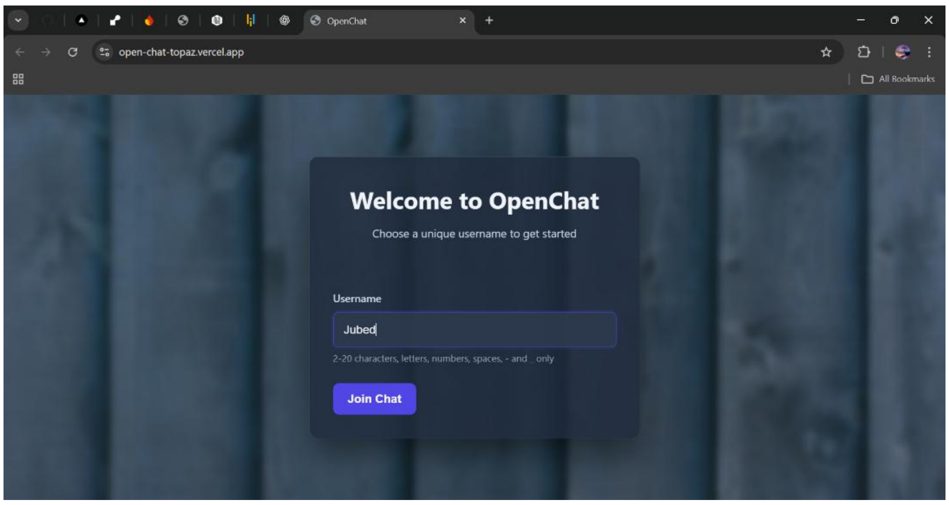
Deployment Strategy

- **Frontend:** Vercel (static hosting, CDN optimized)
- **Backend:** Render (persistent WebSocket support)
- Environment-based server URL detection
- Production-ready WebSocket configuration

Benefits

- Independent scaling
- Faster frontend delivery
- Reliable socket connections

9. Project Visuals


open-chat-topaz.vercel.app

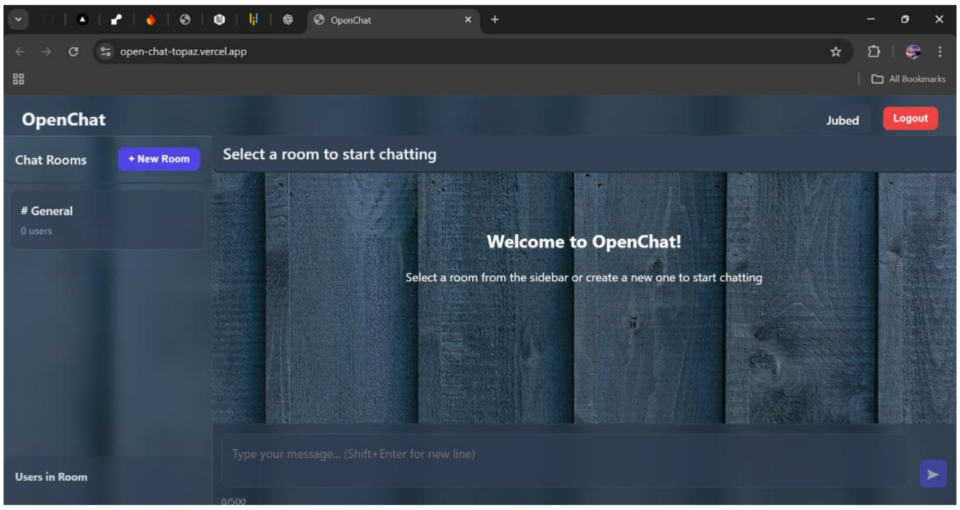
Welcome to OpenChat

Choose a unique username to get started

Username: Jubed

2-20 characters, letters, numbers, spaces, - and _ only

Figure 1: Login / Username screen


open-chat-topaz.vercel.app

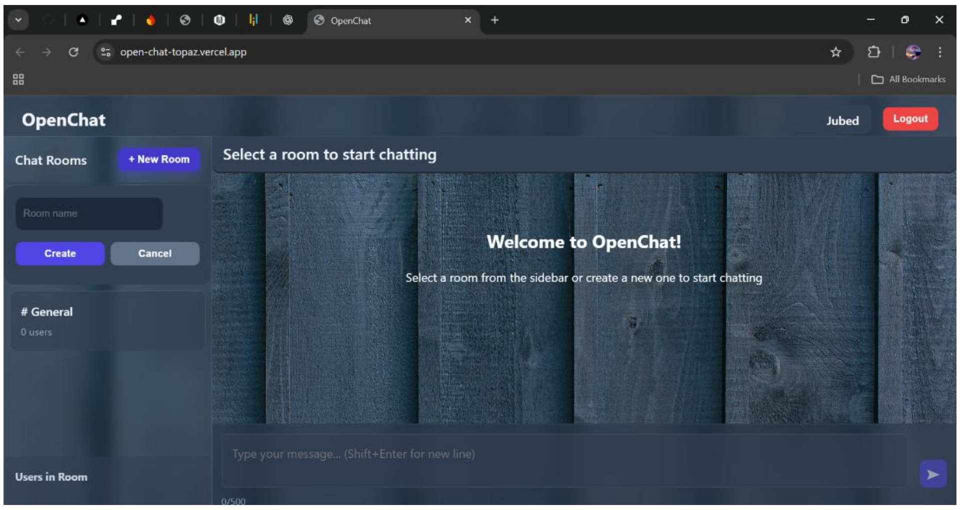
Select a room to start chatting

Welcome to OpenChat!

Select a room from the sidebar or create a new one to start chatting

Type your message... (Shift+Enter for new line)

Figure 2: Main chat interface


open-chat-topaz.vercel.app

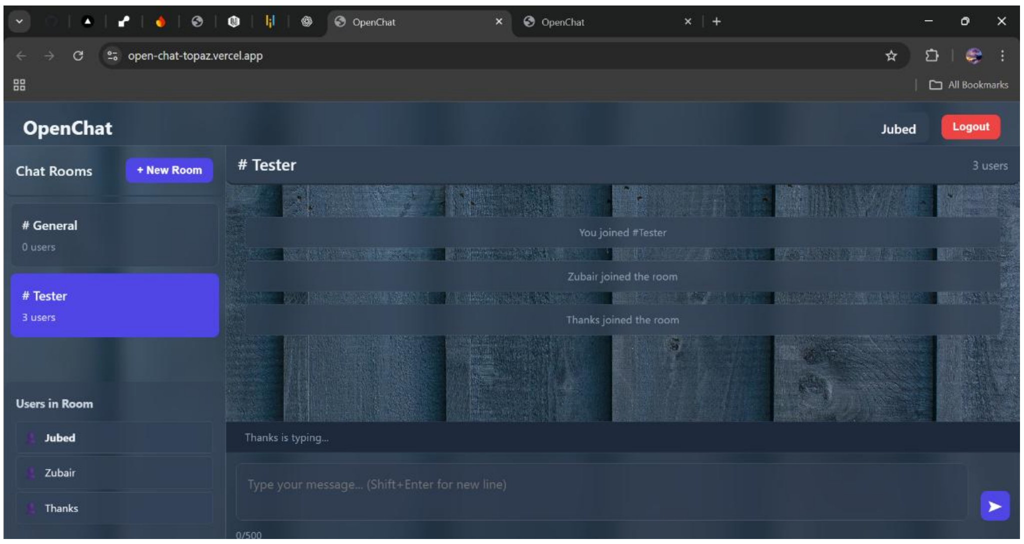
Select a room to start chatting

Welcome to OpenChat!

Select a room from the sidebar or create a new one to start chatting

Type your message... (Shift+Enter for new line)

Figure 3: Room creation UI



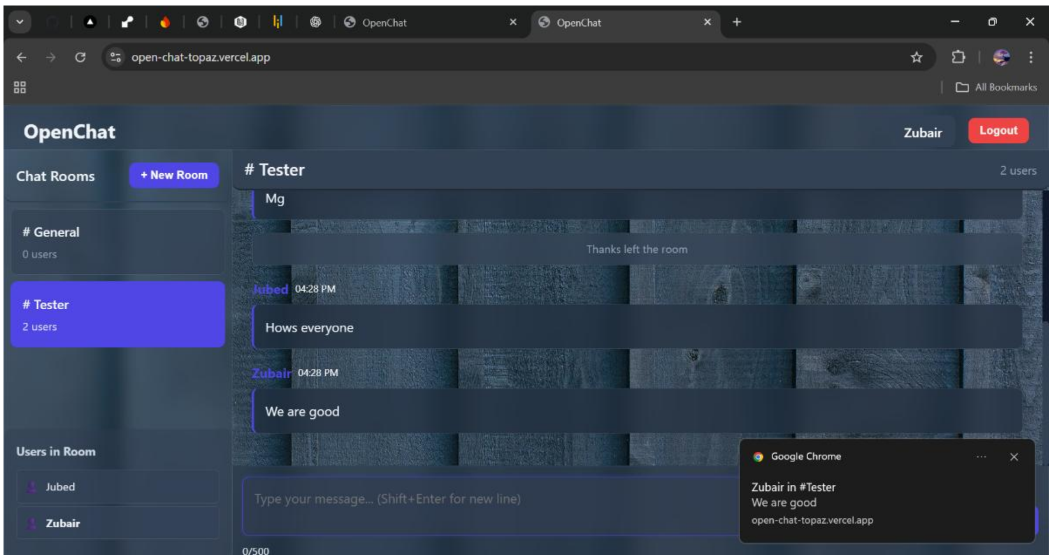
You joined #Tester

Zubair joined the room

Thanks joined the room

Thanks is typing...

Figure 4: Active typing indicator



Mg

Thanks left the room

Jubed 04:28 PM

Hows everyone

Zubair 04:28 PM

We are good

Google Chrome

Zubair in #Tester
We are good
open-chat-topaz.vercel.app

Figure 5: Default Chrome Notification and Users List

10. Conclusion

OpenChat demonstrates a strong understanding of **real-time systems**, **WebSocket communication**, and **secure frontend-backend integration**. The project successfully balances performance, security, and user experience while adhering to modern web development best practices.

Key lessons include efficient state management, the importance of sanitization in real-time applications, and scalable architectural design. OpenChat serves as a solid foundation for advanced real-time platforms and reflects industry-ready development skills.