CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Today, in the age of digitalization, real-time collaboration is a necessity for the efficiency and productivity of a team. Google Docs and Microsoft Office 365 are cloud-based document editors that have changed the way people work together. However, these platforms have their disadvantages, such as dependence on a centralized server and the problem of delay in latency. The platforms need the servers to be available all the time, and the bottlenecks can limit the scalability, especially when a large number of users are working at the same time.

To address these challenges, decentralized and P2P collaboration models have come into the spotlight. WebRTC provides P2P communication. This eliminates the need for servers. Data synchronization in distributed environments can be done using CRDTs such as Yjs. Therefore, a distributed, scalable, fault-tolerant collaborative editing system can be developed.

1.2 PROBLEM STATEMENT

Most of the real-time document collaboration tools that people have now are server dependent. This leads to:

- High infrastructure: The maintenance of centralized servers is not easy. Real-time editing requires significant resources.
- Scalability Limitations: Server-based architectures have a hard time efficiently managing millions of concurrent users.
- Latency Issues: When users are geographically separated, they may experience delays because of server-based synchronization.

Decentralized P2P document collaboration systems can solve these problems by allowing direct communication between users, reducing dependence on central servers, and ensuring efficient data synchronization through CRDTs.

1.3 OBJECTIVES AND CONTRIBUTIONS

LiveDocs aims to provide a decentralized, real-time collaborative document editor that has no dependence on a central server, ensuring low-latency, fault-tolerant, and scalable collaboration. The main contributions of this project are as follows:

- Decentralized P2P Communication: Using WebRTC technology to allow users to communicate directly, minimizing reliance on central servers.
- Conflict-free Real-Time Collaboration: Implementing Yjs (CRDTs) and using its implementation for shared state.
- Authentication Access Control: Adding JWT-based authentication for user security and access control
- Modern and Reliable User Experience: A feature-rich modern text editing UI with real-time updates and granular collaborative editing features.

By this, LiveDocs provides a simple, cost-effective, scalable, and secure alternative to traditional cloud-based document editors, where the centralized server has been eliminated, thus making it the best fit for enterprises, research teams, and large-scale collaboration use cases.

CHAPTER 2

LITERATURE REVIEW

2.1 SURVEY OF RELATED WORK

1. Title: “Performance of real-time collaborative editors at large scale: User perspective”

Authors: Q. -V. Dang and C. -L. Ignat

Year: 2016

Description:

Real-time collaborative editing allows multiple users to edit a shared document at the same time. It received a lot of attention from both industry and academia and gained in popularity due to the wide availability of free services such as Google Docs. While these collaborative editing systems were initially used in scenarios involving only a small set of users such as for writing a research article, nowadays authors notice a change in the scale from several users to communities of users. Group note taking during lectures or conferences is an emerging practice. Delays exist between the execution of one user modification and the visibility of this modification to the other users. They can be caused by network physical communication media, complexity of consistency maintenance algorithms and system architecture. Some user studies have shown that delay affects group performance in collaborative editing. In this paper, authors measure delays in popular real-time collaborative editing systems such as Google Docs and Etherpad and authors study whether these systems could cope with large scale settings from a user perspective. Our results show that these systems are not yet ready for large-scale collaborative activities as either they reject new users connection or a high delay appears when facing an increasing number of users or their typing speeds in the same shared document.

2. Title: “Collaborative Code Editors - Enabling Real-Time Multi-User Coding and Knowledge Sharing”

Authors: K. Virdi, A. L. Yadav, A. A. Gadoo and N. S. Talwandi

Year: 2023

Description:

In today’s dynamic landscape of software development, collaborative code editors have emerged as essential tools for enabling real-time multi-user coding and fostering knowledge sharing among distributed teams. This research paper delves into the intricate fabric of collaborative code editing, aiming to unravel the underlying technologies, challenges, and advancements in this field. The study begins with a comprehensive literature review, tracing the evolution of collaborative code editors and examining the key technologies and protocols employed in this domain. This research study proceeds to outline a detailed methodology, including the selection criteria for technology analysis and the implementation strategy for the collaborative code editor prototype. It discusses the design architecture, the integration of WebSocket communication, and Operational Transformation (OT) algorithms, and strategies for conflict resolution within the prototype. The research presents the findings of performance evaluation metrics and user experience feedback, along with a comparative analysis of the prototype with existing platforms. The implications of the findings on software development practices and global collaboration are thoroughly discussed, highlighting the potential impact of collaborative code editors on enhancing productivity and fostering a culture of collective creativity within development teams. The paper concludes with recommendations for future research and development in this field, emphasizing the need for continued innovation and the exploration of novel strategies to address the persistent challenges faced in collaborative code editing environments

3. Title: “SecureC2Edit: A Framework for Secure Collaborative and Concurrent Document Editing”

Authors: S. Arora and P. K. Atrey

Year: 2024

Description:

Cloud-based online document editing services, such as Google Docs and Office 365, provide an inexpensive and efficient means of managing documents. However, storing data on the cloud also raises certain security and privacy concerns, especially when the data is of confidential and sensitive nature. Authors argue that in online editing environments, user data should never be exposed to the cloud in plaintext form. Significant prior work around secure collaborative editing has several limitations and is not practical. Thus, in this paper, authors propose a secure online editing framework, called SecureC2Edit, which is based on structured peer-to-peer architecture, uses hybrid differential synchronization, allows collaborative and concurrent access to the document, yet ensures security and privacy by encrypting the data before storing it on the cloud. The framework is evaluated for security, operability, and performance.

4. Title: “A Personal Distributed Real-time Collaborative System”

Authors: M. Konstantopoulos, N. Chondros and M. Roussopoulos

Year: 2020

Description:

In this paper, authors present O3 REAL, a privacy-preserving distributed middleware for real-time collaborative editing of documents. O3 REAL introduces a novel approach for building peer-to-peer real-time collaborative applications, using a reliable broadcast channel mechanism for network communication, but at the same time provides for persistent storage management

of collaborative documents using the filesystem interface of a POSIX compliant filesystem. This approach enables real-time, completely decentralized collaboration among users, without the need for a third party to intervene, and significantly simplifies the creation of peer-to-peer collaborative applications. Authors demonstrate that O3 REAL scales well for real-time collaboration use-cases. For example, with 33 users simultaneously collaborating on a document in real time over a WAN with a 50 ms link delay, the average perceived latency is approximately 54 ms, which is very close to the optimal baseline. In comparison, Etherpad exhibits nearly twice the perceived latency.

2.2 OVERVIEW OF EXISTING SYSTEMS

Real-time collaborative editing has emerged as a critical area of research and development, enabling multiple users to modify a shared document simultaneously. Prominent existing systems, such as Google Docs and Etherpad, utilize centralized architectures to maintain document consistency and ensure availability [1]. These platforms have gained widespread adoption due to their user-friendly interfaces and seamless cloud integration. However, as the number of concurrent users increases, these systems experience scalability issues, leading to performance degradation and increased latency [1].

Collaborative coding environments have also seen advancements, particularly in enabling real-time multi-user interactions. Virdi et al. [2] examined various collaborative code editors, which employ WebSocket-based communication and OT algorithms for maintaining synchronization. Despite the benefits of these systems in fostering real-time collaboration among distributed software development teams, persistent issues such as synchronization delays and conflict resolution complexities hinder their efficiency at scale [2].

Security and privacy remain significant concerns in real-time collaborative editing. Arora and Atrey [3] proposed SecureC2Edit, a secure collaborative

editing framework that encrypts user data before storing it on cloud servers. This framework enhances data privacy by ensuring that plain text user content is never exposed to cloud-based storage providers. However, encryption-based solutions introduce additional computational overhead, which may impact the responsiveness of real-time collaboration [3].

A shift towards decentralized collaborative editing systems has been proposed as an alternative to centralized platforms. Konstantopoulos et al. [4] introduced O3 REAL, a P2P system for real-time collaboration. O3 REAL employs a reliable broadcast mechanism for network communication and utilizes a distributed filesystem for document storage. This decentralized approach eliminates reliance on third-party servers, thereby improving fault tolerance and scalability [4]. Notably, O3 REAL demonstrated low latency and high efficiency in collaborative editing, making it a promising direction for future real-time collaborative systems.

2.3 LIMITATIONS OF CURRENT APPROACHES

Despite significant progress in real-time collaborative editing, existing solutions exhibit notable limitations:

- Scalability and Performance Constraints: As noted by Dang and Ignat [1], mainstream collaborative editing platforms such as Google Docs and Etherpad struggle with increased user concurrency, leading to higher latency. This negatively affects user experience and restricts their applicability in large-scale collaborative settings.
- Synchronization and Consistency Challenges: Collaborative code editors, as discussed by Virdi et al. [2], rely on OT-based synchronization mechanisms, which become computationally expensive and complex when handling simultaneous modifications from multiple users.

- Security and Privacy Risks: Cloud-based collaborative systems expose sensitive user data to potential security breaches. While SecureC2Edit [3] mitigates these risks through encryption, it introduces additional processing overhead that impacts real-time responsiveness.
- Dependence on Centralized Infrastructure: The reliance on cloud-based infrastructures results in single points of failure, higher operational costs, and potential vendor lock-in. While decentralized solutions such as O3 REAL [4] address some of these concerns, they do not fully resolve real-time synchronization challenges over P2P networks.

2.4 POSITIONING OF THE PROPOSED WORK

LiveDocs is designed to address the limitations of existing real-time collaborative editing systems by leveraging a decentralized P2P architecture and CRDTs for efficient synchronization. Unlike Google Docs and Etherpad, which rely on cloud-based architectures and experience increased latency with higher user loads, LiveDocs utilizes WebRTC to enable direct P2P communication, reducing server dependency and minimizing delays.

Collaborative coding environments predominantly use OT-based synchronization; however, LiveDocs adopts Yjs, a CRDT-based approach that simplifies conflict resolution while maintaining real-time consistency. This methodology eliminates the complexity associated with traditional operational transformations and improves performance in multi-user editing scenarios. Furthermore, LiveDocs enhances security and privacy by eliminating centralized servers, ensuring that document content remains confined within the P2P network. This decentralized model builds upon the principles introduced by O3 REAL [4], while integrating PostgreSQL for persistence mechanisms to improve data reliability and synchronization efficiency.

By incorporating a robust P2P communication model, CRDT-based synchronization, and enhanced security measures, LiveDocs presents a novel solution that surpasses the scalability, performance, and privacy limitations of existing real-time collaborative editing systems. This framework provides a viable alternative to traditional centralized approaches, paving the way for a more efficient and secure collaborative editing experience.

CHAPTER 3

SYSTEM ARCHITECTURE

3.1 OVERVIEW OF LIVEDOCS

LiveDocs is a P2P real-time collaborative document editor, which means that multiple users can edit the document at the same time, and there is no centralized server involved for collaboration. It uses WebRTC to connect users to each other and Yjs (CRDTs) to sync data that won't conflict. This decentralized architecture allows for low latency, fault tolerance, and scalability. There are three main components of the system:

- Client-Side Application: A web-based text editor offering a real-time collaborative, modern user interface.
- Real-Time Collaboration Module: Uses WebRTC for a P2P communication channel and Yjs for efficient data synchronization.
- Authentication & Access Control: Uses JWT for authentication and role-based permissions for secured document access.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

The development and deployment of LiveDocs require standard modern hardware for both local development and production hosting. For local development, a machine with at least an Intel i5 or AMD Ryzen 5 processor, 8 GB of RAM, and a 256 GB SSD is recommended. However, for an optimal development experience, especially when running multiple services such as the frontend, backend, PostgreSQL server, and development tools concurrently, a system with an Intel i7 or AMD Ryzen 7 processor, 16 GB of RAM, and a 512 GB SSD is ideal. A stable internet connection is necessary to test WebRTC peer-

to-peer features, which require real-time network communication between multiple devices.

For production deployment, a virtual private server (VPS) or cloud-based hosting environment with a minimum of 2 virtual CPUs (vCPUs), 4 GB of RAM, and 20 GB of SSD storage is sufficient to support a small to medium-scale deployment. However, for better performance and scalability, especially when handling multiple concurrent peer-to-peer connections and database interactions, a recommended configuration would include 4 or more vCPUs, 8/16 GB of RAM, and scalable SSD storage. The server should have a high-bandwidth internet connection to facilitate low-latency document synchronization among clients.

Software Requirements:

The LiveDocs platform is built using a modern web development stack. For the backend and frontend, Node.js (version 18 LTS or higher) is used in conjunction with the Next.js framework (version 13 or higher) utilizing the App Router. Package management is handled using npm or pnpm, and TypeScript is used throughout the codebase for type safety. The frontend styling is built using Tailwind CSS version 4, while UI components are powered by the ShadCN/UI library. The rich text editing experience is provided through the TipTap editor, which is based on ProseMirror.

Prisma ORM is used to interface with a PostgreSQL database (version 14 or higher), enabling robust data modeling and efficient querying. Authentication and session management are implemented using bcryptjs for password hashing, jsonwebtoken for JWT-based authentication, and resend for handling OTP-based email verification. Input validation across the application is handled using Zod, and toast notifications are displayed using Sonner for an improved user experience. Additionally, Framer Motion is used for animations to provide a modern and smooth user interface.

For real-time collaboration, the platform integrates WebRTC for peer-to-peer communication and Yjs, a CRDT-based framework, to handle distributed data synchronization. The y-webrtc connector is used to enable synchronization between peers without requiring a central server. Furthermore, essential browser APIs are used to manage session storage, WebRTC signaling, and real-time communication features.

Version control is maintained through Git, with repositories hosted on platforms such as GitHub or GitLab. The recommended development environment is Visual Studio Code, enhanced with extensions for Prettier, ESLint, Tailwind CSS, and Prisma. Together, these tools create a robust environment for developing, debugging, and deploying LiveDocs efficiently.

3.3 ARCHITECTURAL DESIGN

In the architecture of LiveDocs, every module is designed to achieve collaboration without compromising efficiency, security, and scalability.

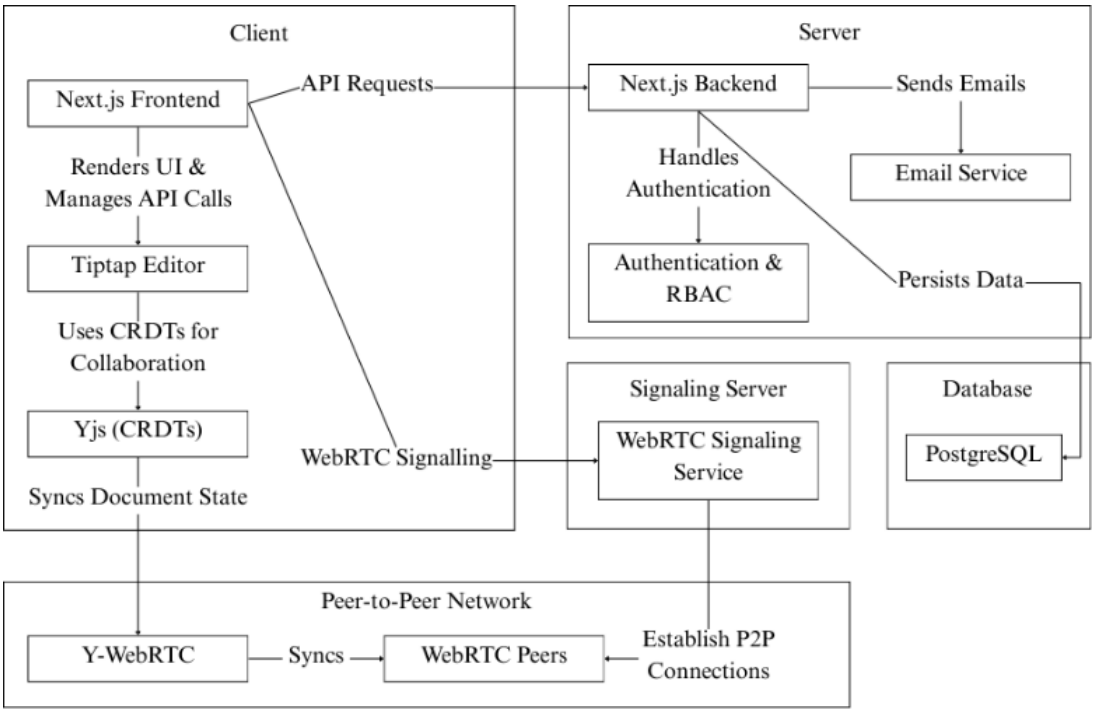

Figure 3.1 Architecture Diagram

Client-Side Components:

The client side of LiveDocs consists of:

- Tiptap-based Text Editor: Provides a rich text editing experience that supports headings, lists, text formatting, and more, with a real-time collaborative experience.
- WebRTC Signaling Mechanism: Establishes P2P connections using a preliminary signaling process (like WebSocket or a lightweight signaling server).
- Yjs Document Synchronization: Provides real-time updates for text changes and efficiently performs operation merging with CRDTs.
- UI Components: Encompasses document management, notifications, and user collaboration tools.

Real-Time Collaboration Module:

The real-time collaboration module is responsible for synchronizing document changes across users. It works as follows:

- WebRTC Connection Establishment: When a user joins a document session, a WebRTC connection is established with other peers.
- Change Detection and Distribution: User edits are captured and distributed to other peers using Yjs.
- Conflict-Free Merging: CRDTs ensure that all document changes are merged seamlessly without conflicts.

P2P Network Topology:

LiveDocs supports different P2P topologies:

- Mesh Network: Each client directly connects to others, ideal for small groups.

- Hybrid P2P with a Lightweight Server: A signaling server helps establish connections but does not handle document data.
- Distributed Relay Mechanism: If direct P2P connections fail due to network restrictions, TURN servers can be used as a fallback.

3.4 DATA SYNCHRONIZATION USING YJS

Yjs (a CRDT library) is used for:

- Efficient Document State Management: Ensures that all peers have a consistent version of the document.
- Optimized Data Transmission: Uses delta-based updates to minimize network overhead.

3.5 AUTHENTICATION AND ACCESS CONTROL

To ensure security, LiveDocs implements:

- JWT-based Authentication: Users authenticate using JWT to gain access.
- Role-Based Access Control: Different permission levels (owner, editor, viewer) are assigned to users.
- Secure Document Sharing: Documents can be shared via encrypted links with access restrictions.

3.6 SCALABILITY AND FAULT TOLERANCE

LiveDocs is designed to handle a high number of users efficiently:

- Decentralized Collaboration: Since there are no central servers, the system scales naturally with more peers.
- Resilient Data Synchronization: Yjs ensures that document consistency is maintained even in cases of network failures.
- Fallback Mechanisms: If direct WebRTC connections fail, alternative routing methods like STUN/TURN servers are used.

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 TECHNOLOGY STACK

LiveDocs is built using modern web technologies to ensure efficient real-time collaboration, scalability, and security. The technology stack includes:

- Frontend: Next.js (React), TypeScript, Tailwind CSS, Tiptap (rich-text editor)
- Backend: Next.js API, PostgreSQL (with an ORM like Prisma)
- Real-Time Collaboration: WebRTC for P2P communication, Yjs (CRDTs) for conflict-free synchronization
- Authentication & Security: JWT-based authentication, role-based access control

Building a modern, real-time collaborative editor involves a blend of advanced frontend, backend, and real-time synchronization technologies. At the heart of the system lies Next.js, chosen for its robust React-based architecture, server-side rendering capabilities, and seamless integration with modern development workflows. The frontend is designed around a custom text editor, built using TipTap, a highly extensible rich-text editor framework based on ProseMirror. This choice provides fine-grained control over editor behaviour and formatting, making it ideal for tailored collaborative experiences.

To ensure real-time collaboration, the editor integrates with Yjs, a powerful CRDT (Conflict-free Replicated Data Type) library. Yjs enables concurrent editing by multiple users, resolving conflicts automatically and maintaining consistency across all peers. As edits are made, changes are broadcasted instantly using WebRTC, a peer-to-peer communication protocol that eliminates the need for centralized relays and dramatically reduces latency. By using the y-webrtc

connector, the application establishes efficient, decentralized synchronization channels between users, creating a smooth, real-time editing experience even over weak or unstable network conditions.

Security is a key consideration in this collaborative platform. To control access and protect user sessions, the system uses JWT (JSON Web Tokens) for authentication. When a user logs in, the server issues a signed token, which the frontend stores and attaches to subsequent requests. This ensures that only authorized users can access and modify documents, and their actions are traceable to authenticated identities. Role-based access control can be layered on top of this, enabling fine-grained permissions for instance, distinguishing between editors, viewers, and admins. This becomes especially important when integrating presence features, such as showing who is currently viewing or editing the document and enforcing different roles within a collaborative session.

Beyond real-time sync, long-term storage and reliability are handled by a PostgreSQL database. While Yjs maintains the document state in-memory and syncs it across peers in real time, the persistent version of each document is periodically stored in PostgreSQL for backup, history tracking, and quick access when users reconnect. This dual-layer architecture (in-memory sync + persistent backup) ensures both high performance and fault tolerance. If a user goes offline or refreshes the page, the latest saved state from the database can be restored instantly, with Yjs resuming real-time updates once the connection is re-established.

For scalability and reliability, special attention is paid to optimizing WebRTC and CRDT operations. Techniques like update throttling, selective broadcasting, and efficient delta compression are applied to reduce bandwidth consumption and CPU load. Furthermore, strategies such as distributed awareness tracking help keep the system responsive when many users are online simultaneously. To support collaborative features like showing user cursors,

highlights, or chat, the system leverages the Yjs awareness protocol, which communicates metadata about user presence and activity without bloating the core document sync.

In summary, this collaborative editing system blends cutting-edge technologies to deliver a secure, responsive, and feature-rich experience. With Next.js and TipTap delivering a customizable frontend, Yjs and WebRTC enabling real-time collaboration, JWT ensuring secure access, and PostgreSQL safeguarding document history, the result is a robust architecture suitable for modern teams and creative professionals. The modular nature of each component allows for future expansion, such as adding AI-assisted editing, offline mode, or cross-device syncing, making it a powerful foundation for any collaborative platform.

4.2 BACKEND IMPLEMENTATION

Authentication Mechanism:

- Authentication is managed using JWT to ensure secure access control and document collaboration.
- Users authenticate via an email/password login system that issues JWTs upon successful authentication.
- Access to documents is validated using a JWT token and the database before establishing a WebRTC connection for real-time collaboration.
- Expired or invalid JWTs prevent access, ensuring only authorized users can participate in document editing.

Document Storage and Retrieval:

- LiveDocs ensures efficient and decentralized document storage and retrieval, balancing real-time collaboration with data persistence.

- Documents are stored in PostgreSQL using Prisma ORM, ensuring structured and reliable data persistence.
- Yjs (CRDTs) enables real-time, conflict-free document editing across multiple users.
- When a user accesses a document, LiveDocs retrieves the latest persisted state from PostgreSQL. The document state is then synchronized using Y-WebRTC, allowing P2P collaboration.
- Changes made during collaboration are periodically persisted to PostgreSQL. When no active users remain, the final document state is saved to ensure data integrity.
- Access to documents is validated using document-specific roles. Only authorized users (editor/viewer) can retrieve or modify document content.

4.3 FRONTEND IMPLEMENTATION

Real-Time Editing Interface:

- TipTap Integration: The editor supports rich-text formatting, comments, and user mentions.
- Live Collaboration: Changes made by users are instantly reflected using Yjs and Y-WebRTC for P2P synchronization.
- Role-Based Editing: Users with editor roles can modify the document, while viewers have read-only access.
- Persistence: Changes are saved to PostgreSQL only when Save is clicked.

User Interaction:

- User Presence Indicators: Display active users with unique colors and cursors.
- Active Users List: Show the profiles of currently active users at the top of the editor.

4.4 P2P COMMUNICATION WITH WEBRTC

- WebRTC Signaling: A signaling server initializes peer connections before switching to direct communication.
- Direct Data Transmission: Once connected, users exchange updates without a central server, reducing latency.
- Secure Access Control: WebRTC connections are validated using JWTs, ensuring only authorized users can participate.

4.5 DATA CONFLICT HANDLING WITH CRDTS

- Yjs (CRDT-based sync): Ensures conflict-free merging of simultaneous edits.
- Automatic Conflict Resolution: Unlike traditional version control, Yjs ensures a single source of truth without merge conflicts.

4.6 PROTOTYPE DEMONSTRATION

The user interface of LiveDocs is composed of several key components, as illustrated in Figures 4.1 to 4.5. Figure 4.1 showcases the Landing Page, which features a clean, modern design with intuitive navigation, allowing unauthenticated users to register or log in and authenticated users to access their documents or profile. Figure 4.2 presents the Document Page, where users can view a list of their documents in a responsive grid layout, with options to create, open, rename, or delete documents. Figure 4.3 displays the Document Editor Page, the central feature of LiveDocs, which integrates the TipTap editor with Yjs to enable low-latency, real-time collaboration through WebRTC, complete with formatting tools and live synchronization. Figure 4.4 highlights the Share Modal, where users can securely invite collaborators via email and assign role-based access (Viewer or Editor), with options to manage permissions dynamically. Lastly, Figure 4.5 shows the User Profile Page, which allows users to view and

manage their account information and monitor active sessions ensuring a user-friendly and secure experience.

Figure 4.1 Landing Page

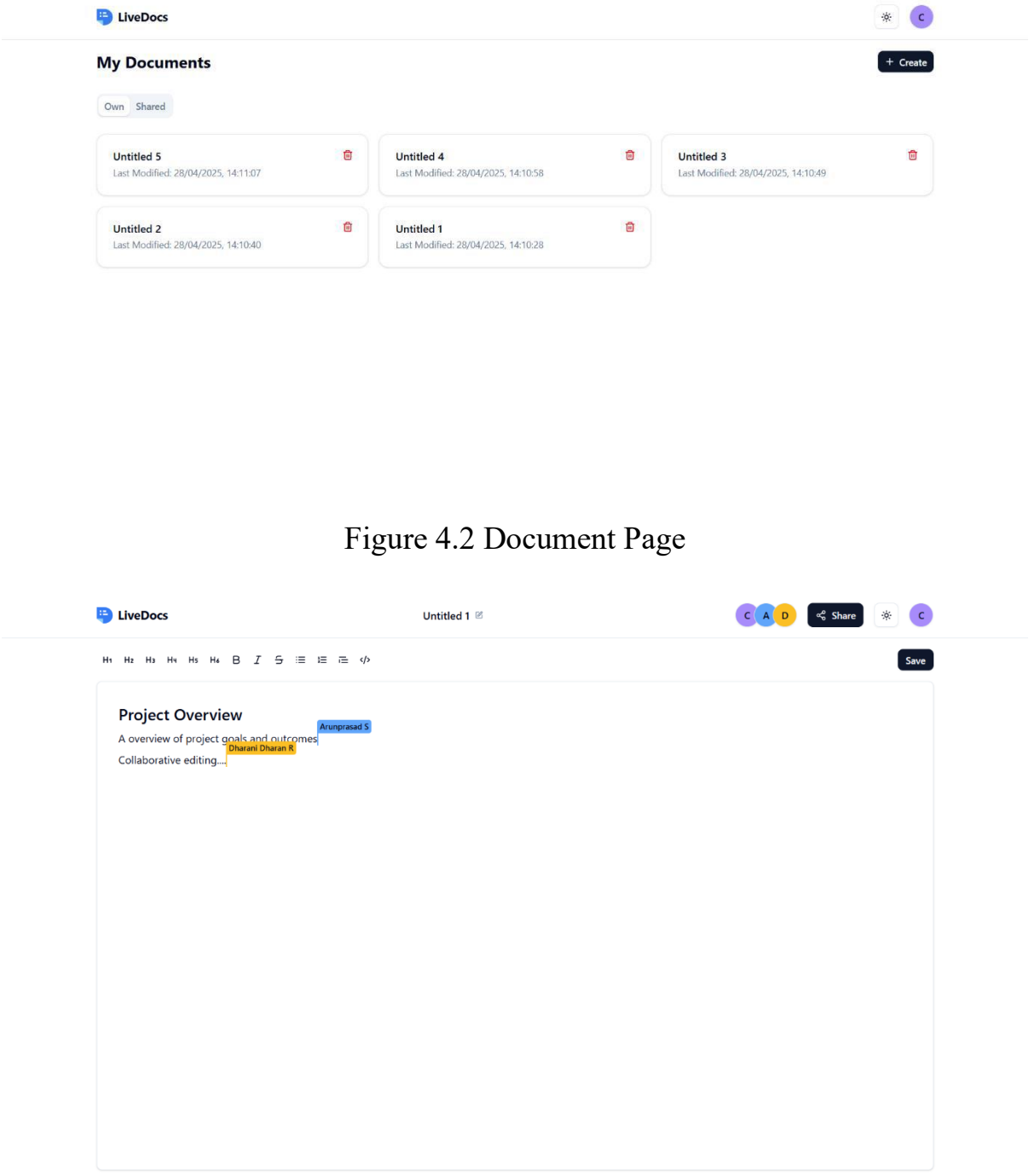

Figure 4.2 Document Page

Figure 4.3 Document Editor Page

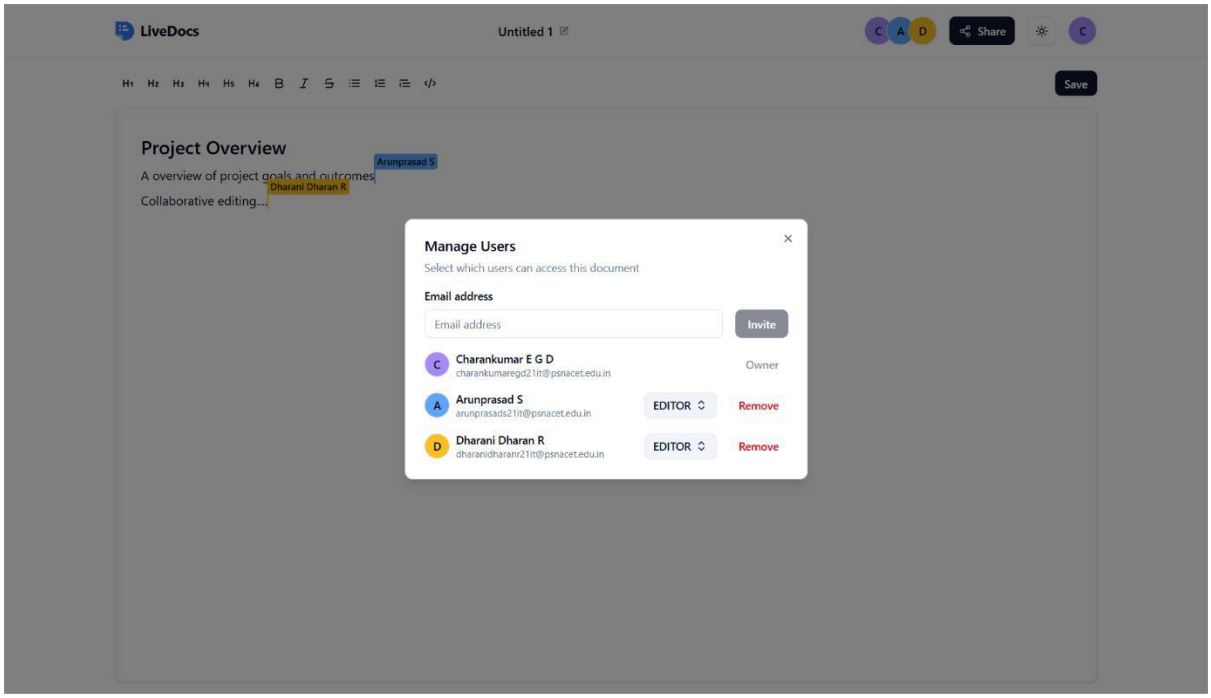

Figure 4.4 Share Modal

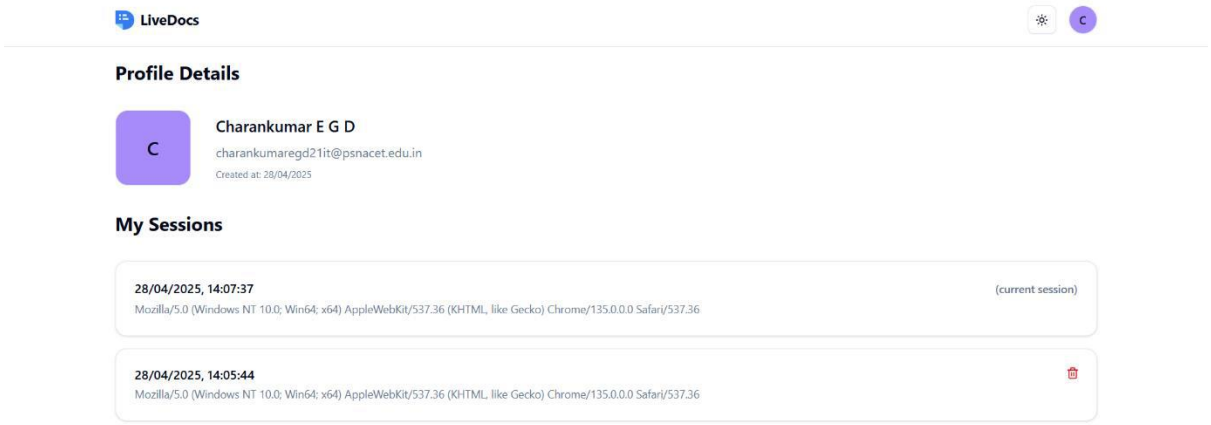

Figure 4.5 User Profile Page

CHAPTER 5

PERFORMANCE EVALUATION

Evaluating the performance of LiveDocs is crucial to understanding its efficiency, scalability, and reliability in real-world usage. This section outlines key evaluation metrics, experimental results, and a comparative analysis with existing solutions.

5.1 LATENCY

In a mesh network, latency is optimized due to direct P2P communication, eliminating the need for centralized servers and reducing transmission delays. Parallel data propagation ensures faster synchronization, as updates can reach multiple peers simultaneously rather than passing through a central relay. WebRTC's adaptive congestion control dynamically adjusts data flow based on network conditions, minimizing lag. Additionally, localized update distribution allows peers to exchange only necessary changes rather than full document states, reducing bandwidth usage and improving real-time responsiveness. These advantages make mesh networks highly efficient for low-latency collaboration, especially in decentralized applications like LiveDocs.

5.2 SCALABILITY

Scalability in LiveDocs is achieved through its decentralized P2P architecture, eliminating server bottlenecks and enabling direct communication between users. In a mesh network, peers dynamically establish connections, distributing the load efficiently without overloading a central server. WebRTC's adaptive bandwidth management ensures stable performance even as the number of participants grows, while Yjs (CRDTs) efficiently synchronize document states, reducing data transmission overhead. To further enhance scalability, peer clustering and selective relay techniques limit redundant connections, optimizing

network traffic. This architecture allows LiveDocs to scale seamlessly to thousands of users, making it ideal for large teams and distributed collaboration.

5.3 SYNCHRONIZATION ACCURACY

Synchronization accuracy in LiveDocs is ensured through Yjs (CRDTs), which guarantees conflict-free merging of concurrent edits without data loss. Since CRDTs operate in a distributed environment, they enable automatic state reconciliation, ensuring that all peers maintain a consistent document version, even in the presence of network partitions or delayed updates. Unlike traditional OT methods, Yjs allows real-time synchronization without requiring a central authority, reducing latency and improving reliability. Additionally, differential updates ensure that only minimal data changes are transmitted, further enhancing efficiency. This approach ensures that every user sees the same document state, regardless of the number of edits or network conditions.

CHAPTER 6

DISCUSSION

6.1 STRENGTHS OF LIVEDOCS

LiveDocs introduces a decentralized, real-time collaborative document editing system that eliminates the limitations of centralized cloud-based solutions. The key strengths of the system include:

- Decentralized Architecture: Unlike traditional document editors that rely on central servers, LiveDocs leverages WebRTC for direct P2P communication, ensuring reduced latency and improved performance.
- Efficient Data Synchronization: By using Yjs (CRDTs), LiveDocs ensures conflict-free, real-time collaboration without requiring a central authority for resolving document edits.
- Scalability: Since LiveDocs operates on a P2P network, the system can scale dynamically without bottlenecks, supporting millions of concurrent users.
- Fault Tolerance: The distributed nature of the system ensures that document collaboration continues even if some peers disconnect, enhancing reliability.
- Security and Privacy: With JWT-based authentication and end-to-end encryption, LiveDocs offers enhanced security, ensuring that only authorized users can access and edit documents.
- Cost Efficiency: Eliminating the need for centralized infrastructure significantly reduces server costs, making the system more sustainable.

6.2 LIMITATIONS AND CHALLENGES

Despite its advantages, LiveDocs faces several challenges that need to be addressed for broader adoption:

- Network Dependencies: WebRTC requires a stable network connection for efficient P2P communication.
- Increased Client-Side Resource Usage: Since document processing is handled on the client side rather than a central server, devices with lower processing power may experience performance issues.
- Security Risks: While JWT provides authentication, additional measures such as end-to-end encryption and access control policies need to be enforced to prevent unauthorized data access.
- High Latency and Network Overhead: Mesh networks require each peer to maintain multiple direct connections, leading to increased bandwidth usage and higher latency as the network scales.

CHAPTER 7

CONCLUSION

LiveDocs presents a decentralized, P2P real-time collaborative document editor that eliminates the dependency on centralized servers, reducing costs while improving scalability, fault tolerance, and privacy. By leveraging WebRTC for direct communication and Yjs (CRDTs) for efficient data synchronization, the system ensures low-latency collaboration, even in large-scale environments.

The project successfully addresses major challenges associated with traditional cloud-based editors, including high infrastructure costs, data privacy risks, and scalability limitations. Additionally, JWT-based authentication enhances security by providing controlled access to documents, while the decentralized nature of the system reduces the risks of single points of failure. Despite its strengths, LiveDocs faces certain limitations, such as network dependency, increased client-side resource consumption, and data persistence challenges. Future enhancements, including hybrid cloud storage, optimized peer discovery, improved mobile support, and AI-driven collaboration tools, will further enhance the system's performance and usability.

Overall, LiveDocs offers a robust, scalable, and privacy-focused alternative to traditional document collaboration tools, making it ideal for enterprises, remote teams, and research communities seeking secure and real-time document editing solutions.

CHAPTER 8

FUTURE ENHANCEMENTS

To further enhance LiveDocs, the following improvements can be considered:

- Optimized Peer Discovery: Enhancing the P2P network with efficient peer discovery mechanisms to reduce connection latency and improve collaboration in large-scale environments.
- Better Mobile Support: Developing optimized WebRTC handling for mobile networks, including adaptive bandwidth management and reduced battery consumption strategies.
- End-to-End Encryption: Strengthening security and ensuring that document content remains private even if intercepted.
- AI-Powered Collaboration Features: Adding AI-driven features such as smart suggestions, real-time grammar correction, and intelligent summarization to enhance user experience.
- Integration with Other Tools: Allowing integration with existing platforms like Git, Notion, and Trello to expand usability and collaboration possibilities.

By addressing these challenges and implementing future enhancements, LiveDocs can become a highly efficient, scalable, and secure solution for real-time decentralized document collaboration, offering an innovative alternative to traditional cloud-based tools.

APPENDICES

APPENDIX A - SAMPLE CODE

client/src/app/user/document/page.tsx

```
“use client”;
import { useRouter } from “next/navigation”;
import { useEffect, useState } from “react”;
import { LoaderIcon, Plus, Trash2 } from “lucide-react”;
import { Button } from “@/components/ui/button”;
import {
  Card,
  CardDescription,
  CardHeader,
  CardTitle,
} from “@/components/ui/card”;
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogFooter,
  DialogDescription,
} from “@/components/ui/dialog”;
```

```
import { Input } from “@/components/ui/input”;

import { Label } from “@/components/ui/label”;

import { Tabs, TabsContent, TabsList, TabsTrigger } from
“@/components/ui/tabs”;

import Loader from “@/components/loader”;

import Header from “@/components/header”;

import { useDocumentStore } from “@/store/document-store”;

export default function Documents() {

  const router = useRouter();

  const documents = useDocumentStore((state) => state.documents);

  const sharedDocuments = useDocumentStore((state) =>
state.sharedDocuments);

  const isLoading = useDocumentStore((state) => state.isLoading);

  const isCreating = useDocumentStore((state) => state.isCreating);

  const isDeleting = useDocumentStore((state) => state.isDeleting);

  const fetchDocuments = useDocumentStore((state) => state.fetchDocuments);

  const createDocument = useDocumentStore((state) => state.createDocument);

  const deleteDocument = useDocumentStore((state) => state.deleteDocument);

  const [newTitle, setNewTitle] = useState<string>(“”);

  const [isCreateModalOpen, setIsCreateModalOpen] =
useState<boolean>(false);

  const [isDeleteModalOpen, setIsDeleteModalOpen] =
useState<boolean>(false);
```

```
const [selectedDocumentId, setSelectedDocumentId] = useState<string | null>(
  null
);
function handleOpenDocument(documentId: string) {
  router.push(`/user/document/${documentId}`);
}
async function handleCreateDocument() {
  await createDocument(newTitle || undefined);
  setNewTitle("");
  setIsCreateModalOpen(false);
}
async function handleDeleteDocument() {
  if (selectedDocumentId) {
    await deleteDocument(selectedDocumentId);
  }
  setIsDeleteModalOpen(false);
}
useEffect(() => {
  fetchDocuments();
}, [fetchDocuments]);
if (isLoading) {
  return <Loader />;
```

```
}
return (
  <>
    <Header variant="user" />
    <main>
      <div className="flex justify-between items-center">
        <h1 className="text-2xl font-bold">My Documents</h1>
        <Button
          size="sm"
          onClick={() => {
            setIsCreateModalOpen(true);
          }}
        >
          <Plus />
          Create
        </Button>
      </div>
      <Tabs defaultValue="own" className="space-y-4">
        <TabsList>
          <TabsTrigger value="own" className="text-muted-foreground">
            Own
          </TabsTrigger>
```

```
          <TabsTrigger value="shared" className="text-muted-foreground">
            Shared
          </TabsTrigger>
        </TabsList>
        <TabsContent value="own">
          {documents.length === 0 ? (
            <p className="text-muted-foreground">
              No documents found. Create One!
            </p>
          ) : (
            <div className="grid gap-4 grid-cols-1 sm:grid-cols-2 lg:grid-cols-
3">
              {documents.map((document) => (
                <Card
                  key={document.id}
                  onClick={() => {
                    handleOpenDocument(document.id);
                  }}
                  className="relative hover:ring hover:ring-ring/50"
                >
                  <div className="absolute right-3 top-3">
                    <Button
```

```
    variant="ghost"
    size="icon"
    onClick={(e) => {
      e.stopPropagation();
      setSelectedDocumentId(document.id);
      setIsDeleteModalOpen(true);
    }}
  >
    <Trash2 className="text-destructive" />
  </Button>
</div>
<CardHeader>
  <CardTitle className="flex items-center justify-between">
    <span>
      {document.title?.length > 25
        ? `${document.title.slice(0, 25).trim()}...`
        : document.title || "Untitled"}
    </span>
  </CardTitle>
  <CardDescription>
    Last Modified: {" "}
    {new Date(document.updatedAt).toLocaleString()}
```

```
            </CardDescription>
          </CardHeader>
        </Card>
      ))}
    </div>
  )}
</TabsContent>
<TabsContent value="shared">
  {sharedDocuments.length === 0 ? (
    <p className="text-muted-foreground">
      No shared documents found.
    </p>
  ) : (
    <div className="grid gap-4 grid-cols-1 sm:grid-cols-2 lg:grid-cols-
3">
      {sharedDocuments.map((document) => (
        <Card
          key={document.id}
          onClick={() => {
            handleOpenDocument(document.id);
          }}
          className="relative hover:ring hover:ring-ring/50"
```

```
>
  <div className="absolute right-3 top-3">
    <Button
      variant="ghost"
      size="icon"
      onClick={(e) => {
        e.stopPropagation();
        setSelectedDocumentId(document.id);
        setIsDeleteModalOpen(true);
      }}
    >
      <Trash2 className="text-destructive" />
    </Button>
  </div>
  <CardHeader>
    <CardTitle className="flex items-center justify-between">
      <span>
        {document.title?.length > 25
          ? `${document.title.slice(0, 25).trim()}...`
          : document.title || "Untitled"}
      </span>
    </CardTitle>
```

```
            <CardDescription>
              Last Modified:{" "}
              {new Date(document.updatedAt).toLocaleString()}
            </CardDescription>
          </CardHeader>
        </Card>
      ))}
    </div>
  )}
 </TabsContent>
</Tabs>
<Dialog
 open={isCreateModalOpen}
 onOpenChange={() => {
  setIsCreateModalOpen(false);
  setNewTitle("");
 }}
>
 <DialogContent
  onOpenAutoFocus={(e) => e.preventDefault()}
  onCloseAutoFocus={(e) => e.preventDefault()}
 >
```

```
<DialogHeader>
  <DialogTitle>Create a New Document</DialogTitle>
  <DialogDescription>
    Enter an optional title for your document.
  </DialogDescription>
</DialogHeader>
<Label htmlFor="title">Title (optional)</Label>
<Input
  id="title"
  value={newTitle}
  onChange={(e) => setNewTitle(e.target.value)}
  placeholder="Untitled"
  disabled={isCreating}
/>
<DialogFooter>
  <Button
    type="submit"
    disabled={isCreating}
    onClick={handleCreateDocument}
  >
    {isCreating ? (
      <LoaderIcon className="animate-spin" />
```

```
            ) : (
              “Create”
            )}
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
    <Dialog open={isDeleteModalOpen}
onOpenChange={setIsDeleteModalOpen}>
      <DialogContent onCloseAutoFocus={(e) => e.preventDefault()}>
        <DialogHeader>
          <DialogTitle>Delete Document</DialogTitle>
          <DialogDescription>
            Are you sure you want to delete this document? This action
            cannot be undone.
          </DialogDescription>
        </DialogHeader>
        <DialogFooter>
          <Button
            variant=“secondary”
            disabled={isDeleting}
            onClick={() => setIsDeleteModalOpen(false)}
```

```
            >
              Cancel
            </Button>
            <Button
              variant="destructive"
              disabled={isDeleting}
              onClick={handleDeleteDocument}
            >
              {isDeleting ? (
                <LoaderIcon className="animate-spin" />
              ) : (
                "Delete"
              )}
            </Button>
          </DialogFooter>
        </DialogContent>
      </Dialog>
    </main>
  </>
 );
}
```

client/src/app/user/document/[document-id]/page.tsx

```
“use client”;
import Link from “next/link”;
import { useParams } from “next/navigation”;
import { useEffect, useState } from “react”;
import { Button } from “@/components/ui/button”;
import Header from “@/components/header”;
import Loader from “@/components/loader”;
import TiptapEditor from “@/components/tiptap/tiptap-editor”;
import { useDocumentStore } from “@/store/document-store”;
import TiptapMenubar from “@/components/tiptap/tiptap-menubar”;
export default function UpdateDocument() {
  const params = useParams();
  const documentId = params[“document-id”] as string;
  const document = useDocumentStore((state) => state.document);
  const isLoading = useDocumentStore((state) => state.isLoading);
  const fetchDocument = useDocumentStore((state) => state.fetchDocument);
  const updateDocument = useDocumentStore((state) =>
state.updateDocument);
  const [content, setContent] = useState<string>(“”);
  async function handleUpdate() {
    await updateDocument(documentId, undefined, content);
```

```
  }
  useEffect(() => {
    fetchDocument(documentId);
  }, [documentId, fetchDocument]);
  useEffect(() => {
    if (document) {
      setContent(document.content || "");
    }
  }, [document]);
  if (isLoading) {
    return <Loader />;
  }
  if (!document) {
    return (
      <main className="flex flex-col items-center justify-center space-y-4 h-96">
        <p className="text-lg font-medium">Document not found!</p>
        <Button variant="secondary" asChild>
          <Link href="/user/document"> Back to Documents</Link>
        </Button>
      </main>
    );
```

```
  }
  return (
    <>
      <Header variant="document" />
      <main className="space-y-4">
        <TiptapMenubar handleUpdate={handleUpdate} />
        <TiptapEditor
          documentId={documentId}
          content={content}
          setContent={setContent}
        />
      </main>
    </>
  );
}
```

client/src/components/tiptap/tiptap-editor.tsx

```
"use client";
import Collaboration from "@tiptap/extension-collaboration";
import CollaborationCursor from "@tiptap/extension-collaboration-cursor";
import { Editor, EditorContent } from "@tiptap/react";
import StarterKit from "@tiptap/starter-kit";
```

```
import { useEffect, useRef } from “react”;
import * as Y from “yjs”;
import { WebrtcProvider } from “y-webrtc”;
import { useEditorStore } from “@/store/editor-store”;
import { useUserStore } from “@/store/user-store”;
import { ActiveUser } from “@/types”;
import { AccessRole } from “@prisma/client”;
import { useDocumentStore } from “@/store/document-store”;
type TiptapEditorProps = {
  documentId: string;
  content: string;
  setContent: (content: string) => void;
};
export default function TiptapEditor({
  documentId,
  content: initialContent,
  setContent: setInitialContent,
}: TiptapEditorProps) {
  const currentUser = useUserStore((state) => state.user);
  const setActiveUsers = useUserStore((state) => state.setActiveUsers);
  const editor = useEditorStore((state) => state.editor);
  const setEditor = useEditorStore((state) => state.setEditor);
```

```
  const userRole = useDocumentStore((state) => state.currentUserRole);
  const ydoc = useRef(new Y.Doc());
  const providerRef = useRef<WebrtcProvider | null>(null);
  useEffect(() => {
    if (!providerRef.current) {
      providerRef.current = new WebrtcProvider(documentId, ydoc.current, {
        signaling: [process.env.NEXT_PUBLIC_SIGNALING_SERVER_URL as
string],
      });
    }
    return () => {
      providerRef.current?.destroy();
      providerRef.current = null;
    };
  }, [documentId]);
  useEffect(() => {
    if (!providerRef.current || !currentUser) return;
    const newEditor = new Editor({
      editorProps: {
        attributes: {
          class:
```

```
        "focus:outline-none bg-card border rounded-md p-8 shadow min-h-
screen",
      },
    },
    extensions: [
      StarterKit.configure({
        history: false,
      }),
      Collaboration.configure({
        document: ydoc.current,
      }),
      CollaborationCursor.configure({
        provider: providerRef.current,
        user: {
          name: currentUser.name,
          color: currentUser.color,
        },
      }),
    ],
    editable: false,
    onUpdate: ({ editor }) => {
      setInitialContent(editor.getHTML());
```

```
  },
});
setEditor(newEditor);
providerRef.current.awareness.setLocalStateField("user", {
  id: currentUser.id,
  name: currentUser.name,
  email: currentUser.email,
  color: currentUser.color,
});
function onAwarenessChange() {
  if (!providerRef.current) return;
  const awarenessStates = providerRef.current.awareness.getStates();
  const activeUsers: ActiveUser[] = Array.from(
    awarenessStates.values()
  ).map((state) => ({
    id: state.user?.id || "",
    name: state.user?.name || "Anonymous",
    email: state.user?.email || "",
    color: state.user?.color || "#F87171",
  }));
  setActiveUsers(activeUsers);
}
```

```
    providerRef.current.awareness.on("change", onAwarenessChange);
    onAwarenessChange();
    return () => {
      providerRef.current?.awareness.off("change", onAwarenessChange);
      newEditor.destroy();
    };
  }, [currentUser, setActiveUsers, setEditor, setInitialContent]);
  useEffect(() => {
    setTimeout(() => {
      if (!providerRef.current || !editor) return;
      const isFirstPeer = providerRef.current.awareness.getStates().size === 1;
      if (isFirstPeer && initialContent !== editor.getHTML()) {
        editor.commands.setContent(initialContent);
      }
      if (userRole !== AccessRole.VIEWER) {
        editor.setEditable(true);
      }
    }, 5000);
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [providerRef, editor, userRole]);
  return <EditorContent editor={editor} />;
}
```

server/src/index.js

```
#!/usr/bin/env node
import { WebSocketServer } from “ws”;
import http from “http”;
import * as map from “lib0/map”;
const wsReadyStateConnecting = 0;
const wsReadyStateOpen = 1;
const wsReadyStateClosing = 2; // eslint-disable-line
const wsReadyStateClosed = 3; // eslint-disable-line
const pingTimeout = 30000;
const port = process.env.PORT || 4444;
const wss = new WebSocketServer({ noServer: true });
const server = http.createServer((request, response) => {
  response.writeHead(200, { “Content-Type”: “text/plain” });
  response.end(“okay”);
});
/**
 * Map froms topic-name to set of subscribed clients.
 * @type {Map<string, Set<any>>}
 */
const topics = new Map();
/**
```

```
 * @param {any} conn
 * @param {object} message
 */
const send = (conn, message) => {
  if (
    conn.readyState !== wsReadyStateConnecting &&
    conn.readyState !== wsReadyStateOpen
  ) {
    conn.close();
  }
  try {
    conn.send(JSON.stringify(message));
  } catch (e) {
    conn.close();
  }
};
/**
 * Setup a new client
 * @param {any} conn
 */
const onconnection = (conn) => {
  /**
```

```
 * @type {Set<string>}
 */
const subscribedTopics = new Set();
let closed = false;
// Check if connection is still alive
let pongReceived = true;
const pingInterval = setInterval(() => {
  if (!pongReceived) {
    conn.close();
    clearInterval(pingInterval);
  } else {
    pongReceived = false;
    try {
      conn.ping();
    } catch (e) {
      conn.close();
    }
  }
}, pingTimeout);
conn.on("pong", () => {
  pongReceived = true;
});
```

```
conn.on(“close”, () => {
  subscribedTopics.forEach((topicName) => {
    const subs = topics.get(topicName) || new Set();
    subs.delete(conn);
    if (subs.size === 0) {
      topics.delete(topicName);
    }
  });
  subscribedTopics.clear();
  closed = true;
});
conn.on(
  “message”,
  /** @param {object} message */ (message) => {
    if (typeof message === “string” || message instanceof Buffer) {
      message = JSON.parse(message);
    }
    if (message && message.type && !closed) {
      switch (message.type) {
        case “subscribe”:
          /** @type {Array<string>} */ (message.topics || []).forEach(
            (topicName) => {
```

```
      if (typeof topicName === “string”) {
        // add conn to topic
        const topic = map.setIfUndefined(
          topics,
          topicName,
          () => new Set()
        );
        topic.add(conn);
        // add topic to conn
        subscribedTopics.add(topicName);
      }
    }
  );
  break;
case “unsubscribe”:
  /** @type {Array<string>} */ (message.topics || []).forEach(
    (topicName) => {
      const subs = topics.get(topicName);
      if (subs) {
        subs.delete(conn);
      }
    }
```

```
          );
          break;
        case “publish”:
          if (message.topic) {
            const receivers = topics.get(message.topic);
            if (receivers) {
              message.clients = receivers.size;
              receivers.forEach((receiver) => send(receiver, message));
            }
          }
          break;
        case “ping”:
          send(conn, { type: “pong” });
      }
    }
  }
);
};
wss.on(“connection”, onconnection);
server.on(“upgrade”, (request, socket, head) => {
  // You may check auth of request here..
  /**
```

```
   * @param {any} ws
   */
  const handleAuth = (ws) => {
    wss.emit("connection", ws, request);
  };
  wss.handleUpgrade(request, socket, head, handleAuth);
});
server.listen(port);
console.log("Signaling server running on localhost:", port);
```

REFERENCES

1. Q. -V. Dang and C. -L. Ignat, “Performance of real-time collaborative editors at large scale: User perspective,” 2016 IFIP Networking Conference (IFIP Networking) and Workshops, Vienna, Austria, 2016, pp. 548-553, doi: 10.1109/IFIPNetworking.2016.7497258.
2. K. Virdi, A. L. Yadav, A. A. Gadoo and N. S. Talwandi, “Collaborative Code Editors - Enabling Real-Time Multi-User Coding and Knowledge Sharing,” 2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bengaluru, India, 2023, pp. 614-619, doi: 10.1109/ICIMIA60377.2023.10426375.
3. S. Arora and P. K. Atrey, “SecureC2Edit: A Framework for Secure Collaborative and Concurrent Document Editing,” in IEEE Transactions on Dependable and Se-cure Computing, vol. 21, no. 4, pp. 2227-2241, July-Aug. 2024, doi: 10.1109/TDSC.2023.3302810.
4. M. Konstantopoulos, N. Chondros and M. Roussopoulos, “A Personal Distributed Real-time Collaborative System,” 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), Hong Kong, 2020, pp. 709-715, doi: 10.1109/ICPADS51040.2020.00101.
5. K. Jahns, Yjs: A CRDT Implementation for Building Collaborative Applications. Available: https://yjs.dev
6. K. Jahns, y-webrtc: WebRTC Provider for Yjs. Available: https://github.com/yjs/y-webrtc
7. Mozilla Developer Network, WebRTC API Documentation. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
8. M. Jones, J. Bradley and N. Sakimura, RFC 7519: JSON Web Token (JWT), Internet Engineering Task Force (IETF), May 2015. Available: https://datatracker.ietf.org/doc/html/rfc7519

9. PostgreSQL Global Development Group, PostgreSQL: An Advanced Open-Source Relational Database. Available: https://www.postgresql.org
10. TipTap, The Headless WYSIWYG Text Editor for the Web. Available: https://tiptap.dev
11. Vercel, Next.js Documentation. Available: https://nextjs.org/docs