## COLLABORATIVE TEXT EDITOR WITH CHATROOM

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

## BACHELOR OF SCIENCE IN COMPUTER TECHNOLOGY

at the Bharathiar University



By

## AKASH.A

## (Reg. No: 211CT004)

Under the Guidance of

**Dr.V.MANIMEKALAI MCA.,M.Phil.,Ph.D.**

Assistant Professor

## DEPARTMENT OF COMPUTER TECHNOLOGY



**Dr. N. G. P. ARTS AND SCIENCE COLLEGE**

(An Autonomous Institution, Affiliated to Bharathiar University, Coimbatore)

Approved by Government of Tamil Nadu & Accredited by NAAC with A++ Grade (3rd Cycle-3.64CGPA) `Dr.N.G.P. - Kalapatti Road, Coimbatore-641 048, Tamil Nadu, India.

Website: [www.drngpasc.ac.in |](http://www.drngpasc.ac.in/) Email: [info@drngpasc.ac.in. |](mailto:info@drngpasc.ac.in) Phone: +91-422-2369100

**APRIL –2024**

# DECLARATION

**AKASH A (Reg.No:211CT004)** hereby declare that the project report entitled **“Excel Template Hub Using Angular”** submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Science in Computer Technology at the Bharathiar University** is a record of original work done during the period of study supervision and under the guidance of **Dr.V.MANIMEKALAI MCA.,M.Phil.,Ph.D. Department of Computer Technology, Dr.N.G.P. Arts and Science College,Coimbatore - 48**, and it has not formed on the basis of award of any Degree/ Diploma/ Associateship/ Fellowship or other similar title to any candidate of any university.

## Place: Coimbatore Date:

**AKASH A**

## Reg.No: 211CT004

**CERTIFICATE**

This is to certify that the project report entitled **“Collaborative Text Editor with Chatroom”** submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Science in Computer Technology at the Bharathiar University** is a record of original work done by **AKASH A (Reg. No: 211CT004)** during the period (2022-2023) of his study in **Department of Computer Technology, Dr. N.G.P. Arts and Science College,Coimbatore-48,** under my supervision and guidance, and it has not formed on the basis of award of any Degree/ Diploma/ Associateship/ Fellowship or other similar title to any candidate of any university.

|  |  |  |
| --- | --- | --- |
| **(Dr.V.Manimekalai)** | **(Dr.M.Rathi)** | **(Dr.K.Ramamurthi**) |
| Project Guide | Associate Professor and Head | Principal |

Place: Coimbatore Date:

Viva-voce Examination held on

## Internal Examiner External Examiner

**ACKNOWLEDGEMENT**

This project was the most significant accomplishment in my life and it would not have been possible without the blessing of God almighty and those who supported and believed in my caliber.

I record my deep sense of gratitude to **Dr. Nalla. G. Palaniswami M.D, AB (USA), Chairman, Dr.N.G.P. Research and Educational Trust and Dr. Thavamani. D. Palaniswami M.D, AB (USA), Secretary, Dr. N.G.P. Arts and Science College, Coimbatore** for providing an opportunity to undergo my study in this prestigious institution.

I record my sincere thanks to Prof**. Dr. K.Ramamurthi M.Com., BL., MBA., M.Phil., Ph.D., Dr. N.G.P. Arts and Science College, Coimbatore,** for permitting me to make use of this infrastructure for this great institution of enduring excellence.

I am immensely greatful to Prof. **Dr. F. Mary Magdalene Jane M.C.A., M.Phil., Ph.D., Dean Faculty of Computer Science, Dr. N.G.P. Arts and Science College, Coimbatore,** for every help she rendered during the project.

I express my sincere thanks to **Dr. M. Rathi MCA., M.Phil., SET, Ph.D., Associate Professor and Head, Department of Computer Technology, Dr. N.G.P. Arts and Science College, Coimbatore** for showing sustained interest and providing help throughout the period of this work.

I would like to extend the sincere thanks to my guide **Dr.V.MANIMEKALAI MCA.,M.Phil.,Ph.D., Assistant Professor, Department of Computer Technology, Dr. N.G.P. Arts and Science College, Coimbatore** for her exemplary guidance and encouragement.

I take this opportunity to acknowledge my sincere thanks to all the staff members of the Department of Computer Applications for their constant inspiration, assistance and resourceful guidance for the completion of this project successfully.

I express my sincere thanks to my family and friends for their encouragement, love, prayer, moral support, advice and sacrifice without which we would not have been able to pursue the course of this study.

**AKASH A**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | Declaration |  |
|  | Certificate |  |
|  | Acknowledgement |  |
| 1 | Abstract |  |
| 2 | Introduction |  |
| 3 | System Specification |  |
|  | 3.1 Software Specification |  |
|  | 3.2 Hardware Specification |  |
| 4 | System Analysis |  |
|  | 4.1 Proposed System |  |
| 5 | System Design |  |
|  | 5.1 Input Design |  |
|  | 5.2Output Design |  |
| 6 | System Development |  |
|  | 6.1 Description of modules |  |
| 7 | Testing and Implementation |  |
| 8 | Conclusion |  |
| 9 | Bibliography |  |
| 10 | Appendices |  |
|  | A.Data Flow Diagram |  |
|  | B.Sample Coding |  |
|  | C.Sample output |  |

**1.ABSTRACT**

Collaborative text editor with an integrated chatroom revolutionizes the way teams collaborate and communicate in real-time. This innovative platform provides a centralized workspace where multiple users can collaboratively edit documents, spreadsheets, presentations, and more, enhancing productivity and streamlining workflow.

The integration of a chatroom complements document editing by facilitating synchronous communication among team members. Whether brainstorming ideas, providing feedback, or sharing updates, users can engage in seamless discussions without interrupting the editing process. This cohesive environment fosters collaboration, creativity, and teamwork, ultimately leading to more efficient and effective outcomes.

Security and accessibility are paramount features of our platform. Robust encryption protocols and user authentication measures ensure that sensitive data remains protected, instilling confidence in users to collaborate freely. Furthermore, our intuitive interface is designed for ease of use, catering to users of all technical proficiencies and enabling smooth navigation throughout the platform.

In summary, our collaborative text editor with an integrated chatroom offers a transformative solution for modern teamwork. By combining real-time document editing with seamless communication, we empower teams to collaborate more effectively, achieve their goals, and drive success in today's dynamic work environments

**2.INTRODUCTION**

In the ever-evolving landscape of collaborative work, efficiency and seamless communication are paramount. Imagine a platform where teams can seamlessly edit documents together in real-time while discussing ideas in a unified chatroom environment. Welcome to our collaborative text editor with integrated chatroom—a dynamic space designed to elevate teamwork to new heights.

At its core, our platform serves as a virtual meeting ground where multiple users can simultaneously edit documents, spreadsheets, presentations, and more. Gone are the days of tedious email chains and cumbersome file transfers; instead, teams can collaborate effortlessly within a single, centralized workspace.

The integration of a chatroom adds an extra layer of synergy to the collaboration process. Whether it's brainstorming ideas, providing feedback, or simply sharing updates, team members can engage in real-time discussions without disrupting the flow of document editing. This synchronous communication fosters a sense of camaraderie and ensures that everyone is on the same page.

Moreover, our platform prioritizes security and accessibility. With robust encryption protocols and user authentication measures, teams can confidently collaborate knowing that their data remains secure at all times. Additionally, our intuitive interface is designed to be user-friendly, allowing team members of all technical backgrounds to seamlessly navigate the platform.

In essence, our collaborative text editor with a built-in chatroom redefines the way teams work together. By combining the power of real-time document editing with seamless communication, we empower teams to achieve their goals more efficiently and effectively than ever before. Welcome to the future of collaboration.

**3.SYSTEM SPECIFICATIONS**

**3.1** **HARDWARE SPECIFICATIONS:**

* **Server Infrastructure:** Adequate resources for hosting the MERN stack application, typically a cloud-based server or dedicated hosting environment**.**
* **Processor:** Multi-core processor with sufficient power to handle concurrent user requests, such as Intel Core i5 or above.
* **Memory (RAM):** Crucial for handling runtime memory requirements and user sessions; at least 4GB recommended, 8GB or more preferable for better performance.
* **Storage:** SSD storage preferred for faster read/write operations; minimum 20GB storage space required, depending on project scale and data storage needs.
* **Network Connectivity:** Reliable internet connectivity essential for accessing the application and managing database interactions**.**
* **Backup and Redundancy:** Essential measures to safeguard against data loss and ensure uninterrupted service availability, including automated backup routines and redundant server configurations.
* **Scalability Considerations:** Designing infrastructure with scalability in mind to accommodate future growth and increased traffic demands without significant performance degradation, utilizing load balancing, horizontal scaling, and containerization technologies.

**3.2 SOFTWARE SPECIFICATIONS:**

* **Node.js**: Required for server-side JavaScript execution and running the backend logic of the MERN stack application.
* **Express.js:** A web application framework for Node.js that simplifies the development of backend APIs and handling HTTP requests/responses.
* **React**: A JavaScript library for building user interfaces, used to develop the frontend components of the MERN stack application.
* **Socket.IO:** Socket.IO is a JavaScript library that enables real-time, bidirectional communication between web clients (such as browsers) and servers.
* **Package Managers (npm or yarn):** Necessary for managing dependencies and installing packages required for building and running the MERN stack application.
* **Web Browser:** Users will access the application using a web browser, so compatibility testing across major browsers like Chrome, Firefox, Safari, and Edge is essential.
* **Integrated Development Environment (IDE):** An IDE such as Visual Studio Code or WebStorm is recommended for development, providing features like syntax highlighting, code completion, and debugging tools.

**4.SYSTEM ANALYSIS**

**4.1 PROPOSED SYSTEM:**

1. **Frontend Interface**:
   * Web Interface: Develop a user-friendly web interface using modern frontend technologies like React.js, Angular, or Vue.js.
   * Text Editor: Implement a text editor component capable of real-time collaborative editing. You can use libraries like Quill.js or Slate.js.
   * Chatroom UI: Integrate a chatroom interface alongside the text editor, allowing users to communicate in real-time.
2. **Backend Server**:
   * API Server: Develop a backend server using Node.js, Django, or Flask to handle client requests and manage collaborative editing sessions.
   * WebSocket Communication: Implement WebSocket communication between clients and the server to facilitate real-time updates in the text editor and chatroom.
3. **Collaborative Editing**:
   * Operational Transformation (OT): Implement OT algorithms to handle concurrent edits from multiple users without conflicts. Libraries like ShareDB or Yjs can assist with this.
   * Version Control: Maintain a version history of documents to allow users to revert changes or track modifications over time.
4. **Authentication and Authorization**:
   * User Authentication: Implement user authentication using JWT (JSON Web Tokens) or OAuth to secure access to the application.
   * Authorization: Define user roles and permissions to control access to documents and chatrooms.
5. **Real-time Chat**:
   * WebSocket Integration: Use WebSocket connections to enable real-time chat communication between users.
   * Message Persistence: Store chat messages in the database to maintain chat history and allow users to access past conversations.

**5.SYSTEM DESIGN**

**5.1 INPUT DESIGN:**

1. **Text Input Area**:
   * Provide a spacious text input area for users to type and edit their content comfortably.
   * Ensure the text input area is resizable to accommodate varying amounts of text.
   * Implement basic formatting options such as bold, italic, underline, font size, and font color.
2. **Collaborative Editing Controls**:
   * Include controls for collaborative editing features such as inviting collaborators, managing permissions, and tracking changes.
   * Integrate buttons or icons for actions like sharing the document link, inviting collaborators via email, and setting permissions for editing or viewing.
3. **Message Input Field**:
   * Offer a designated text input field for users to type their chat messages.
   * Allow for multiline input to accommodate longer messages.
4. **Send Message Button**:
   * Include a prominent button or icon to send chat messages.
   * Enable keyboard shortcuts (e.g., Enter key) for sending messages to improve usability.

**5.2 OUTPUT DESIGN:**

1. **Document Display**:
   * Display the collaborative document in a clear and readable format.
   * Use appropriate fonts, sizes, and spacing to enhance readability.
   * Ensure that text wraps properly within the text editor area to prevent horizontal scrolling.
2. **Collaborator Presence Indicators**:
   * Display visual indicators to show when other collaborators are active in the document.
   * Use different colors or icons to represent each collaborator, along with their username or initials.
3. **Message Display**:
   * Present chat messages in a chronological order, with the newest messages appearing at the bottom of the chatroom window.
   * Use clear formatting and styling to differentiate between messages, including sender names, timestamps, and message content.
4. **Real-time Updates**:
   * Update the chatroom in real-time as new messages are sent by users.
   * Scroll the chatroom automatically to display the latest messages without requiring manual scrolling.
5. **Cursor Tracking and Collaborator Highlighting**:
   * Visualize cursor positions and selections of other collaborators within the document.
   * Highlight text or areas edited by other collaborators using different colors or styles to distinguish changes.

**6.SYSTEM DEVELOPMENT**

**6.1 DESCRIPTION OF MODULES :**

**1. Authentication Module:**

Allows users to securely sign up, log in, and manage their accounts. This module ensures that only authorized users can access the collaborative editor and contribute to documents.

**2. Document Management Module:**

Handles the creation, storage, retrieval, and editing of documents. This module includes features for creating new documents, opening existing documents, saving changes, and organizing documents into folders or categories.

**3. Real-time Collaboration Module:**

Implements real-time collaboration features, enabling multiple users to simultaneously edit the same document. This module utilizes technologies such as WebSockets or WebRTC to synchronize changes in real-time across all connected clients.

**4. Text Editing Interface:**

Provides a user-friendly interface for editing text documents. This module includes features such as text formatting options (e.g., bold, italic, underline), undo/redo functionality, syntax highlighting, and cursor tracking to facilitate collaborative editing.

**5. Version Control Module:**

Manages document versions and revisions to track changes made by different users over time. This module allows users to review past revisions, revert to previous versions, and compare changes between versions to maintain document integrity and traceability.

**6. User Presence Module:**

Displays real-time presence indicators to show which users are currently active and editing a document. This module enhances collaboration by providing visibility into who else is working on the same document at any given time.

**7. Notification Module:**

Sends notifications to users about important events related to document collaboration, such as new document invitations, document sharing requests, or changes made by other users. This module helps keep users informed and engaged in the collaborative editing process.

**8. Permissions and Access Control Module:**

Enforces role-based permissions and access control to regulate user access to documents and editing privileges. This module allows administrators to define user roles (e.g., owner, collaborator, viewer) and set permissions accordingly to ensure data security and privacy.

**7.TESTING AND IMPLEMENTATION**

**Testing**:

Testing is a critical phase in the development of the Collaborative Text Editor with Chatroom to ensure its functionality, reliability, and usability. The testing phase involves several activities, including:

### **1. Unit Testing**:

* **Text Editor Module**:
  + Test text editing functionalities such as formatting options (bold, italic, underline), text alignment, and bullet points.
  + Verify the implementation of operational transformation algorithms for collaborative editing.
* **Chatroom Module**:
  + Test message sending, receiving, and rendering functionality.
  + Verify emoji rendering, file sharing, and message formatting features.
* **WebSocket Communication Module**:
  + Test WebSocket connections and event handling for real-time communication.
  + Ensure that messages are sent and received correctly between clients and the server.

### **2. Integration Testing**:

* **Text Editor and Chatroom Integration**:
  + Test the integration between the text editor and chatroom components to ensure seamless communication and synchronization.
* **Backend Integration**:
  + Test integration with the backend server to verify data exchange and communication protocols.

### **3. End-to-End Testing**:

* **Scenario-based Testing**:
  + Create test scenarios that simulate real-world usage scenarios, such as multiple users collaborating on a document or participating in a chatroom.
  + Verify that all functionalities work as expected from the user's perspective.
* **Cross-browser and Cross-device Testing**:
  + Test the application on different web browsers (e.g., Chrome, Firefox, Safari) and devices (e.g., desktop, laptop, tablet, smartphone) to ensure compatibility and responsiveness.

### **4. Performance Testing**:

* **Load Testing**:
  + Simulate a large number of concurrent users accessing the application to assess its performance under heavy load.
  + Measure response times, throughput, and server resource utilization to identify bottlenecks and optimize performance.
* **Scalability Testing**:
  + Test the application's ability to scale horizontally by adding more server instances or load balancers to handle increased traffic.

### **5. Security Testing**:

* **Authentication and Authorization Testing**:
  + Verify that user authentication mechanisms work correctly and securely.
  + Test authorization to ensure that users can only access authorized features and resources.
* **Data Encryption Testing**:
  + Ensure that sensitive data such as user credentials, chat messages, and documents are encrypted during transmission and storage.

### **6. Usability Testing**:

* **User Interface Testing**:
  + Evaluate the user interface for clarity, intuitiveness, and ease of use.
  + Gather feedback from users through usability testing sessions to identify areas for improvement.
* **Accessibility Testing**:
  + Verify that the application is accessible to users with disabilities, including keyboard navigation, screen reader compatibility, and color contrast.

### **7. Regression Testing**:

* **Automated Regression Testing**:
  + Implement automated regression test suites to ensure that new changes do not introduce regressions or break existing functionalities.
  + Run regression tests regularly as part of the continuous integration and deployment pipeline.

### **8. Acceptance Testing**:

* **User Acceptance Testing (UAT)**:
  + Conduct UAT with real users or stakeholders to validate that the application meets their requirements and expectations.
  + Gather feedback and make necessary adjustments based on user input before the final release.

### **9. Documentation**:

* **Test Documentation**:
  + Document test cases, test plans, and testing results for future reference and knowledge sharing.
  + Provide comprehensive documentation for developers, testers, and other stakeholders to understand the testing process and outcomes.

**Implementation:**

Implementing a collaborative text editor with a chatroom involves several steps, including setting up the development environment, building the frontend and backend components, integrating real-time communication, and deploying the application. Here's a high-level overview of the implementation process:

### **1. Setup Development Environment**:

* Install necessary tools and dependencies such as Node.js, npm (or yarn), and a code editor.
* Choose a frontend framework (e.g., React.js) and a backend framework (e.g., Express.js) for development.

### **2. Frontend Development**:

* Create the user interface for the collaborative text editor and chatroom using HTML, CSS, and JavaScript.
* Implement the text editor component with features like text formatting, cursor tracking, and collaborative editing using libraries like Slate.js or Quill.js.
* Develop the chatroom interface with features such as real-time messaging, user presence indicators, and emoji support.

### **3. Backend Development**:

* Set up a backend server using Node.js and Express.js (or your chosen backend framework).
* Implement RESTful APIs for user authentication, document management, and chatroom functionality.
* Integrate a WebSocket server using Socket.IO or another library for real-time communication between clients and the server.

### **4. Real-time Communication**:

* Implement WebSocket communication between clients and the server for real-time updates in the text editor and chatroom.
* Handle events like document updates, chat messages, and user presence using Socket.IO event handlers.

### **5. User Authentication and Authorization**:

* Implement user authentication using JWT (JSON Web Tokens) or OAuth for securing access to the application.
* Define user roles and permissions to control access to documents and chatrooms.

### **6. Testing**:

* Write unit tests, integration tests, and end-to-end tests to ensure the functionality and reliability of the application.
* Conduct usability testing and gather feedback from users to identify and address any usability issues.
* Configure domain and SSL certificates for secure access to the application.

### **7. Documentation and Training**:

* Document the architecture, design decisions, and implementation details for future reference.
* Provide user documentation and training materials to help users understand how to use the application effectively.

**8.CONCLUSION**

In conclusion, the development of a collaborative text editor with a chatroom presents an exciting opportunity to create a dynamic and interactive web application. By implementing real-time collaboration features alongside instant messaging capabilities, users can seamlessly work together on documents while communicating in real-time.

Throughout the development process, careful attention to detail is crucial. From designing intuitive user interfaces to implementing robust backend systems, every aspect of the application contributes to its overall success. Additionally, thorough testing and quality assurance ensure that the application meets the highest standards of functionality, reliability, and performance.

Ultimately, a well-implemented collaborative text editor with a chatroom can revolutionize the way users collaborate, increasing productivity, fostering teamwork, and enhancing communication. Whether used for remote collaboration, project management, or educational purposes, this type of application has the potential to make a significant impact in various industries and domains.

As technology continues to evolve, so too will the possibilities for collaborative tools like this. By staying attuned to user feedback and emerging trends, developers can continually refine and improve these applications to meet the evolving needs of users in an increasingly connected world.

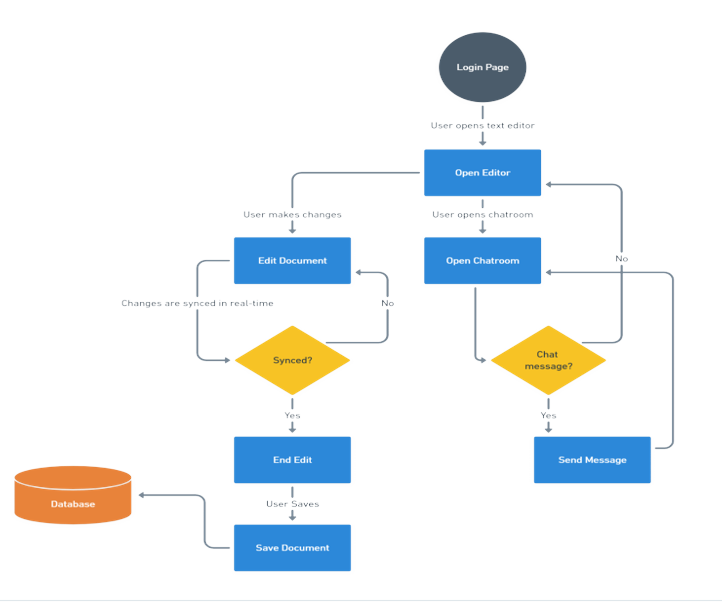
Top of Form

**9.BIBILIOGRAPHY**

1. Abou-Zahra, S., Brajnik, G., Brewer, J., Cáceres, P., Davis, S., Montalvo, A., & O'Connor, J. (2005). Web Content Accessibility Guidelines (WCAG) 2.0. W3C Recommendation.
2. Quill.js Documentation: https://quilljs.com/docs/
3. Socket.IO Documentation: https://socket.io/docs/
4. Yjs Documentation: https://docs.yjs.dev/
5. ShareDB Documentation: <https://github.com/share/sharedb>
6. "Operational Transformation in Real-Time Collaborative Systems" by C. Sun and X. Sun (2016). In Proceedings of the 19th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '16), pp. 47-56.
7. "Collaborative Editing: The Relationship between Document Structure and Behavior" by J. Marlow, R. Yeh, and D. Fox (2016). In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), pp. 1205-1216.
8. "Real-Time Collaborative Editing in Word Processor Documents" by D. Sun, Q. Sun, and Z. Fang (2015). In Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), pp. 1441-1448.

**10.APPENDICES**

**10.1DATA FLOW DIAGRAM:**

****

**10.2.CODING:**

**10.2.1 Frontend :**

import { Route, BrowserRouter as Router, Routes } from "react-router-dom"

import Toast from "./components/toast/Toast"

import EditorLayout from "./layouts/EditorLayout"

import HomePage from "./pages/HomePage"

import GitHubCorner from "./components/GitHubCorner"

import useWindowDimensions from "./hooks/useWindowDimensions"

function App() {

const { isMobile } = useWindowDimensions()

return (

<>

<Toast /> {/\* Toast component from react-hot-toast \*/}

{!isMobile && <GitHubCorner />}

<Router>

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/editor/:roomId" element={<EditorLayout />} />

</Routes>

</Router>

</>

)

}

export default App

import { useEffect, useState } from "react"

function useWindowDimensions() {

const [windowDimensions, setWindowDimensions] = useState({

width: window.innerWidth,

height: window.innerHeight,

})

const [isMobile, setIsMobile] = useState(window.innerWidth < 768)

useEffect(() => {

const updateWindowDimensions = () => {

setWindowDimensions({

width: window.innerWidth,

height: window.innerHeight,

})

setIsMobile(window.innerWidth < 768)

}

window.addEventListener("resize", updateWindowDimensions)

return () => {

window.removeEventListener("resize", updateWindowDimensions)

}

}, [])

return { ...windowDimensions, isMobile }

}

export default useWindowDimensions

import { useEffect, useState } from "react"

import useWindowDimensions from "./useWindowDimensions"

function useResponsive() {

const [isMobileSidebarOpen, setIsMobileSidebarOpen] = useState(false)

const { height } = useWindowDimensions()

useEffect(() => {

if (height < 500) {

setIsMobileSidebarOpen(false)

} else {

setIsMobileSidebarOpen(true)

}

}, [height])

return { isMobileSidebarOpen }

}

export default useResponsive

import { useContext, useEffect } from "react"

import AppContext from "../context/AppContext"

import useLocalStorage from "./useLocalStorage"

function usePageEvents() {

const { settings, updateSettings } = useContext(AppContext)

const { fontSize } = settings

const { setItem } = useLocalStorage()

useEffect(() => {

const beforeUnloadHandler = (e) => {

const msg = "Changes you made may not be saved"

return (e.returnValue = msg)

}

window.addEventListener("beforeunload", beforeUnloadHandler)

return () => {

window.removeEventListener("beforeunload", beforeUnloadHandler)

}

}, [])

useEffect(() => {

const handleWheel = (e) => {

if (e.ctrlKey) {

e.preventDefault()

if (!e.target.closest(".cm-editor")) return

if (e.deltaY > 0) {

updateSettings((prev) => ({

...prev,

fontSize: Math.max(fontSize - 1, 12),

}))

} else {

updateSettings((prev) => ({

...prev,

fontSize: Math.min(fontSize + 1, 24),

}))

}

setItem("settings", JSON.stringify({ ...settings, fontSize }))

}

}

window.addEventListener("wheel", handleWheel, { passive: false })

return () => {

window.removeEventListener("wheel", handleWheel)

}

}, [fontSize, settings, setItem, updateSettings])

}

export default usePageEvents

function useLocalStorage() {

const setItem = (key, value) => {

localStorage.setItem(key, value)}

const getItem = (key) => {

return localStorage.getItem(key)

} const removeItem = (key) => {

localStorage.removeItem(key)}

return { setItem, getItem, removeItem }

}

export default useLocalStorage

**10.2.2**

const express = require("express");

const app = express();

require("dotenv").config();

const http = require("http");

const cors = require("cors");

const ACTIONS = require("./utils/actions");

app.use(express.json());

app.use(cors());

const { Server } = require("socket.io");

const server = http.createServer(app);

const io = new Server(server, {

cors: {

origin: "\*",

},

});

const userSocketMap = {};

function getAllConnectedClient(roomId) {

return Array.from(io.sockets.adapter.rooms.get(roomId) || []).map(

(socketId) => {

return {

socketId,

username: userSocketMap[socketId]?.username,

status: userSocketMap[socketId]?.status,

};

}

);

}

io.on("connection", (socket) => {

// Handle user actions

socket.on(ACTIONS.JOIN, ({ roomId, username }) => {

userSocketMap[socket.id] = { username, roomId, status: ACTIONS.ONLINE };

socket.join(roomId);

const clients = getAllConnectedClient(roomId);

socket.broadcast.to(roomId).emit(ACTIONS.JOINED, {

username,

socketId: socket.id,

});

// Send clients list to all sockets in room

io.to(roomId).emit(ACTIONS.UPDATE\_CLIENTS\_LIST, { clients });

});

socket.on("disconnecting", () => {

const rooms = [...socket.rooms];

rooms.forEach((roomId) => {

const clients = getAllConnectedClient(roomId);

clients.forEach(({ socketId }) => {

io.to(socketId).emit(ACTIONS.DISCONNECTED, {

username: userSocketMap[socket.id]?.username,

socketId: socket.id,

});

});

});

delete userSocketMap[socket.id];

socket.leave();

});

// Handle file actions

socket.on(ACTIONS.SYNC\_FILES, ({ files, currentFile, socketId }) => {

io.to(socketId).emit(ACTIONS.SYNC\_FILES, { files, currentFile });

});

socket.on(ACTIONS.FILE\_CREATED, ({ roomId, file }) => {

socket.broadcast.to(roomId).emit(ACTIONS.FILE\_CREATED, { file });

});

socket.on(ACTIONS.FILE\_UPDATED, ({ roomId, file }) => {

socket.broadcast.to(roomId).emit(ACTIONS.FILE\_UPDATED, { file });

});

socket.on(ACTIONS.FILE\_RENAMED, ({ roomId, file }) => {

socket.broadcast.to(roomId).emit(ACTIONS.FILE\_RENAMED, { file });

});

socket.on(ACTIONS.FILE\_DELETED, ({ roomId, id }) => {

socket.broadcast.to(roomId).emit(ACTIONS.FILE\_DELETED, { id });

});

// Handle user status

socket.on(ACTIONS.OFFLINE, ({ roomId, socketId }) => {

userSocketMap[socketId] = {

...userSocketMap[socketId],

status: ACTIONS.OFFLINE,

};

socket.broadcast.to(roomId).emit(ACTIONS.OFFLINE, { socketId });

});

socket.on(ACTIONS.ONLINE, ({ roomId, socketId }) => {

userSocketMap[socketId] = {

...userSocketMap[socketId],

status: ACTIONS.ONLINE,

};

socket.broadcast.to(roomId).emit(ACTIONS.ONLINE, { socketId });

});

// Handle chat actions

socket.on(ACTIONS.SEND\_MESSAGE, ({ roomId, message }) => {

socket.broadcast.to(roomId).emit(ACTIONS.RECEIVE\_MESSAGE, { message });

});

});

const PORT = process.env.PORT || 3000;

app.get("/", (req, res) => {

res.send("API is running successfully");

});

server.listen(PORT, () => {

console.log(`Listening on port ${PORT}`);

});

**C.SCREENSHOTS:**