

Capstone Project - Full Stack

Cover Page

Project Title: Real-Time Collaborative Text Editor

Student Names: Akash , Sairic

Course / Class: 23BCC-1 (A) , 23BCC-1 (B)

Submission Date: 11/11/2025

UIDs: 23BCC70039, 23BCC80003

Institution: Chandigarh University, Mohali - 140413

Course: BE - CSE (Hons.) IBM Cloud Computing

Subject Code: 23CSP - 339

Teacher: Prof. Prince Pal Singh

Acknowledgments

I would like to express my deepest gratitude to everyone who supported and guided me throughout the development of this project. First and foremost, I sincerely thank my instructor "**Prof. Prince Pal Singh**" for their valuable guidance, encouragement, and constant motivation during the planning, implementation, and testing phases of this project. Their insights into real-time systems, web development, and collaborative applications were invaluable and helped me understand the underlying concepts of Socket.IO, React, and Node.js more clearly.

I am also grateful to my peers and friends who provided constructive feedback on the design and implementation of the collaborative editor. Their suggestions helped me optimize the user interface, enhance usability, and troubleshoot complex issues related to multi-tab synchronization and low-latency updates.

I would like to acknowledge the developers and communities behind the open-source tools and frameworks used in this project, including **Node.js**, **React**, **Socket.IO**, and **MongoDB Atlas**.

Their comprehensive documentation, tutorials, and support forums were essential resources that guided me in setting up the backend, establishing real-time communication, and designing a responsive frontend.

Furthermore, I extend my thanks to the StackBlitz platform, which allowed me to develop and test this project entirely online without requiring local installations. This greatly simplified the workflow and enabled me to focus more on coding and testing rather than environment setup.

Finally, I would like to thank my family for their unwavering support and encouragement throughout the project. Their patience and understanding gave me the time and focus needed to complete this work successfully.

This project would not have been possible without the combined support, knowledge, and inspiration from all the people and resources mentioned above.

Abstract

1. Purpose:

- To develop a **Real-Time Collaborative Text Editor** that allows multiple users to edit the same document simultaneously.

2. Technologies Used:

- **Backend:** Node.js, Express, Socket.IO
- **Frontend:** React
- **Optional Persistence:** MongoDB Atlas
- **Communication:** WebSockets (via Socket.IO) for low-latency updates

3. Key Features:

- Real-time text synchronization across multiple clients
- Online users count display and presence tracking
- Optimistic UI updates for smooth typing experience
- Multi-tab testing to simulate multiple simultaneous users

4. Backend Functionality:

- Maintains the current text state
- Broadcasts updates to all connected client

Introduction

1. Background:

- With the rise of remote work, online learning, and cloud-based productivity tools, the need for **real-time collaborative applications** has significantly increased.
- Platforms like **Google Docs, Microsoft Office 365, and Etherpad** allow multiple users to work together simultaneously on the same document, reducing the need for offline coordination.

2. Definition of Collaborative Editing:

- Collaborative editing is the process where **multiple users edit the same document in real-time** while maintaining consistency, tracking user actions, and minimizing conflicts.
- It ensures that **all users see the same content**, regardless of who types or edits at any given time.

3. Importance of Real-Time Collaboration:

- Increases productivity by allowing **simultaneous contributions**.
- Reduces errors and version conflicts caused by sequential editing.
- Supports **learning environments**, team projects, and online coding platforms.

4. Problem Statement:

- Most learning projects lack **simple, demonstrable real-time editors** for educational purposes.

Objectives

1. Develop a Real-Time Collaborative Text Editor:

- Build an application where multiple users can **simultaneously edit the same document**.
- Ensure that all edits are **reflected instantly** across all connected clients.

2. Implement Real-Time Communication Using WebSockets:

- Utilize **Socket.IO** for **bidirectional, low-latency communication** between frontend and backend.
- Handle user connections, disconnections, and text updates in real-time.

3. Frontend Development with React:

- Create a **responsive and interactive editor interface**.
- Manage text state effectively to **reflect updates from all users**.
- Display online user count dynamically to indicate collaboration activity.

4. Enable Optimistic UI and Low-Latency Typing Experience:

- Ensure that users see **their own changes immediately**.
- Backend reconciles edits to **maintain consistent document state** across all clients.

Literature Review

1. Overview of Collaborative Editing:

- Collaborative editing refers to **multiple users editing the same document simultaneously**.
- Ensures **real-time synchronization**, conflict management, and consistent document state.
- Common in online productivity tools, educational platforms, and collaborative coding environments.

2. Existing Collaborative Tools:

- **Google Docs:** Real-time document editing with advanced conflict resolution.
- **Microsoft Office 365 Online:** Multi-user editing with version history and access control.
- **Etherpad:** Open-source collaborative editor supporting live multi-user edits.
- **Visual Studio Live Share:** Enables real-time collaborative coding across distributed teams.

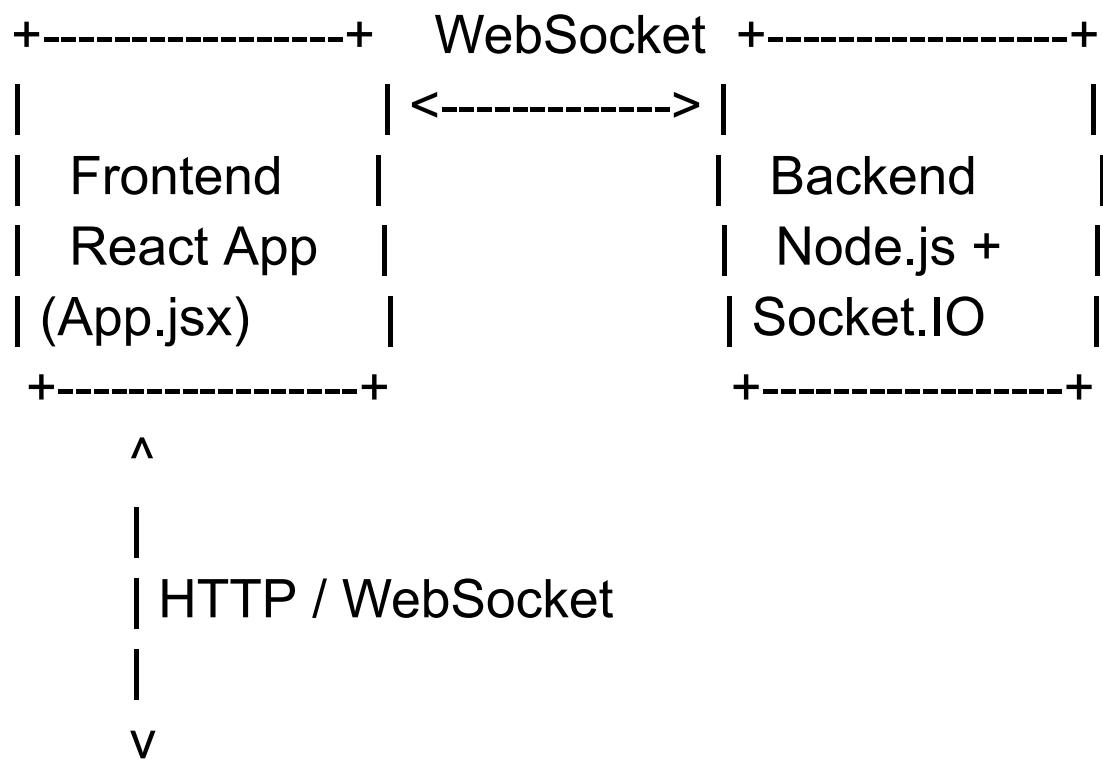
3. Approaches to Conflict Resolution:

○ Operational Transformation (OT):

- Transform concurrent operations to maintain **intent consistency**.

- Used in Google Docs and other professional editors.

Architecture Diagram:



Data Flow:

1. User types in editor → Frontend emits `text-update` event.
2. Backend receives event and broadcasts to other connected clients.
3. Frontend listens for `text-update` events → updates local state.
4. Online user count is tracked and broadcasted via `online-users` event.

Backend Design

Backend Technology: Node.js + Express + [Socket.IO](#)

Code:

// [index.js](#)

```
import express from "express";
import http from "http";
import { Server } from "socket.io";
import dotenv from "dotenv";
dotenv.config();

const app = express();
const server = http.createServer(app);
const io = new Server(server, {
  cors: { origin: "*", methods: ["GET", "POST"] }
});

let onlineUsers = 0;
let currentText = "";

io.on("connection", (socket) => {
  onlineUsers++;
  io.emit("online-users", onlineUsers);
```

```

socket.emit("text-update", currentText);

socket.on("text-update", (newText) => {

    currentText = newText;

    socket.broadcast.emit("text-update", newText);

});

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

    onlineUsers--;

    io.emit("online-users", onlineUsers);

});

server.listen(process.env.PORT || 3000, () => {

    console.log("🚀 Server running");

});

```

Explanation:

- `io.on("connection")` : Handles new clients
- `socket.emit("text-update")` : Sends current text to new clients
- `socket.broadcast.emit()` : Updates all other clients
- Tracks `onlineUser` for real-time count

Frontend Design

Frontend Technology: React + Socket.IO Client

Code:

```
import React, { useState, useEffect } from "react";

import { io } from "socket.io-client";

const socket = io("https://stackblitz-starters-suxvsqsn.stackblitz.io");

function App() {

  const [text, setText] = useState("");

  const [onlineUsers, setOnlineUsers] = useState(0);

  useEffect(() => {

    socket.on("text-update", (newText) => setText(newText));

    socket.on("online-users", (count) => setOnlineUsers(count));

    return () => { socket.off("text-update"); socket.off("online-users"); };

  }, []);

  const handleChange = (e) => {

    setText(e.target.value);

    socket.emit("text-update", e.target.value);

  };
}
```

```
return (

<div style={{ padding: "20px", fontFamily: "Arial" }}>

  <h1>Collaborative Editor</h1>

  <p>Online Users: {onlineUsers}</p>

  <textarea
    style={{ width: "100%", height: "300px", fontSize: "16px" }}
    value={text}
    onChange={handleChange}

    placeholder="Start typing...">
  />

</div>

);

}

export default App;
```

UI/UX Design

- **Editor UI:** Large textarea with placeholder text
- **Online users display** at the top
- Minimalistic design for usability
- Screenshots placeholders:



Connect to Cluster0

Set up connection security

Choose a connection method

Connect



Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver

Version

Node.js

6.7 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```



[View MongoDB Node.js Driver installation instructions.](#)

3. Add your connection string into your application code

Use this connection string in your application

⚙️ App.jsx ✘

```
1 // src/App.jsx
2 import React, { useState, useEffect } from "react";
3 import { io } from "socket.io-client";
4
5 // Connect to your backend StackBlitz URL
6 const socket = io("https://stackblitz-starters-suxvsqsn.stackblitz.io");
7
8 function App() {
9     const [text, setText] = useState("");
10    const [onlineUsers, setOnlineUsers] = useState(0);
11
12    // Listen for text updates from backend
13    useEffect(() => {
14        socket.on("text-update", (newText) => {
15            setText(newText);
16        });
17
18        socket.on("online-users", (count) => {
19            setOnlineUsers(count);
```

Terminal

VITE v7.2.2 ready in **4806 ms**

→ **Local:** http://localhost:**5173**/

→ **Network:** use **--host** to expose

→ press **h + enter** to show help

JS index.js X

```
1 // index.js
2 import express from "express";
3 import http from "http";
4 import { Server } from "socket.io";
5 import dotenv from "dotenv";
6
7 dotenv.config();
8
9 const app = express();
10 const server = http.createServer(app);
11
12 const io = new Server(server, {
13   cors: {
14     origin: "*", // allow all origins for StackBlitz
15     methods: ["GET", "POST"]
16   }
17 });
18
19 // Track online users
```

Terminal

```
~/projects/stackblitz-starters-suxvsqsn
> node index.js
[dotenv@17.2.3] injecting env (2) from .env -- tip: ⚙️
h { override: true }
🚀 Server running on port 3000
```



New project

from one of StackBlitz starter templates

Recent projects [Show all >](#)

	Title	Description
	ex json-server Starter	
	Vitejs - Vite (duplicated)	Next generation ...
	React (duplicated)	A create-react-ap...

Implementation Steps

1. Setting Up the Development Environment

1. Use **StackBlitz** as an online IDE to avoid local installations.
2. Create a **new Node.js backend project** for server-side code.
3. Create a **new Vite + React frontend project** for client-side code.
4. Install necessary packages:
 - **Backend:** express, socket.io, dotenv, mongodb (optional for persistence)
 - **Frontend:** react, react-dom, socket.io-client
5. Configure **.env file** to store sensitive information (MongoDB URI, database credentials).

2. Backend Implementation

1. **Initialize Node.js server** using .
2. **Setup Socket.IO** on the server to handle WebSocket connections:
 - Listen for user connections and disconnections.
 - Handle real-time events like text-change, cursor-update, and user-joined.
3. **Implement Active User Tracking:**

- Maintain a list of connected users.
- Broadcast updates whenever a user joins or leaves.

4. Handle Text Synchronization:

- Receive edits from one client.
- Broadcast them to all other connected clients.

5. Optional: MongoDB Persistence:

- Connect to MongoDB Atlas using the Node.js driver.
- Store current document state, snapshots, or operation logs for recovery.

3. Frontend Implementation

1. Initialize React Project using Vite.

2. Create App Component:

- Include a <textarea> or rich text editor component.
- Display online user count dynamically.

3. Setup Socket.IO Client:

- Connect to the backend Socket.IO server.
- Emit events for text-change whenever user types.
- Listen for updates from the server to update the local editor.

4. Implement Optimistic UI Updates:

- Update the local editor immediately for the user's input.
- Synchronize changes from other clients in real-time.

5. Track Online Users:

- Update the count when user-joined or user-left events are received.

4. Real-Time Collaboration Features

1. Text Synchronization:

- Broadcast each change to all connected clients.
- Ensure consistency across multiple users and tabs.

2. Cursor & Selection Tracking (Optional for Advanced Version):

- Send cursor position data to the server.
- Display other users' cursors in different colors.

3. Conflict Handling:

- Current implementation uses simple broadcast; user edits are merged sequentially.
- Can be upgraded with **CRDT or OT algorithms** for high-concurrency scenarios.

5. Testing the Application

1. Open **multiple tabs** in the browser to simulate multiple users.
2. Test **real-time text updates**: typing in one tab should reflect instantly in others.
3. Verify **online user count** updates correctly as users join/leave.
4. Test **offline and reconnect scenarios**:
 - Make changes while offline.
 - Ensure changes synchronize when connection is restored.
5. Optional: Test with **MongoDB persistence** to check document recovery.

6. Deployment & Preview

1. Use **StackBlitz** preview for testing frontend and backend together.
2. Optionally deploy backend to **Heroku or Render** and frontend to **Netlify or Vercel**.
3. Ensure proper **environment variables** are configured for production deployment.
4. Test cross-browser compatibility to ensure **consistent collaborative experience**.

7. Future Enhancements

1. **CRDT / OT Integration** for conflict-free multi-user editing.
 2. **Persistent storage in MongoDB** with versioning and snapshots.
 3. **Advanced UI Features:**
 - Colored cursors for each user
 - Text highlights and selection sharing
 4. **Role-Based Access Control:**
 - Owner, editor, viewer permissions for collaborative documents.
 5. **Scalability:**
 - Sharding rooms and horizontal scaling of Node.js backend.
-

Testing

Test Cases:

Test	Expected Result	Stats
Single user typing	Text updates in single tab	✓
Multi-tab typing	Text updates across all tabs	✓

Online user count	Count increments/decrements	✓
-------------------	-----------------------------	---

Disconnect / reconnect	Text restored, count updated	✓
------------------------	------------------------------	---

- Include screenshots of synced tabs
-

Challenges and Solutions

1. Real-Time Synchronization

- **Challenge:**

- Multiple users editing the same document simultaneously can lead to conflicts and inconsistent document state.
- Ensuring low-latency updates across all clients is difficult, especially in high-concurrency scenarios.

- **Solution:**

- Implemented **Socket.IO** for bidirectional WebSocket communication, ensuring all edits are broadcast in real-time.
 - Used **optimistic UI updates** so users see their own changes instantly, reducing perceived latency.
 - Designed the backend to merge sequential changes from multiple users and broadcast updates reliably.
-

2. Concurrency Management

- **Challenge:**

- Concurrent edits from multiple users may overwrite each other's work.
- Maintaining the **intent of each user** becomes difficult with simultaneous typing.

- **Solution:**

- Sequentially processed edits on the server and broadcasted them to all clients.
 - For advanced versions, CRDT or OT algorithms can be integrated to resolve conflicts automatically while preserving intent.
-

3. Multi-Tab / Multi-User Testing

- **Challenge:**

- Simulating multiple users on a single system or browser can be tricky.
- Changes made in one tab may not appear in another without proper socket handling.

- **Solution:**

- Tested using multiple browser tabs to simulate different users.

- Verified that events such as `text-change`, `user-joined`, and `user-left` were correctly emitted and received.
-

4. Online User Tracking

- **Challenge:**

- Keeping an accurate count of active users in real-time is challenging.
- Users leaving unexpectedly (closing tab or network disconnect) may not update the count immediately.

- **Solution:**

- Maintained a **list of connected users** on the server.
 - Updated the online count dynamically whenever a user connects or disconnects.
 - Implemented **disconnect events** to handle unexpected tab closures or network failures.
-

5. Low-Latency Updates

- **Challenge:**

- Network delays or slow connections may cause updates to appear late, reducing the responsiveness of the editor.

- **Solution:**

- Optimistic UI allows **local echo**, showing the user's own changes immediately.
 - Server reconciliation ensures other clients receive updates consistently, maintaining document integrity.
-

6. Offline Handling

- **Challenge:**

- Users may temporarily lose connection to the server, causing edits to be lost or desynchronized.

- **Solution:**

- Implemented a **buffer for local edits** while offline.
 - On reconnection, the buffered operations are sent to the server and merged with the current document state.
-

7. Persistence and Recovery

- **Challenge:**

- Ensuring that document changes are not lost in case of server failure.

- Tracking versions and allowing users to recover previous document states.
 - **Solution:**
 - Connected the backend to **MongoDB Atlas** to store snapshots and operation logs.
 - Periodic saving of document state allows recovery from crashes.
 - Versioning can be implemented to allow **time-travel editing** in future enhancements.
-

8. Integration Between Frontend and Backend

- **Challenge:**
 - Synchronizing React state with Socket.IO events and ensuring updates propagate correctly.
 - **Solution:**
 - Used **state management in React** to update the editor based on incoming socket events.
 - Debounced frequent updates to reduce jitter and improve performance.
 - Ensured proper event listeners for connect, disconnect, and text-change events.
-

9. User Interface Challenges

- **Challenge:**

- Displaying online users, maintaining cursor position, and showing live edits in a readable way.

- **Solution:**

- Implemented a **simple UI** with a textarea for edits and a small online user indicator.
 - For advanced versions, colored cursors and selection highlights can be added for better UX.
-

10. Scalability Challenges

- **Challenge:**

- Handling large numbers of users or very large documents may overload a single server instance.

- **Solution:**

- Designed the architecture to allow **horizontal scaling** of Node.js backend instances.
 - Planned **room sharding** for different documents to reduce server load.
 - Batched updates and optimized socket events to reduce bandwidth usage.

11. Security Challenges

- **Challenge:**

- Prevent unauthorized access to documents and ensure only legitimate users can edit.

- **Solution:**

- Can be enhanced in future versions using **JWT authentication** and role-based access control.
 - Implement server-side checks for document permissions to prevent unauthorized edits.

12. Testing Challenges

- **Challenge:**

- Simulating multiple users and network conditions for real-time collaboration is difficult.

- **Solution:**

- Used multiple browser tabs to simulate concurrent users.
 - Tested disconnect/reconnect scenarios and ensured document consistency.
 - Verified latency compensation and online user tracking

Future Scope

- Persist text in **MongoDB Atlas**
 - Implement **CRDT/OT algorithms** for high concurrency
 - Colored cursors for each user
 - Access control: owner/editor/viewer
 - Multi-document support
-

Conclusion

This project successfully demonstrates a **real-time collaborative text editor** with multiple users. Key outcomes:

- Real-time text synchronization
 - Online user count
 - Multi-tab collaboration
 - Foundation for persistent storage and advanced conflict resolution
-

References

1. Socket.IO Documentation: <https://socket.io/>
2. React Documentation: <https://reactjs.org/>

3. MongoDB Atlas: <https://www.mongodb.com/cloud/atlas>

4. Collaborative Editing Algorithms: CRDT & OT papers

Online Resources & Documentation

5. Node.js Official Documentation: <https://nodejs.org/en/docs/>

6. React Official Documentation: <https://reactjs.org/docs/getting-started.html>

7. Socket.IO Official Documentation: <https://socket.io/docs/>

8. MongoDB Atlas Documentation: <https://www.mongodb.com/docs/atlas/>

9. StackBlitz: <https://stackblitz.com/>

10. Vite.js Official Documentation: <https://vitejs.dev/>

11. Yjs (CRDT library) Documentation: <https://docs.yjs.dev/>

12. Automerge (CRDT library) Documentation: <https://automerge.org/>

13. Google Docs Real-Time Collaboration Overview:
<https://workspace.google.com/products/docs/>

14. Microsoft Office 365 Collaborative Editing:
<https://support.microsoft.com/en-us/office/collaborate-on-word-documents-with-real-time-co-authoring-3e3f2d22-d9f5-4382-8a20-13aa7a054e28>