CodeAlpha_Real-time Collaborative Editing Tool

Developing a **real-time collaborative editing tool** requires integrating multiple technologies to ensure seamless communication and user collaboration. Here's an overview of the architecture and a high-level implementation with key code snippets.

Architecture

1. **Frontend**: React.js for a dynamic and interactive user interface.
2. **Backend**: Node.js with WebSocket integration for real-time communication.
3. **Database**: MongoDB for storing document versions and user data.
4. **WebSocket Server**: Enables real-time updates across multiple clients.
5. **Version Control**: Tracks changes to documents and allows rollback.

Features

1. **Real-time Editing**: Users see changes made by others instantly.
2. **Cursor Synchronization**: Shows each user's cursor location and activity.

3. **User Presence Indicators**: Displays active users editing the document.
4. **Document Versioning**: Saves document versions and allows restoration.
5. **Conflict Resolution**: Handles simultaneous edits without errors.

Implementation Details

Frontend

React.js + WebSocket Integration

1. Install necessary libraries:

```
npm install react react-dom socket.io-client axios
```

2. **React App Structure**
   a. `App.js`: Main component managing WebSocket connection and state.
   b. `Editor.js`: Collaborative editor component (e.g., using `contenteditable` or libraries like Quill.js).

```
// App.js
import React, { useState, useEffect } from "react";
import io from "socket.io-client";
import Editor from "./Editor";

const socket = io("http://localhost:4000");

const App = () => {
    const [document, setDocument] = useState("");

    useEffect(() => {
        socket.on("document-update", (data) => {
            setDocument(data);
        });
    }, []);

    const handleDocumentChange = (updatedText) => {
        setDocument(updatedText);
        socket.emit("update-document", updatedText);
    };

    return (
        <div>
            <h1>Collaborative Editing</h1>
            <Editor content={document}
onChange={handleDocumentChange} />
        </div>
    );
};

export default App;
// Editor.js
import React from "react";

const Editor = ({ content, onChange }) => {
    const handleInput = (e) => {
        onChange(e.target.innerHTML);
    };
    return (
        <div
            contentEditable
            dangerouslySetInnerHTML={{ __html:
content }}
            onInput={handleInput}
            style={{
                border: "1px solid #ccc",
                padding: "10px",
                minHeight: "200px",
            }}
        ></div>
    );
};

export default Editor;
```

Backend

Node.js + WebSocket Server

1. Install necessary libraries:

```
npm install express socket.io mongoose
```

2. **Server.js**

```
const express = require("express");
const http = require("http");
const { Server } = require("socket.io");
const mongoose = require("mongoose");

const app = express();
const server = http.createServer(app);
const io = new Server(server);

mongoose.connect("mongodb://localhost:27017/collab-
edit", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});

const DocumentSchema = new mongoose.Schema({
    content: String,
});

const Document = mongoose.model("Document",
DocumentSchema);

io.on("connection", (socket) => {
    console.log("A user connected");

    socket.on("get-document", async () => {
        const doc = await Document.findOne() || await
Document.create({ content: "" });
        socket.emit("document-update", doc.content);
    });

    socket.on("update-document", async (content) => {
        await Document.updateOne({}, { content });
        socket.broadcast.emit("document-update",
content);
    });

    socket.on("disconnect", () => {
        console.log("A user disconnected");
    });
});

server.listen(4000, () => {
    console.log("Server is running on port 4000");
});
```

Database Setup

1. Install MongoDB locally or use a cloud service like MongoDB Atlas.
2. Create a `collab-edit` database and allow Node.js to connect.

Run the Project

1. Start MongoDB: `mongod`

2. Start the Node.js server: `node server.js`

3. Start the React frontend: `npm start`

Deployment

1. Use **Docker** for containerization.
2. Deploy frontend on **Vercel** or **Netlify**.
3. Deploy backend on **Heroku** or **AWS EC2**.
4. Use **MongoDB Atlas** for a managed database service.