Computer Networks And Internet Of things Lab

Docify: Collaborative Document Editor

Project Synopsis

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

SUBMITTED BY

GAUTAM GROVER (2010368)

DHRUV VISHNOI (20103322)

PRERNA SINGH (20103090)

Group Number	**4**
Project Title	**Docify: Collaborative Document Editor**
File Name	
Lab Faculty	**Parmeet Kaur Sodhi,Vivek Kumar Singh**

Group Members and Contribution Details

Sno	Roll Number	Name	E-mail	Contribution in this work (write your contribution in this work such as task done, tools explored, knowledge gained and presented, etc.)
1	20103068	Gautam Grover	Gautam34563@gmail.com	Made the React Frontend using quill editor
2	20103090	Prerna Singh	singhPrerna455@gmail.com	Used the socket.io library to make the documents collaborative
3	20103322	Dhruv Vishnoi	dhruvvishnoi136@gmail.com	Saved all the documents to the mongoDB database

Declaration

I/We hereby declare that this submission is my/our own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text. I/We accept the use of the material presented in this report for Education/Research/Teaching purpose by the faculty.

Signature	Signature	Signature
Name:Prerna Singh Date :- 24/04/2023 Place:-JIIT NOIDA-62	Name:Gautam Grover Date :- 24/04/2023 Place:-JIIT NOIDA-62	Name:-Dhruv Vishnoi Date :- 24/04/2023 Place:-JIIT NOIDA-62

Abstract

Docify is a collaborative editor designed to enable multiple users to work on the same document simultaneously in real-time. The application is built using a combination of cutting-edge technologies, including ReactJS, Socket.IO, ExpressJS, and MongoDB.

ReactJS is a popular JavaScript library used to build user interfaces. Docify leverages ReactJS to create a smooth and responsive user experience that enables users to create, edit, and collaborate on documents seamlessly.

Socket.IO is a real-time event-driven communication library that enables real-time, bidirectional communication between clients and servers. Docify leverages Socket.IO to facilitate real-time collaboration, allowing multiple users to work on the same document simultaneously.

ExpressJS is a flexible and powerful web application framework that is used to build scalable and robust web applications. Docify uses ExpressJS to manage server-side logic and to provide RESTful API endpoints for handling document creation, editing, and sharing.

Finally, MongoDB is a flexible and scalable NoSQL database that enables efficient and fast data storage and retrieval. Docify uses MongoDB to store and manage documents, as well as user authentication and authorization data.

Overall, the combination of these technologies enables Docify to provide a seamless and efficient collaborative editing experience that enables multiple users to work on the same document simultaneously, in real-time, while ensuring efficient data storage and retrieval, scalability, and reliability.

Scope of the project:-

The scope of the Docify project is to develop a collaborative editing application that enables multiple users to work on the same document simultaneously, in real-time. The application aims to provide a seamless and efficient editing experience that enables users to create, edit, and collaborate on documents with ease.

Docify's target users are individuals and teams who need to collaborate on documents remotely, such as remote teams, students, teachers, and professionals in various fields.

a) Functional Requirements:-

Document creation: Users should be able to create new documents and save them for future editing.

Real-time collaboration: Multiple users should be able to work on the same document simultaneously, with real-time updates to changes made by others.

Text editing: Users should be able to add, delete, and modify text in the document, including font type, size, color, and style.

Formatting: Users should be able to apply formatting to text, including bold, italic, underline, and strikethrough.

Search: Users should be able to search for specific text within the document.

Spell-checking: The application should have a built-in spell-checker to help users avoid errors in their documents.

b) Non Functional Requirements:-

Performance: The application should be able to handle a large number of users and documents without experiencing slowdowns or crashes. It should be designed to be scalable, efficient, and reliable.

Usability: The application should be intuitive and easy to use, with a clean and user-friendly interface. It should also be accessible to users with disabilities and support multiple languages.

Compatibility: The application should be compatible with different web browsers and devices, including desktops, laptops, tablets, and smartphones.

Availability: The application should be available 24/7, with minimal downtime for maintenance or upgrades.

Tools and technologies used:-

ReactJS is a popular JavaScript library for building user interfaces that allows you to create reusable UI components and manage application state in a predictable and efficient way.

Socket.IO is a JavaScript library that enables real-time, bidirectional communication between web clients and servers using web sockets.

ExpressJS is a popular and lightweight web application framework for Node.js that simplifies the process of creating and managing web applications by providing a set of robust and flexible features for routing, middleware, and more.

MongoDB is a popular NoSQL database management system that uses a document-oriented data model to provide scalability, flexibility, and performance for modern web and mobile applications.

Quill is a JavaScript library that provides an open source WYSIWYG editor for creating rich text content with a modern, user-friendly interface.

Implementation Details

Backend server:

- Used Express.js to create a server that listens for incoming requests and responds with appropriate data.
- Used MongoDB using mongoose to store the documents in the database
- Used Socket.IO to enable real-time collaboration between multiple users. Socket.IO is a popular library for building real-time applications that works seamlessly with Node.js and can handle bidirectional communication between clients and the server.

Frontend UI:

- Used ReactJS to create the user interface for the Google Docs clone editor. ReactJS is a popular JavaScript library for building UI components that can be easily reused and updated.
- Used React Router to manage the different views of the application. React Router is a popular library for managing client-side routing in single-page applications.
- Used The Quill Editor to make a document editor so that users can write and edit things, it also has autocorrect functionality.
- Some of the main reasons we chose Quill are its ease of use, customization options, awesome documentation, and the fact that it is open-source.
- Quill supports all formats found in Trix, and also supports text color, font, background, size, superscript, subscript, underline, text alignment, text direction, syntax highlighted code, videos, and formulas, which are not supported in Trix.

Real-time collaboration:

- Used Socket.IO to enable real-time communication between multiple users. When a user connects to the server, they should join a specific room based on the document they want to edit. The server can then broadcast any changes made to that document to all users in the room.
- Created a collaborative text editor that allows multiple users to edit the same document in real-time. The editor should be able to handle changes made by different users and display them in real-time.

Code:-

Client/src/App.js

```
import TextEditor from "./TextEditor"
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Redirect,
} from "react-router-dom"
import { v4 as uuidV4 } from "uuid"

function App() {
  return (
    <Router>
      <Switch>
        <Route path="/" exact>
          <Redirect to={`/documents/${uuidV4()}`} />
        </Route>
        <Route path="/documents/:id">
          <TextEditor />
        </Route>
      </Switch>
    </Router>
  )
}

export default App
```

Client/src/index.js

```
import React from "react"
import ReactDOM from "react-dom"
import App from "./App"
import "./styles.css"

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
)
```

Client/src/TextEditor.js

```
import { useCallback, useEffect, useState } from "react"
import Quill from "quill"
import "quill/dist/quill.snow.css"
import { io } from "socket.io-client"
import { useParams } from "react-router-dom"

const SAVE_INTERVAL_MS = 2000
const TOOLBAR_OPTIONS = [
  [{ header: [1, 2, 3, 4, 5, 6, false] }],
  [{ font: [] }],
  [{ list: "ordered" }, { list: "bullet" }],
  ["bold", "italic", "underline"],
  [{ color: [] }, { background: [] }],
  [{ script: "sub" }, { script: "super" }],
```

```
  [{ align: [] }],
  ["image", "blockquote", "code-block"],
  ["clean"],
]

export default function TextEditor() {
  const { id: documentId } = useParams()
  const [socket, setSocket] = useState()
  const [quill, setQuill] = useState()

  useEffect(() => {
    const s = io("http://localhost:3001")
    setSocket(s)

    return () => {
      s.disconnect()
    }
  }, [])

  useEffect(() => {
    if (socket == null || quill == null) return

    socket.once("load-document", document => {
      quill.setContents(document)
      quill.enable()
    })

    socket.emit("get-document", documentId)
  }, [socket, quill, documentId])
```

```
useEffect(() => {
  if (socket == null || quill == null) return

  const interval = setInterval(() => {
    socket.emit("save-document", quill.getContents())
  }, SAVE_INTERVAL_MS)

  return () => {
    clearInterval(interval)
  }
}, [socket, quill])

useEffect(() => {
  if (socket == null || quill == null) return

  const handler = delta => {
    quill.updateContents(delta)
  }
  socket.on("receive-changes", handler)

  return () => {
    socket.off("receive-changes", handler)
  }
}, [socket, quill])

useEffect(() => {
  if (socket == null || quill == null) return
```

```
    const handler = (delta, oldDelta, source) => {
      if (source !== "user") return
      socket.emit("send-changes", delta)
    }
    quill.on("text-change", handler)

    return () => {
      quill.off("text-change", handler)
    }
  }, [socket, quill])

  const wrapperRef = useCallback(wrapper => {
    if (wrapper == null) return

    wrapper.innerHTML = ""
    const editor = document.createElement("div")
    wrapper.append(editor)
    const q = new Quill(editor, {
      theme: "snow",
      modules: { toolbar: TOOLBAR_OPTIONS },
    })
    q.disable()
    q.setText("Loading...")
    setQuill(q)
  }, [])
     return   <><h3   className="firstH1">Docify:Collaborative   Document
Editor</h3><div className="container" ref={wrapperRef}></div></>
}
```

Server/Document.js

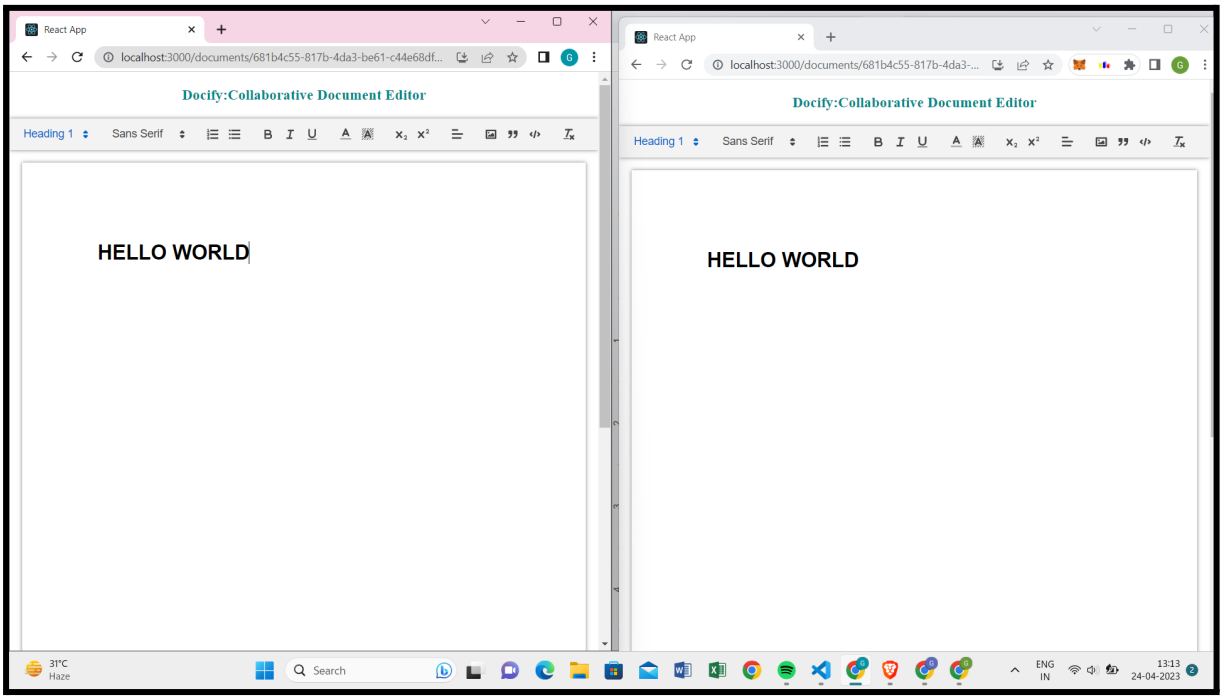
```
const { Schema, model } = require("mongoose")


const Document = new Schema({
  _id: String,
  data: Object,
})


module.exports = model("Document", Document)
```

Server/index.js

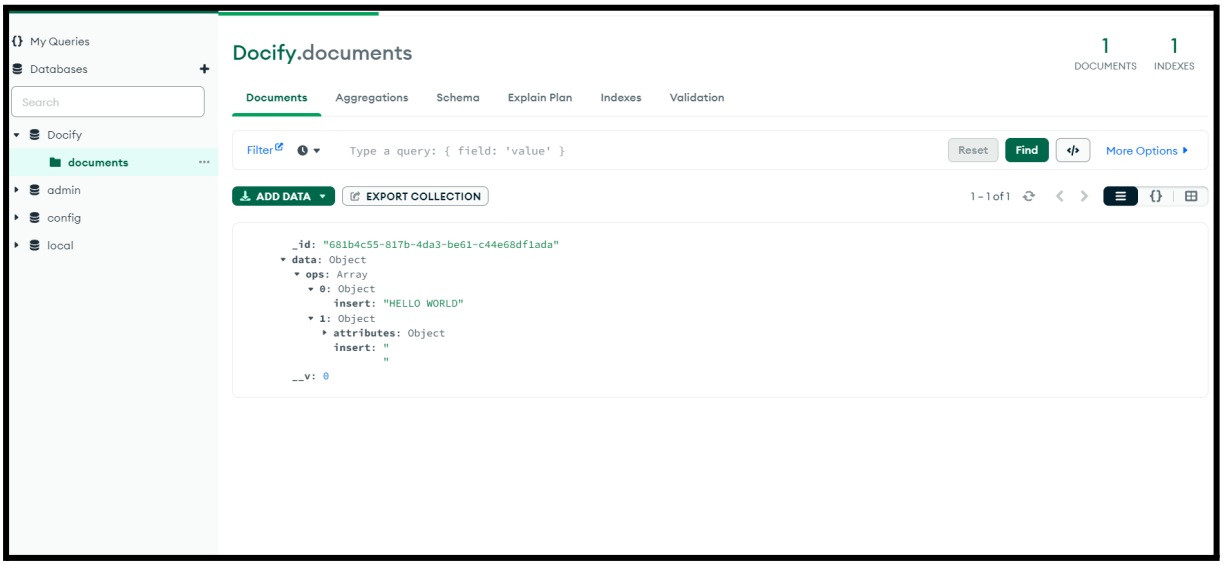
```
const mongoose = require("mongoose")
const Document = require("./Document")


mongoose.connect("mongodb://127.0.0.1:27017/Docify", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
  useCreateIndex: true,
})


const io = require("socket.io")(3001, {
  cors: {
    origin: "http://localhost:3000",
    methods: ["GET", "POST"],
  },
})


const defaultValue = ""
```

```
io.on("connection", socket => {
  socket.on("get-document", async documentId => {
    const document = await findOrCreateDocument(documentId)
    socket.join(documentId)
    socket.emit("load-document", document.data)

    socket.on("send-changes", delta => {
      socket.broadcast.to(documentId).emit("receive-changes", delta)
    })

    socket.on("save-document", async data => {
      await Document.findByIdAndUpdate(documentId, { data })
    })
  })
})

async function findOrCreateDocument(id) {
  if (id == null) return

  const document = await Document.findById(id)
  if (document) return document
  return await Document.create({ _id: id, data: defaultValue })
}
```

Screenshots and explanation



As we can see that in different browsers the same link is searched and changes are appearing concurrently



And from above we can see that all of our files are getting saved in MongoDB

References

- https://www.geeksforgeeks.org/react-js-introduction-working/
- https://www.geeksforgeeks.org/mongodb-an-introduction/
- https://socket.io/how-to/use-with-react
- https://www.techtarget.com/searchnetworking/definition/client-server#:~:text=What%20is%20client%2Dserver%3F,computing%20model%20used%20by%20mainframes.
- https://www.geeksforgeeks.org/client-server-model/