Program: Computer Engineering and software systems

Course Code: CSE 354

Course Name: Distributed Computing

Spring 2022

Names:

Mahmoud Mourad Youssef   *18P6555*

Mohamed Ashraf Fathy   *18P5993*

Youssef Hany Onsy   *18P1789*

Table of Contents

Table of figures

1. Links

GitHub link: https://github.com/MohamedAshraf67/Distributed-Computing

Video (demo) link: https://drive.google.com/drive/folders/1yAM75mdhTZk46nupE4yOLfrbAf-7Mz6o

Source code:
https://drive.google.com/drive/folders/1LRCvLyj6oHMfyUgK0moUG72HFRi8V09N?usp=sharing

2. Introduction

Distributed system which are also known as distributed computing systems, are systems made up of many components not just one, and these components are located on different machines, communicating and coordinating actions through message passing in the background in order to appear for the user as a single coherent system, these components are also called autonomous nodes. There are a lot of components that can be added to the system like physical servers, VMs, memory, containers ...etc. all these can be considered as a part of our distributed computing system. Although these nodes run without a global clock, they also run simultaneously.

Distributed systems have many pros, one of the main advantages is that with time passing technologies grow a lot bigger than before, hence larger files and more tasks, which makes computing all these tasks and files by one computer significantly harder and slower and more complicated, since the distributed systems contain multiple components (autonomous nodes) hence the work load is distributed across different machines, moreover since we have a lot of machines automatically there is a higher fault tolerance therefore we can depend on the system more so if one node faulted or halted or even totally broke down there are a number of other machines to continue the task. Distributed computing systems can grow bigger and bigger by adding more CPUs to increase the computing power hence speeding up the process.

These systems have many types including, Client-Server, Peer-to-Peer(P2P), the three-tier and the n-tier.

In this system we chose to use the Client-Server model, as it is one of the most popular and well-known type. This model has a central server to store and/or process in other words to compute data to reach our goal, along with multiple networked computer nodes.

3. Project description

This text-editor project is a web application which allows reading, writing, editing and saving documents. There are multiple autonomous nodes in this system which are responsible for sharing resources and performing in real-time.

There are a number of clients introduced to the system where any change done to the document is stored on the spot(real-time).

In this project we chose to use the Client-Server model as previously stated, react for the UI, Quill for text editing, MongoDB(database) to store the documents and any changes made to them along with Socket.io to implement the communication within the client and the server

4. Project beneficiaries

This text-editor is suitable for anyone who wants to collaborate with other users in order to make a document with every change seen and saved automatically to the database (real-time as explained before)

This project is similar to "Google docs clone".

5. Detailed Analysis

The project communicates through a client server communication which is illustrated in client-server architecture.
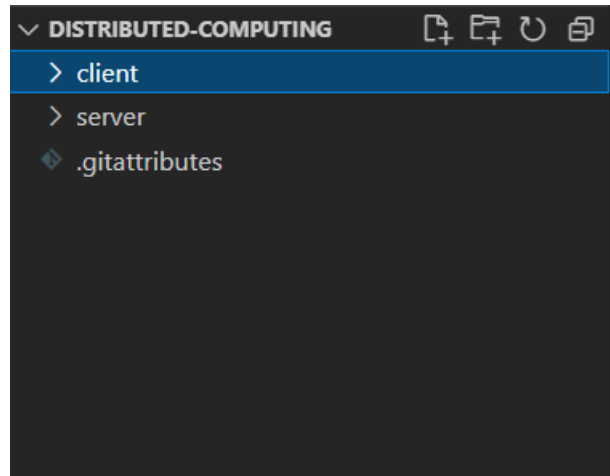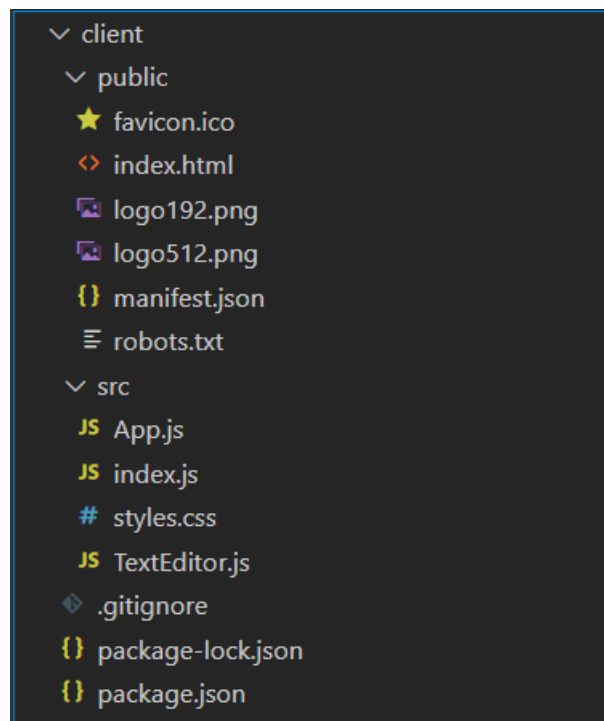

Figure 1

5.1. Client Side

All the user interaction will deal with the client side of the project contains our HTML, CSS, and java script files (mainly react) for front end usage and user interface.

- App.js

Importing uuidv4 to generate a document with a unique id to be saved later in our database the function app directs the website path to the document using our routers and switches.

- Index.js

By calling app function in the app.js we render our document into the webpage.

- Texteditor.js

We set the time interval for updates to 2 seconds if the client doesn't reach the server or the server crashed then it won't load the document and will simply type "loading…" waiting for the server.

Importing quill text editor and io from socket.io function text editor initialize the document by giving the document an id, socket, and linkage to the quill text editor. We set the parameter of the io to process.env.REACT_APP_SERVER to open a socket from the client to the server. And then test if the socket isn't equal to NULL then we continue by loading documents, if the document we load doesn't exist we create a new document with a new id, then we save the document. Then there is a function to check for modifications by updating contents and receive changes on the document. Wrapper is used to write on the webpage using innerHTML.

5.2. Server Side
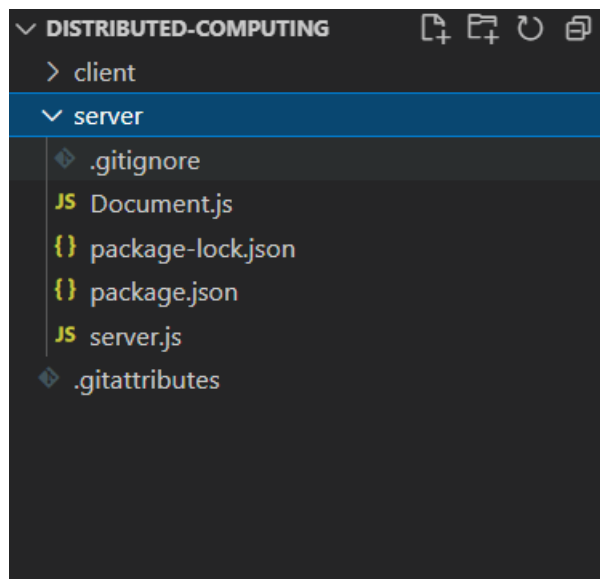
- Document.js

Simply creating a database schema for every document containing the document id and data.

- Server.js

Importing mongoose database object for mongodb, we use mongoose.connect to connect to our cloud database using the database URL and with username and a password for the user using this

database, we require the socket.io to open the port for the server to connect to the client, we simply initialize the origin to * to accept any host, and the methods used is get and post which is similar to the request-reply message in the client-server architecture. After we connect a socket is opened and wait to find or create a document.

5.3.   Database

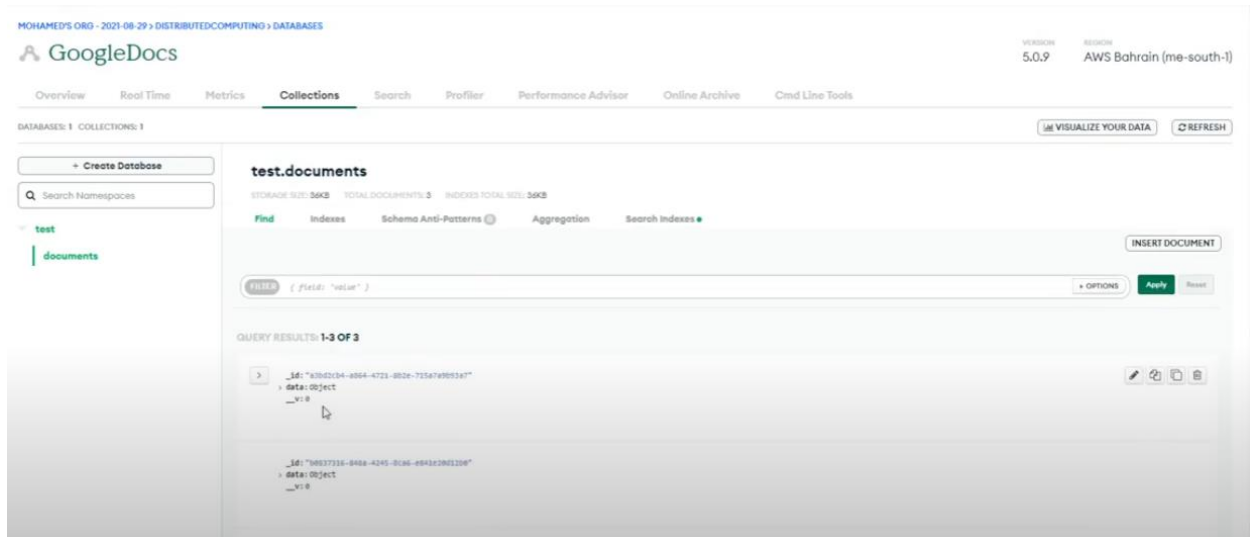We used mongo DB to create our database specifically we used a cloud service (mongo ATLAS) here we see our saved documents in our database.

Figure 2

Then set the IP access entry list to 0.0.0.0 to allow any user to access our documents.

Figure 3

5.4.   Deploying on Heroku

- Taking the git ignore file in the client copying it into the server.

```
3    # dependencies
4    /node_modules
5    /.pnp
6    .pnp.js
7
8    # testing
9    /coverage
10
11   # production
12   /build
13
14   # misc
15   .DS_Store
16   .env.local
17   .env.development.local
18   .env.test.local
19   .env.production.local
20
21   npm-debug.log*
22   yarn-debug.log*
23   yarn-error.log*
24
```

Figure 4

- linking the client's socket to the server

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

  useEffect(() => {
    const s = io(process.env.REACT_APP_SERVER)
    setSocket(s)

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

Figure 5

- adding start node server.js in the package.json file in the server.

```
  "scripts": {
    "start":"node server.js",
    "devStart": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

Figure 6

- Opening a port in the server file to the client

```
const io = require("socket.io")(process.env.PORT, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
  },
})
```

Figure 7

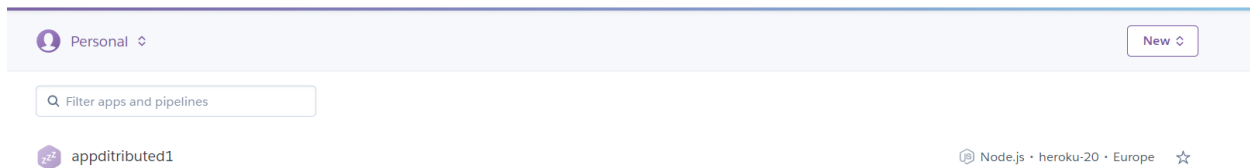- Creating two apps in the Heroku dashboard one for the client and other for the server by clicking on New

Personal ⌄   New ⌄

🔍 Filter apps and pipelines

appditributed1   Node.js · heroku-20 · Europe   ☆

Figure 8

- After creating the two apps we start deploying the server app by following the line of commands in the deploy section.

Install the Heroku CLI

Download and install the Heroku CLI.

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Clone the repository

Use Git to clone serverdist1's source code to your local machine.

```
$ heroku git:clone -a serverdist1
$ cd serverdist1
```

Deploy your changes

Make some changes to the code you just cloned and deploy them to Heroku using Git.

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

Figure 9

- Then we start deploying the client app by following these line of commands

Install the Heroku CLI

Download and install the Heroku CLI.

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Clone the repository

Use Git to clone appditributed1's source code to your local machine.

```
$ heroku git:clone -a appditributed1
$ cd appditributed1
```
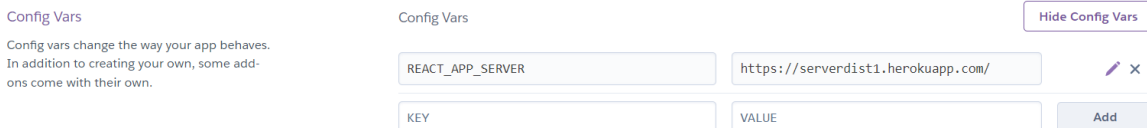
Deploy your changes

Make some changes to the code you just cloned and deploy them to Heroku using Git.

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

Figure 10

- Then go to the settings of the client app to adjust some settings by recognizing a key to redirect us to the server app URL

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Config Vars		Hide Config Vars
REACT_APP_SERVER	https://serverdist1.herokuapp.com/	✏ ×
KEY	VALUE	Add

Figure 11

- now the application is ready to use to use click on the open app button.

Personal ◇ > appditributed1   ☆ Open app   More ◇

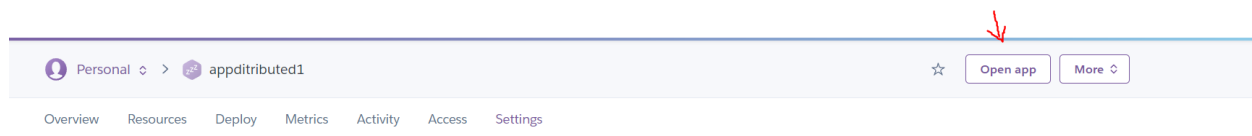Overview   Resources   Deploy   Metrics   Activity   Access   Settings

Figure 12

6. Architecture diagrams

We use client-server architecture to communicate between the client and the server the main levels are User level, application layer, data layer.

- User level: the user simply types in what he wants to add in the text editor interacting with the client app.
- Application layer: the client side we used quill text editor due to its many features it can notify you with any changes, offer multi-user editing by saving the modifications, without overwriting any of the text editor content. The server-side listens to the changes done by the client by requests these requests is saved into the database with its own unique document id can be fetched at any given time then is sent back to the client by reply message, the client and the server communicate using socket.io which is a basic library for socket programming they use the methods initialized GET and POST.
- Data layer: the database for storing the documents with its own ID, we used mongo DB as stated above using mongo ATLAS as a cloud online service to store the documents in it.
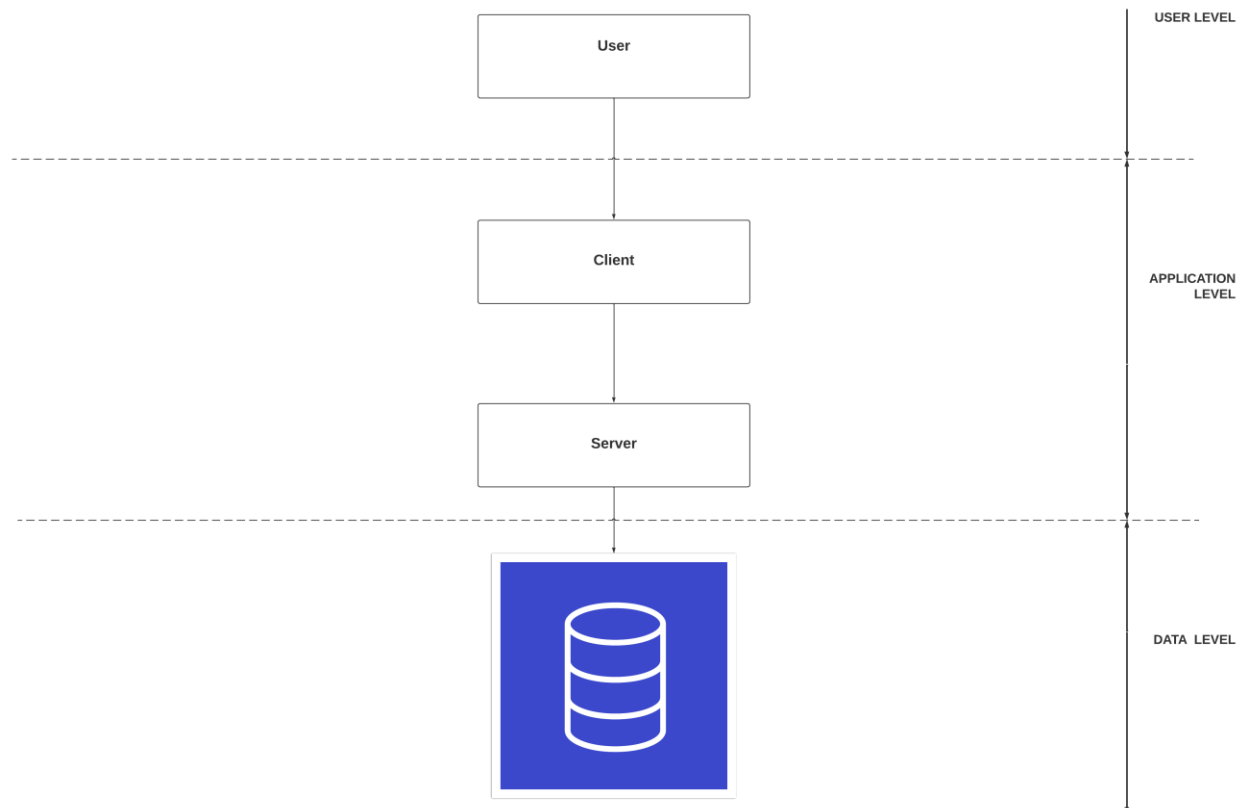

Figure 13

7. Testing application

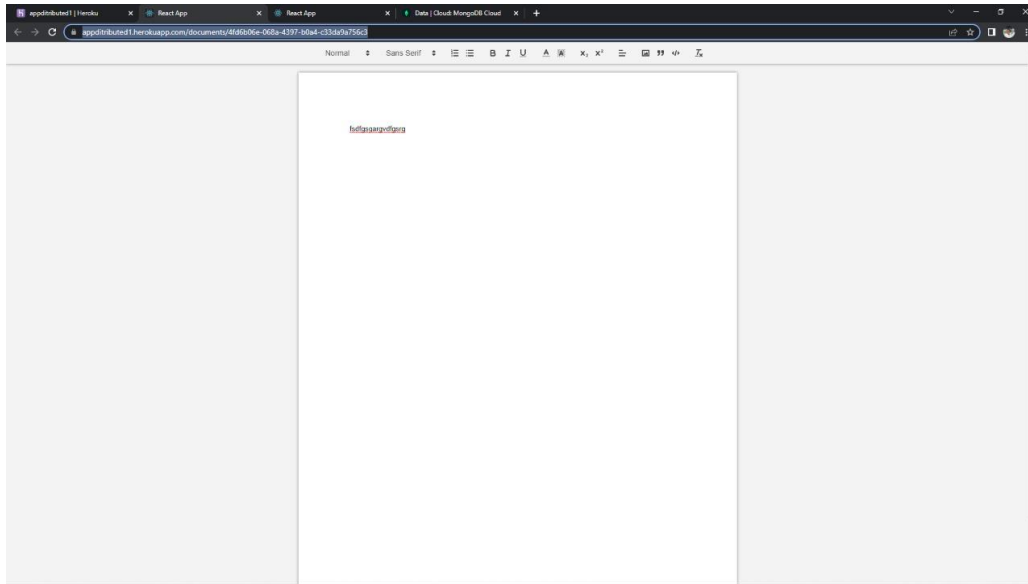- Opening two documents with same id (copying the URL into a new tab)
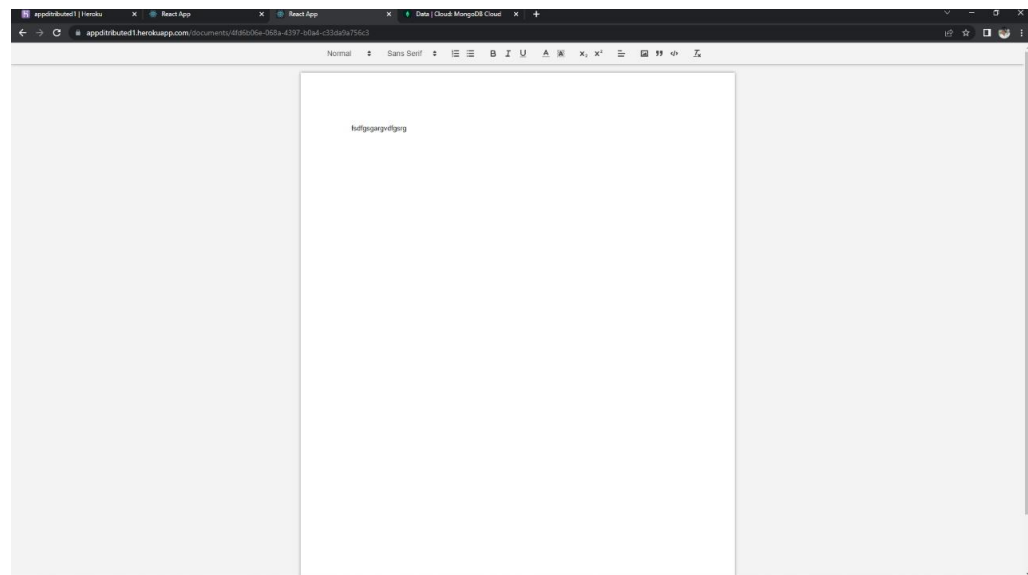

Figure 14

Figure 15

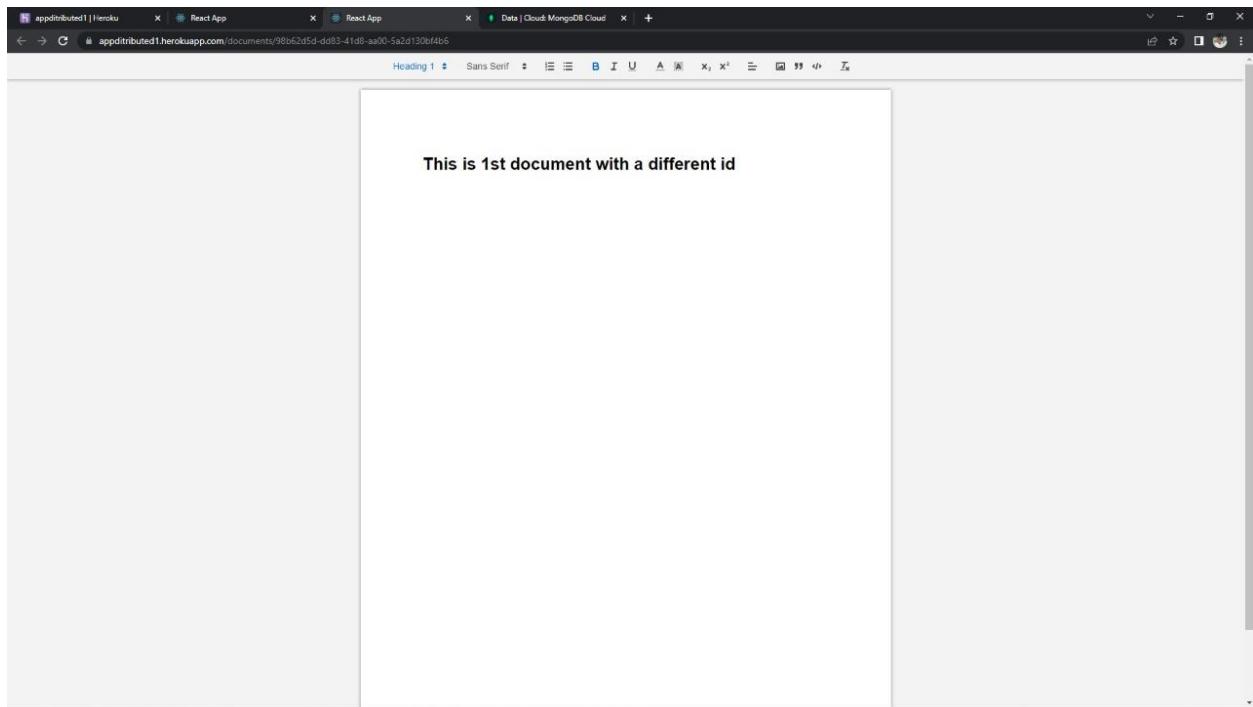- Opening two documents with different ids (clicking open app)
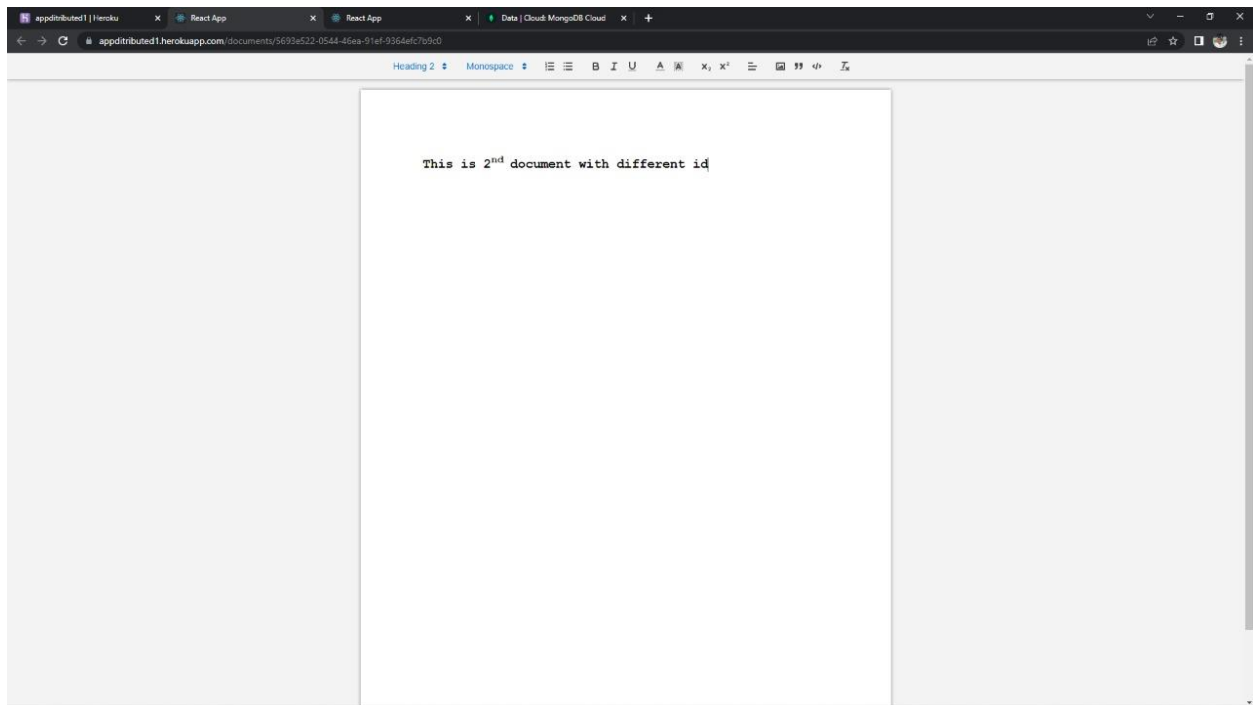

Figure 16

Figure 17

- If the server is unavailable

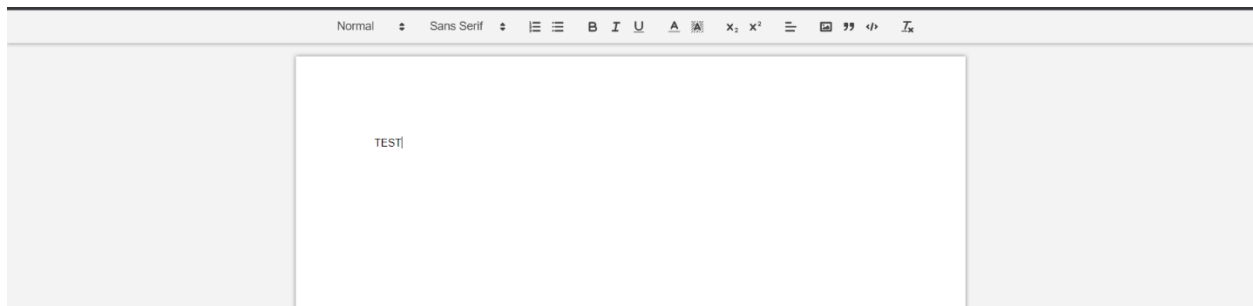Before: before we terminate the server.
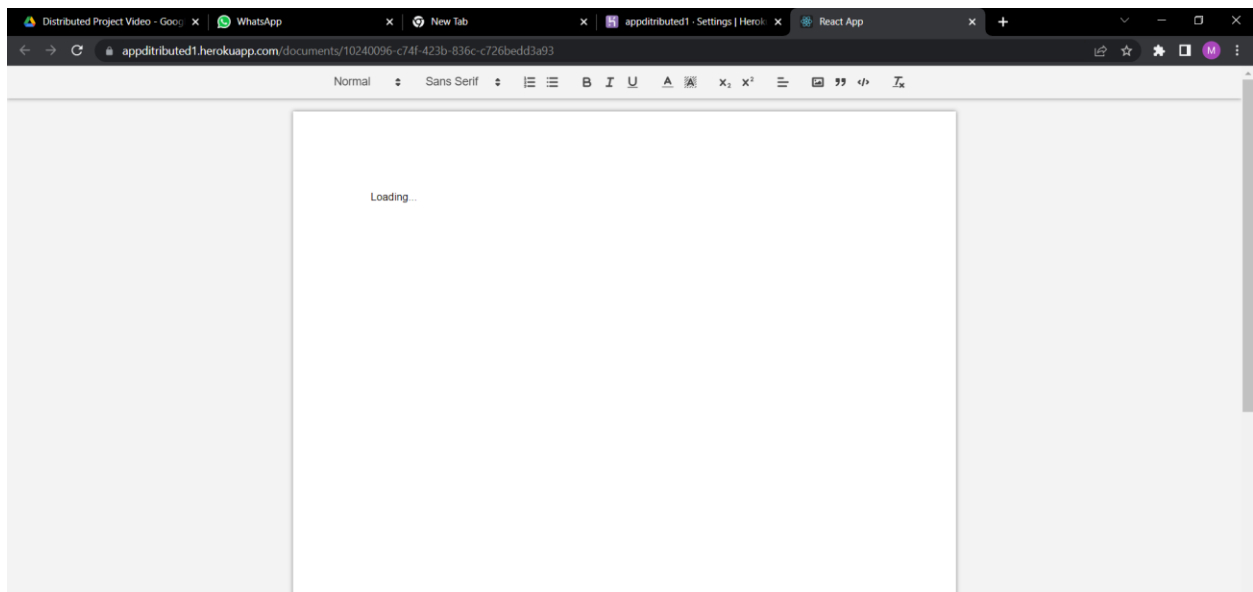

Figure 18

After: we terminate the server

Figure 19

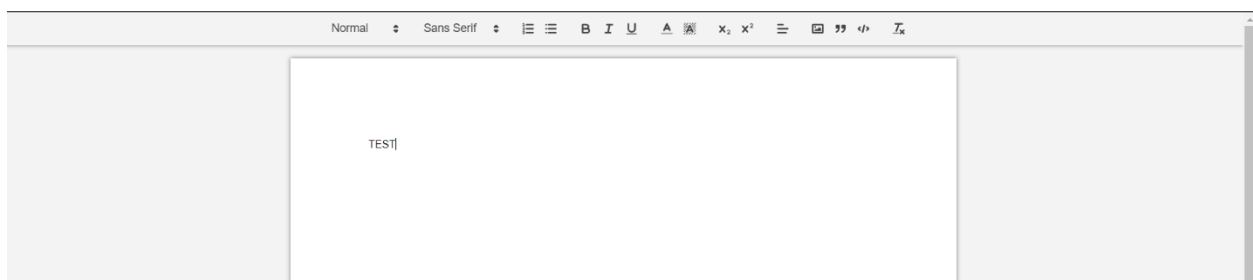Then we restore the server we find our data still saved.

Figure 20
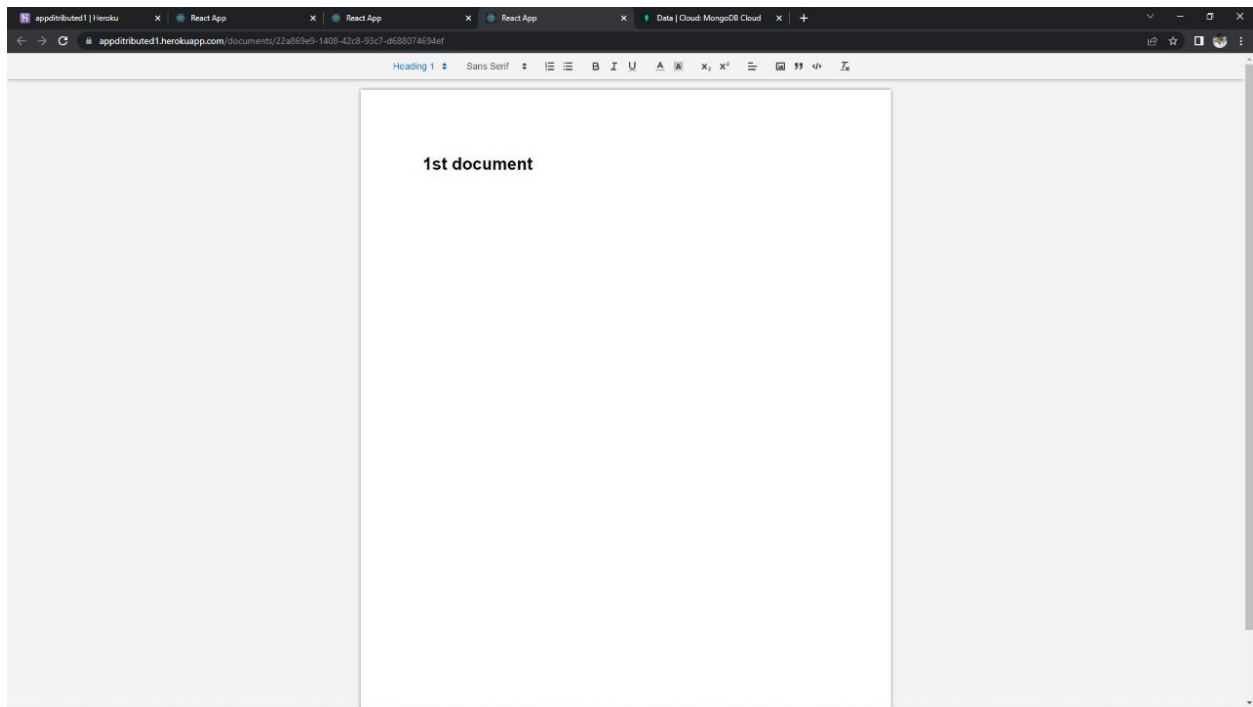
14

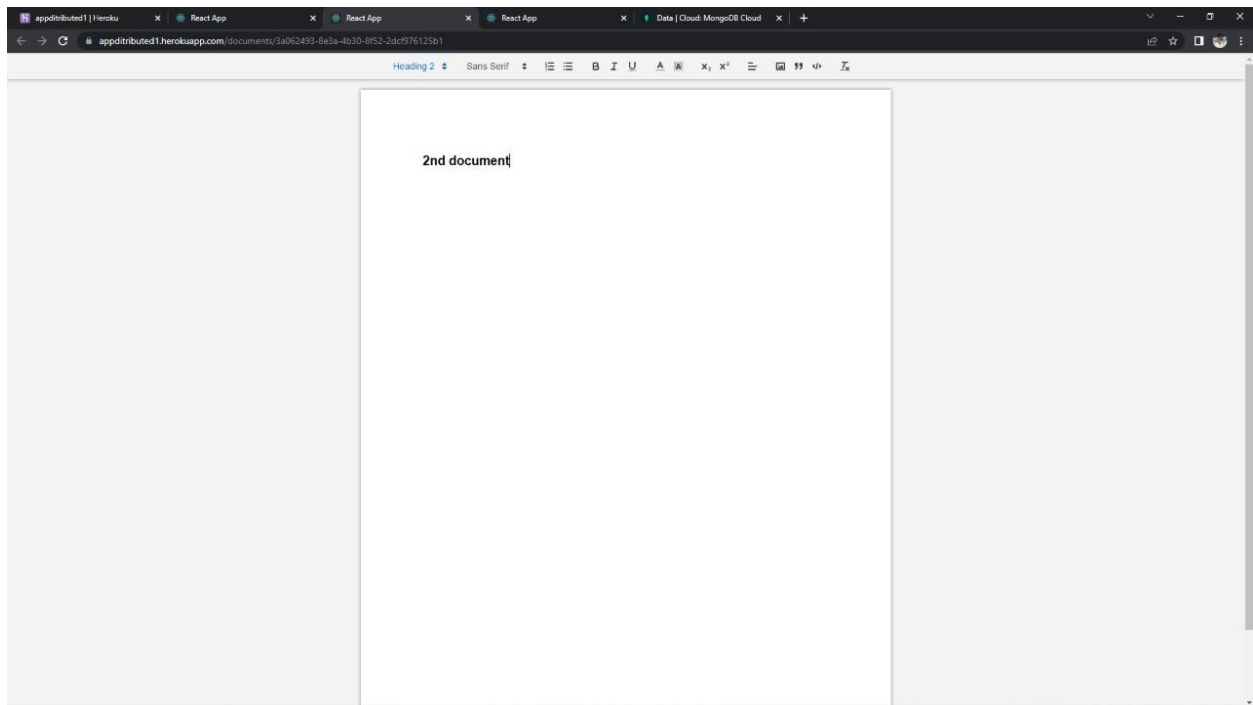- Opening several documents with different id
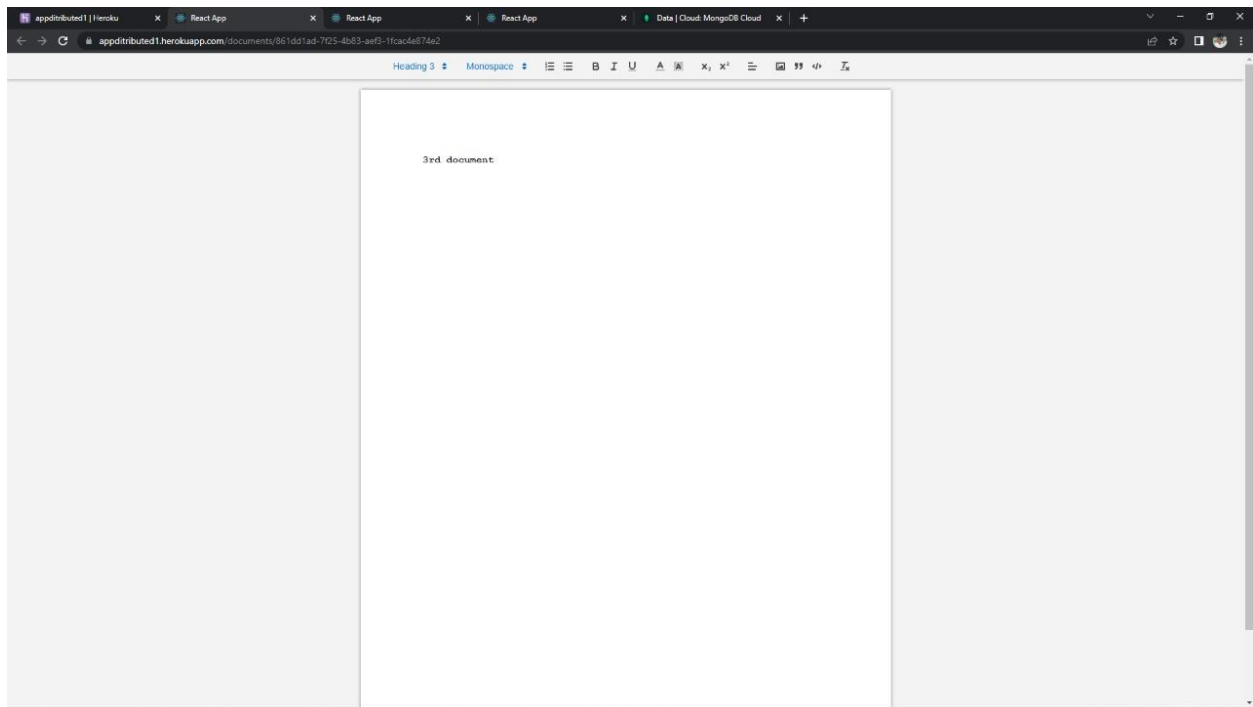

Figure 21

Figure 22

Figure 23

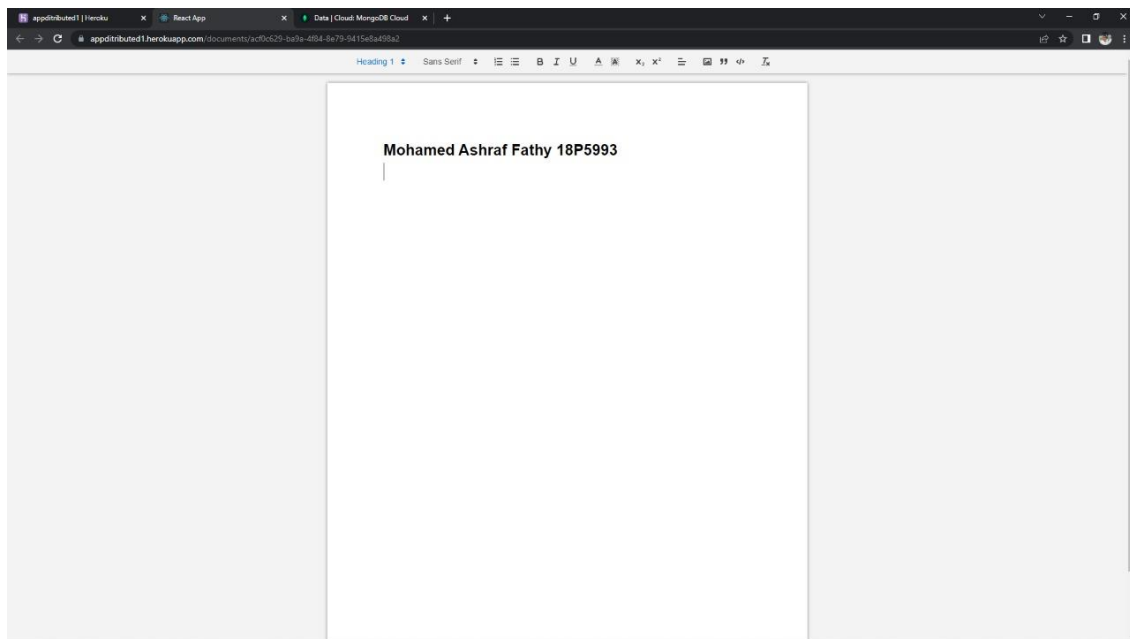- Multiple users open the same document
  First user screen

Figure 24

Second user screen
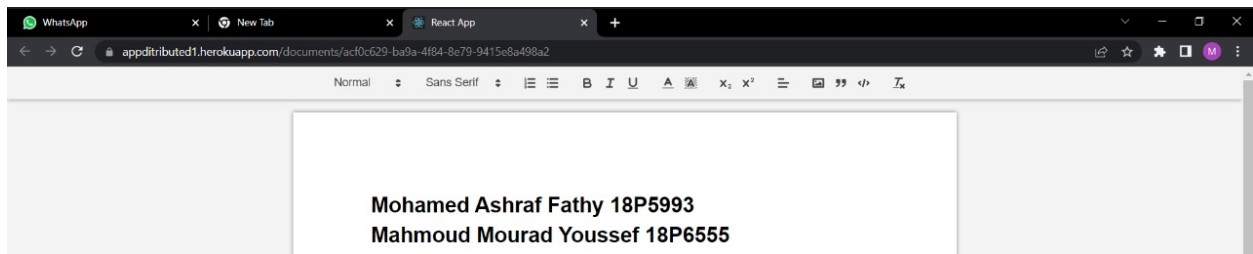
Figure 25

Third user screen

Figure 26

8. End User Guide

https://appditributed1.herokuapp.com/
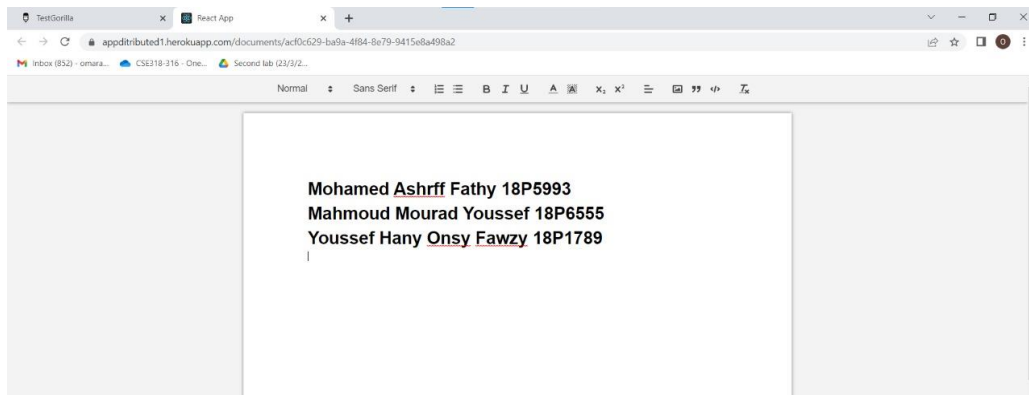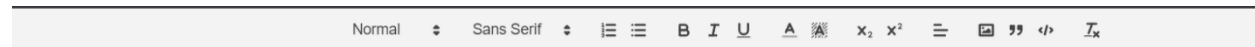
To create a new document, follow the previous link.

To load an existing document, add to the previous link /document/id where the id is the document id saved in our database.

Explain text editor features (tool bar):

Normal ⇕ Sans Serif ⇕ ≣ ≣ B I U A 🖊 x₂ x² ≡ 🖼 99 </> I𝐱

- Text type (normal, heading 1, heading 2, …, heading 6)
- Text font (sans serif, serif, monospace)
- Ordered lists
- Bulleted lists
- Text weight (bold, italic, underlined)
- Text color
- Text shadow color
- Sub script and super script
- Text alignment
- Picture upload
- Quotation marks
- Insert code area
- Default button

Every user who has the document ID can edit the document at the same time, and the changes will appear to the other users on the document area.

9. Task Breakdown and roles

Mohamed: created application (app.js), text editor (texteditor.js), application deployment (client) with its git ignore file.

Mahmoud: created server application (server.js), document schema (document.js), server deployment with its git ignore file.

Youssef: created public folder that has index.html containing all web page content, and style.css styling the webpage, documentation.

Milestones:

Client setup: in the client folder there is react application we created then text editor file contains code for the text editor.

Quill: used to implement its text editor along with its styling.

Server setup: in server folder we installed socket io for communication between client and server

Changes: a function is created to take changes done by the client to the server.

Documents: we created multiple documents using routing and uuid we give each document its own unique id.

Database: we setup mongo DB cloud service to store the documents by their unique id with its own content.

10. Conclusion

Finally, our distributed computing system is based on the Client-Server model which can and will help, multiple users to create and edit documents simultaneously with anyone overwriting another, moreover our project has a lot of pros such as flexibility, reliability, dependability, scalability and much higher performance since it is a distributed system

11. References

- Accessed June 2022, https://socket.io/
- Accessed June 2022, https://www.npmjs.com/package/socket.io
- Accessed June 2022, https://www.mongodb.com/docs/atlas/getting-started/
- Accessed June 2022, https://quilljs.com/guides/why-quill/