

DELHI TECHNOLOGICAL UNIVERSITY

2020-2021

COMPUTER NETWORKS PROJECT REPORT (IT-303)



PROJECT TITLE:-

COLLABX: Document Collaborator with Chat

(<https://github.com/RitikGarg7/CollabX>)

Submitted by:-

RITIK GARG

2K18/IT/099

ROHIT AGGARWAL

2K18/IT/101

Submitted to:- Mrs. Anamika Chauhan

Objective:-

The aim of this project is to implement an application through which multiple users can collaborate in real time to write a document and chat. Using this application, users can collaborate with each other on projects/documents and other such things on real-time similar to google docs, but the main advantage of our application is that real-time chat facility which is not there in google docs is also provided. Application of this kind can be very helpful in these hard times where sometimes many students are required to make projects in groups.

Motivation behind project:-

The project we have created focuses on resolving a practical problem, rather than a hypothetical one. We decided not to address an obsolete or a common issue which has been taken care of in numerous ways, instead we preferred to go for something new.

We as students have to collaborate on many different occasions to complete a task. These tasks many times require us to share our portion of the task with each other and hence work accordingly, as a team. 'CollabX' allows all the team members to work simultaneously in a coordinated manner, as if they are present next to each other physically. At this platform, all authorized team members can view the progress of work done by other members and coordinate effectively to increase productivity.

Technical Aspects:-

We will be using HTML and CSS for our front end styling of the application. The main component in this application will be use of sockets to setup a real time end-to-end connection between server and clients. Socket Programming provides the facility to implement real time analytic, binary streaming, instantaneous messaging and document collaboration. We will be using Socket.IO. It is a custom real time transport protocol. Socket.IO requires the usage of socket.IO libraries on both server and client side. There is end-to-end communication as in WebSocket, it allows broadcasting to multiple nodes. The multiple clients send connection

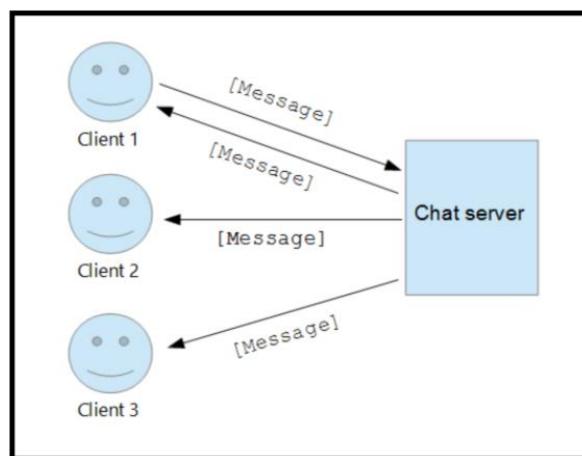
requests and the server accepts the requests by creating a new socket for each with its local address and port. Thus, the socket can make connections and send or receive the data from the socket. We will use MySQL as the database to store information regarding users and documents.

Deep-dive into Socket-IO:-

Socket.IO allows bi-directional communication between client and server. Bi-directional communication is enabled when a client has Socket.IO in the browser, and a server has also integrated the Socket.IO package.

Socket.IO brings to mind WebSockets. WebSockets are also a browser implementation allowing bi-directional communication however, Socket.IO does not use this as standard. First, Socket.IO creates a long-polling connection using xhr-polling. Then, once this is established, it upgrades to the best connection method available. In most cases, this will result in a WebSocket connection.

With sockets, when the server receives a new message, it sends it to the client and notifies them, bypassing the need to send requests between client and server. A simple chat application shows how this works.



The server constantly listens the incoming requests from the client-side and send the required responses. Under the hood, the following process takes place in our application -

1. So first, this is the screenshot taken from the client-side and this code will be used to send a message to the server.

```
// handle sending message to server & input reset
function sendMessage(e) {
  // prevent form submission refreshing page
  e.preventDefault();
  // send input value to server as type 'message'
  socket.emit("message", input.value);
  // reset input value
  input.value = "";
}
```

2. The server-side then matches the argument, which is “message” and then emits/broadcasts the message to all the connected sockets or clients.

```
io.on("connection", function(socket) {
  io.emit("user connected");
  socket.on("message", function(msg) {
    io.emit("message", msg);
  });
});
```

3. Now, on the client side again the client will receive a message from the server and will display it on the browser-side.

```
// watch for socket to emit a 'message'
socket.on("message", addMessageToHTML);
```

Why use web-sockets, when we can use techniques like short pooling, long-pooling etc?

In short-polling, a client constantly makes requests to the server on a specified interval and the resource is responding every time. This repeats on a very short cycle.

In long polling, a client initiates a request to the resource and if the server does not have any data, it keeps that connection open to respond later but if it does have data, it responds right away and closes the connection. It is generally a good idea to use long polling, it works well when there is very less traffic and also can scale up very quickly when traffic gets really high. Therefore, it is a good choice under many circumstances.


In WebSockets, a full-duplex connection is established between the client and the resource and as data changes on the backend that data gets pushed onto the client. These are useful, but only in certain circumstances, for instance we have an application where we require very little latency between the client and the resource or if we have something like a web chat application with many users in a chat room and we need to push a message from a resource server to multiple client, web sockets will be the best to use in this case.

Resources Required:-

1. HTML, CSS and Javascript (for front-end design).
2. Nodejs (Javascript runtime environment on client-side).
3. Express (back end web application framework for Node.js)
4. MySQL (for database of users and collaborated documents)
5. Socket-IO (Javascript Library for implementing sockets)

Some Screenshots of the Salient Features of the application:-

- **Sign in/sign up**



Welcome to CollabX !

Multiple Users can share their docs with each other and collaborate on an online environment. Chat facility also available

Copyright @ CollabX

Need an account? [Signup here](#)

Sign in


☐ Remember Me

Password?

OR


- **create new doc/find my doc/collobrate**

[</> Collabfy](#) [Home](#) Signed In As: ritik7garg@gmail.com [Logout](#)




Create New Doc

Enter details to create a new doc



Find My Docs

Find your existing Docs Heree



Collaborate

Search documents from our database to collaborate

- **see docs which you have created and in which you collaborated in separately in below table shown in find My Docs**

Created :	Collaborated In :				
	<table> <tr> <td>Theory Of Relativity</td><td>Fri Nov 20 2020 10:23:30 GMT+0530 (India Standard Time)</td></tr> <tr> <td>1</td><td></td></tr> </table>	Theory Of Relativity	Fri Nov 20 2020 10:23:30 GMT+0530 (India Standard Time)	1	
Theory Of Relativity	Fri Nov 20 2020 10:23:30 GMT+0530 (India Standard Time)				
1					




- **create your own document**

Create Your Own Document

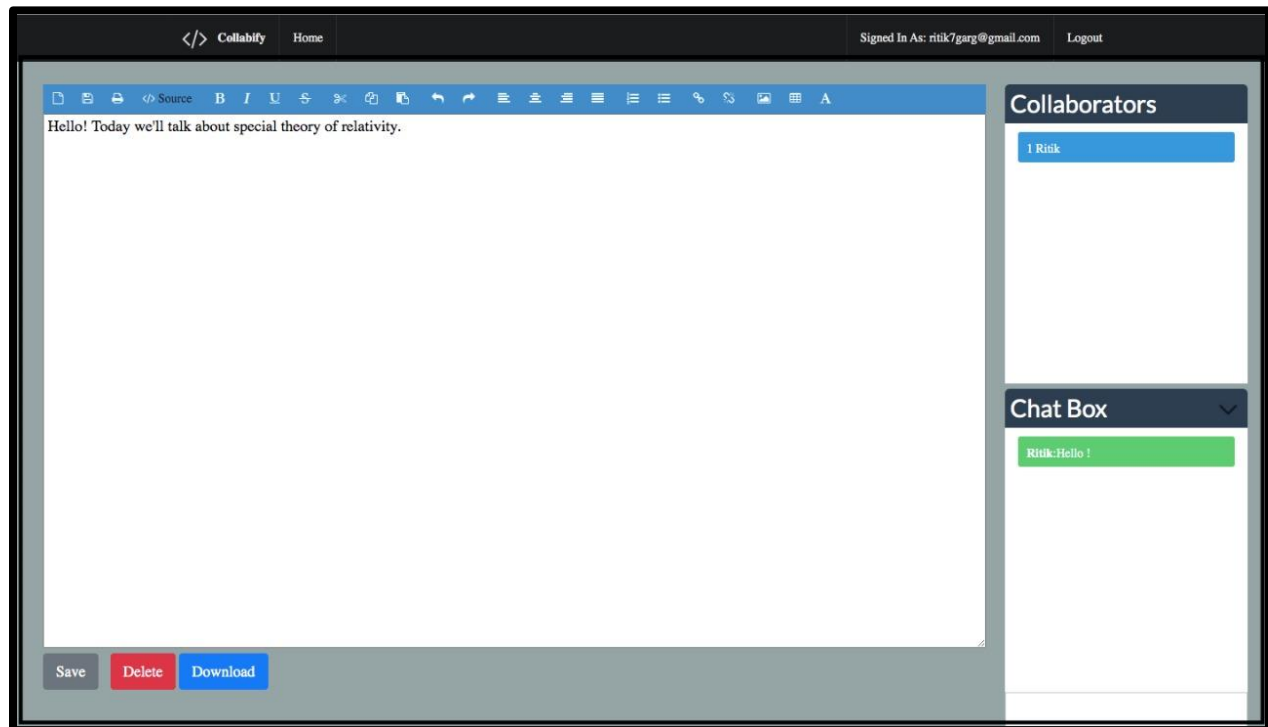
☐ Remember me
 [Forgot password?](#)

Already a member [Sign in](#)

or sign in with:

- **Collabrate/chat interface**



Result/Conclusion:-

We have successfully implemented a end-to-end networking application which in our case is a real time document collaborator web application with chat facility using socket programming.

Project's Github Link:-

The online hosted link of the project is provided in the project's github repository description. The project's github repository link is given below.

<https://github.com/RitikGarg7/CollabX>

FUTURE SCOPE/WORK:-

The project can be improved by adding the the following functionalities upon which we will work in future. The functionalities are:-

- **File upload:-** In future, we will work upon a feature to upload file on the app so that it can be shared and can be collaborated on by multiple users.
- **Video chat:-** we will try to implement a real time video chat feature so that users can communicate better and work on their collaborated documents.

REFERENCES:-

- <https://socket.io>
- <https://www.geeksforgeeks.org/socket-in-computer-network/>
- <https://www.geeksforgeeks.org/nodejs-tutorials>
- <https://stackoverflow.com/questions/tagged/socket.io>
- <https://www.freecodecamp.org/news/simple-chat-application-in-node-js-using-express-mongoose-and-socket-io-ee62d94f5804>
- <https://www.smartsheet.com/document-collaboration>
- <https://www.scnsoft.com/blog/online-document-collaboration>
- <https://medium.com/swlh/working-with-real-time-document-collaboration-tools-1b62e0f63311>
- M. Xue and C. Zhu, '**The socket programming and software design for communication based on client/server**', *Proc. 2009 Pacific-Asia Conf. Circuits, Commun. Syst. PACCS 2009*, pp. 775–777, 2009, doi: 10.1109/PACCS.2009.89