Lab-5 Report

In this report, I talk about implementing a Toy Chat (Real-Time) Web Application.
For the client side, I have used ReactJs to build my frontend server. This is the server which communicates with the browser to interact with the web user.
For the backend sir, I have used a NodeJs server using ExpressJs. ExpressJs helps to build the API's really fast by taking care of the overhead code, which we would have to write in NodeJs. Now I have used the JWT library for authentication purposes, which builds a token from the user information, which can be used to verify the authenticity of requests. I have used "Bearer <Token>" as the authorisation in the request.
Now the frontend server communicates with the backend server without refreshing the page and I have used socket.io library to send notifications regarding a new message being received from the server to the client.

In this lab, I learnt the following

- NodeJs
- ExpressJs
- Json Web Token
- Backend Servers and API's.
- npm and some of its libraries.
- HTTP Requests
- Javascript (await,async,promise)
- ReactJS
- Client Side Frontend.
- Client-Server Architecture

Features of the Chat-App :

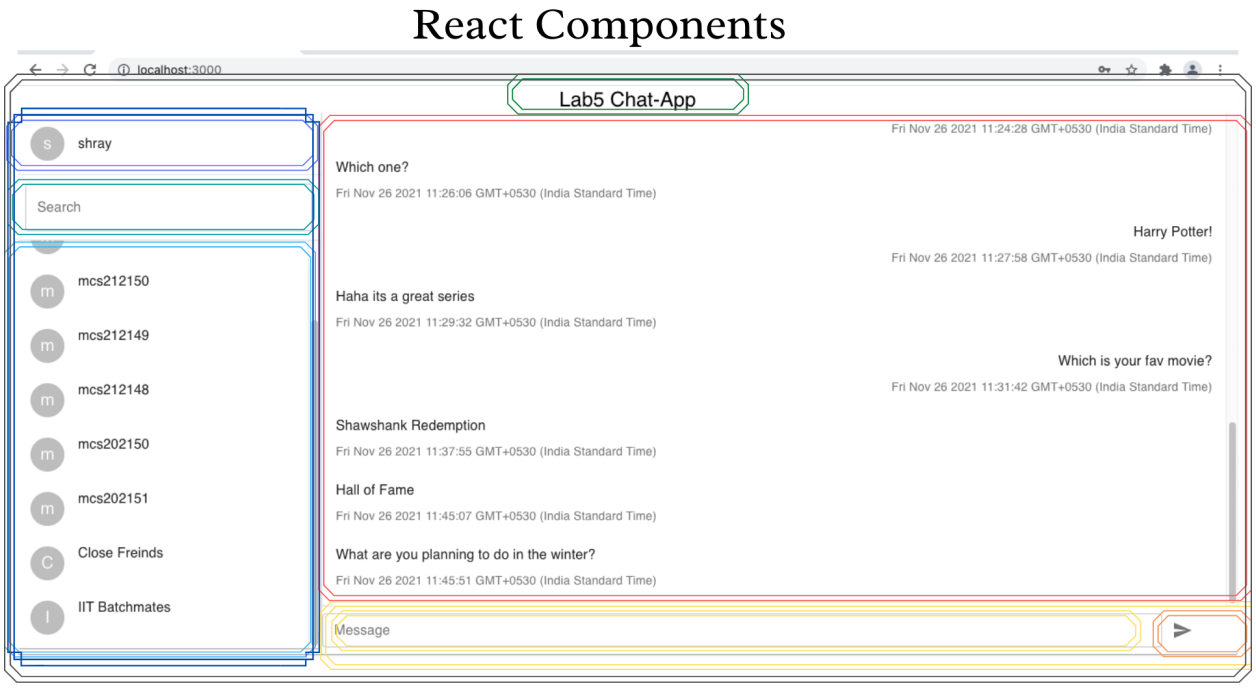
- ➢ Login Authentication
- ➢ Real time Private One-To-One Chat
- ➢ Real time Group Chat
- ➢ History of Chat is saved in database

I have discussed the below following subjects in my report :

a) React Components
b) Mongodb Collections
c) NodeJs Server
d) Performance Graph
e) Further Improvements
f) Important Observations
g) References

React Components

Now in the frontend, I have made 2 major components (Login Component and Chat Component). Server shows the Login component if the user is not authenticated and thus the user can never proceed to chat unless he/she login first.
Now in the chat component, I have divided into other simple components, making the structure easy to understand and modify.



Concept of useEffect hooks in React ::

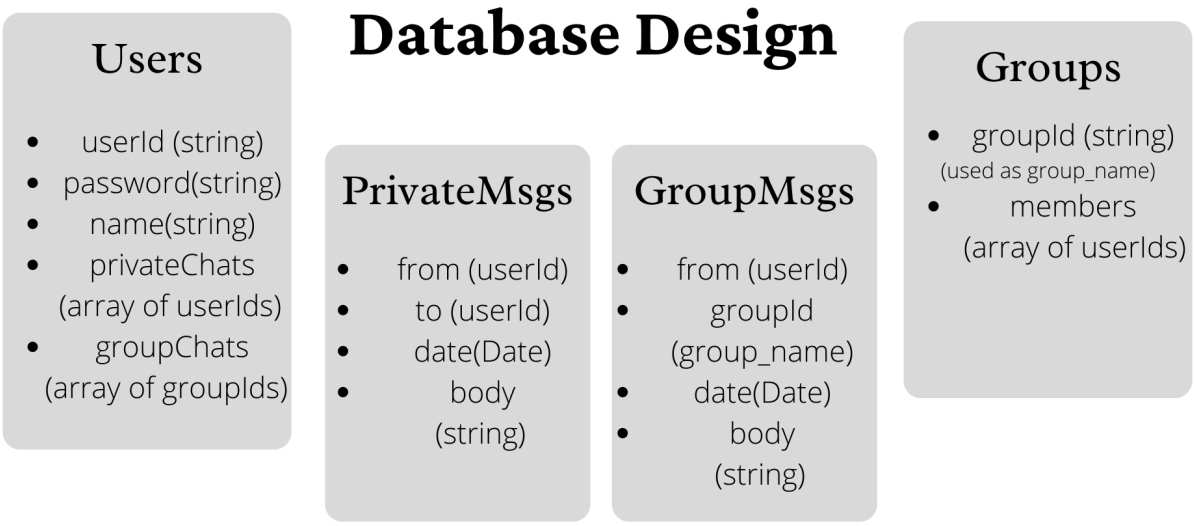
- Now, after loading the page, we load several needed components using “behind the scenes” requests to the backend server and load the data which is needed.
  Such as, we may need to populate the chat list for the user, we may need to populate the messages of the selected chat, from the backend server. We load all these data without refreshing the page and using the react library.

Concept of useState in React ::

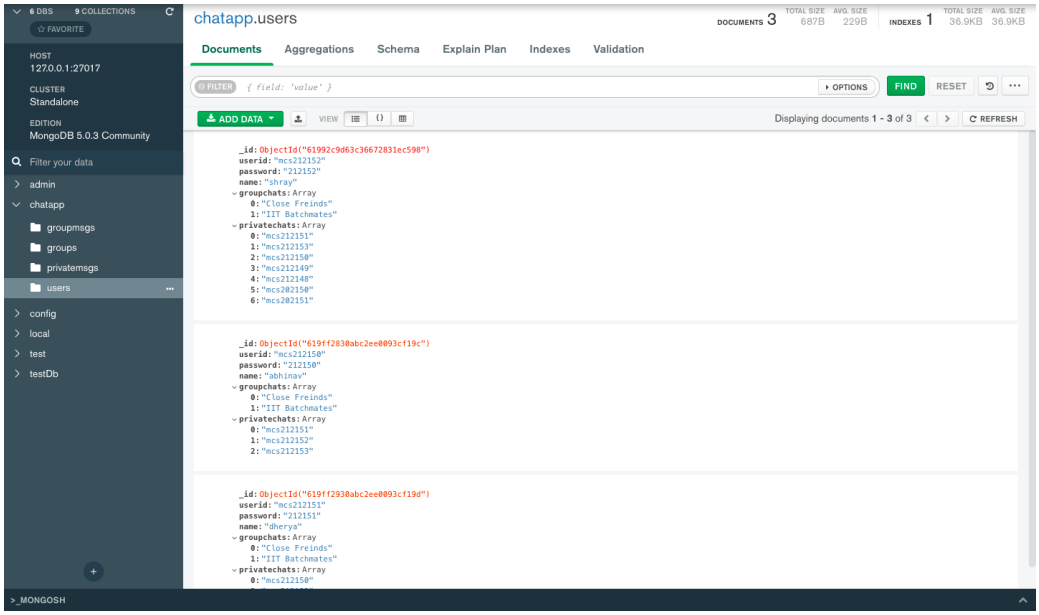
- Now, we may want to store particular data and then formulate logic based on this data to decide what needs to be shown to the web user and communicate with the web user.
  We can do this in the react “useState” hook in react.

```
function App() {
  const [token, setToken] = useState(false);
  const [chatUserList, setChatUserList] = useState([]);
  const [currentUser,setCurrentUser] = useState();
  const [messageList,setMessageList] = useState([]) ;
  const [selectedChat,setSelectedChat] = useState();
  const [searchResultList,setSearchResultList] = useState([]);
  const [groupList,setGroupList] = useState([]);
  if(!token) {
    return (<Login setToken={setToken} setCurrentUser={setCurrentUser} />);
  }
  return (<Chat chatUserList={chatUserList} setChatUserList={setChatUserList}
                messageList={messageList} setMessageList={setMessageList}
                currentUser={currentUser}
                selectedChat={selectedChat} setSelectedChat={setSelectedChat}
                searchResultList={searchResultList} setSearchResultList={setSearchResul
                groupList={groupList}  setGroupList={setGroupList}
          />);
}
export default App;
```

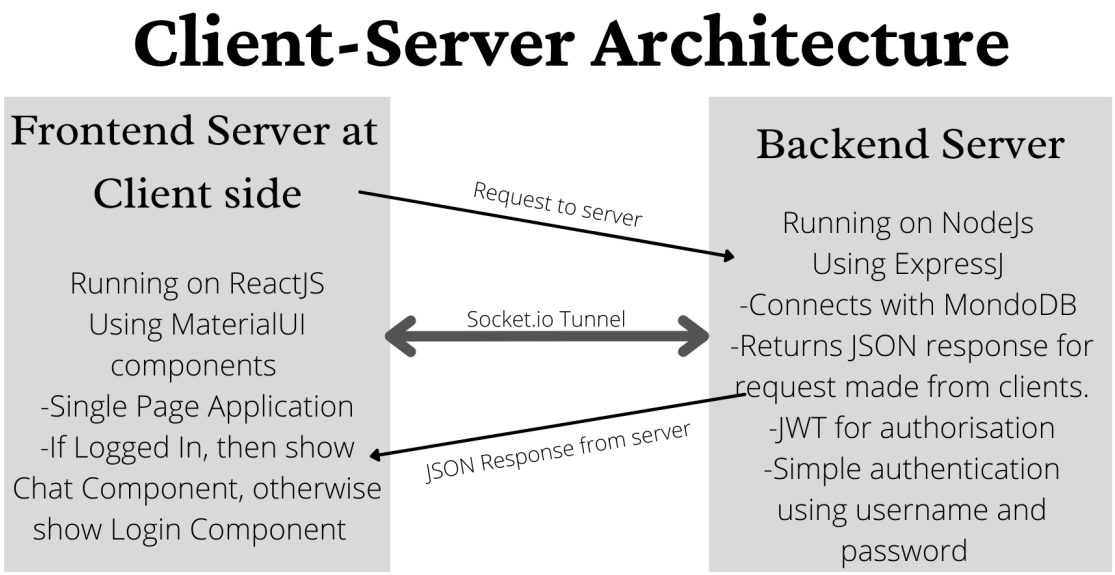
Mongodb Collections



In this lab, I have used mongodb to store the users information and all the messages. Given below is the basic schema of the “chatapp” database.
Mongodb is a no-sql database and doesn’t need a fixed schema for its tables (called as collections in mongodb). Choosing this helps us to change our collections easily without affecting our whole database. I have used “*moongoose*” library to connect to database.
We can start our mongodb server by starting its service. Further we can use Mongodb Compass application for a GUI interaction with our Mongodb.



Backend (Nodejs Server)



API's

I have tested my API's using the POSTMAN application before integrating the client side to the backend server.

- ***/login*** :: It expects a username and password and returns a JSON object which contains details whether there was a successful login or a failure. Post a successful login, (i.e. credential match is successful), the server returns a success response with a JWT which the client side uses as an authorisation for the successive requests to the server.

- ***/groupmsgs*** :: It expects a "group_name" and returns a JSON object which contains messages pertaining to the particular "group_name".

- ***/privatemsgs*** :: It expects a "from" and "to" user id's and returns a JSON object which contains the private chat messages pertaining to the particular user ids.

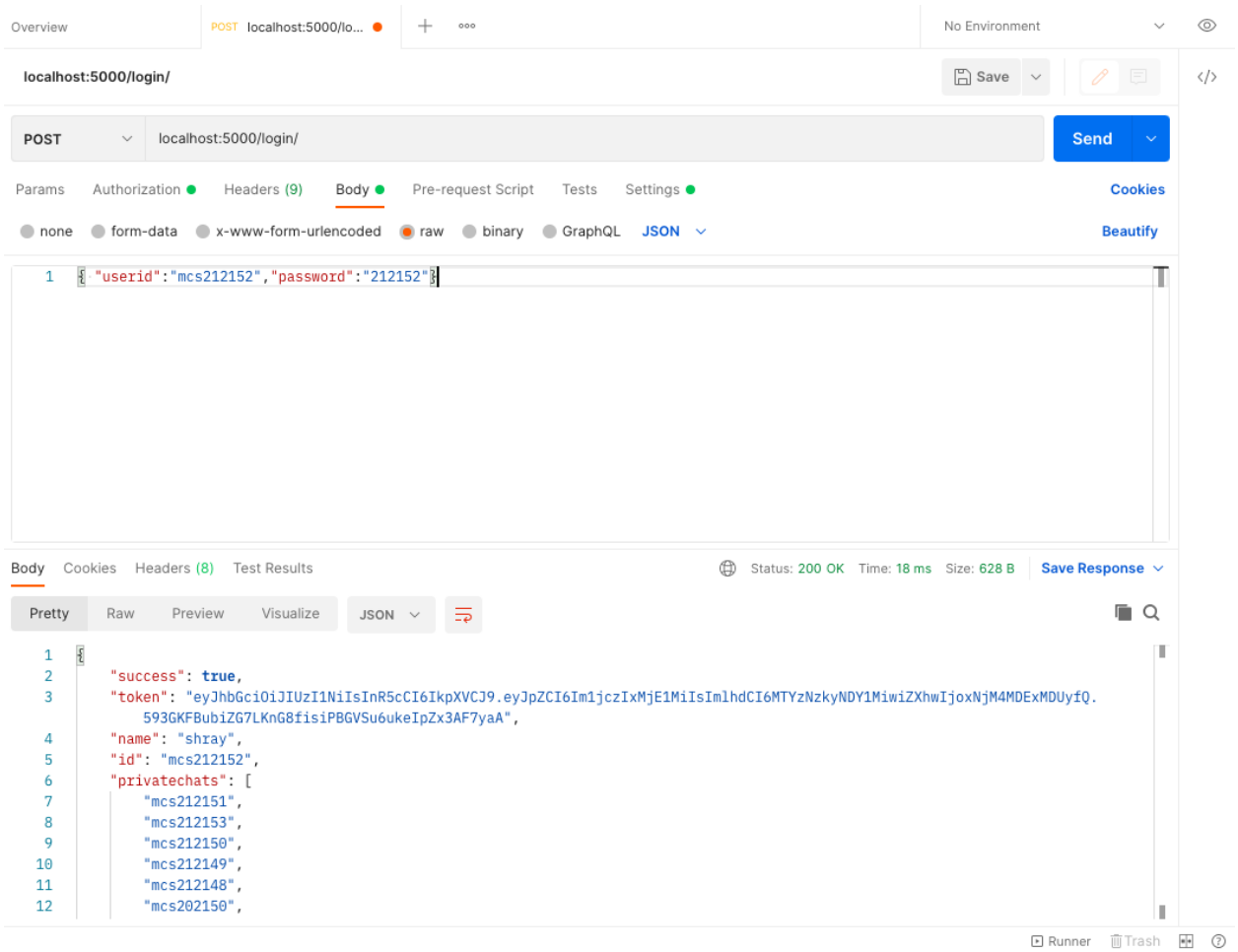
- ***/chatlist*** :: It expects a user id and returns a JSON object which contains the list of chats the particular user has with. (An array of user id's and group names).

- ***/allgroupnames*** :: It simply returns the list of all group names contained in the database.

- ***/newprivatemsg*** :: It is used to store a new private message into the database and the server also notifies the users of a new message in the client side.

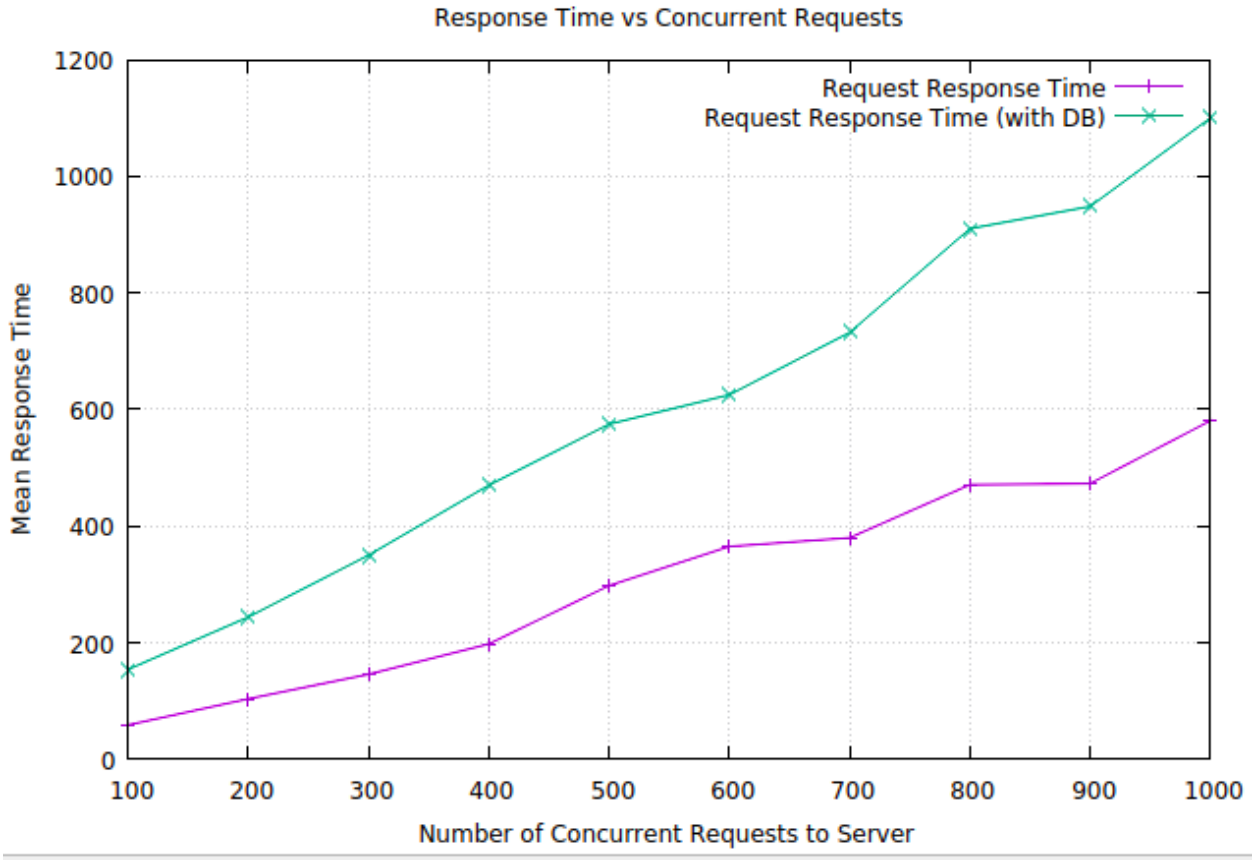
- ***/newgroupmsg*** :: It is used to store a new group message into the database and the server also notifies the users of a new message in the client side.

(Notification are sent using the socket.io library support)
For the error handling, I have used try/catch and then printed the errors in the console only. No errors are shown to the user.



Benchmarking Results

For the benchmarking results, I have used the tool “ab” in linux.
Following were the results ::



We can see that the request on which the server had to communicate with the mongodb database, the response time is higher.
Here the results shown, takes the total response time, which is the connection time plus the processing time of the server. A similar trend is seen in case of the connection times of the concurrent requests.

Data

Num Of Concurrent Requests	Mean Time Of Response (for a request which makes server do a DB access)	Mean Time Of Response
100	154	59
200	244	104
300	350	146
400	470	198
500	575	298
600	625	365
700	732	380
800	910	471
900	948	473
1000	1100	581

Further Improvements

- Currently a group chat is identified by it's "group_name" which is the name of the group. We can improve its identification by a unique GroupId, which can thus allow different sets of users to use the same group name for different groups.
- Session Logins. Currently sessions are not maintained and a user will have to re-login in case he wishes to open the chat in another tab. In case of a full refresh of a page, the user will have to login again.
- Currently notification from the server is broadcast in nature. We can improve on this by sending notification only to the required users.
- Addition of Files Support.
- Addition of an Admin Account.
- Addition of Emoji's Support.

Important Observations

- Building servers on NodeJs using ExpressJs is a really fast way to develop web applications. It helps us in maintaining the whole code in a single language. The support of modules/libraries from the open source community is very vast in nature.
- Nodemon is a very handy module of Nodejs which restarts the server in case of a change in our codebase.
- We should always add an OnSubmit event on the Form element, not on a single button.
- HTTP “GET” requests do not contain a request body. So “GET” requests must contain any required data in the header fields only. (Sometimes, the required data is added to the url path.)
- HTTP “POST” requests can contain a request body, and we can send files over the network using the body.
- React and Material UI help to focus on the content and the structure of the client side and the developer doesn’t have to spend much time with HTMLs and CSS anymore.

References :

1. Traversy Media Tutorials on Web Development
2. Digital Ocean Blogs
3. Stackoverflow Q&A
4. GeeksForGeeks Blogs
5. ExpressJS Docs
6. Mongoose Library
7. Medium Articles