

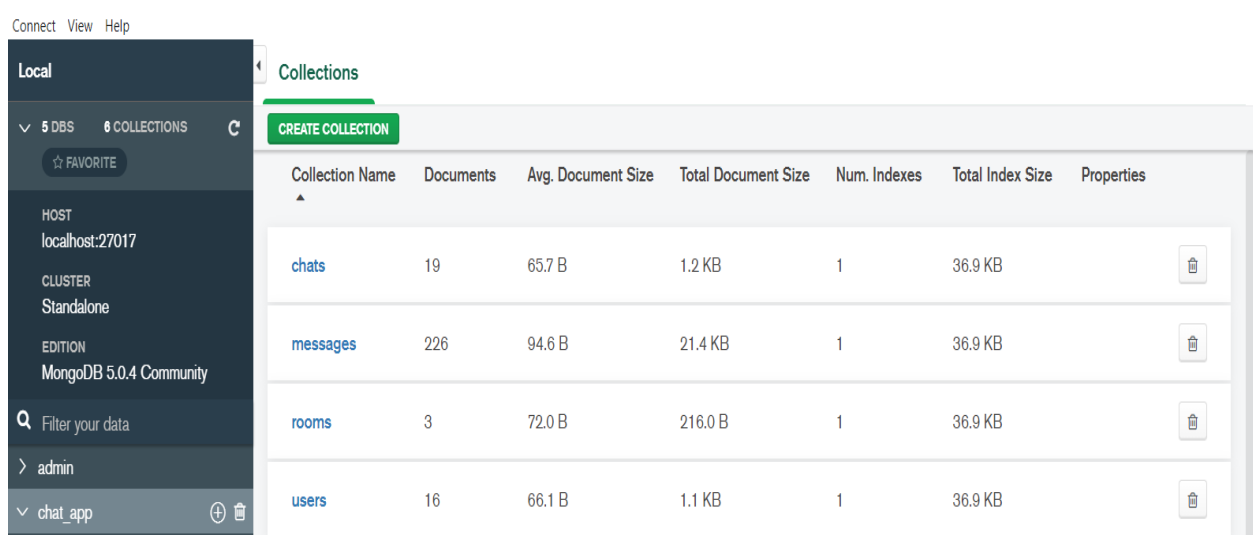
# A REPORT ON TOY CHAT APPLICATION

## Technologies used:

For the development of the application, I used MERN stack which basically stands as follows:

- 1) **Node Js** – backend server with express js as backend framework.
- 2) **Mongo Db** – No SQL database
- 3) **React Js** – Front end UI
- 4) **Socket.io** – Bidirectional communication

## 1) Mongo Db Collections



The screenshot shows the MongoDB Compass interface. On the left sidebar, the 'Local' connection is selected, showing 'localhost:27017' and 'MongoDB 5.0.4 Community'. The main panel displays a table of collections. The table has columns: Collection Name, Documents, Avg. Document Size, Total Document Size, Num. Indexes, Total Index Size, and Properties. There are four collections listed: 'chats' (19 documents, 65.7 B avg size, 1.2 KB total size, 1 index), 'messages' (226 documents, 94.6 B avg size, 21.4 KB total size, 1 index), 'rooms' (3 documents, 72.0 B avg size, 216.0 B total size, 1 index), and 'users' (16 documents, 66.1 B avg size, 1.1 KB total size, 1 index). Each collection has a trash icon in the Properties column.

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
chats	19	65.7 B	1.2 KB	1	36.9 KB	
messages	226	94.6 B	21.4 KB	1	36.9 KB	
rooms	3	72.0 B	216.0 B	1	36.9 KB	
users	16	66.1 B	1.1 KB	1	36.9 KB	

I have created four models for my chat app database which stores the following data:

**Chats** – Stores the chat history for every conversation that's happened.

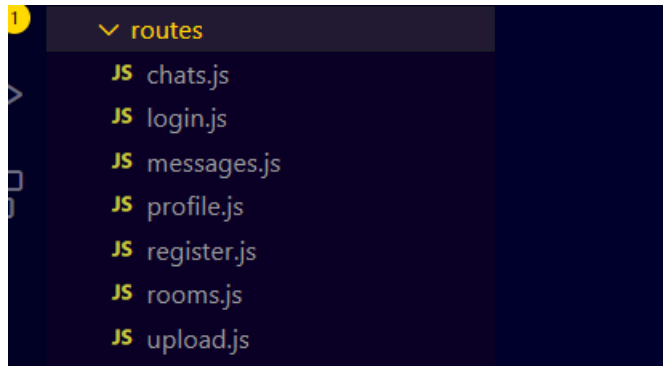
**Messages** – Stores all the messages between two users

**Rooms** – Stores the list of the chat rooms that are currently available.

**Users** – Stores the user credentials for authentication.

## 2) Rest Api's

I used express js to create the rest api's. They are as described below:



- 1) **Register Api**- To register the user. It just inputs the username and password to the users collection in the database. It also checks if the username has already been used. The post request posts the username and password.
- 2) **Login Api** – To login the registered user and authentication. It matches the input parameters with the one stored in the database. In case of success, it redirects it to the messenger page else it sends authentication failed error. The post request posts the username and password for authentication. There is no get request.
- 3) **Context Api** – To save the context of the user so that it can be used later in the program. It used action, context and reducer to save the context of the user. Whenever a user tries to log in, three different possible actions are there, first the user sends the info where the data and the error is null. Second either the login is success in which case the response should contain the data of the user, and third if the login is failure then the response should contain an error message . I used useContext library from react for this context purpose.
- 4) **Chat Api** – To load the conversation of all the saved users. It has two requests get and post. Whenever any chat is clicked, the post request is triggered which stores the name of the given conversation in the database. During the chat history fetch, the get request fetches all the relevant conversation matching to the given user that is stored in the database.
- 5) **Message Api** – To send and receive the messages. Whenever a message is created in chatroom or the private chat, this api is called and the given message is stored using the post request which contains <sup>information</sup> about the sender and the receiver as well. Thus, whenever a message history is required, the get request fetches all the message having desired sender and receiver from the database.

- 6) **Room Api** – This api is used to create the chatrooms for a particular user. The post request creates a chatroom which can be joined by anyone. The get requests fetches all list of chatrooms that are available to join currently.
- 7) **Upload Api** – This api is used to save the images in case there are some files in the message conversation. It uses react's multer library. Whenever a message contains a url, it looks for the particular file location stored by this upload api.

Problem Statements and requirements:

- 1) To verify the identity of the student using Kerberos credentials:

I created a simple register and login mechanism for this purpose which checks the user username and password and logs in into the system. Register and login api is used in this case.

**Toy Chat Application by Abesh**

**User Name**  
Username

**Password**  
Password

Login

[Not a user yet? Register Here](#)

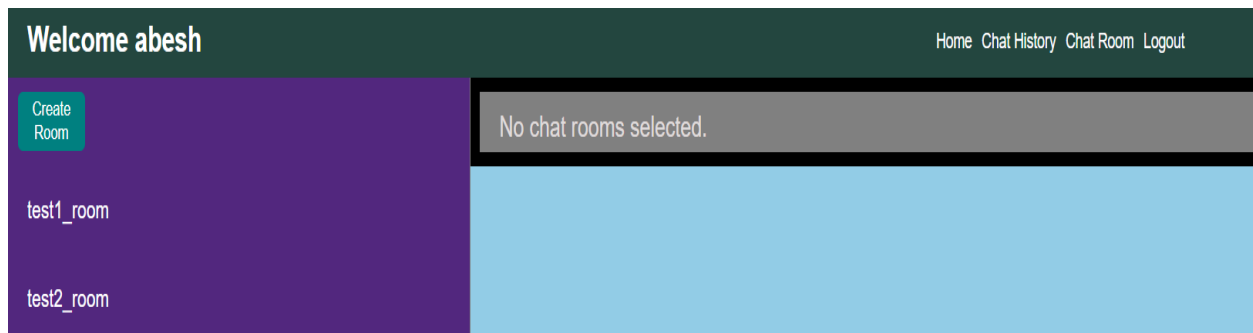
## 2. The students can chat among themselves

For this I created a private chat option using socket.io. Whenever a user login is successful, a connection is established to the socket server and thus this socket is displayed to all the other users as well. Whenever a connection is closed, that socket is removed from the list of online users and others can't chat with that particular socket. When a username is clicked, the socket associated with that username is called and the conversation messages are directed to that particular socket.



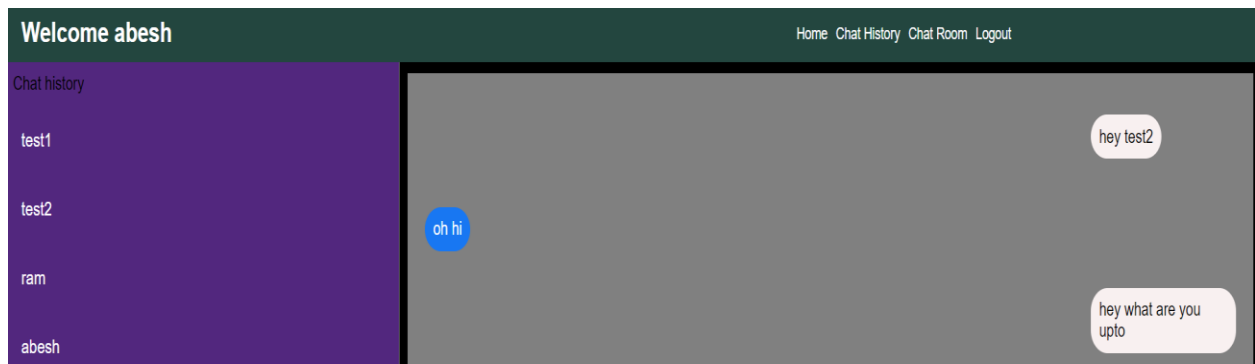
## 3. The students can create a chat room to have a group chat.

In this I designed in such a way, that any user who is logged in can either create their own room or join the room that has already been created. If a user joins a particular room, all the messages sent over that room are loaded as well. Two users in different rooms cannot see messages of another room. These rooms are not controlled by admin rather users can choose a room in which they want to be a part of.

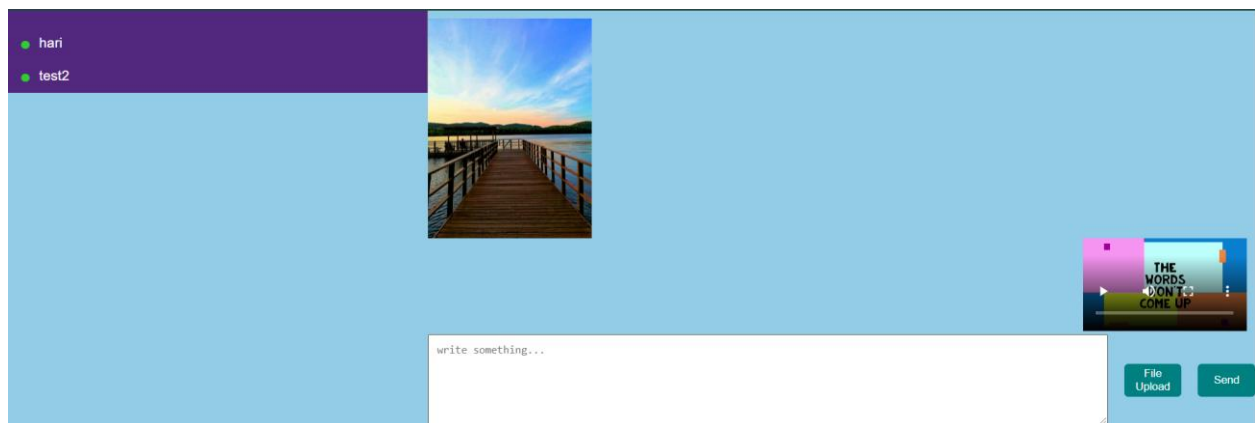


## 4. They must not lose their data after logout from the application or a system shutdown or application crash.

For this I created a chat history option which stores the list of all the conversation a particular user had. The history messages also get loaded during chatting. But if the user is offline, we would not be able to see the previous messages and hence the user can go into the chat history menu and load all the previous conversations.



5. The students may share any media file or meme, or reels among their friends.



For uploading the files, I used the file selector and on select I used the multer library of react which stores the given selected file to the backend server. However, the messages do not contain the image/video rather they are simple urls in normal text format. But when rendering the messages, if the frontend finds a url, it fetches that particular image/video file from the location.

**Graph to show how the time to process a request varies with the increase in number of requests to the application.**

To plot the graph I used the postman tool to send the number of requests to the application, I used a tool call artillery which is used for load testing of the rest api. I used one of my Api's get request and gradually increased the number of requests per second and noted down the average response time for each request. My script is as follows:

```
test.txt X
C: > Users > Abesh > Desktop > test.txt
1  config:
2    target: http://127.0.0.1:5000/api/rooms/
3    phases:
4      - duration: 5
5        | arrivalRate: 1
6
7      - duration: 5
8        | arrivalRate: 10
9
10     - duration: 5
11       | arrivalRate: 50
12
13     - duration: 5
14       | arrivalRate: 100
15
16     - duration: 5
17       | arrivalRate: 200
18
19
20
21     defaults:
22       | headers:
23 scenarios:
24   - flow:
25     | - get:
26       | | url: "/"
```

```

-----
Metrics for period to: 16:48:00(+0545) (width: 8.594s)
-----
vusers.created_by_name.0: ..... 48
vusers.created.total: ..... 48
vusers.completed: ..... 48
vusers.session_length:
  min: ..... 14.6
  max: ..... 202.3
  median: ..... 21.5
  p95: ..... 125.2
  p99: ..... 169
http.request_rate: ..... 10/sec
http.requests: ..... 48
http.codes.200: ..... 48
http.responses: ..... 48
http.response_time:
  min: ..... 0
  max: ..... 40
  median: ..... 12.1
  p95: ..... 29.1
  p99: ..... 40

```

An example of the statistics.

The final graph obtained is as follows:

