Assignment 5

COP-701

Toy Chat Application

Name : Sourav Sharma

Entry No. 2021MCS2154

Department of Computer Science & Engineering

Indian Institute of Technology Delhi

1 Directory Structure

```
BackEnd
├─ middleware
│  ├─ auth.js
│  └─ simpleauth.js
├─ models
│  ├─ chatmsg.js
│  ├─ chatuser.js
│  ├─ message.js
│  ├─ room.js
│  ├─ roommsg.js
│  ├─ roomuser.js
│  └─ user.js
├─ routes
│  ├─ auth.js
│  ├─ chat.js
│  ├─ message.js
│  ├─ room.js
│  └─ user.js
├─ socket
│  └─ socket.js
├─ userfiles
├─ index.js
└─ package.json
```

```
FrontEnd
├─ src
│  └─ components
│     ├─ common
│     │  ├─ form.jsx
│     │  ├─ input.jsx
│     │  ├─ protectedRoute.jsx
│     │  ├─ searchBox.jsx
│     │  └─ select.jsx
│     ├─ chat.jsx
│     ├─ chatForm.jsx
│     ├─ loginForm.jsx
│     ├─ logout.jsx
│     ├─ navBar.jsx
│     ├─ registerForm.jsx
│     └─ roomForm.jsx
├─ services
│  ├─ authServices.js
│  ├─ chatServices.js
│  ├─ httpServices.js
│  └─ userServices.js
├─ App.css
├─ App.js
├─ index.css
├─ index.js
├─ package.json
└─ README.md
```

2 Back-end for Toy Chat application

2.1 Introduction to NodeJS

Node js or node, is an open source and cross platform runtime environment for executing js code outside of a browser.We use node to build back-end services, also called API's or **Application Programming Interfaces**. These are the services that power our client application. Node is used to build highly scalable applications. Node is not a programming language neither it is a framework.

Node is ideal for Highly-scalable, data-intensive and real-time apps. This is because of Non-blocking or Asynchronous nature of node. A single thread is used to handle multiple requests. If a request is made, we use that thread to handle that request. And if we need to query a database.Our thread doesn't wait for the db to return the data.While the db is executing the query that thread is used to serve another client.When db prepares the result it put that in what we called an Event Queue .Node is continuously monitoring this queue in background, when it found any event in this queue. It takes it out and processes it.

And this makes node ideal for building for I/0 intensive apps.which includes more disk and network access. We can serve more clients without the need of throwing extra hardware.And that's why node applications are highly scalable.

2.2 NPM: Node Package Manager

It's a kind of registry for all the Third Party libraries that we can add to our applications, which literally help us to write more clean code by abstracting most things.

Before starting a new project the first thing that we need to do is to make a kind of metadata about the project, which stores what kind of libraries our application is using. The file is known as package.json. To make this we use the command **npm init –yes**.

2.3 Express Framework

Express is framework which help us to create web applications which can respond to the HTTP requests. Here we define different routers and create a http server which listen to these request and send a response accordingly. To install express write **npm i express**.

Express come with different things. One of those is express.json(). It is used to parse the body of the request into json objects which is a key value pair. To create a server we use the command **http.createServer(app)**, where app is an instance of

2.3.1 Methods provided by express for routing

1. **router.post('api_end_point', middle-ware(s), callback function)** : This is used to send a post receive any post request to the given api end point. Post mean pushing fresh new data.

2. **router.put('api_end_point', middle-ware(s), callback function)** : This is used to send a put request. Put mean updating already existing data.

3. **router.get('api_end_point', middle-ware(s), callback function)** : Get is used when we requesting for some data from the end point.

4. **router.delete('api_end_point', middle-ware(s), callback function)** : To send any delete request to the api end point.

2.4 Mongoose Library

To install mongoose use the command **npm i mongoose**. Mongoose is an object modeling library for Mongo DB. It provides features like schema definition and schema validation. It basically communicate through the mongo driver to the mongo db. Methods that are used from the mongoose library.

2.4.1 Queries methods on mongoose

1. **Model.find({filter})** : It is used to filter through the model by providing query. It returns a promise and thus we need to await that promise.

2. **Model.findById({_id: id})** : This methods finds documents by matching the id passed in query parameter. And this also returns a promise which either resolves or get rejects.

3. **Model.findOne({Query})** : This method finds the first document which matches with the query pattern.

4. **Model.findOneAndUpdate({Query}, {Update})**: This method will find as well as update the document with update object passed where we pass what we need to update.

2.5 Socket.io

Socket.io is a library that is used to have a real-time and bidirectional and event based communication between the server and the browser. We first require this in our project main file by using **require('socket.io')**. We pass the first argument to this as an instance of the http server. We set it to the global object. Since we need this thing in our socket.js file.

In this whenever user make a first time connection to the server after a successful login it emit a default 'connection' which is then listen by **io.on('connection')**. In return it give us a socket object. Whenever user send any message from the front end it listen to that. First it verify that user. In case if that is valid. Then it check the format type of the message if it is text then it creates a new Message object from the message model by setting appropriate property.

Along with that it also update the last message time in that private chat or room chat. In case of any invalid format it emits 'Invalid-Format'. Else it save the message. And then it emits the message either in the room or the chat to whatever it belongs.

Another event on which socket listen is the join-room or join-chat. It basically join the user that socket if it was not joined. Or if user is the first one to create that socket then it first create and then make user to connect to that socket.

2.6 Joi Library

This is a library which comes very handy when we need to validate things. Specifically when we need to validate input fields. To main methods that are used in this project of Joi are. To install it use **npm i joi**.

1. **Joi.object{schema}** : Here we define the whole schema as an object.
2. **Schema.validate(model)** : Here we validate the model over the Schema that we have defined.

2.7 Multer Library

Multer is used as a middle ware in node js. This is mainly used for the file uploading. To have the multer library use the command **npm i multer**. To start using multer in the project write require('multer') in the project. Then provide the destination folder name in a object like {dest: "path" }. Then use it as the middle ware like upload.single("filename"). To access the file use the req.file. It uses the upload function to upload the file.

3 MiddleWare

3.1 Auth Middleware

This is a middle ware which is used in the pipeline for all the API services which need to authorize the login user before provide it any response from any service. It check whether token is send in the request header. In case if it is not then it response with the message that no token is provided. In case if there is a token then it need to validate that token. For that **JWT** provides a method call **verify** which takes the jwt private key along with the token and return the payload. This payload was used at the time of generating this token. And since we using the unique id from the mongo db for each entry it store. We fetch this thing from the verify function and then make a database call to the user documents to check whether there is any user with same id or not. In case if there is one then we set that user in request body and send it to the next one in the pipeline. For this we need to use next().

4 API Calls

4.1 API Calls For Authentication

1. **POST Request::/api/auth/register** : This API call is requested with the data in the request body from the front-end part. And it basically first check whether use is valid user with unique user id and email id. To do so it call **findOne** method to the database and check if user exists with same email or not. In case if does then it returns response in a json object with a error message field and also with status of the current response. One thing in this is if new user which is register is valid then it first use **bcrypt** method hash to encode the password using a salt value and then only store it to the database so password are not openly stored on the database. In the return of the response we set the header field of the response object with the **'x-auth-token'** set to the token return by the jwt token generator to which we have passed some payload. In case if there is some **Internal server error** we send a response back with the **status 500**. A successful response have a status 200.

2. **POST Request::/api/auth/login** : This API call is used to send a response to a login request by the user. It is passed with the email and password. We first search whether their is any user register with the current email or not. If it does then we use **bcrypt.compare()** function to compare the hash value of the password. In case it is a valid password for the login then we generate a authentication token also called as json web token, which from now onward will be send along with any request body that is generated by the login user from the front end so that we can validate whether user is allowed to call that api end point or not.

4.2 API Calls For Rooms-Chat

1. **POST request:: /api/room/createRoom** : This API call is used when we have to create a new Room. From the request body it takes the room name only. One important thing here is we only want that a valid login user can only create room. So thats why we have a **middleware function** auth. It basically check for the token that is send with request body And if it is a valid user then it also return the user id in the req body. Which is passed to the next function in the pipeline.

2. **POST request:: /api/room/addNewMembers** : This API call is used to add new member or members to the room. Although from the front end we are only sending a request to add a single member. But the API is written considering any modification in the future. Also in this case we need the middle ware auth which verify the user token and then allow it to make this call. Again it take care of things like if the user is already a member of the room or not. If it is then we don't add it to the room.

3. **GET request:: /api/room/getRoom** : This API call returns the rooms which the user requesting service from this API belongs. It populate through the database and returns the roomname and the last message time. This information is used by the front to sort the rooms according to the messages order they received. Like the rooms with latest message are shown first than the others. Again it also need the middle ware auth which verify the user token and then allow it to make this call.

4. **GET request:: /api/room/leaveRoom** : This API is kept but is not used in the front end. Although is there for any future modification to the project. This allow user to leave the room. First it validate whether user actually belongs to the room or not. And one more thing it check is that if the user who is leaving the room is the **admin** of the room then whole room is removed from the database.

5. **GET request:: /api/room/getRoomMsg/:roomid**: This is an API get call where we request for the messages to the current room. This is generated as soon as someone click on the room chat. Messages are fetched and send to the front end.

6. **POST request: /api/room/remove** : This is a API call that can be used by the admin only to remove the member from the room. It first validate user which is requested to remove from the room is a valid room member or not. In case if it then it remove it's entry from the database for it's belonging to the this room.

4.3 API Calls For Private-Chat

1. **POST request:: /api/chat/createChat** : This API call is used to create a new private chat between two users. This basically takes a user id in the request body with whom private chat need to be start. It first checks whether there is any private chat already going on between users. In case if there is none then only it creates a new chat window between those two users. So thats why we have a **middleware function** auth. It basically check for the token that is send with request body And if it is a valid user then it also return the user id in the req body. Which is passed to the next function in the pipeline.

2. **GET request:: /api/chat/getUser/:val** : This a get method call to fetch the user. This is used when we want to create a new chat. This call returns the user whose name is send in the request body.

3. **POST request:: /api/chat/delChat** : This is post method which is used to delete the chat between two users. In case if any user is deleting chat we will delete the chat for the both users. In the current project this feature is not provided in the front end and thus kept left for the future updates to this project. Again it also need the middle ware auth which verify the user token and then allow it to make this call.

4. **GET request:: /api/chat/getChat** : This API is used to fetch all the private chat that are done by this users(only chat not messages). And other by the users to this current user who is requesting this service from the backend.

5. **GET request:: /api/chat/getChatMsg/:chatid**: This is used when we want to fetch all the messages of the current user with a particular user.

4.4 API Calls For Users

1. **GET request:: /api/user/getAllUsers** : This API call is used to fetch all the users present in the toy-chat application and in the response to this service it returns all the user other than user which is requesting this service. We also need to verify whether the user requesting for this service is a valid user in our database or not so that's why we have a **middleware function** auth. It basically check for the token that is send with request body And if it is a valid user then it also return the user id in the req body. Which is passed to the next function in the pipeline.

4.5 API Calls For uploading media files

1. **POST request:: /api/msg/uploadMediaFiles** : This API call used the methods of the **multer** package to upload the files. This basically initialize the storage and returns the url of the uploaded file in the response body. To make sure that if two files with same name are asked to send then it prefix those with the data and thus ensures the uniqueness of the files. We also need to verify whether the user requesting for this service is a valid user in our database or not so that's why we have a **middle ware function** auth. It basically check for the token that is send with request body And if it is a valid user then it also return the user id in the req body. Which is passed to the next function in the pipeline.

5 Model Design for the Document Database

5.1 Model for the User

Model design for the user contain following fields. Out of this few have constraints. And few are in required also. To do so I used the **Joi** library to validate this which are entered because **Mongo DB** it self doesn't provide any feature to do so. And thus we need some validation mechanism to validate entries. Some of are like email must of valid email type. Similarly a phone number must be 10 digit. And a user name should be at-least some constant value long. Fields in the user database are:

1. **First Name** : It is required and min-length should be 2 and max 25 and should be a string..
2. **Last Name** : It is also required by current design choice and it should be a string
3. **User Id**: A string which is required and must be unique.
4. **Email Id**: A valid email id which is also required. This is also used during login new user.
5. **Password**: A valid string with atleast 5 character.
6. **Phone**: A valid phone number of exactly 10 character.

Along with this model we also having generateAuthToken function because this is the model which control it and thus kept along with this data.

Then we call the mongoose.model passing the above schema defined.

5.2 Model for the Room

This schema for the model contains only two fields

1. **Room Name**: This is a unique name which mean an user can't create two rooms of same name. We are also having Joi validation on this field. This should be atleast of length 1 and maximum it can goes 255.
2. **Last Message Time**: This basically store the last message time associated with the current room. With every room we fetch this thing and in the front end this help us to sort the rooms or private chat according to the time of the last message they have received.

5.3 Model for Room-Users

This schema for the model contains information about room id, user id and the whether the user id is admin of the room id or not.

1. **Room Id**: Stores the unique room id which refer to the id of the room model.
2. **User Id**: Store the unique user id which refer to the id of the user model.
3. **is Admin** : This stores the information about the admin property which later will help us to decide the what a user in the given room can do.

5.4 Model for the Messages

This is used to store the information about each message. Either it could be media file or it could be simple text message

1. **Format Type** : This is a flag which refer whether this message contains media or text.
2. **Text**: This store the message and it is of type string.
3. **Media**: This store information about the multi-media message.

5.5 Model for the Private Chat-Users

This is a model which is used to store the private chat information.

1. **User1 Id**: It store information about either of the one user communicating in the private chat window. It refer user model to have the valid user entry.
2. **User2 Id**: It stores information about the other user. It also refer to the user model
3. **Last Message Time**: It stores the last message time of the message that is happen between these two users. It is used in the front end to sort the chat according to the latest time they have received message.

5.6 Model for Private Chat-Message

This is a model which contains detail about each message of the conversation between the two chat. Like what is the sender id and what is the message id.

1. **Chat id**: It basically refer to the Private Chat-User model for the id of the chat.
2. **Message Id**: This refers to the message model.And it can be populate in case if we need the details of the message.
3. **Sender Id**: This refers to the user model and it is also a required field.

5.7 Model for Room-Message

This is a model which contains detail about each message of the conversation in the Room. Like what is the sender id and what is the message id.

1. **Room id**: It basically refer to the Room model for the id of the room. It is a required field.
2. **Message Id**: This refers to the message model.And it can be populate in case if we need the details of the message.
3. **Sender Id**: This refers to the user model and it is also a required field.

6 Front End for the Toy chat application

6.1 Introduction to React JS

React is a JS library for building fast interactive user interfaces. It was developed at Facebook in 2011 and is currently the most popular JS library to build interfaces. Every react applications has components (essentially a piece of user interface). So when building applications with React we build a bunch of independent , isolated and reusable components and compose them to build complex user interfaces.

Every application has at least one component which is referred to as the root component.This component represents the entire application and contains other child components.So every React application consists of tree of components. In terms of implementation: A component is typically implemented as JS class that has some state and render method. Render return react element(js object) which map to DOM

The State here is the data that we want to display when the component is render and the render will describe what and how the UI will look like.The output of this render method is react element - which is a simple plain JS object that maps to a DOM element. It's not a real DOM element. It's just a plain JS object that represents the DOM element in memory. So React keeps a light weight representation of DOM in memory which we will refer to as Virtual DOM . Unlike real DOM this virtual DOM is cheap to create. To begin with the front end project write the command **create-react-app app-name**.

7 Library used in Frontend part

7.1 React Router and React Router DOM

React Router is package for the routers. The react router api are Router, Route. The router keep the UI in sync with the url. Routes are used to render UI components according to the URL. Browser Router is used because that provide things like history api. To install this use the command **npm i react-router-dom**

Switch will make the path matching exclusive. We may also need to define exact path property. And by default if user is not login we make it to move to the login route.

7.2 Axios in React JS

Axios is a data fetching library in react js. To have it in the project write the command **npm i axios**. Axios is a http client library that allow to make request to a given end point. This can be any external api or the service or your own application. Axios provide us with all kind of http requests. It is much good to user axios that other things like fetch because by default we are working with json objects. In Fetch api we need to set our headers and also need to convert the request body to a JSON string. Axios function matches exactly with the http services name. Axios also throw errors

7.3 Bootstrap

To have bootstrap in your project write the command **npm i react-bootstrap**. It is open-source CSS framework that is used to develop responsive front end development. It contains template realted to the CSS.

8 Components used in the Frontend part

8.1 Login Form

The state of this component have data part and error. Data store an object which contains two things email and password. And we also have the render method. This Login form component extends to a common component in the project called form component. We rendered two input filed name email and password. And we have used a submit button. On Submit we have a handler function which first thing we do is we send the data to the api end point called login user. And it then if it is a valid user it returns a token which we set to the localStorage by set item on the field token with token value.

In case if it is a invalid email or password then we show the error using toastify container. In case if it was a valid login then we change the router of the login user to the chat window.

8.2 Registration Form

The state of this component have data part and error. Data store an object which contains first name, last name, email id, user id, password and phone number. We also having a joi schema which basically validate the entry in the form fields. We have six input fields and one submit button.

Submit button on click goes to the hanldeSubmit function. Which first call the api end point with a http post request to register the user. If user registration has some invalid data then we show that using toastify containers. Else if it a valid user we take the token and set it to the local storage.

8.3 Room Form

The state of this component have data part which contain room name and other is error object. We are also having schema to validate entries. We create a form with a single input field which takes the unique room name from a particular user. And lastly we have a button to render which on submit make a post request to the api in the backend to service the createRoom for the current user. In case if it is valid request we route back the user to the chat window.

8.4 Navigation Bar

First things it contains name of the application i.e toy chat which is also clickable and take us to the chat route. Other elements on nav bar are conditionally rendered. Like if user is already login then his First name and second name are shown on the top right corner and another is a logout link. In other case when the user is not login. Then we show tow nav link login and register to the user.

8.5 Chat Form

The state of this component contains data object which contains user id and other thing is an array of the users. For the form we have a form submit button which on click have a registered handler. In the component did mount phase of this component. We fetch the user and all the chats in which user is active. And then we take out those users from all users which have active chat with current user. And store the result in the state. In the handle submit we send a post request to api end point and then we return user to the chat window.

8.6 Logout Form

In its component did mount phase we remove the token key from the local storage and then route the user back to the home page of application.

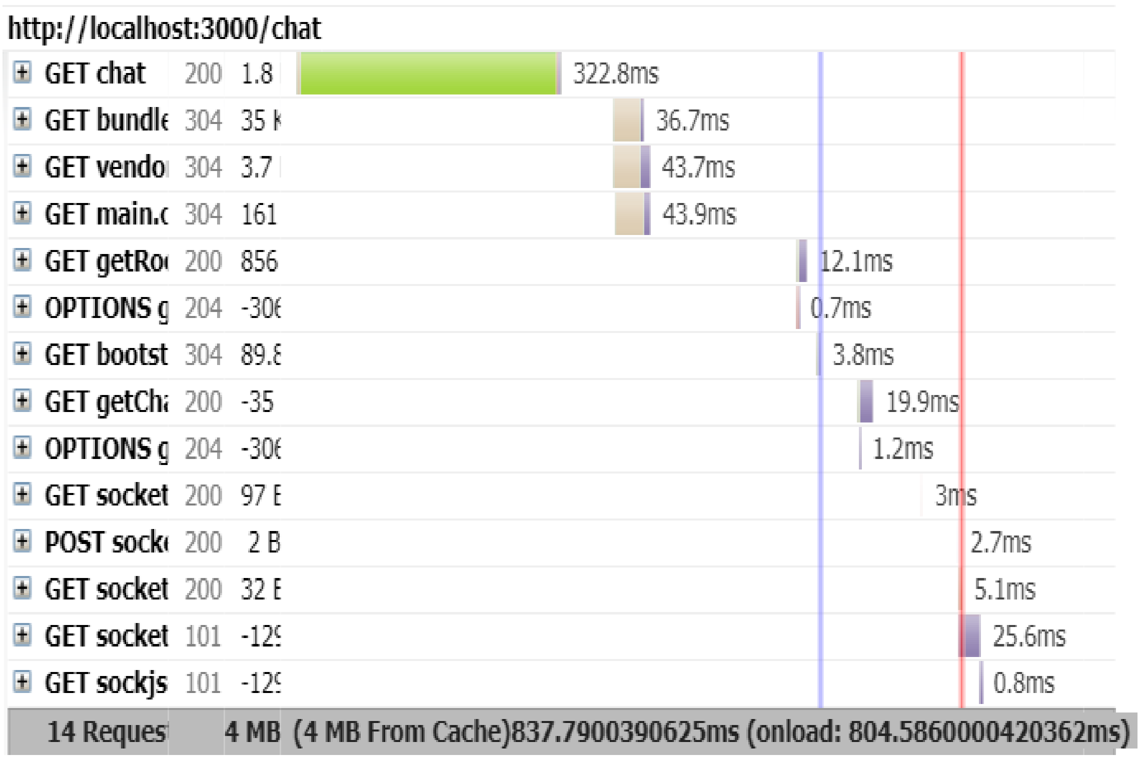
8.7 Chat Window and User panel

The state of this component contains lot of things like chat array, cur selected chat, chat name, room members, chat messages, There are lot of handler also for handling different events.

9 Project Requirement

1. Node version 12.13.1
2. Git version control
3. React version 17.0.2

10 Performance Graph



http://localhost:3000/chat

GET chat	200	1.8	322.8ms
GET bundl	304	35 k	36.7ms
GET vendo	304	3.7	43.7ms
GET main.c	304	161	43.9ms
GET getRo	200	856	12.1ms
OPTIONS g	204	-30	0.7ms
GET bootst	304	89.	3.8ms
GET getCh	200	-35	19.9ms
OPTIONS g	204	-30	1.2ms
GET socket	200	97 B	3ms
POST sock	200	2 B	2.7ms
GET socket	200	32 B	5.1ms
GET socket	101	-12	25.6ms
GET sockjs	101	-12	0.8ms
14 Reques		4 MB	(4 MB From Cache)837.7900390625ms (onload: 804.5860000420362ms)

11 References

1. https://www.npmjs.com
2. https://icons.getbootstrap.com
3. https://getbootstrap.com
4. https://nodejs.org/en/
5. https://socket.io
6. https://reactjs.org

7. https://www.freecodecamp.org/news/how-to-use-axios-with-react/

8. https://create-react-app.dev