CeeScoresLive

Project Engineering

Year 4

Connor Ngouana

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2023/2024

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

_____Connor Ngouana____

Acknowledgements

In this section I wish to acknowledge all my lectures in ATU for what they have thought me over the 4 years, in which allowed me to create this project, I would also like to thank the project organisers Paul Lennon, Niall O’Keeffe, Michelle Lynch, and my supervisor Brian O’Shea.

Table of Contents

1 Summary

CeeScoresLive is a website made for football lovers like me that are also too busy with other commitments to watch the matches live. This project was made to help people see score from live matches, statistics from your favorite teams and a live chat to talk to your friends about the match or you teams.

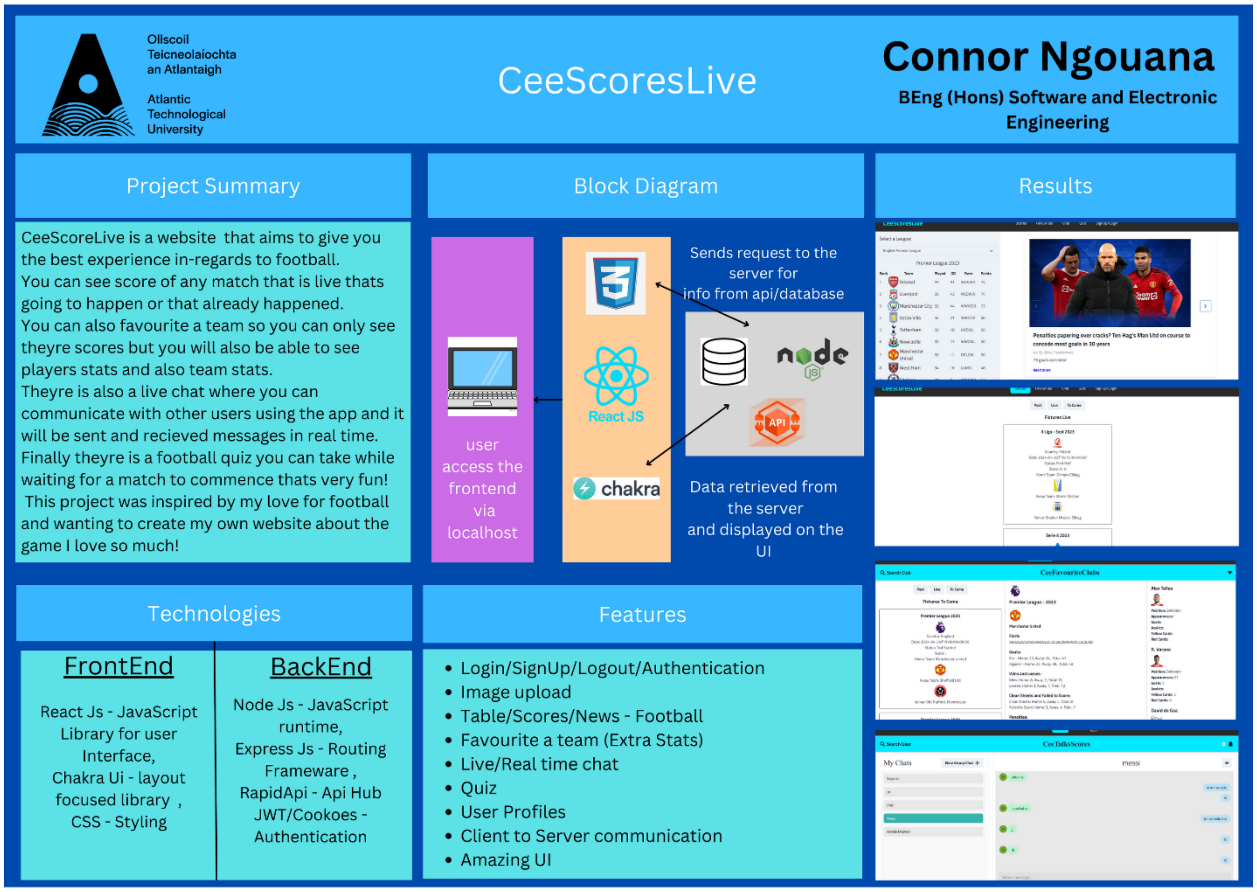
The goal of CeeScoresLive is to provide a website for people to get real time scores for ongoing, upcoming and past fixtures you can also get detailed statistics for teams and players. A key feature of my website is the ability to favorite team which will filter out and only show the team you favorited and see the team's statistics and player statistics. You will also get a live Chat feature with real time communication among users. For entertainment during halftime or while waiting for the match to start there is also a quiz feature about football.

The scope of this project uses both front-end and back-end development making sure the responsive user interface and server communication. The website uses React Js for user interface, chakra UI for layout design and CSS for styling in the front-end. The server uses Node Js and Express Js for server operations, JWT/Cookies for secure user authentication. MongoDb compass to store website details such as user information. Finally, Rapid Api to generate the football data into my website.

The approach was to build a responsive, user-friendly website that supports real time interactions and displays data. Using backend to handle API requests and server-side data from the API/Database.

All and all CeeScoresLive is a fun website that tracks scores with other features creating an amazing website for football fans to use when looking for a place to look for scores in real time with other amazing features.

2 Poster



Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

CeeScoresLive

Connor Ngouana

BEng (Hons) Software and Electronic Engineering

Project Summary

CeeScoreLive is a website that aims to give you the best experience in-regards to football.
You can see score of any match that is live thats going to happen or that already happened.
You can also favourite a team so you can only see theyre scores but you will also be able to see players stats and also team stats.
Theyre is also a live chat where you can communicate with other users using the app and it will be sent and recieved messages in real time.
Finally theyre is a football quiz you can take while waiting for a match to commence thats very fun!
This project was inspired by my love for football and wanting to create my own website about the game I love so much!

Block Diagram

user access the frontend via localhost
React JS
chakra
Sends request to the server for info from api/database
API
Data retrieved from the server and displayed on the UI

Results

Technologies

FrontEnd	BackEnd
React Js - JavaScript Library for user Interface, Chakra Ui - layout focused library , CSS - Styling	Node Js - JavaScript runtime, Express Js - Routing Frameware , RapidApi - Api Hub JWT/Cookoes - Authentication

Features

- Login/SignUp/Logout/Authentication
- Image upload
- Table/Scores/News - Football
- Favourite a team (Extra Stats)
- Live/Real time chat
- Quiz
- User Profiles
- Client to Server communication
- Amazing UI

3 Introduction

CeeScoresLive is a web application made to help improve user experience when looking for real time matchday scores and pervious scores and shows upcoming matches. This was inspired by a website called LiveScore. I wanted to develop it myself because on Live Score there was too much going on and I wanted to expand on what they had adding more features. People can come on my website and check out scores and league table for top 5 leagues and see recent news headlines happening in the football world. You will need to then log in to be able to use the additional features that I have created that other websites like mine done have such as the favouriting of a team, real time chat and a quiz. This website aims to make a fun interactive football experience for football fans of any age group all around the world.

Features?

Register - To register you just need to enter your name, password, email address and if you choose you can enter a profile picture from your camera role.

Login – Enter the email address and the password you used to register.

Main page – It will show you the league table of the top 5 leagues in the world and will also show you recent news which you can read about if the headline piques your curiosity.

Favourite Page – It allows you search you any team in the world and favourite that team showing you that teams match (live, future, past), the teams statistics and player statistics.

Chat Page – You can chat to any other user that has created the app by just searching for their name and you can create group chats so you guys can talk about the matches going on.

Quiz Page – you can play a quiz with 15 questions you have 5 hints and 2 50/50 that shows your scores at the end.

All these features together are what I came up with to make hopefully one of the best football live score websites in the world, making an easy to use and visually pleasing website that shows all the features listed above and gives the user the best experience you can have compared to any other website or app like mine.

4 Technologies and Tools

4.1 React Js

React. js is an open-source JavaScript library, crafted with precision by Facebook, that aims to simplify the intricate process of building interactive user interfaces [1]. It enables efficient updates and rendering of web components, making it ideal for real-time applications, I used React Js for frontend development of my website.

4.2 Chakra UI

Chakra UI is a simple, modular and accessible component library that gives you the building blocks you need to build your React applications [2]. I used chakra ui to help with the styling of my overall project to make it more professional using its layouts.

4.3 CSS

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes. I used this for styling of different components like the NavBar.

4.4 Node Js

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts [3]. I used to build the backend/server-side of the website.

4.5 Express Js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [4]. It is used to build a single page, multipage, and hybrid web application. It's a layer built on the top of the Node js that helps manage servers and routes [5]. I used Express Js for the routing and middleware in my backend/server-side code.

4.6 MongoDB compass

MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment [6]. This is a document-based database which I used to store information used in my website such as the user information and your favourite teams you choose.

4.7 Rapid API

RapidAPI is a platform that allows developers to easily access and integrate various APIs into their applications. It provides a unified interface for developers to discover, test, and manage APIs from different providers [7]. This is where I found the API that allows me to get the real time football data.

4.8 JWT/Cookies

The server generates both an access token (JWT) and a refresh token. The access token has a relatively short expiration time while the refresh token has a longer expiration time. Then, the server sends the JWT and Refresh token to the client. The refresh token is usually stored in a secure cookie [8]. I used this for authentication JWT Json Web Token and cookies to help secure my users information making the website more secure.

4.9 Socket.io

Socket.IO is a library that enables real-time, bidirectional and event-based communication between the browser and the server. It consists of:

- a Node.js server: Source | API
- a Javascript client library for the browser (which can be also run from Node.js) [9]

I used to allow my chat to be real time so you can get instant messages without needing to refresh your page for you to render other users messages.

These technologies together are what made the project possible, making an easy to use and visually pleasing website that shows all the features listed above and gives the user the best experience you can have compared to any other website or app like mine.

5 Project Architecture

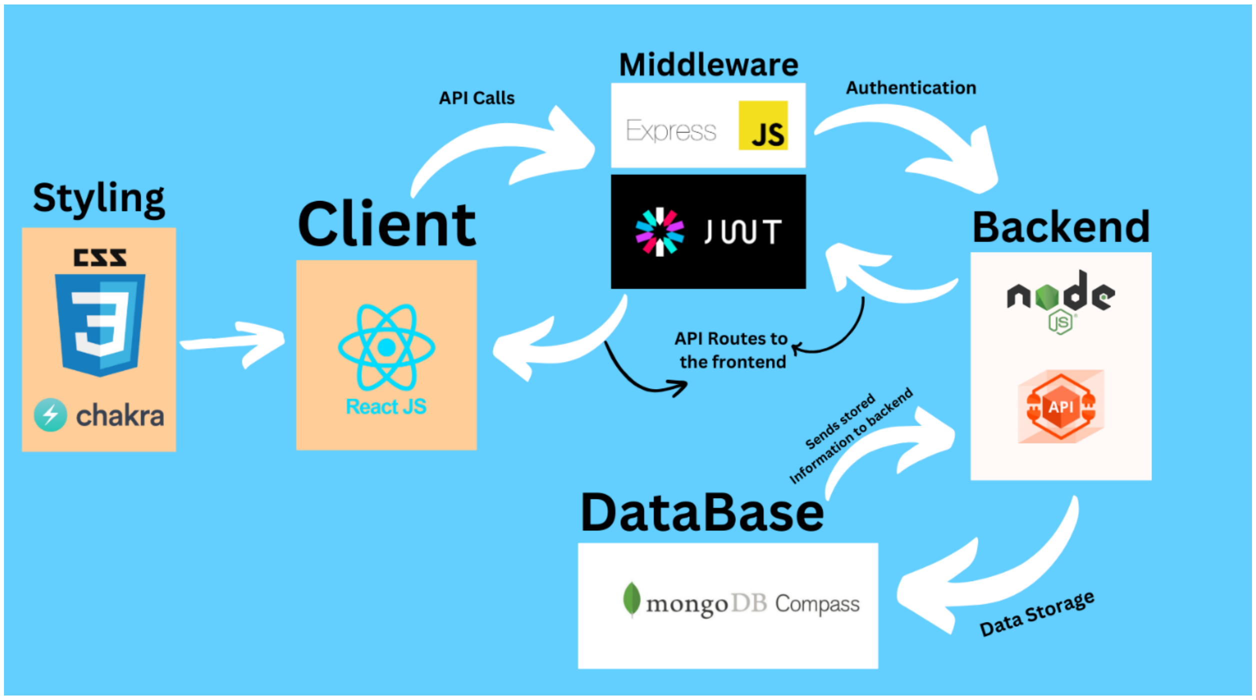
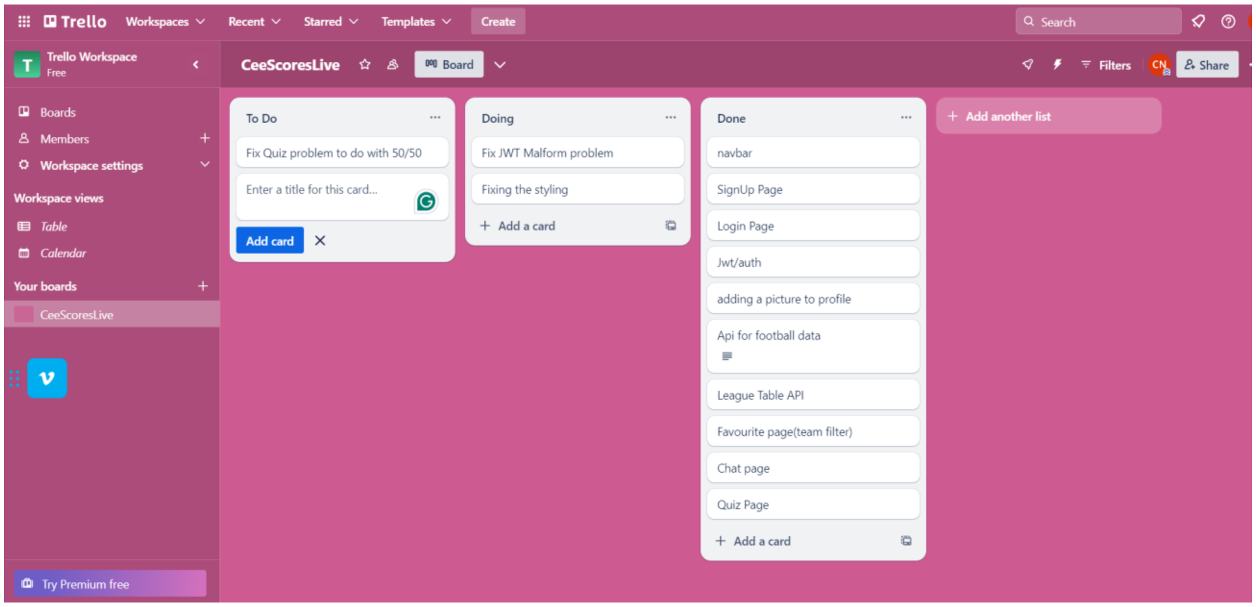


Figure 5-1 Architecture Diagram

6 Project Plan

I used Trello as my project management tool. Trello is the visual tool that empowers your team to manage any type of project, workflow, or task tracking. Add files, checklists, or even automation: Customize it all for how your team works best. Just sign up, create a board, and you're off! [10] and Microsoft one note for project planning, I used kanban as me of planning my work and getting my work done as I planned. The Kanban process focuses on breaking a project down into workflow stages and managing the flow and volume of tasks through those stages [11]. So to this I had 3 headings to do, doing, and done so at the start I had written a log of things I needed to do like research or actual coding then when I was ready to do it I would move it to the doing card and whenever I got it done I would move it to the done card so it was an easy to track what I needed to do, do and what was done.

Here is an example of how it looked using Trello and Kanban.



To Do
- Fix Quiz problem to do with 50/50
- Enter a title for this card...
- Add card

Doing
- Fix JWT Malform problem
- Fixing the styling
- Add a card

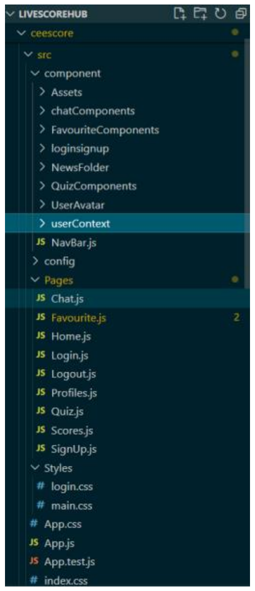
Done
- navbar
- SignUp Page
- Login Page
- Jwt/auth
- adding a picture to profile
- Api for football data
- League Table API
- Favourite page(team filter)
- Chat page
- Quiz Page
- Add a card

7 Project Code

In this section ill be explaining both my frontend and backend code and their functionalities and how the code works to make my project. I will be splitting this into 2 sections first is the Frontend code:

7.1 Frontend Code

In the frontend I created several key components and pages to my code to make it work all together. I have Pages folder which contains my pages that users will see, and I have a lot of components with the functionality of what happens on the screens here is a screenshot of my frontend looks like:

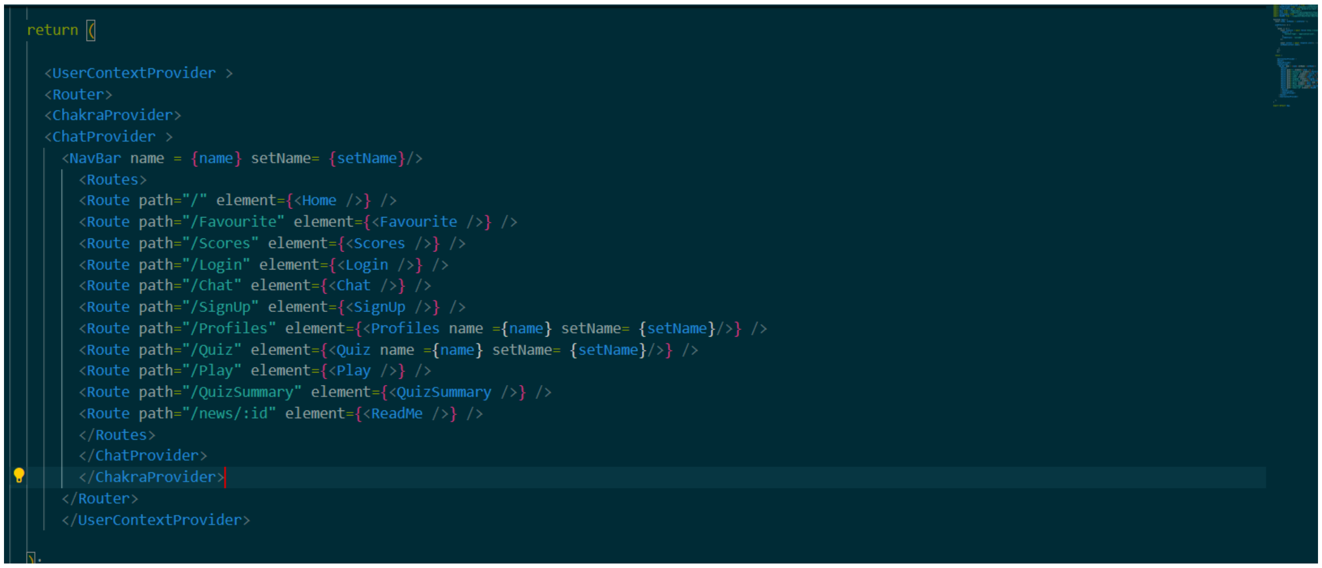


```
LIVESCOREHUB
  ceescore
    src
      component
        Assets
        chatComponents
        FavouriteComponents
        loginsignup
        NewsFolder
        QuizComponents
        UserAvatar
        userContext
        NavBar.js
      config
      Pages
        Chat.js
        Favourite.js                2
        Home.js
        Login.js
        Logout.js
        Profiles.js
        Quiz.js
        Scores.js
        SignUp.js
      Styles
        login.css
        main.css
      App.css
      App.js
      App.test.js
      index.css
```

Here you can see I have multiple pages and components for each and have reusable components like the useContext component which I will explain later.

7.2 App Js

I will start with App.js code to see how I liked all the pages together so the user can simply move from page to page then I will go through each page and the components linked to that page.



```
return (

  <UserContextProvider >
  <Router>
  <ChakraProvider>
  <ChatProvider >
    <NavBar name = {name} setName= {setName}/>
      <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/Favourite" element={<Favourite />} />
      <Route path="/Scores" element={<Scores />} />
      <Route path="/Login" element={<Login />} />
      <Route path="/Chat" element={<Chat />} />
      <Route path="/SignUp" element={<SignUp />} />
      <Route path="/Profiles" element={<Profiles name ={name} setName= {setName}/>} />
      <Route path="/Quiz" element={<Quiz name ={name} setName= {setName}/>} />
      <Route path="/Play" element={<Play />} />
      <Route path="/QuizSummary" element={<QuizSummary />} />
      <Route path="/news/:id" element={<ReadMe />} />
      </Routes>
      </ChatProvider>
      </ChakraProvider>
    </Router>
    </UserContextProvider>

)
```

In this code you can see from the top down I have wrapped the code in a userContextProvider and a chatProvider which ill explain properly later but the overall function is to pass variables to other parts of the project like the JWT token.

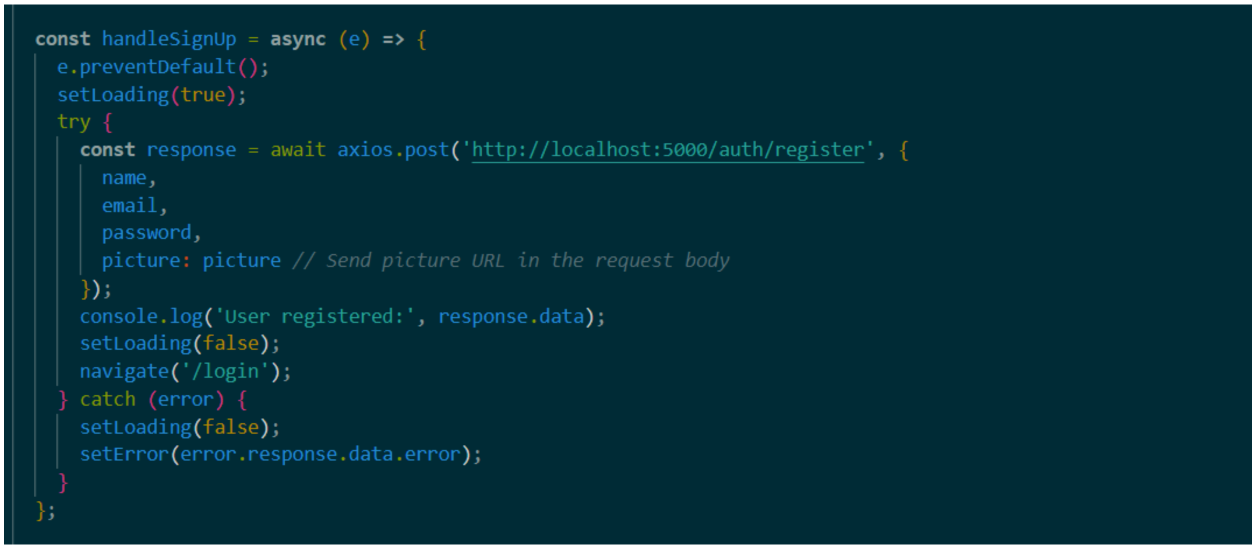
Then I have Router, Routes and route these were all imported from the react-router-dom to handle routing to the different pages that I have created.

Finally, I have ChakraProvider this is from Chakra UI to give all the different pages the styling page layout for my website.

I have also added navbar to the top and with no routes because I was navbar to be on the top of every page.

7.3 SignUp Page

Next, I will be moving onto the SignUp page and explain how the code works:



```
const handleSignUp = async (e) => {
  e.preventDefault();
  setLoading(true);
  try {
    const response = await axios.post('http://localhost:5000/auth/register', {
      name,
      email,
      password,
      picture: picture // Send picture URL in the request body
    });
    console.log('User registered:', response.data);
    setLoading(false);
    navigate('/login');
  } catch (error) {
    setLoading(false);
    setError(error.response.data.error);
  }
};
```

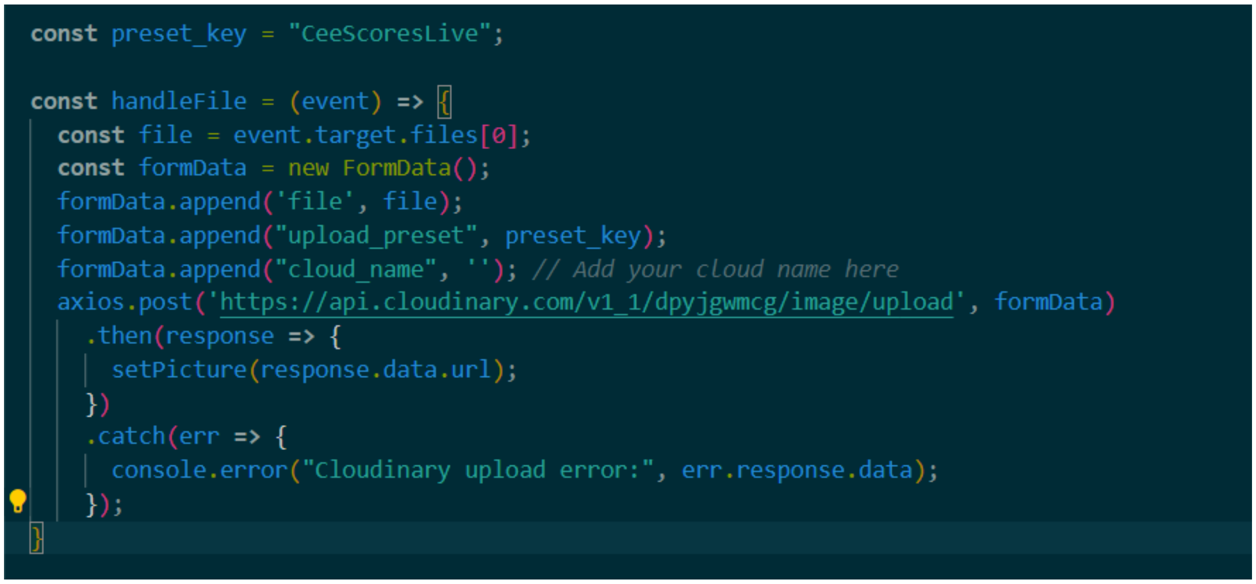
This the function that allows the user to successfully create an account and pass the information to the backend and ultimately stores it in the database so it can be reused through the application.

So we have a e.preventDefault which prevent the default form submission which stops the page from reloading.

Next, I'm using setloading this is to show a loading symbol to indicate that signup is process just taking a while to run.

Im using axios.post to make post request to the api endpoint /auth/register which is in the backend of my code which will pass the name, email, password and picture of the users choosing.

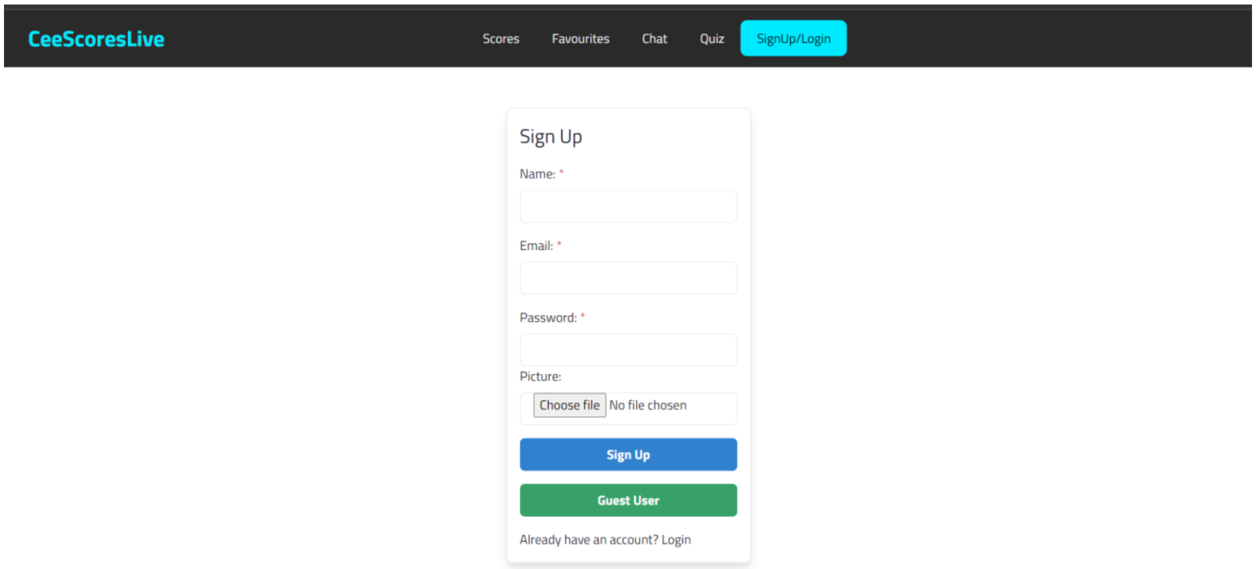
If it is successful, it will show User registered and always show all the user's information that was sent to the backend and will stop the loading with was set earlier and will navigate the user to the login and if it is unsuccessful, it will show the user an error.



```
const preset_key = "CeeScoresLive";

const handleFile = (event) => {
  const file = event.target.files[0];
  const formData = new FormData();
  formData.append('file', file);
  formData.append("upload_preset", preset_key);
  formData.append("cloud_name", ''); // Add your cloud name here
  axios.post('https://api.cloudinary.com/v1_1/dpyjgwmcg/image/upload', formData)
    .then(response => {
      setPicture(response.data.url);
    })
    .catch(err => {
      console.error("Cloudinary upload error:", err.response.data);
    });
}
```

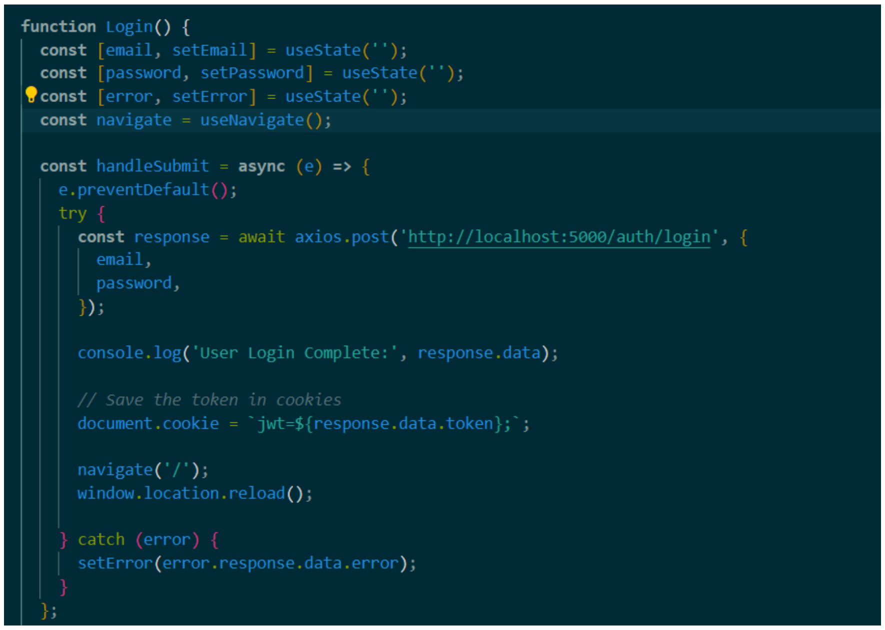
This the code of how I allowed the user to pick an image from their gallery and use it as their profile picture. It is storing the picture you chose to a third-party cloud service which is cloudinary to be used and viewed by anyone using the application.

Results



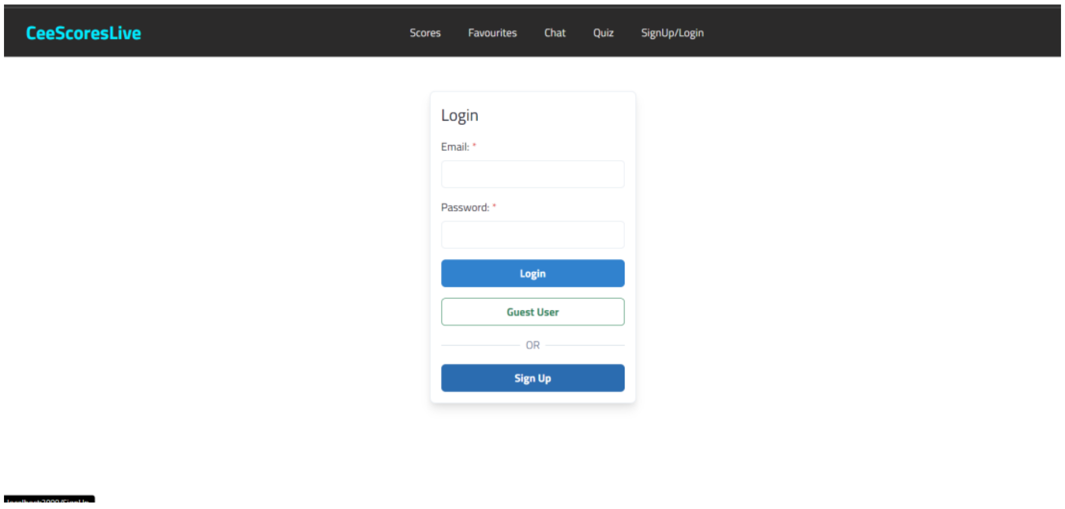
7.4 Login Page

Next is the login code:

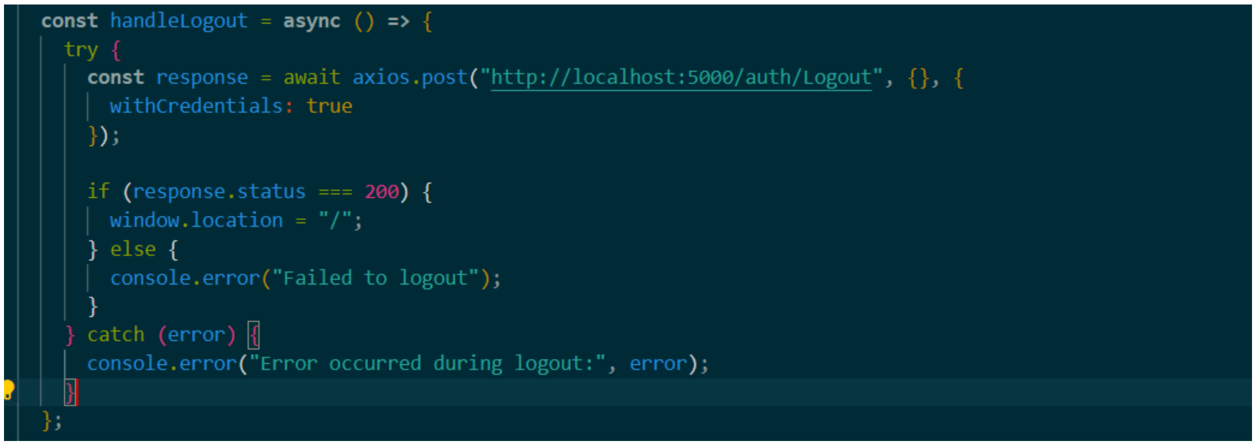


```
function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://localhost:5000/auth/login', {
        email,
        password,
      });

      console.log('User Login Complete:', response.data);

      // Save the token in cookies
      document.cookie = `jwt=${response.data.token};`;

      navigate('/');
      window.location.reload();

    } catch (error) {
      setError(error.response.data.error);
    }
  };
```

This code is the same as the SignUp code it uses the e.preventDefault and then uses axios.post to post the users email and password to see if they match with any other users information stored in the backend if it passes and there is a user then it shows User Login Complete and the login information. It then sends the token which was generated in the backend and stores it in the cookies and navigates the user to the home page.



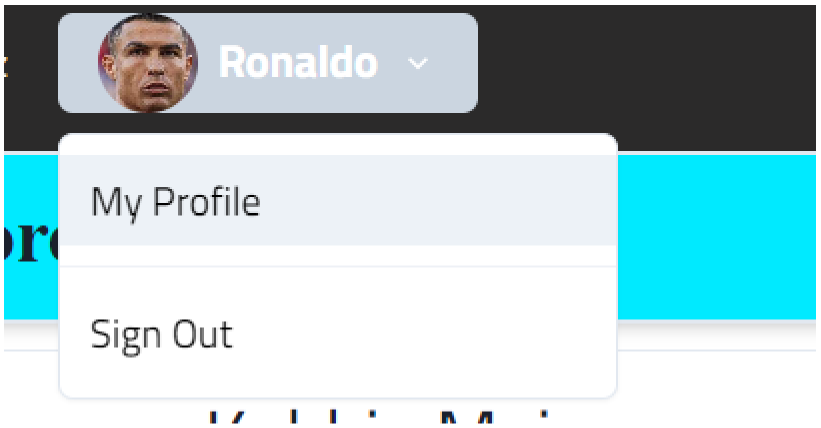
7.5 NavBar/LogOut

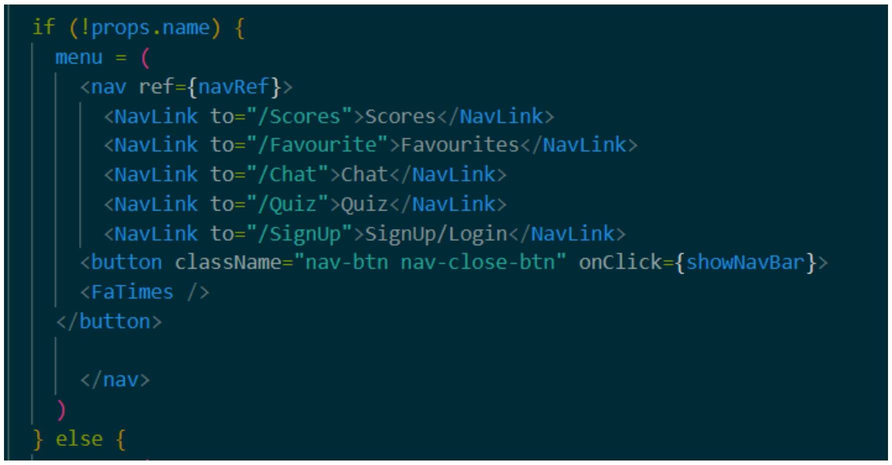


```
const handleLogout = async () => {
  try {
    const response = await axios.post("http://localhost:5000/auth/Logout", {}, {
      withCredentials: true
    });

    if (response.status === 200) {
      window.location = "/";
    } else {
      console.error("Failed to logout");
    }
  } catch (error) {
    console.error("Error occurred during logout:", error);
  }
};
```

Here Is where I have the is the code I made to logout, it uses axios.post to the endpoint called /Logout in the backend and if it is successful, it will navigate the user back to the home page if not you will get an error message.

Result

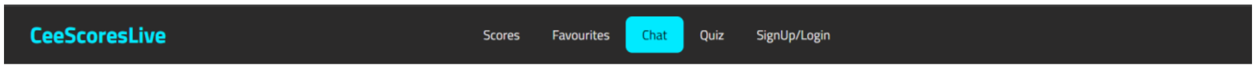


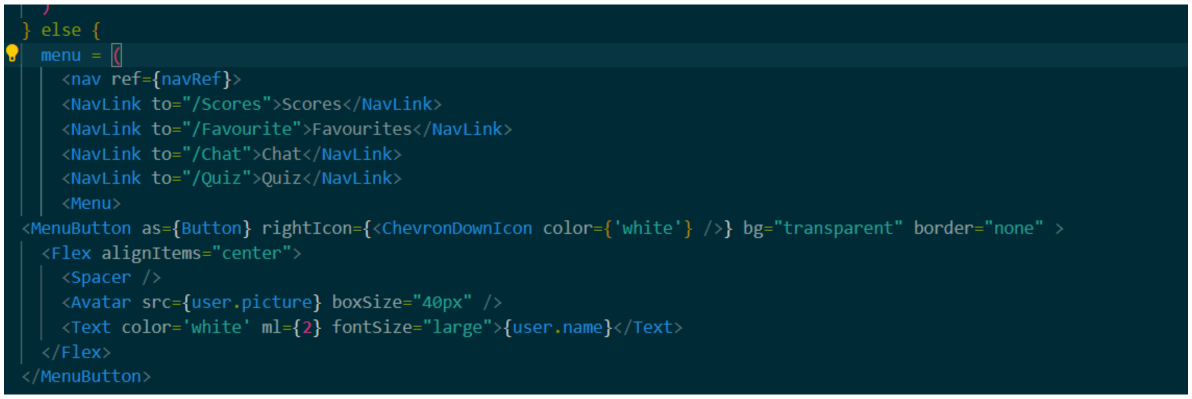


```
if (!props.name) {
  menu = (
    <nav ref={navRef}>
      <NavLink to="/Scores">Scores</NavLink>
      <NavLink to="/Favourite">Favourites</NavLink>
      <NavLink to="/Chat">Chat</NavLink>
      <NavLink to="/Quiz">Quiz</NavLink>
      <NavLink to="/SignUp">SignUp/Login</NavLink>
    <button className="nav-btn nav-close-btn" onClick={showNavBar}>
    <FaTimes />
  </button>

    </nav>
  )
} else {
```

This is the Navbar code to show the different pages you can navigate too.

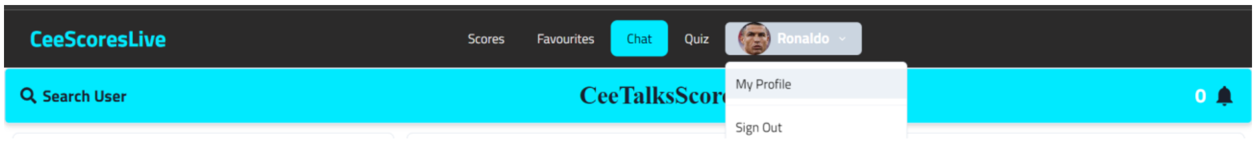
In this snippet it shows that if there is no prop.name then it will show this version of the navbar allowing the user to login or signup but still shows the rest of the pages available to the user.





```
} else {
  menu = (
    <nav ref={navRef}>
    <NavLink to="/Scores">Scores</NavLink>
    <NavLink to="/Favourite">Favourites</NavLink>
    <NavLink to="/Chat">Chat</NavLink>
    <NavLink to="/Quiz">Quiz</NavLink>
    <Menu>
<MenuButton as={Button} rightIcon={<ChevronDownIcon color={'white'} />} bg="transparent" border="none" >
  <Flex alignItems="center">
    <Spacer />
    <Avatar src={user.picture} boxSize="40px" />
    <Text color='white' ml={2} fontSize="large">{user.name}</Text>
  </Flex>
</MenuButton>
```

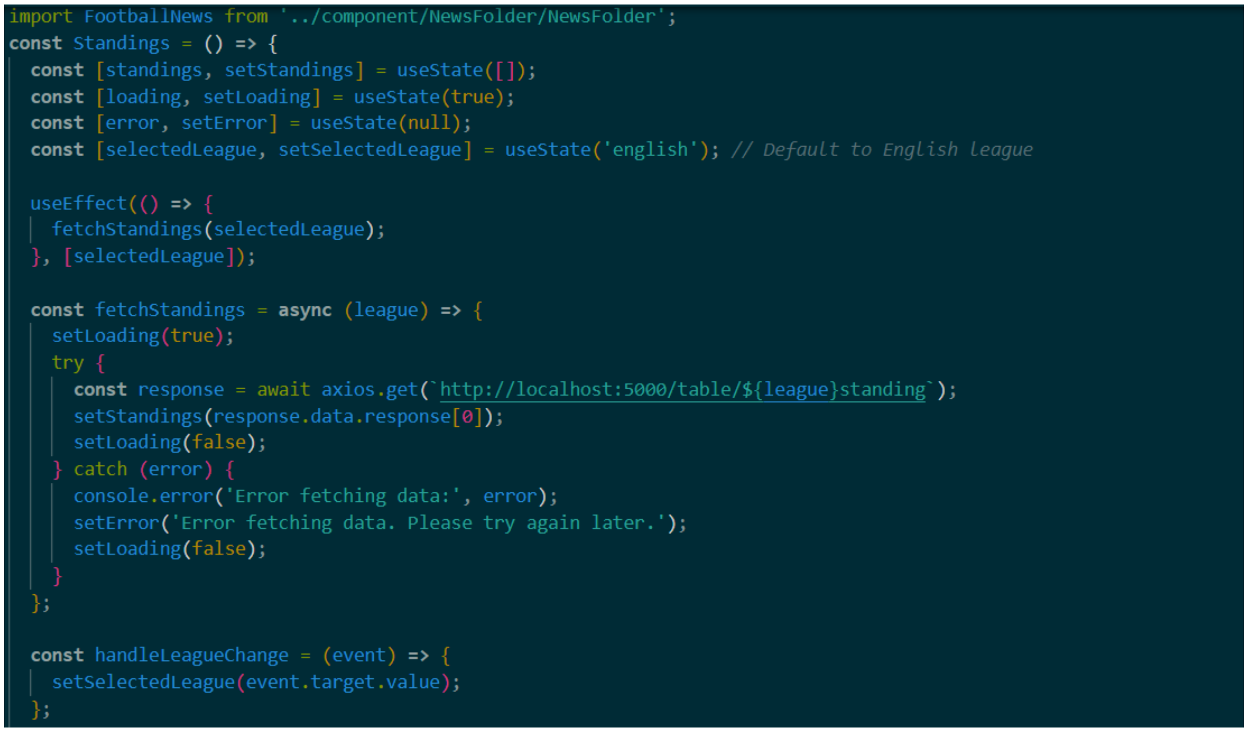
In this snippet it is the else statement which basically shows the navbar that will show when there is a prop.name and it will show the users name and the users profile picture and with a drop-down menu to show the user profile or a sign-out button.



So summary is if you are logged in it shows the profile and if you are not logged in it will give you the option to sign up or login.

7.6 Home Page

In this page I have the league table of the top 5 leagues in the work and I also show the news of the recent headlines in football.



```
import FootballNews from '../component/NewsFolder/NewsFolder';
const Standings = () => {
  const [standings, setStandings] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [selectedLeague, setSelectedLeague] = useState('english'); // Default to English league

  useEffect(() => {
    fetchStandings(selectedLeague);
  }, [selectedLeague]);

  const fetchStandings = async (league) => {
    setLoading(true);
    try {
      const response = await axios.get(`http://localhost:5000/table/${league}standing`);
      setStandings(response.data.response[0]);
      setLoading(false);
    } catch (error) {
      console.error('Error fetching data:', error);
      setError('Error fetching data. Please try again later.');
      setLoading(false);
    }
  };

  const handleLeagueChange = (event) => {
    setSelectedLeague(event.target.value);
  };
```

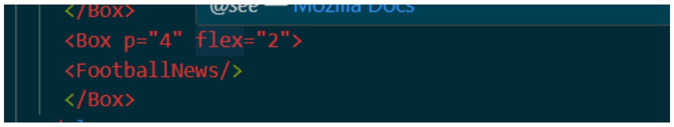
In this code snippet it shows the code I used to get the league tables:

First, I used useState to manage the states of the of each variable I created.

Then I used useEffect hook to trigger the fetchStandings function to whatever league has been chosen selectedleague but the default league is the English league as set using the useState.

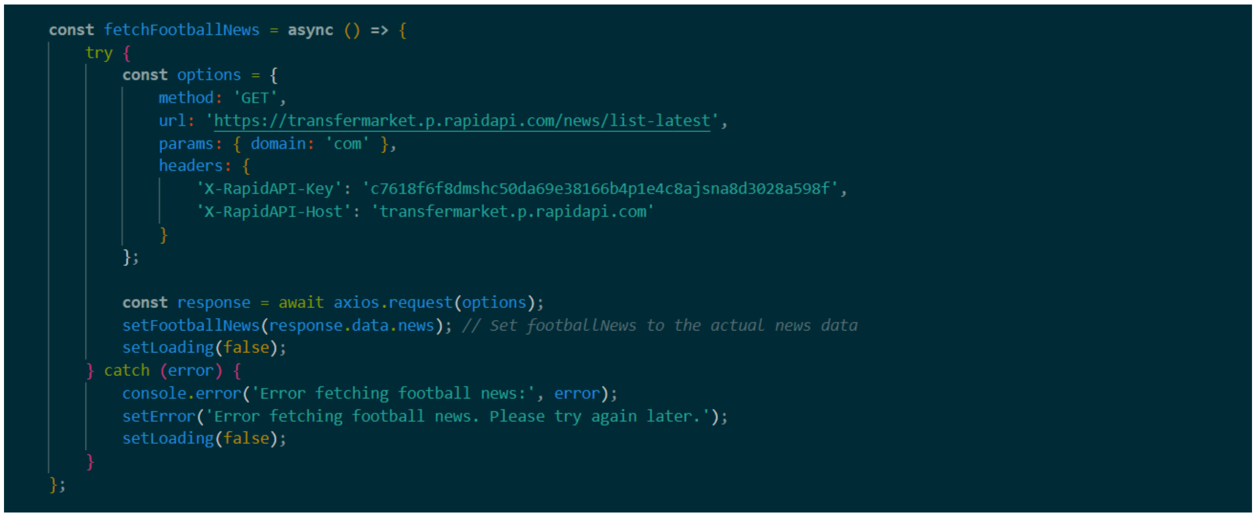
Like explained in the SignUp/Login page I also use axios but this time to get information from the from the endpoint, not post information. From the endpoint im getting the league table from an API in the backend if it gets the league table data it updates the standings state with the data sets loading false as done before.

HandleLeagueChange updates the selectedLeague state from whatever league the users select from the dropdown menu with all the teams on it.



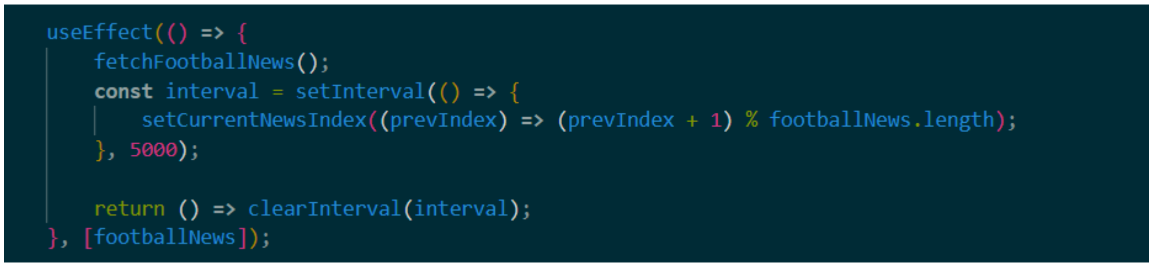
```
</Box>          @see — Mozilla Docs
<Box p="4" flex="2">
<FootballNews/>
</Box>
```

This is how I will call the footballNews to show the football news that was done from a different API and used styling to separate the football news from the league table using box and flex.



```
const fetchFootballNews = async () => {
    try {
        const options = {
            method: 'GET',
            url: 'https://transfermarket.p.rapidapi.com/news/list-latest',
            params: { domain: 'com' },
            headers: {
                'X-RapidAPI-Key': 'c7618f6f8dmshc50da69e38166b4p1e4c8ajsna8d3028a598f',
                'X-RapidAPI-Host': 'transfermarket.p.rapidapi.com'
            }
        };

        const response = await axios.request(options);
        setFootballNews(response.data.news); // Set footballNews to the actual news data
        setLoading(false);
    } catch (error) {
        console.error('Error fetching football news:', error);
        setError('Error fetching football news. Please try again later.');
        setLoading(false);
    }
};
```

This is how I was able to get the news data. Fetching the data through an API this time I'm fetching it from the frontend straight away instead of doing it through the backend then calling it to the frontend if I'm able to get the information from the API it stores the data in the setFootballNews which we will call again to display it.



```
useEffect(() => {
    fetchFootballNews();
    const interval = setInterval(() => {
        setCurrentNewsIndex((prevIndex) => (prevIndex + 1) % footballNews.length);
    }, 5000);

    return () => clearInterval(interval);
}, [footballNews]);
```

I then set and interval in the useEffect so that every 5 seconds it would change the news article to the next one, so it cycles around to show you the headlines when the page is open.

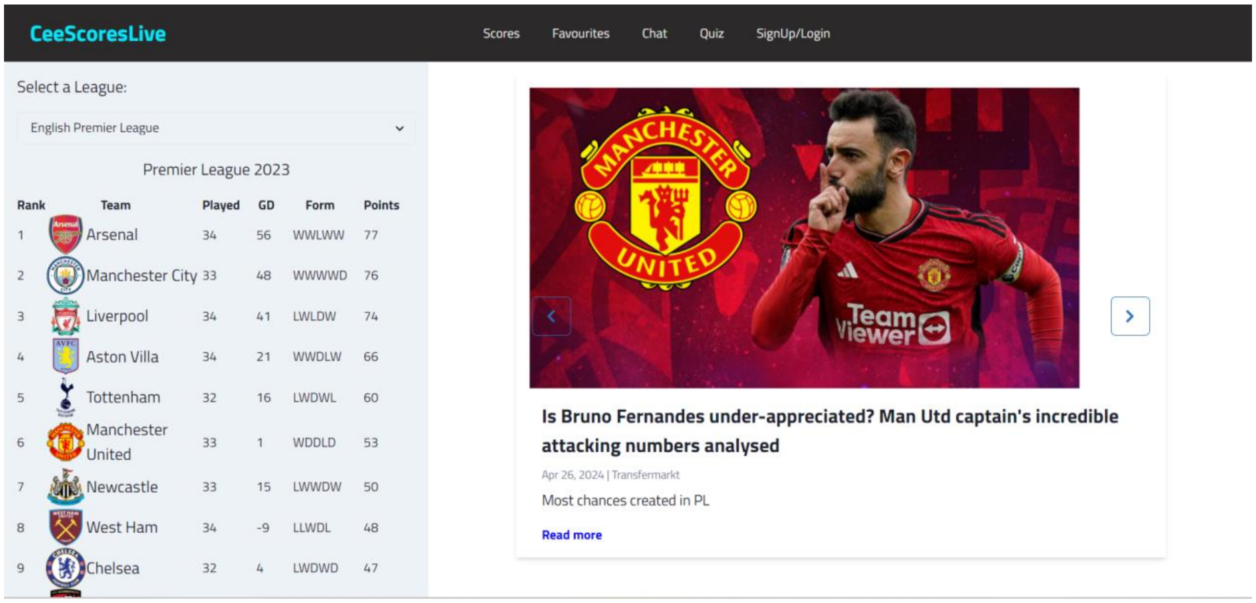
```
const handleNextNews = () => {
    setCurrentNewsIndex((prevIndex) => (prevIndex + 1) % footballNews.length);
};

const handlePrevNews = () => {
    setCurrentNewsIndex((prevIndex) => (prevIndex - 1 + footballNews.length) % footballNews.length);
};
```

You can also just move to the next news headline manually if you wish and you can always look at the pervious news headline allowing the user to move back and forth looking at each headline.

```
    <Link to={`/news/${footballNews[currentNewsIndex]?.id}`} style={{ fontSize: 'sm', color: 'blue', fontWeight: 'bold' }}>
        Read more
    </Link>
</Box>
```

Then this code which links you to another page which lets you read the article of the headline that catches your attention and in that page is the same as this just calls an API that allows you to get the full article.

Results



Premier League 2023

Rank	Team	Played	GD	Form	Points
1	Arsenal	34	56	WWLWW	77
2	Manchester City	33	48	WWWWD	76
3	Liverpool	34	41	LWLDW	74
4	Aston Villa	34	21	WWDLW	66
5	Tottenham	32	16	LWDWL	60
6	Manchester United	33	1	WDDLD	53
7	Newcastle	33	15	LWWDW	50
8	West Ham	34	-9	LLWDL	48
9	Chelsea	32	4	LWDWD	47

Is Bruno Fernandes under-appreciated? Man Utd captain's incredible attacking numbers analysed

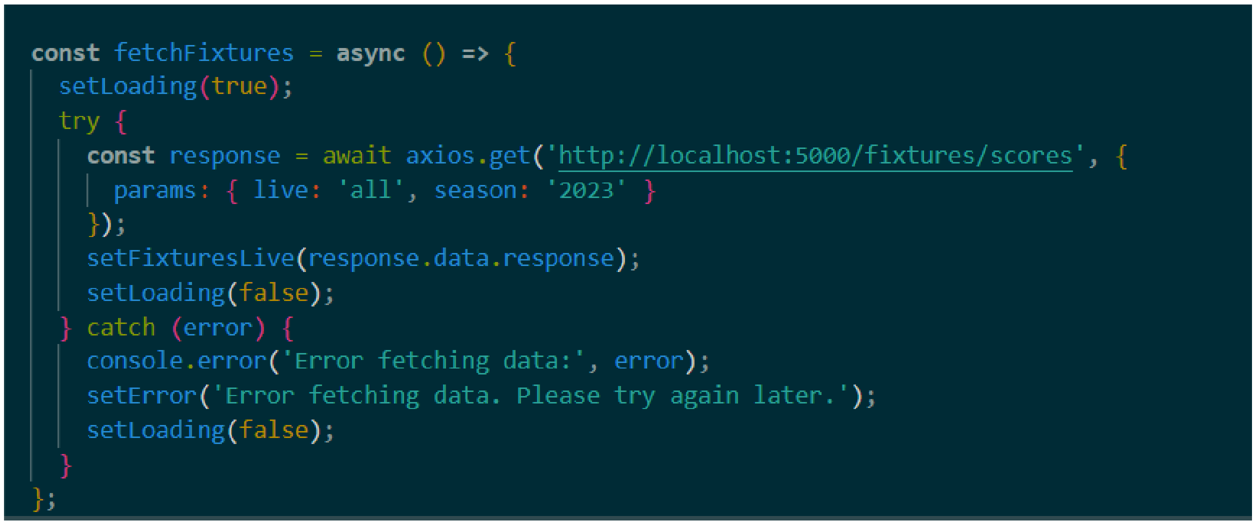
Apr 26, 2024 | Transfermarkt

Most chances created in PL

Read more

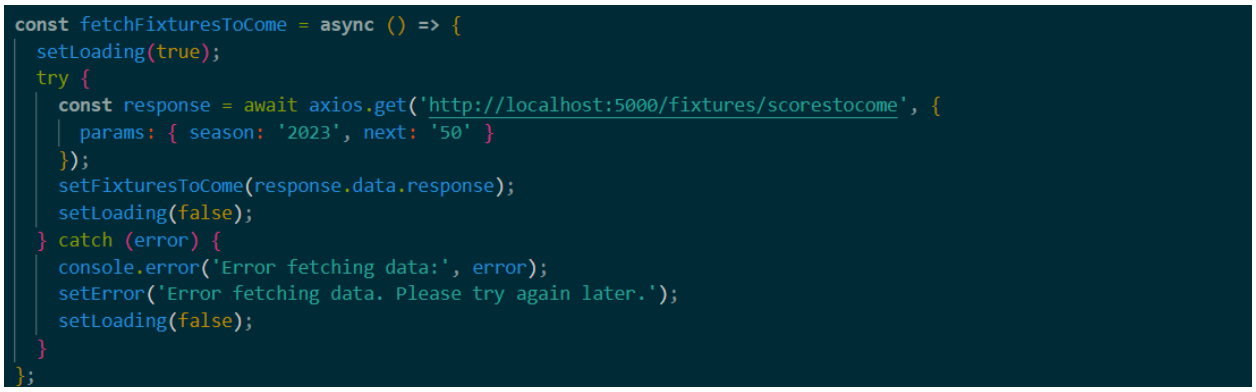
7.7 Score Page

In this I will be talking about the how I was able to show the scores of the past, future, and live matches.



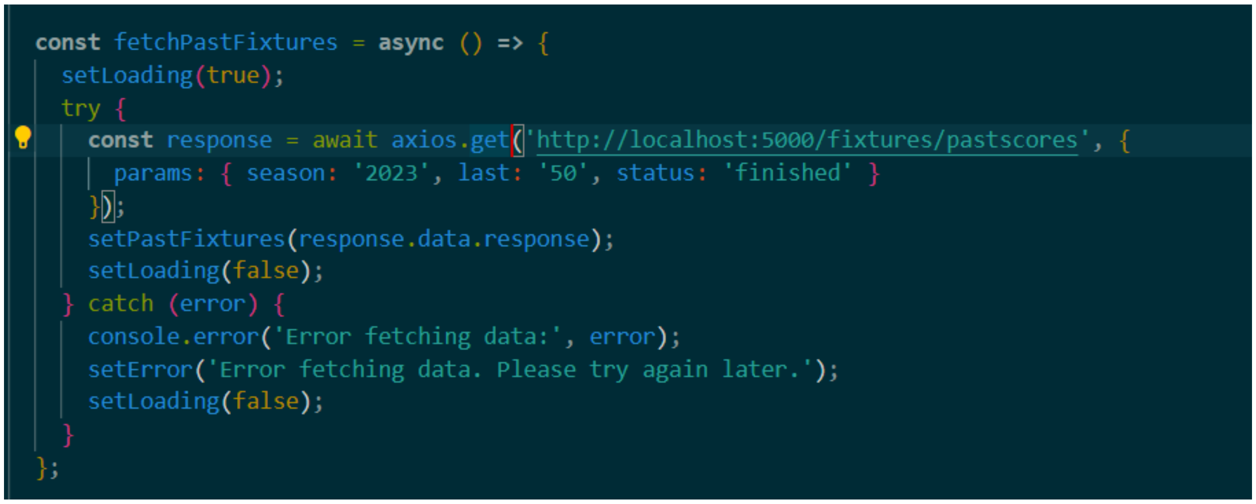
```
const fetchFixtures = async () => {
  setLoading(true);
  try {
    const response = await axios.get('http://localhost:5000/fixtures/scores', {
      params: { live: 'all', season: '2023' }
    });
    setFixturesLive(response.data.response);
    setLoading(false);
  } catch (error) {
    console.error('Error fetching data:', error);
    setError('Error fetching data. Please try again later.');
    setLoading(false);
  }
};
```

This code is how I can show live scores, I got the information for the scores in the backend where I stored the API, and I called here using the params needed to make it live and I set the data from the API into a variable called setFixtureLive.



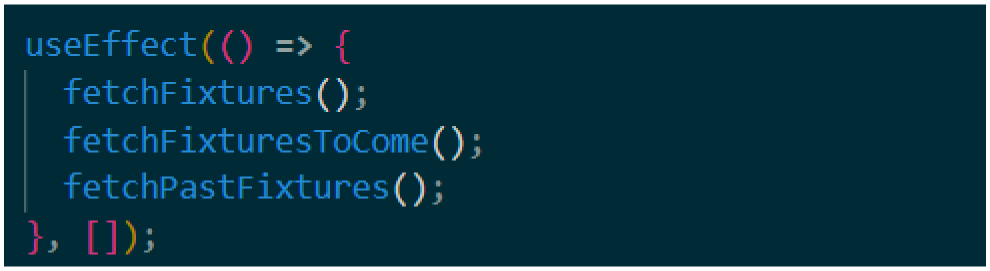
```
const fetchFixturesToCome = async () => {
  setLoading(true);
  try {
    const response = await axios.get('http://localhost:5000/fixtures/scorestocome', {
      params: { season: '2023', next: '50' }
    });
    setFixturesToCome(response.data.response);
    setLoading(false);
  } catch (error) {
    console.error('Error fetching data:', error);
    setError('Error fetching data. Please try again later.');
    setLoading(false);
  }
};
```

This code is how I can show future scores, I got the information for the scores in the backend where I stored the API, and I called here using the params needed to make it grab the information for the next 50 fixtures and I set the data from the API into a variable called setFixtureToCome.



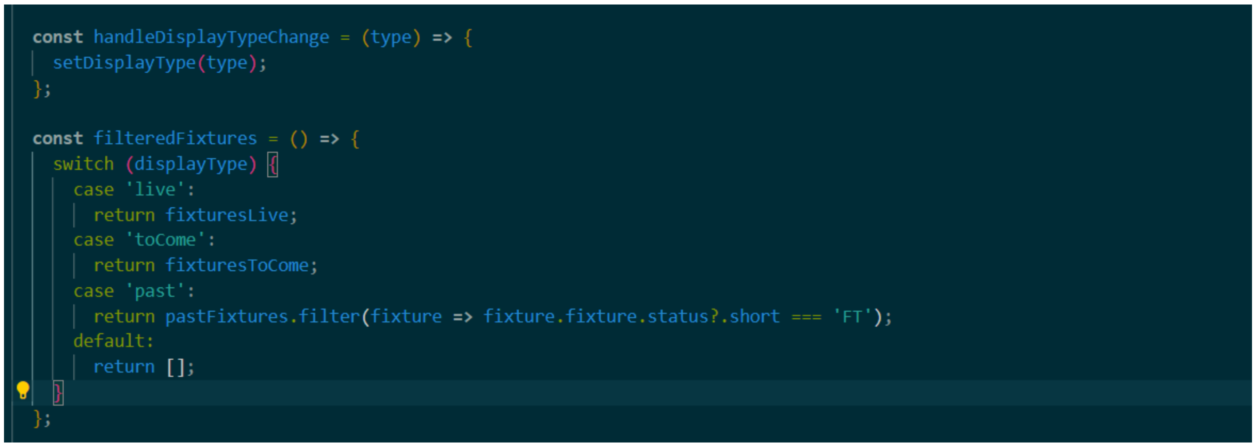
```
const fetchPastFixtures = async () => {
  setLoading(true);
  try {
    const response = await axios.get('http://localhost:5000/fixtures/pastscores', {
      params: { season: '2023', last: '50', status: 'finished' }
    });
    setPastFixtures(response.data.response);
    setLoading(false);
  } catch (error) {
    console.error('Error fetching data:', error);
    setError('Error fetching data. Please try again later.');
    setLoading(false);
  }
};
```

This code is how I can show past scores, I got the information for the scores in the backend where I stored the API, and I called here using the params needed to make it show the results of the last 50 games and I set the data from the API into a variable called setPastFixtures.



```
useEffect(() => {
  fetchFixtures();
  fetchFixturesToCome();
  fetchPastFixtures();
}, []);
```

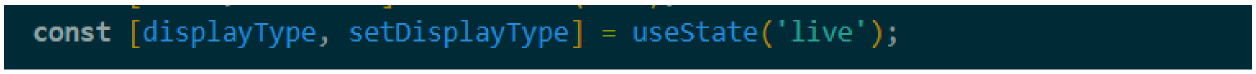
I then use useEffect to update the function every time I refresh the website so that scores or fixtures are up to date.



```
const handleDisplayTypeChange = (type) => {
  setDisplayType(type);
};

const filteredFixtures = () => {
  switch (displayType) {
    case 'live':
      return fixturesLive;
    case 'toCome':
      return fixturesToCome;
    case 'past':
      return pastFixtures.filter(fixture => fixture.fixture.status?.short === 'FT');
    default:
      return [];
  }
};
```

Here is how I made the button to pick which fixtures you want to do see so I have a handle display type change function that shows whatever one of the filteredfixtures that you have chosen.

I have made switch and cases to display whichever of the fixture types provided and if you haven't picked a fixture type by default it is set up to show the live score using this:

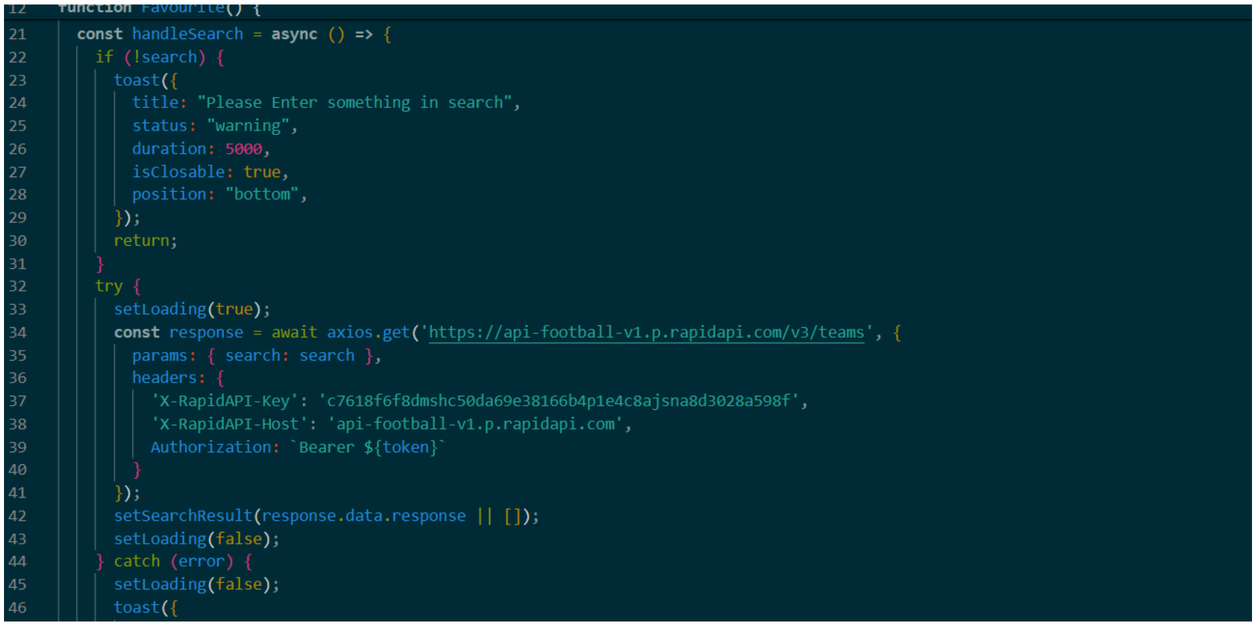


```
const [displayType, setDisplayType] = useState('live');
```

7.8 Favourite Page

On this page is where I allow the user to be able to pick a favourite team/team in a search so it only shows that teams information on this page it will show that teams fixtures, statistics, and the players statistics.

Here is the code breakdown:



```
function Favourite() {
  const handleSearch = async () => {
    if (!search) {
      toast({
        title: "Please Enter something in search",
        status: "warning",
        duration: 5000,
        isClosable: true,
        position: "bottom",
      });
      return;
    }
    try {
      setLoading(true);
      const response = await axios.get('https://api-football-v1.p.rapidapi.com/v3/teams', {
        params: { search: search },
        headers: {
          'X-RapidAPI-Key': 'c7618f6f8dmshc50da69e38166b4p1e4c8ajsna8d3028a598f',
          'X-RapidAPI-Host': 'api-football-v1.p.rapidapi.com',
          Authorization: `Bearer ${token}`
        }
      });
      setSearchResult(response.data.response || []);
      setLoading(false);
    } catch (error) {
      setLoading(false);
      toast({
```

This is where I make the code to allow the user to be able to search for the team they want to favourite, how this works is:

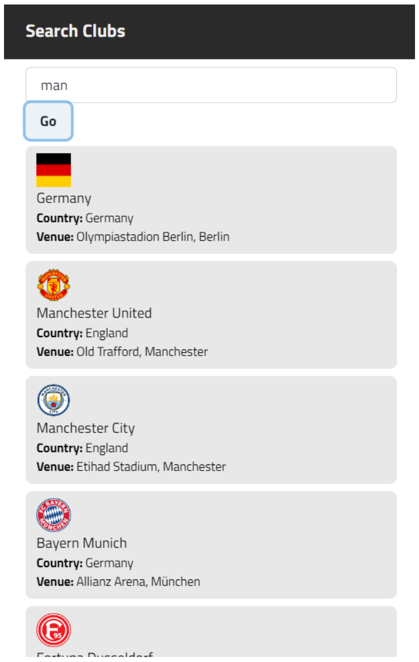
It first checks if you have anything written in the search parameters and if you don't you will get a warning telling you enter something in the search.

Next, I use axios again to get information team through a search param so in the search param it will allow you to search for a team in the API endpoint.

The information you get from the API or the team you chose will them be set as the setSearchResult which we use to display the data you chose.

If you couldn't get anything from the API, you will be given an error message.

Results



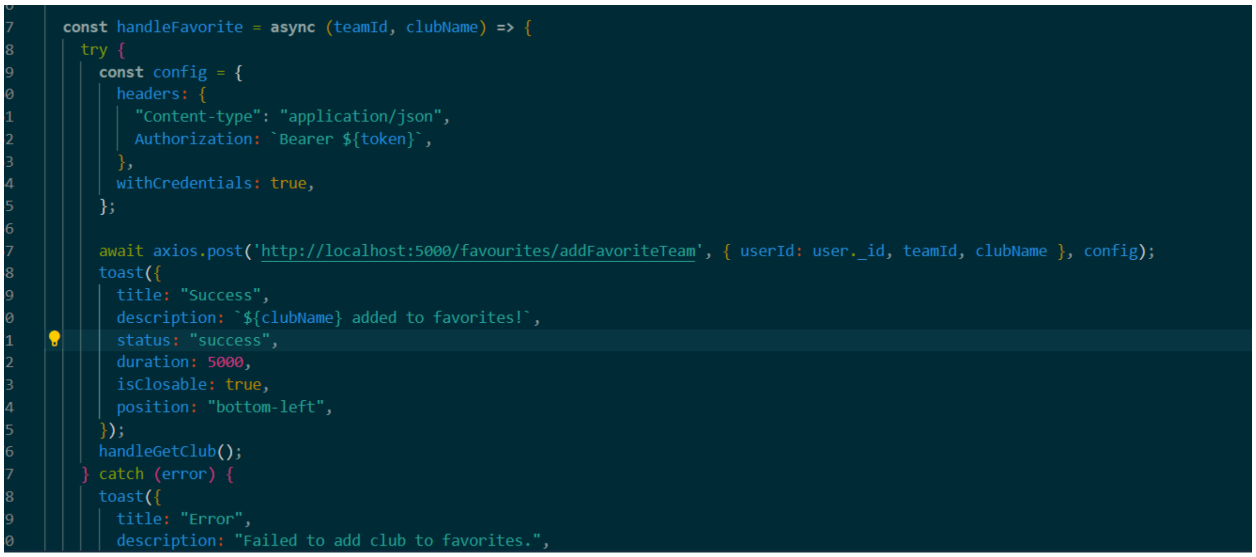
Germany
Country: Germany
Venue: Olympiastadion Berlin, Berlin

Manchester United
Country: England
Venue: Old Trafford, Manchester

Manchester City
Country: England
Venue: Etihad Stadium, Manchester

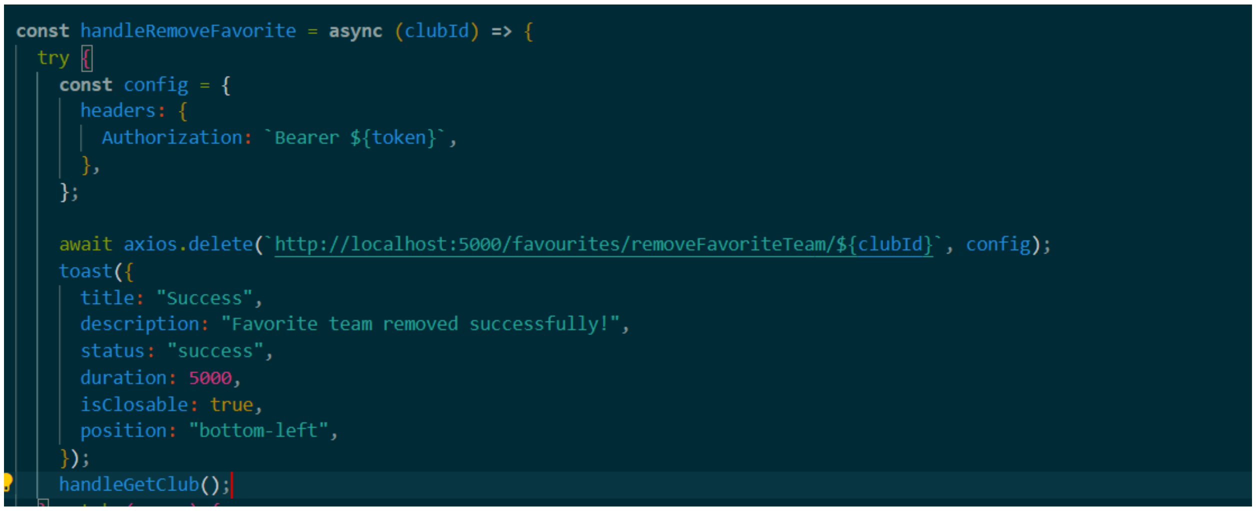
Bayern Munich
Country: Germany
Venue: Allianz Arena, München

Next is the function to add the team and to post the teams details back to the backend using axios.



```
const handleFavorite = async (teamId, clubName) => {
  try {
    const config = {
      headers: {
        "Content-type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      withCredentials: true,
    };

    await axios.post('http://localhost:5000/favourites/addFavoriteTeam', { userId: user._id, teamId, clubName }, config);
    toast({
      title: "Success",
      description: `${clubName} added to favorites!`,
      status: "success",
      duration: 5000,
      isClosable: true,
      position: "bottom-left",
    });
    handleGetClub();
  } catch (error) {
    toast({
      title: "Error",
      description: "Failed to add club to favorites.",
```

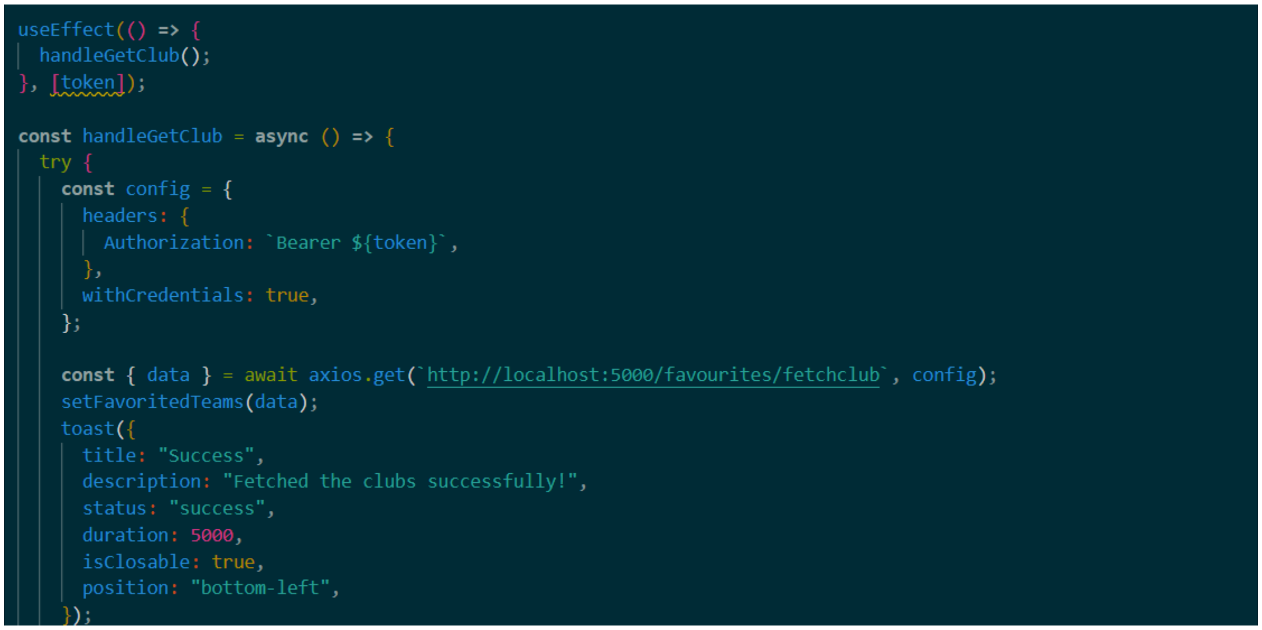
In this code the user posts the users the club's name and the club Id in the body, it then stores it in the backend so it can be processed and use the information to add the team as one of your favourite teams.



```
const handleRemoveFavorite = async (clubId) => {
  try {
    const config = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    await axios.delete(`http://localhost:5000/favourites/removeFavoriteTeam/${clubId}`, config);
    toast({
      title: "Success",
      description: "Favorite team removed successfully!",
      status: "success",
      duration: 5000,
      isClosable: true,
      position: "bottom-left",
    });
    handleGetClub();
```

This code is what I used to delete a club from the favourite team section.

This time I used axios.delete and not axios.post or get so this will delete. So, in this code snippet it deletes clubId which will delete the club for the database making the team be gone and you can add it back using the favourite.



```
useEffect(() => {
  handleGetClub();
}, [token]);

const handleGetClub = async () => {
  try {
    const config = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
      withCredentials: true,
    };

    const { data } = await axios.get(`http://localhost:5000/favourites/fetchclub`, config);
    setFavoritedTeams(data);
    toast({
      title: "Success",
      description: "Fetched the clubs successfully!",
      status: "success",
      duration: 5000,
      isClosable: true,
      position: "bottom-left",
    });
```

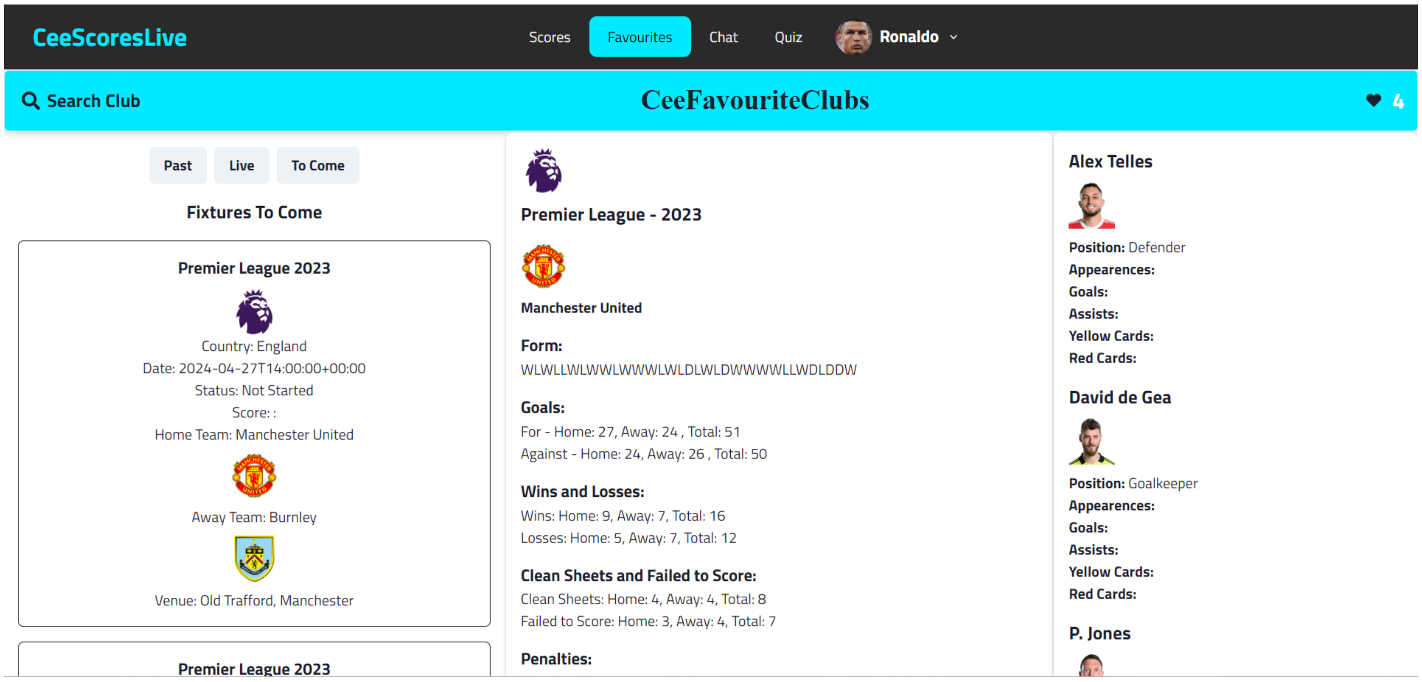
This code is to get and show the teams that you have added to the as a favourite team database and I put the function into a useEffect so that whenever you make a change it will update the favourite teams so it's always up to date whenever something happens.

```
    <Box >
      <FavouriteFixtures teamId={filteredTeamId} />
    </Box>
  </Box>
```

I then call this function to show components like I did previously so here I'm calling Favourite teams fixtures; team stats and players stats and I'm also passing the favourite teams Id so it can filter and only show that team's information.

I then call other API like the one I used to get the league table I get information on the player statistics, team statistics and fixtures but I use the id that I sent through props to them filter out the team so that it only shows information for that team.

Results



Fixtures To Come

Premier League 2023
Country: England
Date: 2024-04-27T14:00:00+00:00
Status: Not Started
Score: :
Home Team: Manchester United
Away Team: Burnley
Venue: Old Trafford, Manchester

Premier League 2023

Premier League - 2023
Manchester United
Form:
WLWLLWLWWLWWWLWLDLWLDWWWWLLWDLDDW
Goals:
For - Home: 27, Away: 24 , Total: 51
Against - Home: 24, Away: 26 , Total: 50
Wins and Losses:
Wins: Home: 9, Away: 7, Total: 16
Losses: Home: 5, Away: 7, Total: 12
Clean Sheets and Failed to Score:
Clean Sheets: Home: 4, Away: 4, Total: 8
Failed to Score: Home: 3, Away: 4, Total: 7
Penalties:

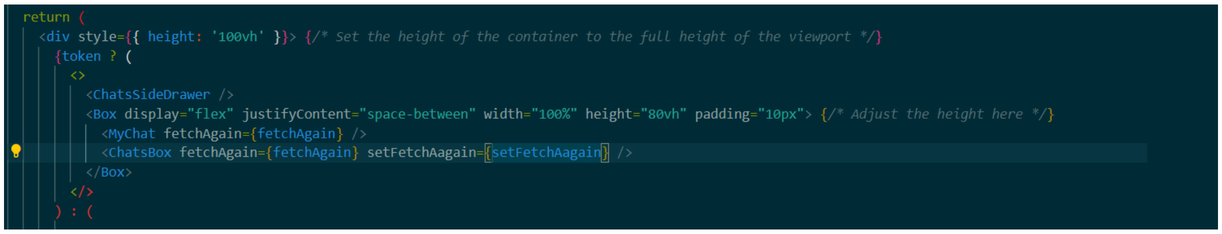
Alex Telles
Position: Defender
Appearences:
Goals:
Assists:
Yellow Cards:
Red Cards:

David de Gea
Position: Goalkeeper
Appearences:
Goals:
Assists:
Yellow Cards:
Red Cards:

P. Jones

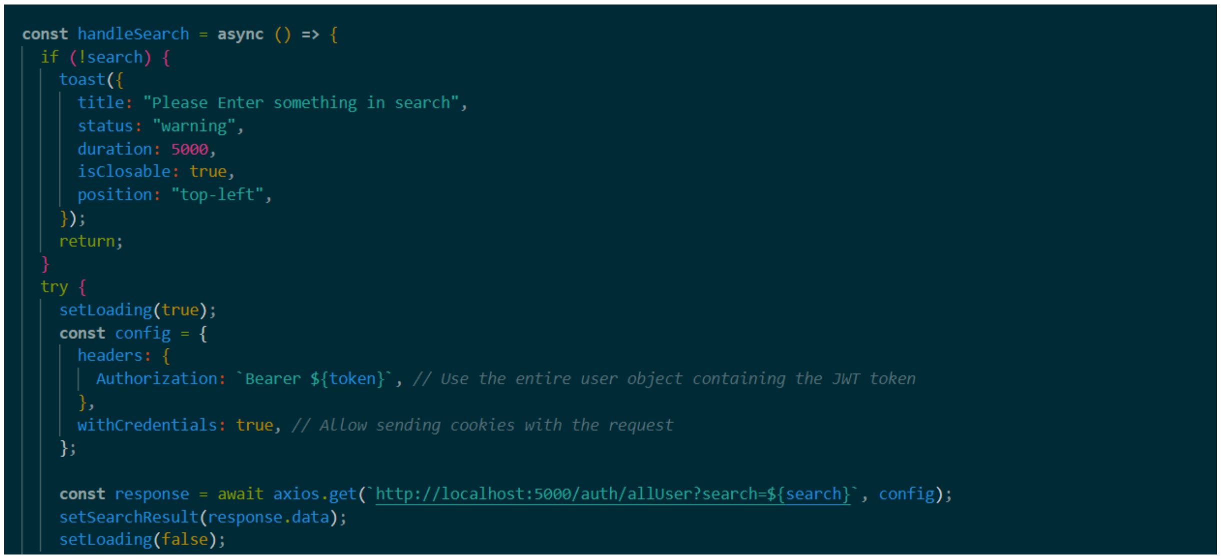
7.9 Chat Page

Here I will be talking about how I make my fully functional real time chat for my users to be able to communicate with each other. We will start off with the chat page itself:



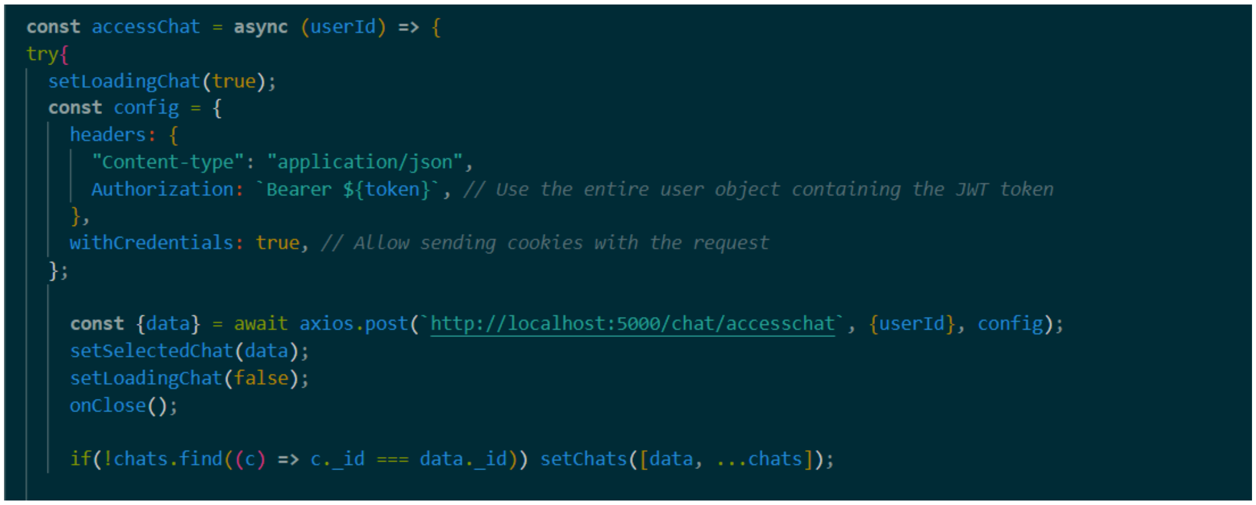
```
return (
  <div style={{ height: '100vh' }}> {/* Set the height of the container to the full height of the viewport */}
    {token ? (
      <>
        <ChatsSideDrawer />
        <Box display="flex" justifyContent="space-between" width="100%" height="80vh" padding="10px"> {/* Adjust the height here */}
          <MyChat fetchAgain={fetchAgain} />
          <ChatsBox fetchAgain={fetchAgain} setFetchAagain={setFetchAagain} />
        </Box>
      </>
    ) : (
```

In this code I’m just importing the components I used that handles different aspects of the chat page ill start from the top and explain each of the chat components:



```
const handleSearch = async () => {
  if (!search) {
    toast({
      title: "Please Enter something in search",
      status: "warning",
      duration: 5000,
      isClosable: true,
      position: "top-left",
    });
    return;
  }
  try {
    setLoading(true);
    const config = {
      headers: {
        Authorization: `Bearer ${token}`, // Use the entire user object containing the JWT token
      },
      withCredentials: true, // Allow sending cookies with the request
    };

    const response = await axios.get(`http://localhost:5000/auth/allUser?search=${search}`, config);
    setSearchResult(response.data);
    setLoading(false);
```

In the chatSideDrawer I use the same type of code that I did for the favourite page which allows the user to search for other users that are stored in the database unlike before how I got the from an API and it stores the data into the setSearchResult variable.

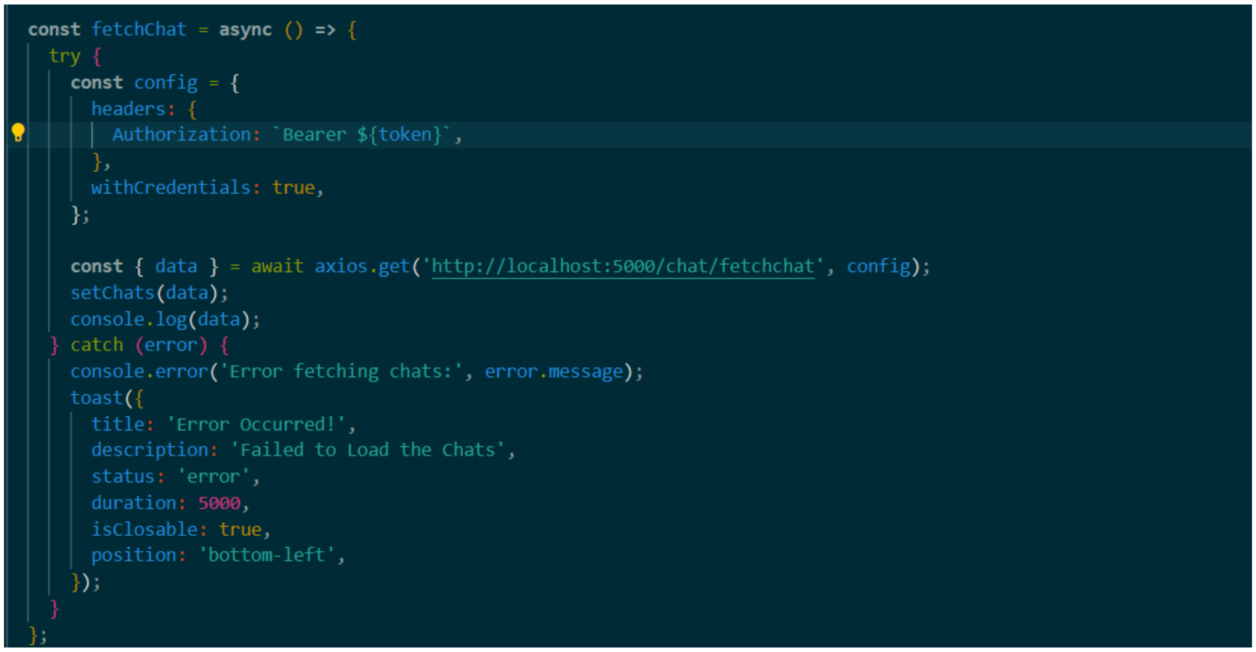


```
const accessChat = async (userId) => {
try{
  setLoadingChat(true);
  const config = {
    headers: {
      "Content-type": "application/json",
      Authorization: `Bearer ${token}`, // Use the entire user object containing the JWT token
    },
    withCredentials: true, // Allow sending cookies with the request
  };

    const {data} = await axios.post(`http://localhost:5000/chat/accesschat`, {userId}, config);
    setSelectedChat(data);
    setLoadingChat(false);
    onClose();

    if(!chats.find((c) => c._id === data._id)) setChats([data, ...chats]);
```

This how I create/access chats with other users here. How it works is I use axios.post and I'm passing the userId of the user I wish to talk to back to the backend so it will either create a new chat with the user linked to that Id or it will get the chat with user if you have already created a chat with the user.

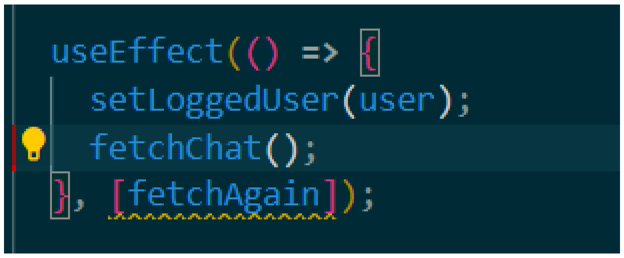
Next it stores the data, stops the loading, hits the onClose(); so when I click and successfully pass the userId it will close the chatSidedrawer.

Finally, the last line is used to check if the chat is already in the list of chats you already made if not it will create the new chat with the person so ultimately it is used to update the chat list.



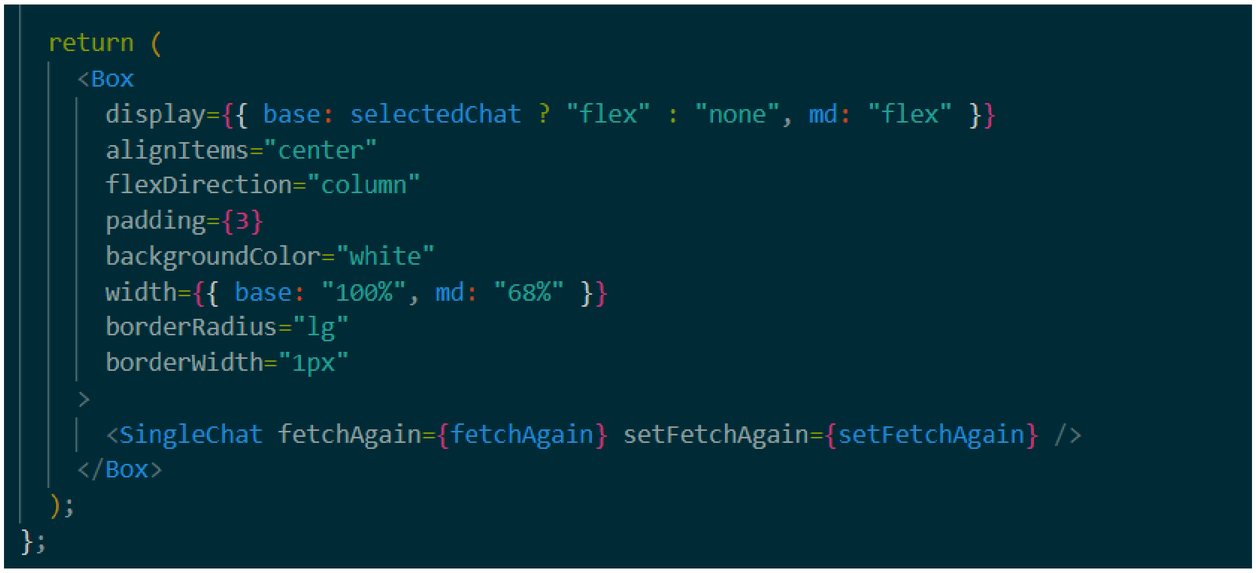
```
const fetchChat = async () => {
  try {
    const config = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
      withCredentials: true,
    };

    const { data } = await axios.get('http://localhost:5000/chat/fetchchat', config);
    setChats(data);
    console.log(data);
  } catch (error) {
    console.error('Error fetching chats:', error.message);
    toast({
      title: 'Error Occurred!',
      description: 'Failed to Load the Chats',
      status: 'error',
      duration: 5000,
      isClosable: true,
      position: 'bottom-left',
    });
  }
};
```

This code snippet is found in the next component called MyChats and all it does is use axios.get to get all the chats that user has created and stores the data in a variable and displays it.



```
useEffect(() => {
  setLoggedUser(user);
  fetchChat();
}, [fetchAgain]);
```

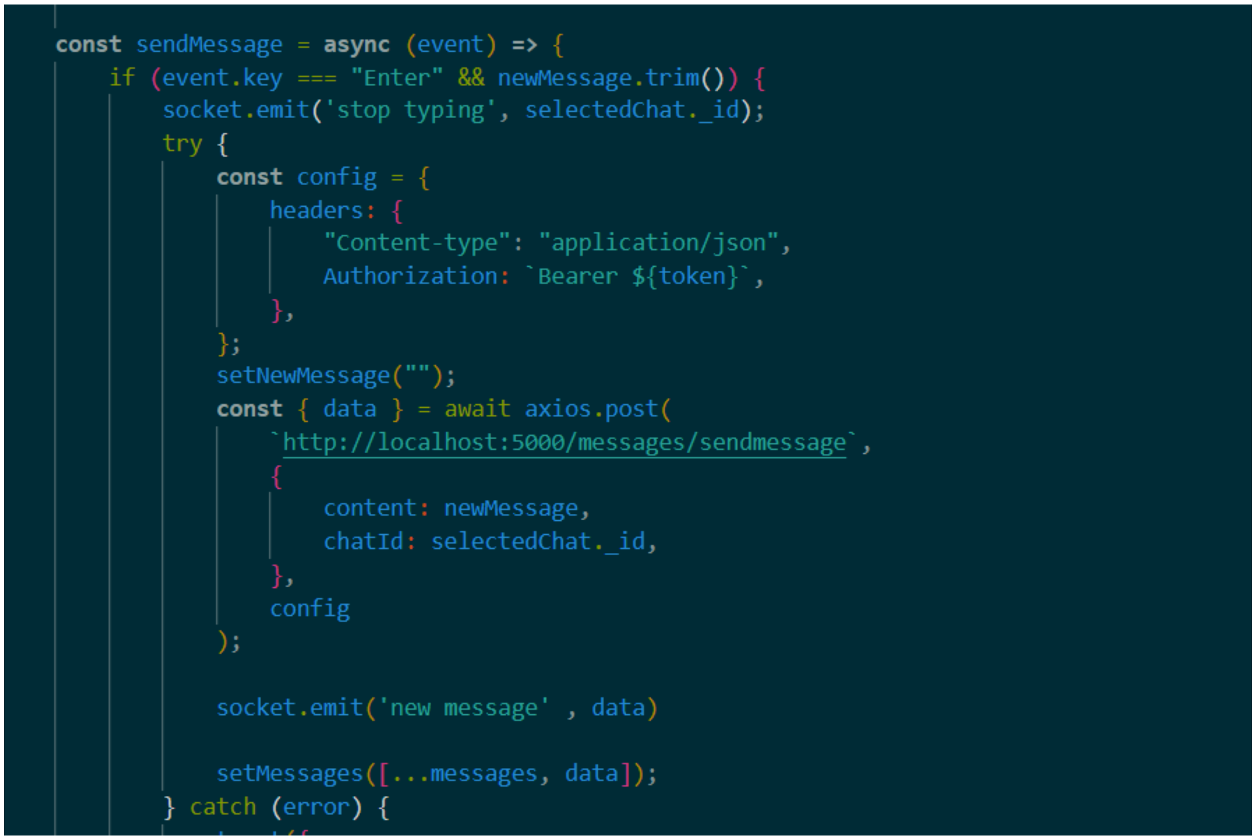
Once again, I use useEffect to make sure the chats are all up to date whenever a change is made to the chats like the user delete a chat or just adding a new chat and sets the logged in user to user that is logged in so if a new user logs in it will use that user information.



```
  return (
    <Box
      display={{ base: selectedChat ? "flex" : "none", md: "flex" }}
      alignItems="center"
      flexDirection="column"
      padding={3}
      backgroundColor="white"
      width={{ base: "100%", md: "68%" }}
      borderRadius="lg"
      borderWidth="1px"
    >
      <SingleChat fetchAgain={fetchAgain} setFetchAgain={setFetchAgain} />
    </Box>
  );
};
```

This is the code for the chatBox so here it makes chat screen more responsive and adapts to different screen sizes using Chakra UI. It will also use selectedChat from the previous code snippet to show the selectedChat in the chatBox.

It also imports SingleChat component to this screen and it is responsible for showing the individual chat/conversation the user and the other users, here is the code breakdown for it:



```
const sendMessage = async (event) => {
    if (event.key === "Enter" && newMessage.trim()) {
        socket.emit('stop typing', selectedChat._id);
        try {
            const config = {
                headers: {
                    "Content-type": "application/json",
                    Authorization: `Bearer ${token}`,
                },
            };
            setNewMessage("");
            const { data } = await axios.post(
                `http://localhost:5000/messages/sendmessage`,
                {
                    content: newMessage,
                    chatId: selectedChat._id,
                },
                config
            );

            socket.emit('new message' , data)

            setMessages([...messages, data]);
        } catch (error) {
```

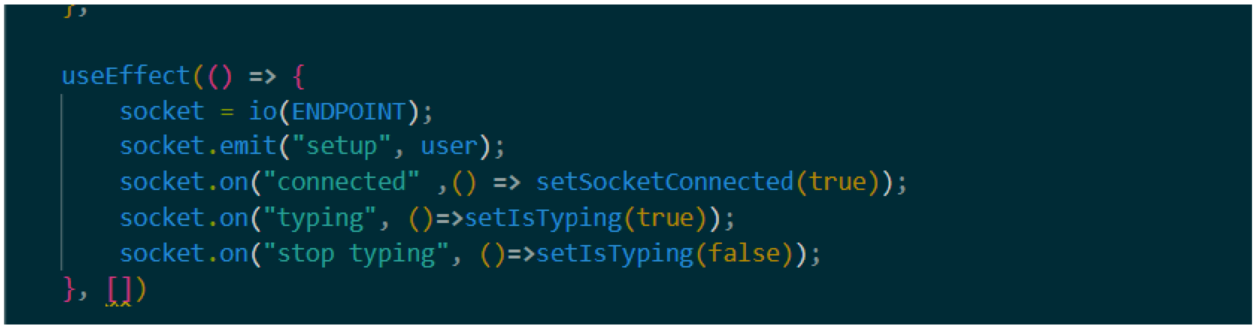
This is a code snippet that allows the user to send messages to other users. The first line checks if the enter key was pressed and it will also remove the whitespace from both ends of the message.

If the condition is true it tells the backend of the code that a user in selectedChat._id has stopped typing and emits ‘stop typing’ to that selectedChat._id socket server.

It then makes newMessage as an empty string.

Passes the content of the message and the chatId in the request body to the backend and stores the data retrieved into the variable setMessages.

When the message is sent it emits ‘new message’ event to the socket server with the data retrieved from the Post request this make the chat messages real time. Then sets the variable state for messages to the updated message.



```
};

useEffect(() => {
    socket = io(ENDPOINT);
    socket.emit("setup", user);
    socket.on("connected" ,() => setSocketConnected(true));
    socket.on("typing", ()=>setIsTyping(true));
    socket.on("stop typing", ()=>setIsTyping(false));
}, [])
```

In this code this is how I got socket.io to work with my website to make the chat real time.

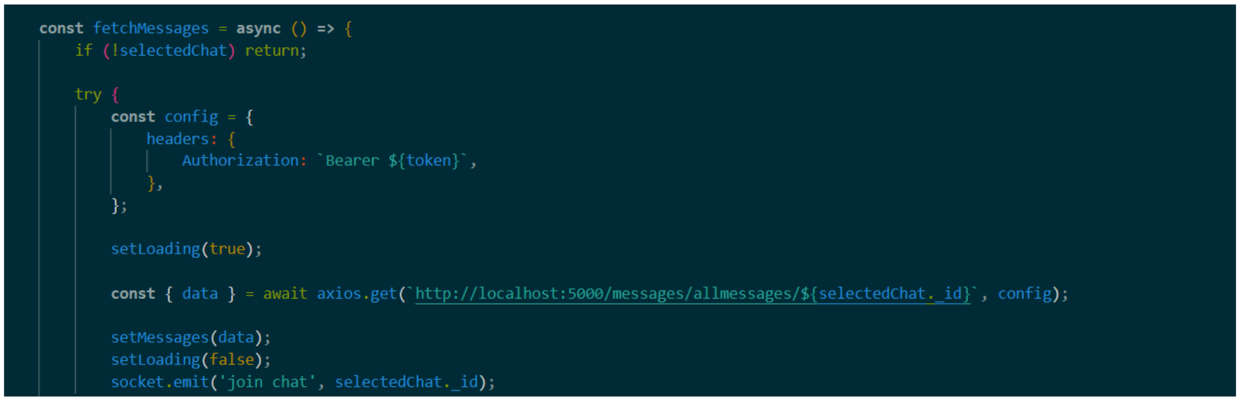
Socket = io (ENDPOINT) creates a new WebSocket connection to the server at the endpoint which is called in the backend.

Socket.emit(“setup”, user); sends a ‘setup event to the server with the user’s data this is done to registers the user with the socket server when they connect or join rooms.

Next line sends an event called ‘connected’ that calls setSocketConnected(true) which tracks the connection status of the socket.

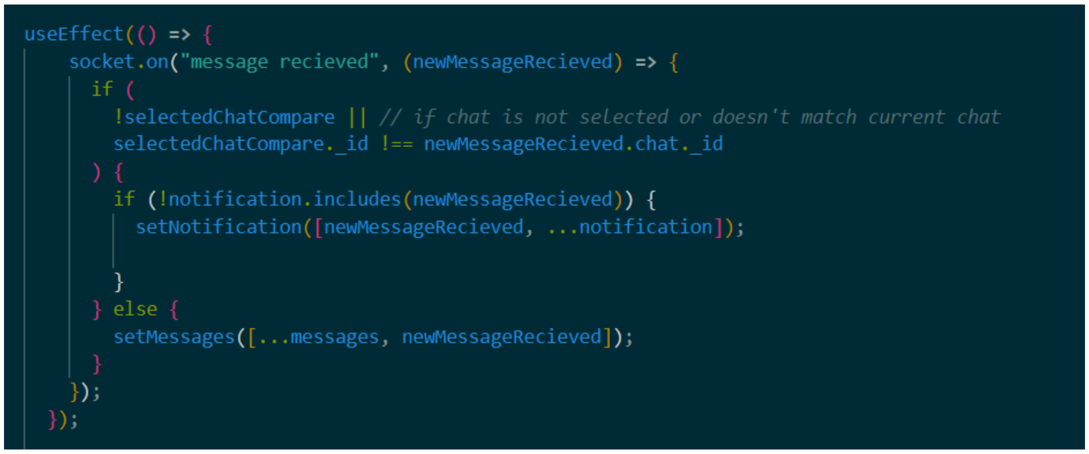
It then indictates if a user is typing or not by sending the events ‘typing’ and ‘stop typing’ to the socket server.

I then you use useEffect to update the socket.io whenever a change is made.



```
const fetchMessages = async () => {
    if (!selectedChat) return;

    try {
        const config = {
            headers: {
                Authorization: `Bearer ${token}`,
            },
        };

        setLoading(true);

        const { data } = await axios.get(`http://localhost:5000/messages/allmessages/${selectedChat._id}`, config);

        setMessages(data);
        setLoading(false);
        socket.emit('join chat', selectedChat._id);
```

First, I check if there are any selected chats and if there are none then it returns. Then it gets the all the messages from the chat with the selectedChat Id from the backend using axios.get and sets the data to the setMessages with the messages from the selected chat. Finally I use socket.emit('join chat', selectedChat._id) so once you join/enter the chat with the Id provided it notifies the socket server a user has joined the chat.

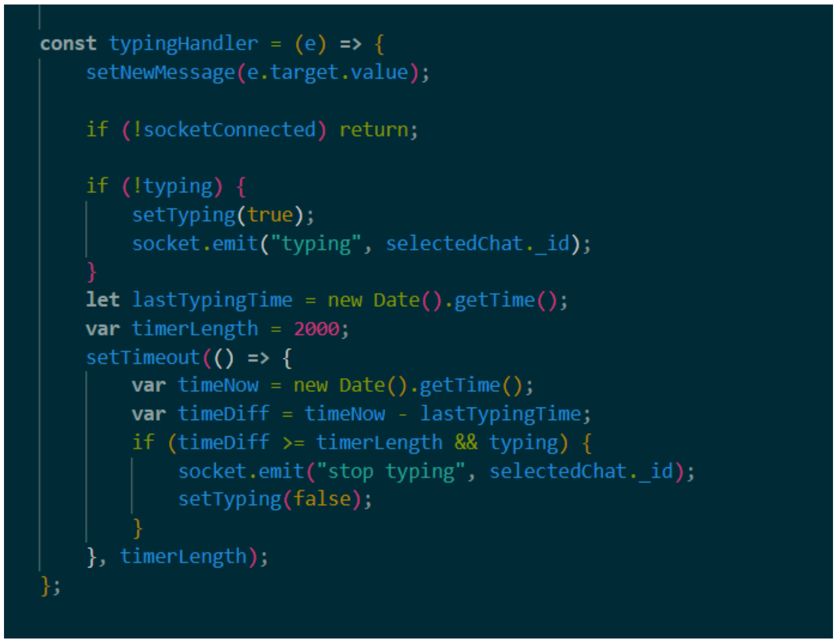


```
useEffect(() => {
    socket.on("message recieved", (newMessageRecieved) => {
      if (
        !selectedChatCompare || // if chat is not selected or doesn't match current chat
        selectedChatCompare._id !== newMessageRecieved.chat._id
      ) {
        if (!notification.includes(newMessageRecieved)) {
          setNotification([newMessageRecieved, ...notification]);

        }
      } else {
        setMessages([...messages, newMessageRecieved]);
      }
    });
  });
```

In this code it is how I made the notification for the messages first it sends and event to the socket server called 'message received ', (messageReceived) to send the message received back. After it checks if there is a selected chat, if there is no selected chat or if the selected chat doesn't match the Id of the chat that is receiving the message, then it moves to next check.

The next If statement checks if the received message is included in the notification state if it is not included then it will setNotification with newMessageRecieved.

Else it will setMessages to update the messages state to the new message received and all the pervious messages (…messages).



```
const typingHandler = (e) => {
    setNewMessage(e.target.value);

    if (!socketConnected) return;

    if (!typing) {
        setTyping(true);
        socket.emit("typing", selectedChat._id);
    }
    let lastTypingTime = new Date().getTime();
    var timerLength = 2000;
    setTimeout(() => {
        var timeNow = new Date().getTime();
        var timeDiff = timeNow - lastTypingTime;
        if (timeDiff >= timerLength && typing) {
            socket.emit("stop typing", selectedChat._id);
            setTyping(false);
        }
    }, timerLength);
};
```

The final part of the code in the single chat component is the typing handler which shows a typing icon when the user is typing and if the user stops typing for 2 seconds, then the typing icon will disappear till the user starts typing again.

Here is the breakdown:

First when the types in the chat box setNewMessage(e.target.value) is called to set the value of NewMessage to the whatever is typed.

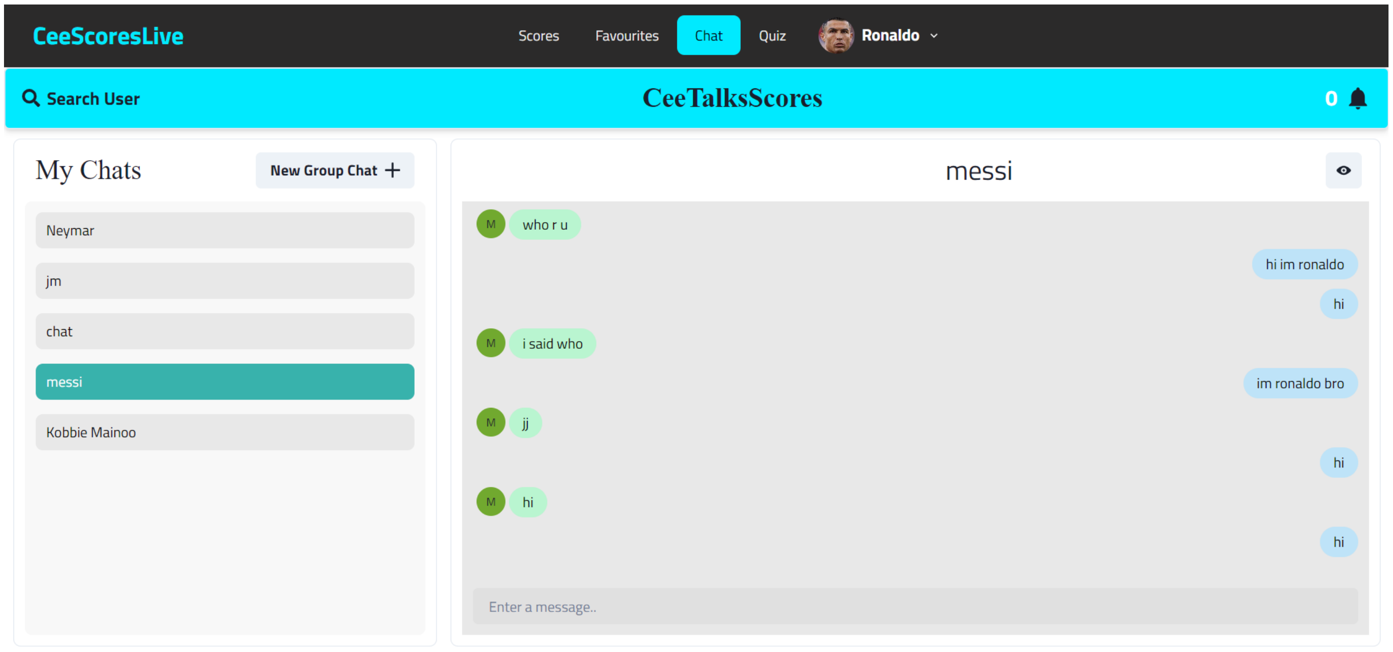
It checks if socket is connected if not it will return early.

Next is if the user is not typing setTyping to true is called to update the typing state and it will emit the event 'typing' to the socket with the selectedchat._id.

It will then set a 2 second timer, so after 2 seconds it calculates the time difference between now and the last time you typed, if the time difference is greater than or equal to timer length and the user is shown to be typing it will emit 'stop typing' and set the typing to false, this will show the user has stopped typing or has paused.

I used this video to understand/learn how to build a real time chat page and merged it with my code to get it to work with my web application [20].

Results



M who r u

hi im ronaldo

hi

M i said who

im ronaldo bro

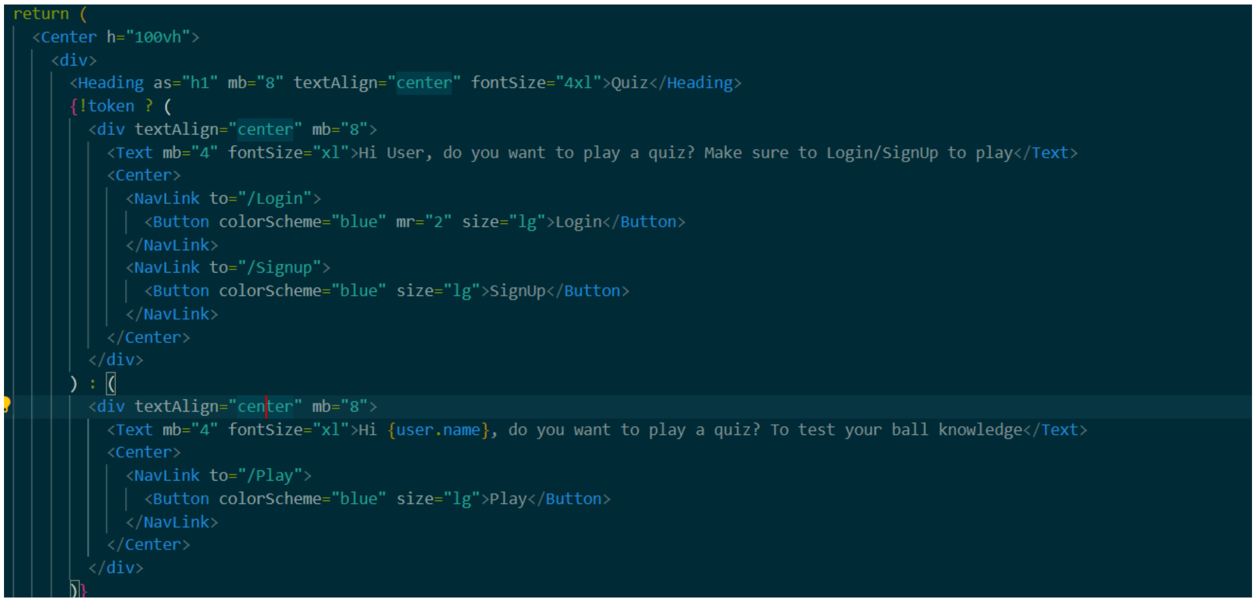
M jj

hi

M hi

hi

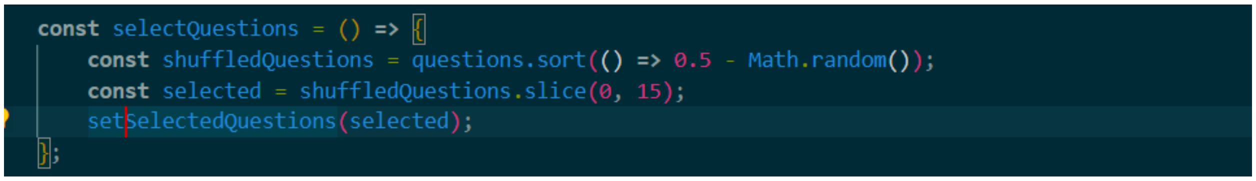
7.10 Quiz Page



```
return (
  <Center h="100vh">
    <div>
      <Heading as="h1" mb="8" textAlign="center" fontSize="4xl">Quiz</Heading>
      {!token ? (
        <div textAlign="center" mb="8">
          <Text mb="4" fontSize="xl">Hi User, do you want to play a quiz? Make sure to Login/SignUp to play</Text>
          <Center>
            <NavLink to="/Login">
              <Button colorScheme="blue" mr="2" size="lg">Login</Button>
            </NavLink>
            <NavLink to="/Signup">
              <Button colorScheme="blue" size="lg">SignUp</Button>
            </NavLink>
          </Center>
        </div>
      ) : (
        <div textAlign="center" mb="8">
          <Text mb="4" fontSize="xl">Hi {user.name}, do you want to play a quiz? To test your ball knowledge</Text>
          <Center>
            <NavLink to="/Play">
              <Button colorScheme="blue" size="lg">Play</Button>
            </NavLink>
          </Center>
        </div>
      )}
```

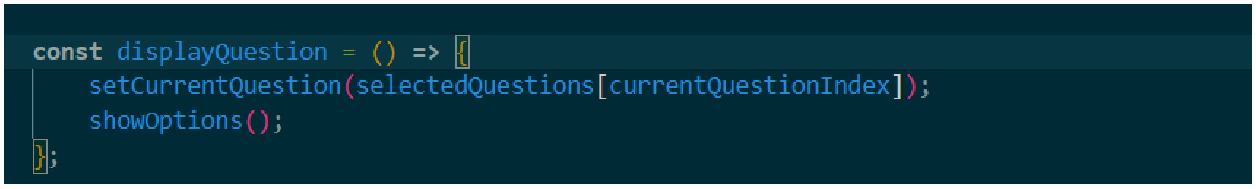
This the code for the quiz page the functionality behind it is that if there is no token or if a user is not found then it will say Hi user, do you want to play the quiz make sure you login or sign up then it shows you the link for both the login and signup pages for the user to sign up or login.

But if there is user or a token then you will be shown a play button that allows you to play the quiz, I will breakdown the code for the play quiz now:



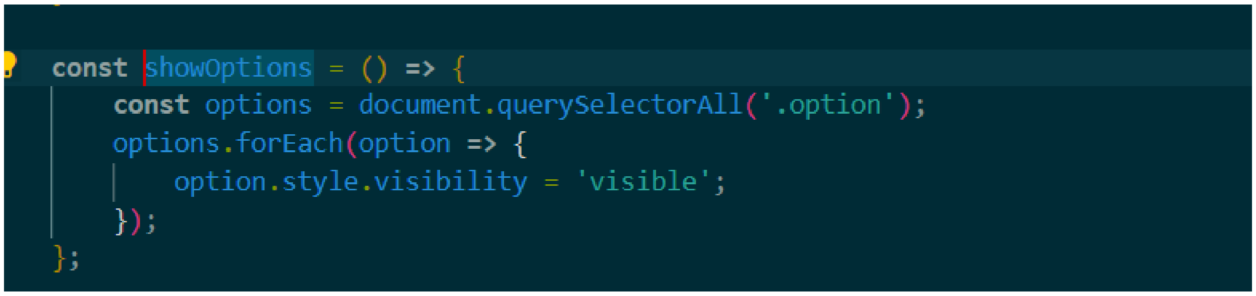
```
const selectQuestions = () => {
    const shuffledQuestions = questions.sort(() => 0.5 - Math.random());
    const selected = shuffledQuestions.slice(0, 15);
    setSelectedQuestions(selected);
};
```

This code is how I select the questions randomly, I use sort() => 0.5 and Math.random() to shuffle the 50 questions that I imported that were generated by chatgpt the next line slices the 50 questions and only picks 15 of the 50 questions randomly and then sets the selectected questions to selected which is the selected questions that were shuffled.



```
const displayQuestion = () => {
    setCurrentQuestion(selectedQuestions[currentQuestionIndex]);
    showOptions();
};
```

This is how the current question is displayed so I call the variable setCurrentQuestion and I show the current question based on the on what the current question index is, and the current question index is the index number of the array storing the questions. So, if the index number is 0 in the array it will show the question linked to that.

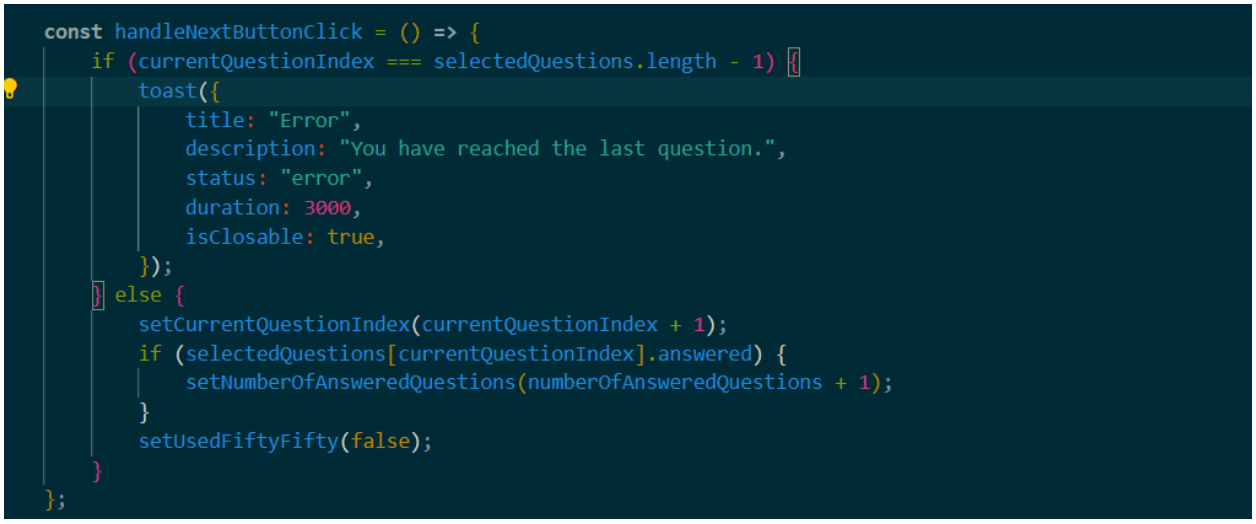


```
const showOptions = () => {
    const options = document.querySelectorAll('.option');
    options.forEach(option => {
        option.style.visibility = 'visible';
    });
};
```

I then call this function showOptions so:

The first line uses querySelectorAll to select all the elements with the class name ‘option’.

Then use foreach so that every element in the class will the visible until I use a lifeline like 50/50 to make them invisible.



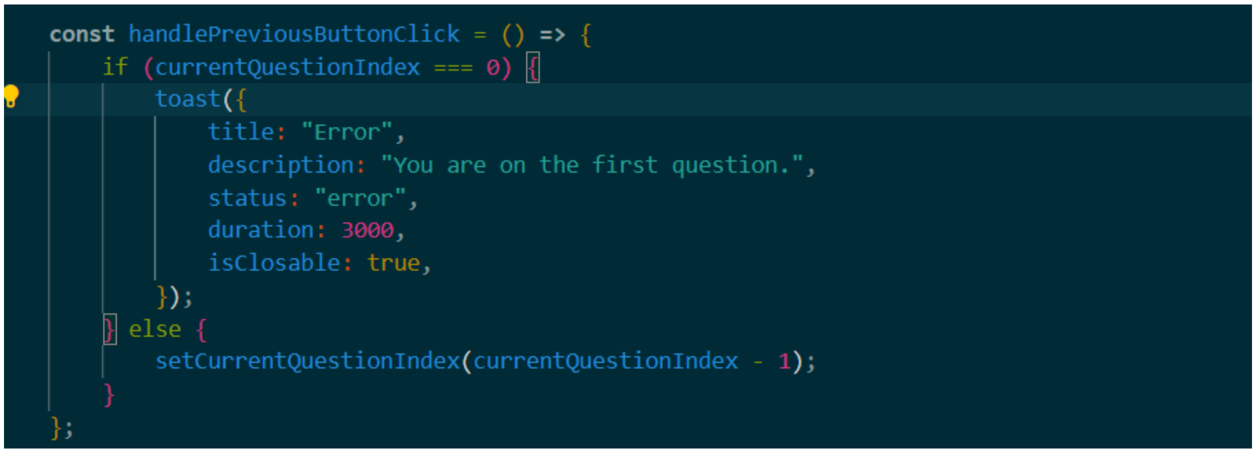
```
const handleNextButtonClick = () => {
    if (currentQuestionIndex === selectedQuestions.length - 1) {
        toast({
            title: "Error",
            description: "You have reached the last question.",
            status: "error",
            duration: 3000,
            isClosable: true,
        });
    } else {
        setCurrentQuestionIndex(currentQuestionIndex + 1);
        if (selectedQuestions[currentQuestionIndex].answered) {
            setNumberOfAnsweredQuestions(numberOfAnsweredQuestions + 1);
        }
        setUsedFiftyFifty(false);
    }
};
```

Here I am making the button to go to the next question:

The first line of the code checks if you are on the last question by checking if the currentQuestionIndex is equal to the selectedQuestion.length – 1 which is 14.

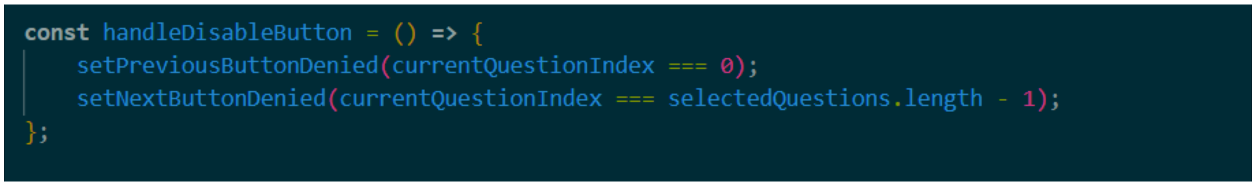
So, if you are not on the last question, you set the current question index to + 1 which means if you are on 0 in the array it will be 1 after you click the button changing the question.

Then checks if the question has been answered and if it has then it adds to the number of answered questions while also resetting the 50/50 lifeline.



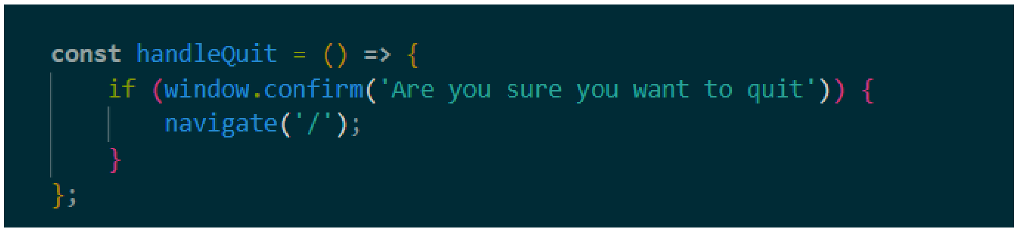
```
const handlePreviousButtonClick = () => {
    if (currentQuestionIndex === 0) {
        toast({
            title: "Error",
            description: "You are on the first question.",
            status: "error",
            duration: 3000,
            isClosable: true,
        });
    } else {
        setCurrentQuestionIndex(currentQuestionIndex - 1);
    }
};
```

Here is to go back a question it is the same code but just -1 to go back and first line checks if you are on the first question checking if currentQuestionIndex is 0.



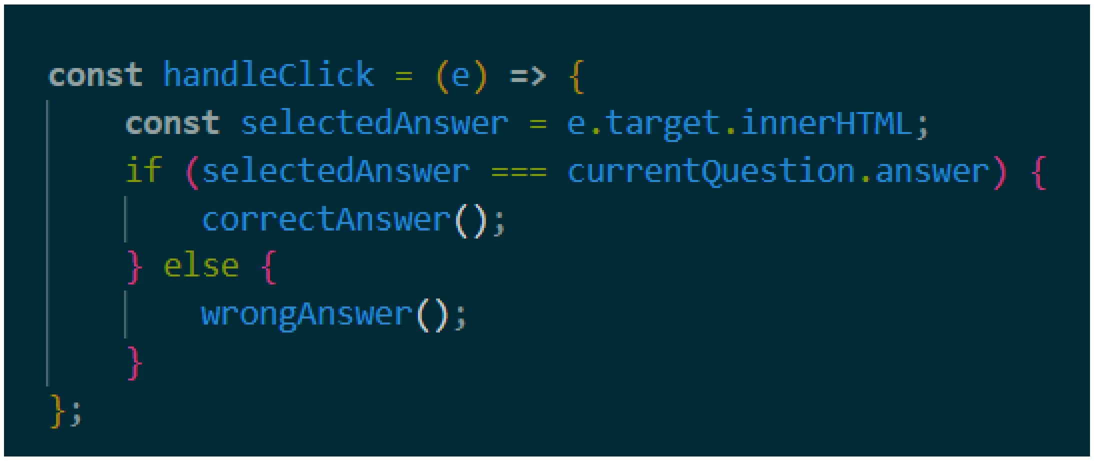
```
const handleDisableButton = () => {
    setPreviousButtonDenied(currentQuestionIndex === 0);
    setNextButtonDenied(currentQuestionIndex === selectedQuestions.length - 1);
};
```

This is to make sure the user is not allowed to go forward a question if you are in the next question and denies the user to go back a question if you are on the first question.



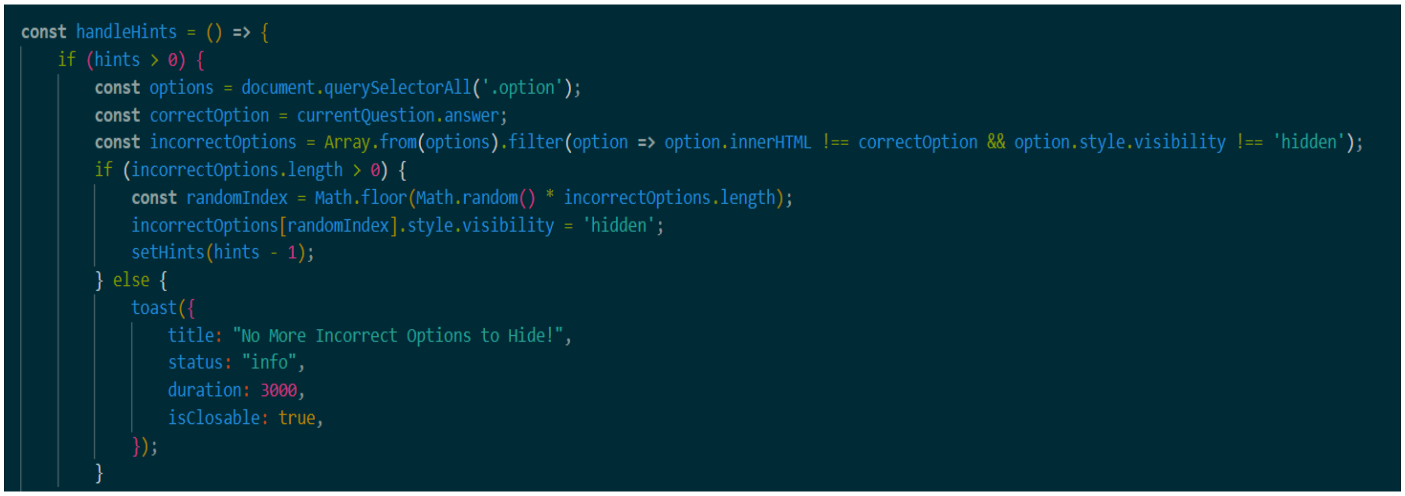
```
const handleQuit = () => {
    if (window.confirm('Are you sure you want to quit')) {
        navigate('/');
    }
};
```

This function is for the quit button so if you click the button, you will get this pop up that asks are you want to quit and if you click okay it will navigate you back to the home page.



```
const handleClick = (e) => {
    const selectedAnswer = e.target.innerHTML;
    if (selectedAnswer === currentQuestion.answer) {
        correctAnswer();
    } else {
        wrongAnswer();
    }
};
```

This is to answer the question it I use e.target.innerHTML to so whatever the user clicks will be used as the selectedAnswer it will then check if the answer is correct or wrong calling the correctAnswer or wrongAnswer functions.



```
const handleHints = () => {
    if (hints > 0) {
        const options = document.querySelectorAll('.option');
        const correctOption = currentQuestion.answer;
        const incorrectOptions = Array.from(options).filter(option => option.innerHTML !== correctOption && option.style.visibility !== 'hidden');
        if (incorrectOptions.length > 0) {
            const randomIndex = Math.floor(Math.random() * incorrectOptions.length);
            incorrectOptions[randomIndex].style.visibility = 'hidden';
            setHints(hints - 1);
        } else {
            toast({
                title: "No More Incorrect Options to Hide!",
                status: "info",
                duration: 3000,
                isClosable: true,
            });
        }
```

This is the code to use hints to hide some of the incorrect answers if you don't know the answer and you only have 5 hints.

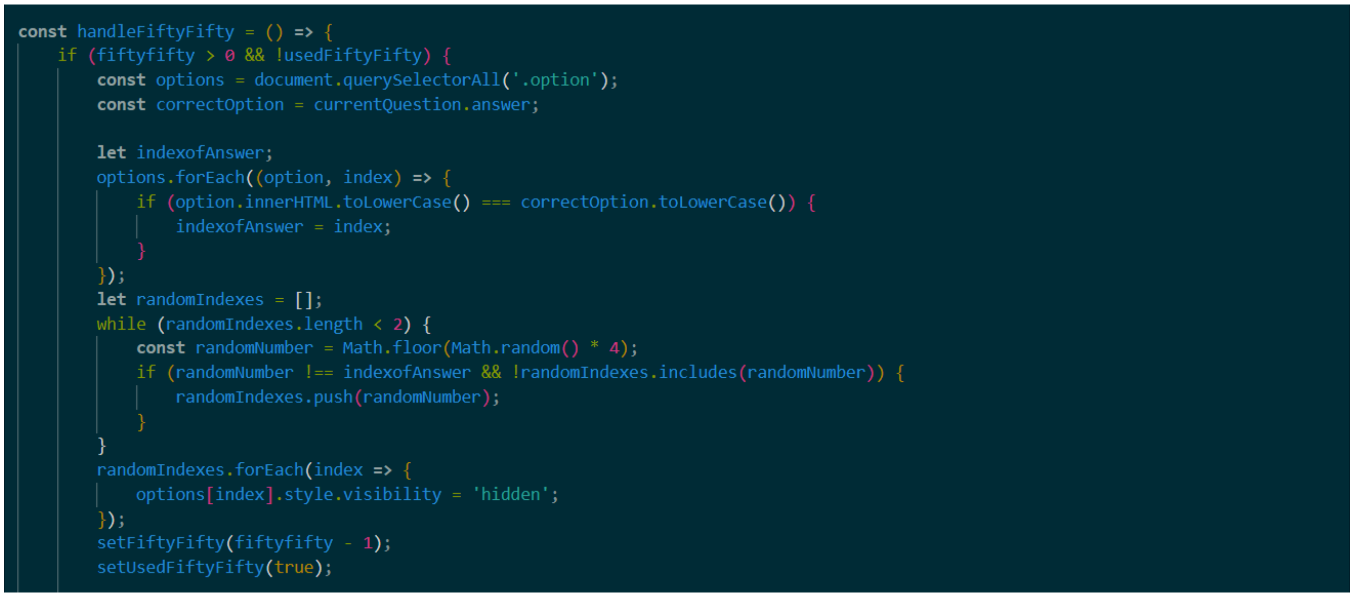
Once again, I use query Selector all to select all the elements in the option class.

I then get the correct answer from the array and assign it to correct option.

Then I filter out the correctOption that I just got and filter out all the answers that have already been hidden and use them in the incorrectOption.

Then if there are more than 0 incorrectOptions then it will randomly pick one of the available incorrectOptions and make it hidden.

Then it sets the hints to be -1 so everytime you use a hint the number of hints you have will go down.



```
const handleFiftyFifty = () => {
    if (fiftyfifty > 0 && !usedFiftyFifty) {
        const options = document.querySelectorAll('.option');
        const correctOption = currentQuestion.answer;

        let indexofAnswer;
        options.forEach((option, index) => {
            if (option.innerHTML.toLowerCase() === correctOption.toLowerCase()) {
                indexofAnswer = index;
            }
        });
        let randomIndexes = [];
        while (randomIndexes.length < 2) {
            const randomNumber = Math.floor(Math.random() * 4);
            if (randomNumber !== indexofAnswer && !randomIndexes.includes(randomNumber)) {
                randomIndexes.push(randomNumber);
            }
        }
        randomIndexes.forEach(index => {
            options[index].style.visibility = 'hidden';
        });
        setFiftyFifty(fiftyfifty - 1);
        setUsedFiftyFifty(true);
```

This is the 50/50 code so when you are also stuck you can get rid of 50% of the incorrect answers and leaves 2 but you can only use it 2 times.

First it checks if you have more than 0 50/50 and if you have any 50/50 available.

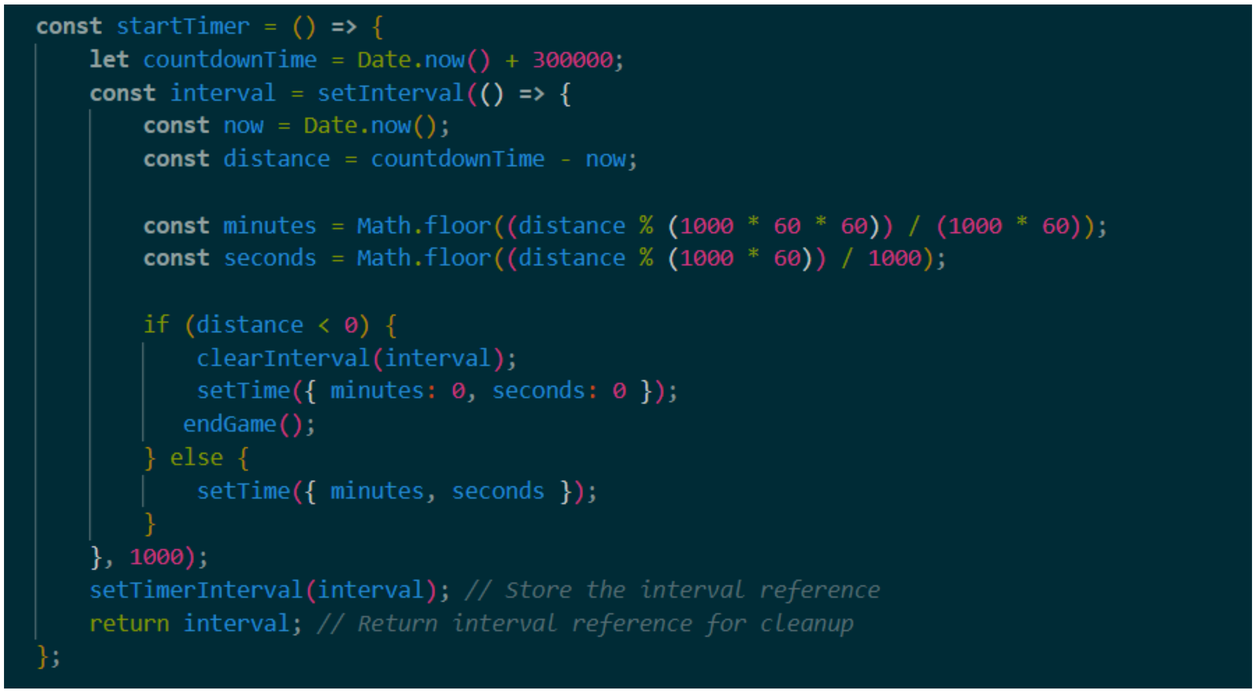
Once again, I use query Selector all to select all the elements in the option class.

It finds the correctOptions again then it finds the index of the correct option and that is stored in the indexOfAnswer variable.

Once again it creates, and empty array called randomIndex and then it will populate the randomIndex with 2 random numbers that are not the index of the answer.

It then hides two of the random indexes in the randomIndex so the user can’t see 2 of the answers.

Finally, it sets 50/50 to -1 which leaves you with only one left and sets used50/50 to true.



```javascript
const startTimer = () => {
    let countdownTime = Date.now() + 300000;
    const interval = setInterval(() => {
        const now = Date.now();
        const distance = countdownTime - now;

        const minutes = Math.floor((distance % (1000 * 60 * 60)) / (1000 * 60));
        const seconds = Math.floor((distance % (1000 * 60)) / 1000);

        if (distance < 0) {
            clearInterval(interval);
            setTime({ minutes: 0, seconds: 0 });
           endGame();
        } else {
            setTime({ minutes, seconds });
        }
    }, 1000);
    setTimerInterval(interval); // Store the interval reference
    return interval; // Return interval reference for cleanup
};
```

This to add the timer for the quiz so the way I have it is so that you have 5 minutes to complete the quiz, here is the breakdown:

First, I set the countdown time to be 300,000 which is 5 minutes. I then create an interval so every second it runs the callback function.

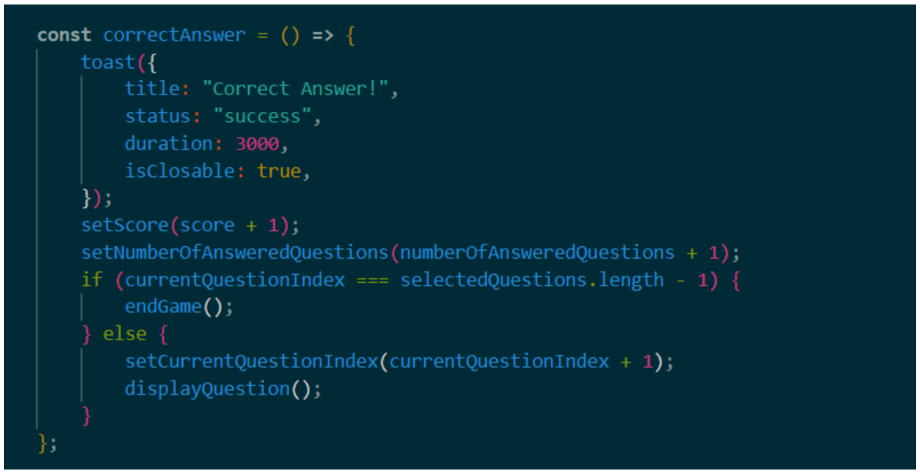
The callback function calculates how much time till the countdown is over by subtracting the countdowntime to the time now and assigns it to 'distance'.

I then calculate the time into minute and seconds using Math.floor and the distance I got earlier.

Then if distance is less than 0 it means the timer is over and you are out of time, it then clears the interval to stop the timer, then sets the time to 0 minutes and 0 seconds to show the timer is over. Then it runs the end game function which I will explain later.

Else if the distance isn't 0 and the timer is still going it will update time to show the calculated minutes and seconds.

It will then setTimeInterval to interval to ensure that the interval is cleared.

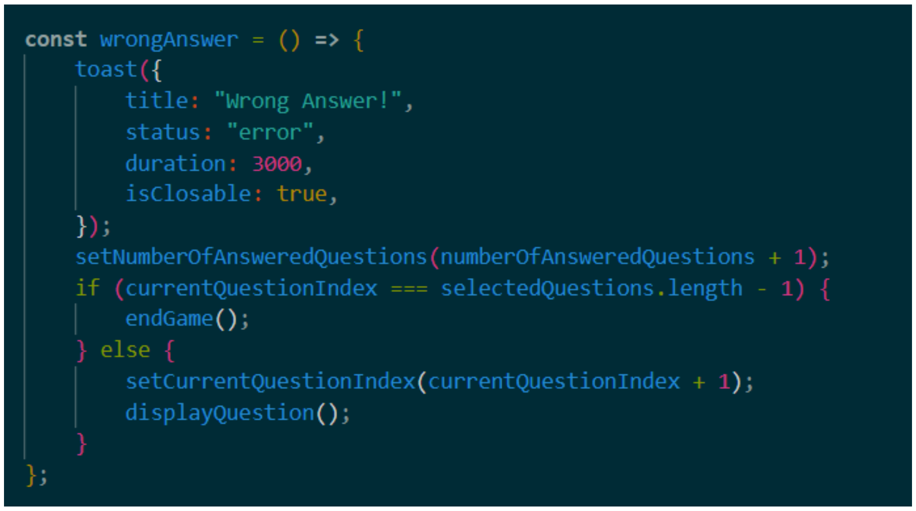


```javascript
const correctAnswer = () => {
    toast({
        title: "Correct Answer!",
        status: "success",
        duration: 3000,
        isClosable: true,
    });
    setScore(score + 1);
    setNumberOfAnsweredQuestions(numberOfAnsweredQuestions + 1);
    if (currentQuestionIndex === selectedQuestions.length - 1) {
        endGame();
    } else {
        setCurrentQuestionIndex(currentQuestionIndex + 1);
        displayQuestion();
    }
};
```

This is the that checks if the answer is correct.

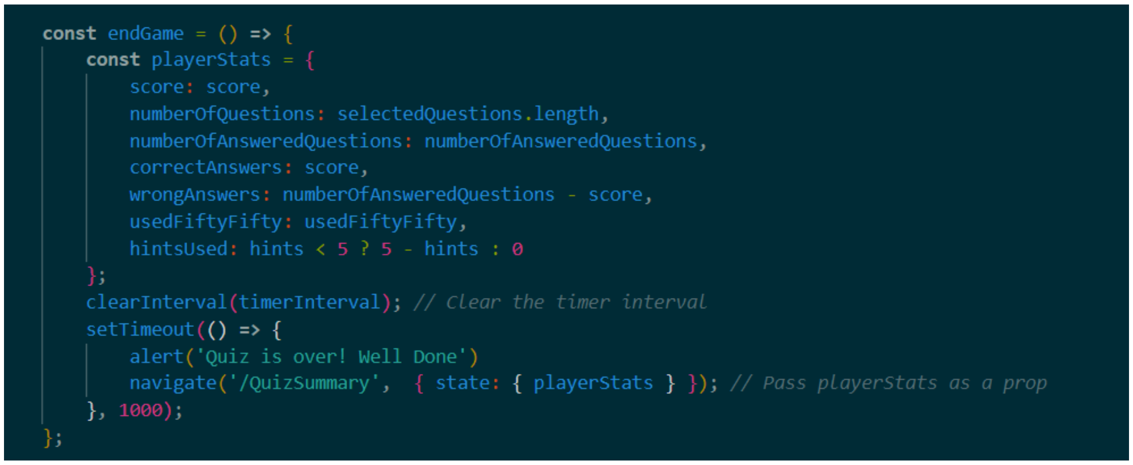
First it updates the score to score + 1 if you have answered the question correctly. Then it sets the number of answered questions to number of answered questions + 1 meaning that if the number of answered questions is 1 then it will go to 2.

I will then check if you are on the last question and if you are it will run the endgame function, I will explain later. But if there are still more questions it will display the questions remaining.



```javascript
const wrongAnswer = () => {
    toast({
        title: "Wrong Answer!",
        status: "error",
        duration: 3000,
        isClosable: true,
    });
    setNumberOfAnsweredQuestions(numberOfAnsweredQuestions + 1);
    if (currentQuestionIndex === selectedQuestions.length - 1) {
        endGame();
    } else {
        setCurrentQuestionIndex(currentQuestionIndex + 1);
        displayQuestion();
    }
};
```

This the WrongAnswer code that does the same thing as above but just does it for the wrong answers this time.



```
const endGame = () => {
    const playerStats = {
        score: score,
        numberOfQuestions: selectedQuestions.length,
        numberOfAnsweredQuestions: numberOfAnsweredQuestions,
        correctAnswers: score,
        wrongAnswers: numberOfAnsweredQuestions - score,
        usedFiftyFifty: usedFiftyFifty,
        hintsUsed: hints < 5 ? 5 - hints : 0
    };
    clearInterval(timerInterval); // Clear the timer interval
    setTimeout(() => {
        alert('Quiz is over! Well Done')
        navigate('/QuizSummary',  { state: { playerStats } }); // Pass playerStats as a prop
    }, 1000);
};
```

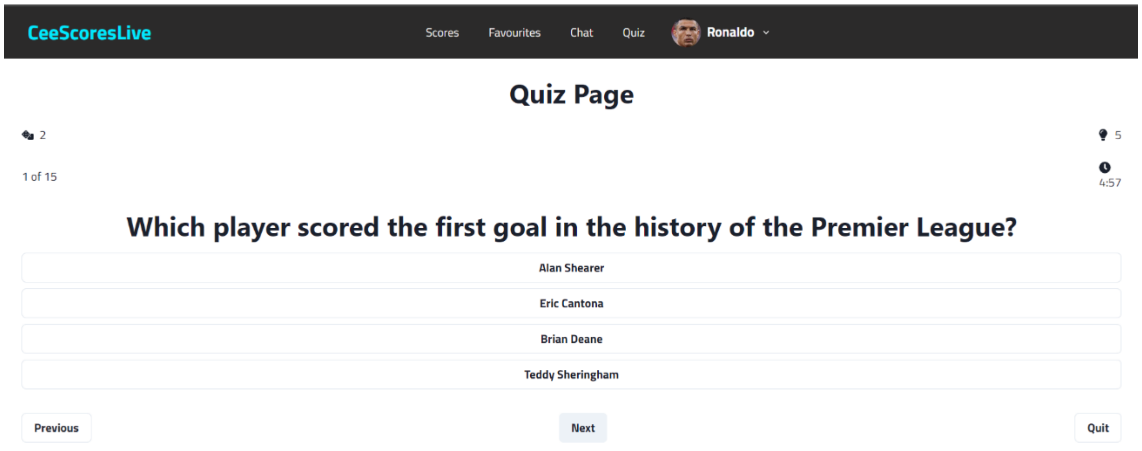
This is the code for the endgame I had called multiple times earlier.

This code will get all the players statistics from the quiz they took and store it into the playerStats variable. Then it will clear the interval timer that I had spoken about earlier.

When the timeout or the quiz is over you will get an alert telling you that the quiz is over and will navigate you to the quiz summary page where I passed the playerStats to as a prop so I can display the statistics the player got from the quiz.

I used this video to understand/learn how to build quiz page and merged it with my code to get it to work with my web application [21].

Results



Quiz Page

1 of 15

Which player scored the first goal in the history of the Premier League?

Alan Shearer

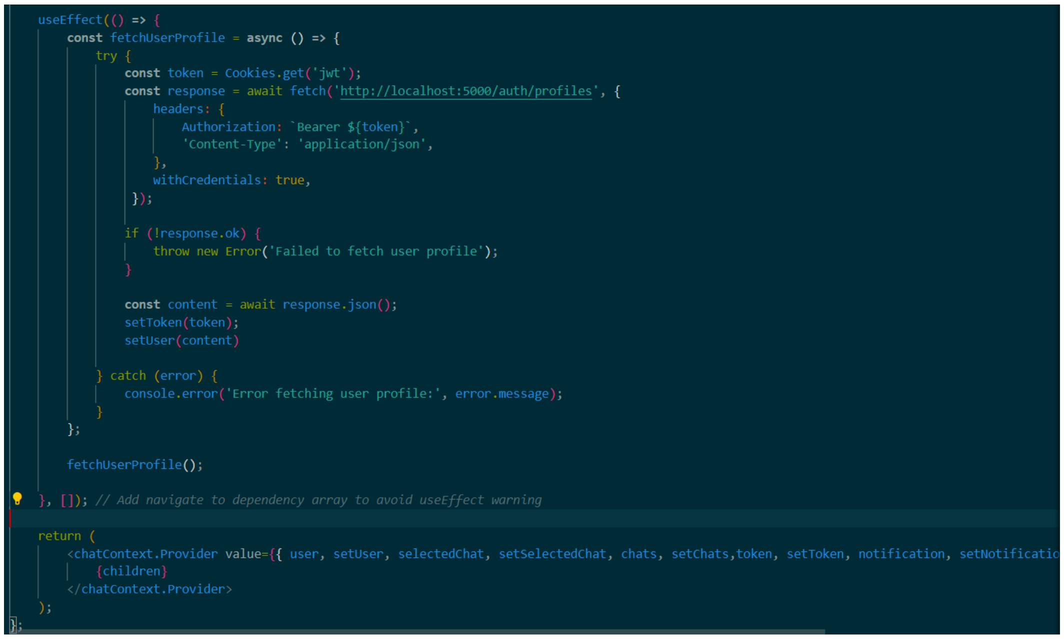
Eric Cantona

Brian Deane

Teddy Sheringham

7.11 Additional Code/Features

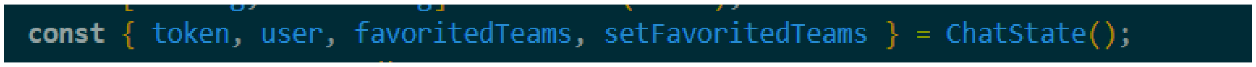
7.11.1 Context Hook



```
useEffect(() => {
    const fetchUserProfile = async () => {
        try {
            const token = Cookies.get('jwt');
            const response = await fetch('http://localhost:5000/auth/profiles', {
                headers: {
                    Authorization: `Bearer ${token}`,
                    'Content-Type': 'application/json',
                },
                withCredentials: true,
            });

            if (!response.ok) {
                throw new Error('Failed to fetch user profile');
            }

            const content = await response.json();
            setToken(token);
            setUser(content)

        } catch (error) {
            console.error('Error fetching user profile:', error.message);
        }
    };

    fetchUserProfile();

}, []); // Add navigate to dependency array to avoid useEffect warning

return (
    <chatContext.Provider value={{ user, setUser, selectedChat, setSelectedChat, chats, setChats,token, setToken, notification, setNotificatio
        {children}
    </chatContext.Provider>
);
};
```

React Context is a way to manage state globally. It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone [12].

In this code the information that I decided to pass mainly was the token which I had previously stored in the Cookies so you can see that I used cookie.get to get the jwt token and set it to setToken and added it as one of the values that I wanted to pass to other parts of the project like this:



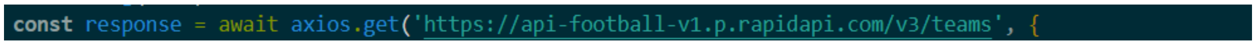
```
const { token, user, favoritedTeams, setFavoritedTeams } = ChatState();
```

You can also I'm also fetching to get the users information for the backend and storing it in setUser to use it in other parts of the code too like you can also see me add it as a value and call it in that code snippet.

7.11.2 Axios

I have mentioned axios a lot in my code here is a what axios is used for and why I used it:

Axios supports features such as interceptors, handling request and response headers, and handling different types of data, like JSON. It is widely used in web development to fetch data from APIs and interact with servers.[13]



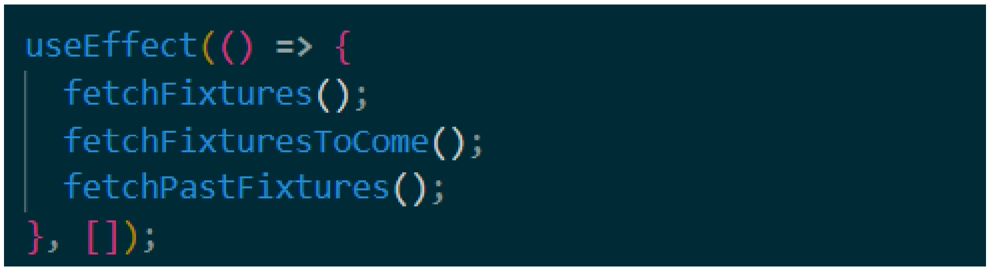
```
const response = await axios.get('https://api-football-v1.p.rapidapi.com/v3/teams', {
```

I used it in my code to handle the requests and to help the frontend communicate to the backend passing information back and forth.

7.11.3 UseEffect

I have mentioned useEffect a lot in my code here is a what useEffect is used for and why I used it:

React useEffect hook handles the effects of the dependency array. The useEffect Hook allows us to perform side effects on the components. fetching data, directly updating the DOM and timers are some side effects. It is called every time any state if the dependency array is modified or updated.[14]

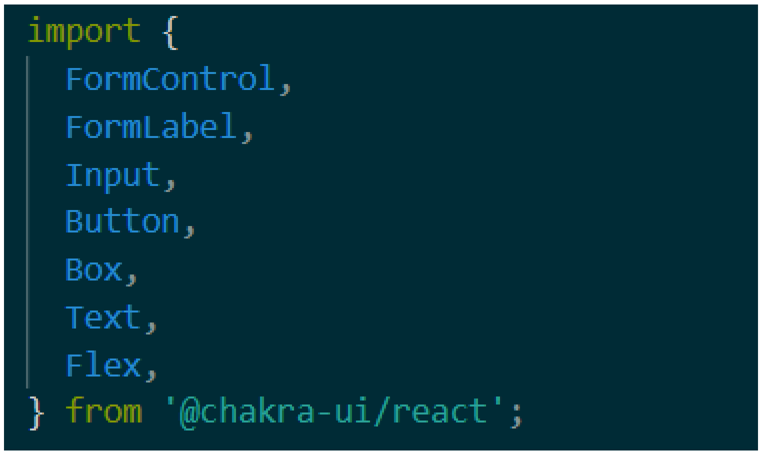


```
useEffect(() => {
  fetchFixtures();
  fetchFixturesToCome();
  fetchPastFixtures();
}, []);
```

I have explained why I used useEffect multiple times now, but I use it to update information/data whenever I made a change, or the user makes a change. In the code snippet I need it to update the information coming from the API to keep the fixtures up to date day to day.

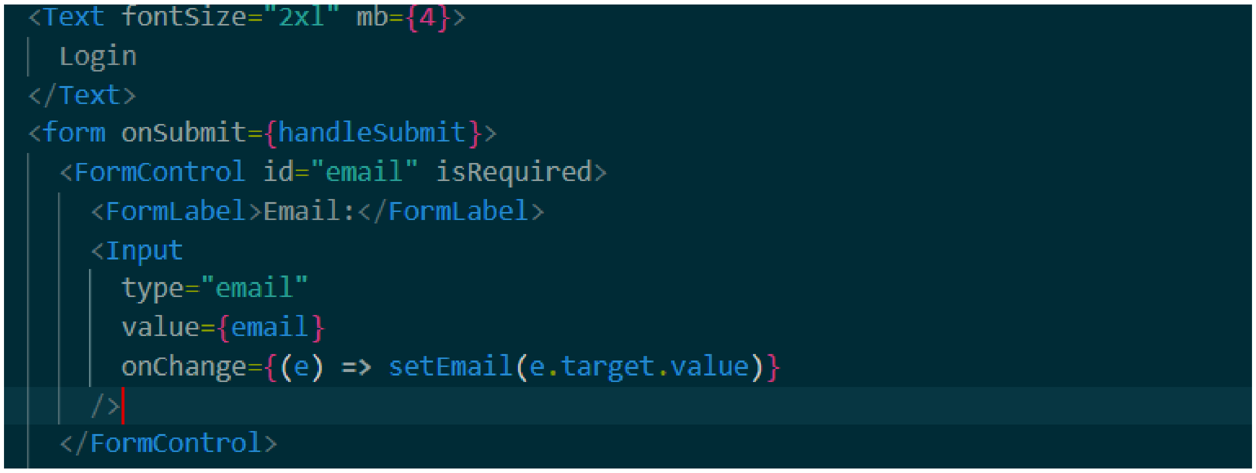
7.11.4 Chakra UI/Styling/Layout

I haven't really explained my styling so here is a quick example of what my styling looks like throughout my project using chakra UI:



```
import {
  FormControl,
  FormLabel,
  Input,
  Button,
  Box,
  Text,
  Flex,
} from '@chakra-ui/react';
```

Here is how you would import the styling components that you would like you can find out more about them here [15] that is the doc I used to become more familiar with chakra UI and there styling.

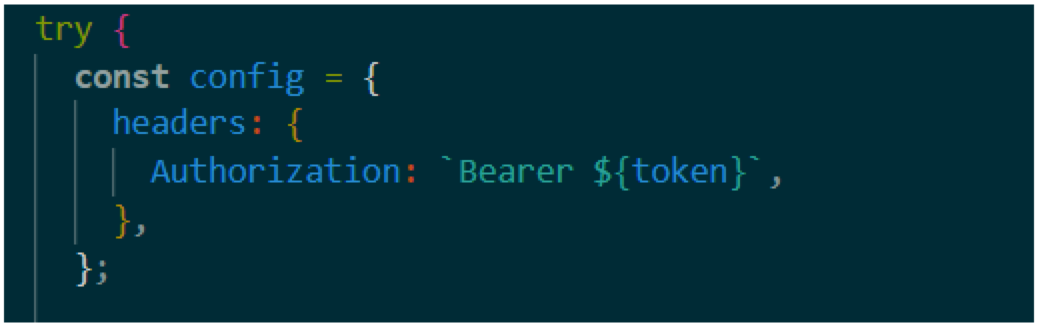


```
<Text fontSize="2xl" mb={4}>
  Login
</Text>
<form onSubmit={handleSubmit}>
  <FormControl id="email" isRequired>
    <FormLabel>Email:</FormLabel>
    <Input
      type="email"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
    />
  </FormControl>
```

Here is the code snippet of the login page I styled using chakra UI, I used the different components to make the page look how it did if you want to see it again you can have a look at figure 5 above. You can see me use FormLabel to label the form email and Input so that it where I would enter the email I would like to use and formControl to make sure that the user enters an email.

7.11.5 Authentication

I used an authentication a lot throughout my project to make sure that the request and all the information is authenticated and secure letting the authentication middleware deal with the rest which is explained in the backend code section.

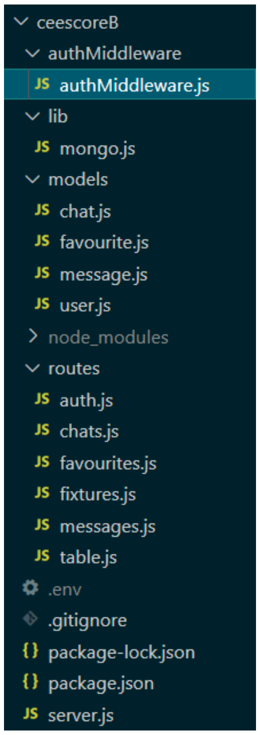


```
try {
  const config = {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  };
```

This is an example of it being used here I'm header called Authorization, and, in the authorization, I set the bearer to token, which is the JWT token that I used in the backend to make sure my application is secure and protected. So doing this will help verify the user and keep the information connected to that user secure.

8 Backend Code

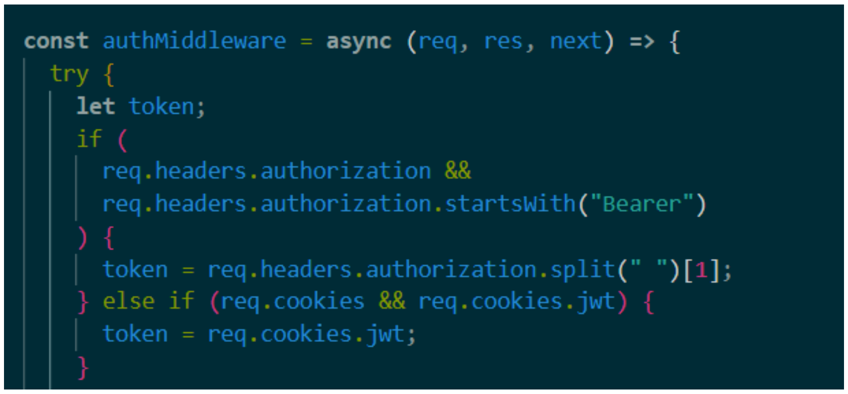
In the backend I created several key components to my code to give each of my frontend components functionality. I have a lot of folders which all have a different function that will play a role in storing information to the database and communicating with the frontend, looking at the screen show I will go from the top down explain what each component of the backend does:



```
ceescoreB
  authMiddleware
    authMiddleware.js
  lib
    mongo.js
  models
    chat.js
    favourite.js
    message.js
    user.js
  node_modules
  routes
    auth.js
    chats.js
    favourites.js
    fixtures.js
    messages.js
    table.js
  .env
  .gitignore
  package-lock.json
  package.json
  server.js
```

8.1 Authentication Middleware

The Auth middleware is used to secure/protect the routes in the backend so before the route can get accessed it will need to be checked so that the token created in the login and assigned to the user is the same token being used to access the routes. This will protect the routes and information from random people or hackers.



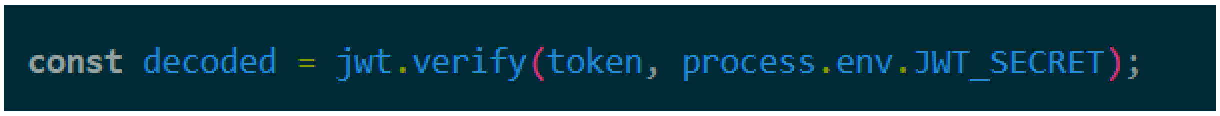
```
const authMiddleware = async (req, res, next) => {
  try {
    let token;
    if (
      req.headers.authorization &&
      req.headers.authorization.startsWith("Bearer")
    ) {
      token = req.headers.authorization.split(" ")[1];
    } else if (req.cookies && req.cookies.jwt) {
      token = req.cookies.jwt;
    }
```

In this code snippet it creates a variable called token.

It then has an if statement to check if an authorization header exists and if there is authorization that starts with ‘Bearer’.

If there is a header with that has a authorization that starts with bearer it splits the header so the bearer is in the “ ” and the token is where the one is. So, it looks like this (“bearer”) [token].

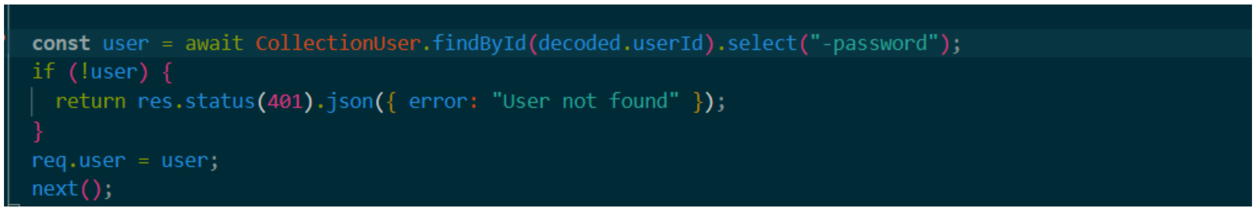
Then there is an else if statement that makes token the value of the req.cookies.jwt which is the jwt token stored in the cookies.



```
const decoded = jwt.verify(token, process.env.JWT_SECRET);
```

In this code I create a variable called decoded that will store the decoded token.

I then call jwt.verify that will verify the token and also the jwt secret key which is stored in the .env file which is hidden from everyone but me to keep the information secret/private.



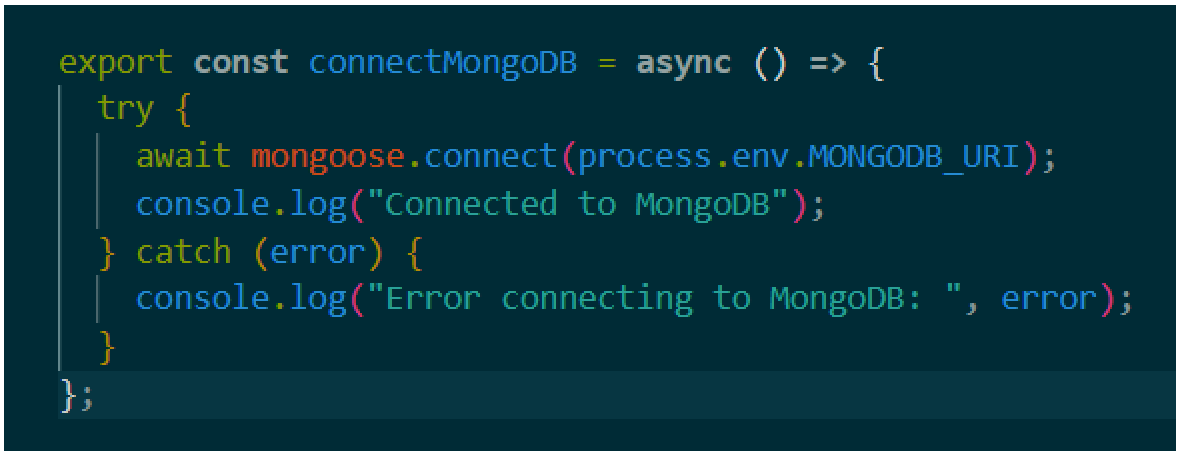
```
const user = await CollectionUser.findById(decoded.userId).select("-password");
if (!user) {
  return res.status(401).json({ error: "User not found" });
}
req.user = user;
next();
```

In this code snippet it finds the decoded.userId in the mongo database by the id and it will also remove the password so that is not used to keep the users password hidden and secure.

Then it does if there is not user it will return an error message saying User not found.

After that if there is a user in the database it will store it in the request that's how we have the req.user = user. Then it calls next(); which stop the middleware and let the user access the endpoint if the authentication goes smoothly.

8.2 Connection to MongoDB



```
export const connectMongoDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI);
    console.log("Connected to MongoDB");
  } catch (error) {
    console.log("Error connecting to MongoDB: ", error);
  }
};
```

This is code to connect mongoDb to this project so I can use it to store the information I would like to reuse or to be stored for that specific user.

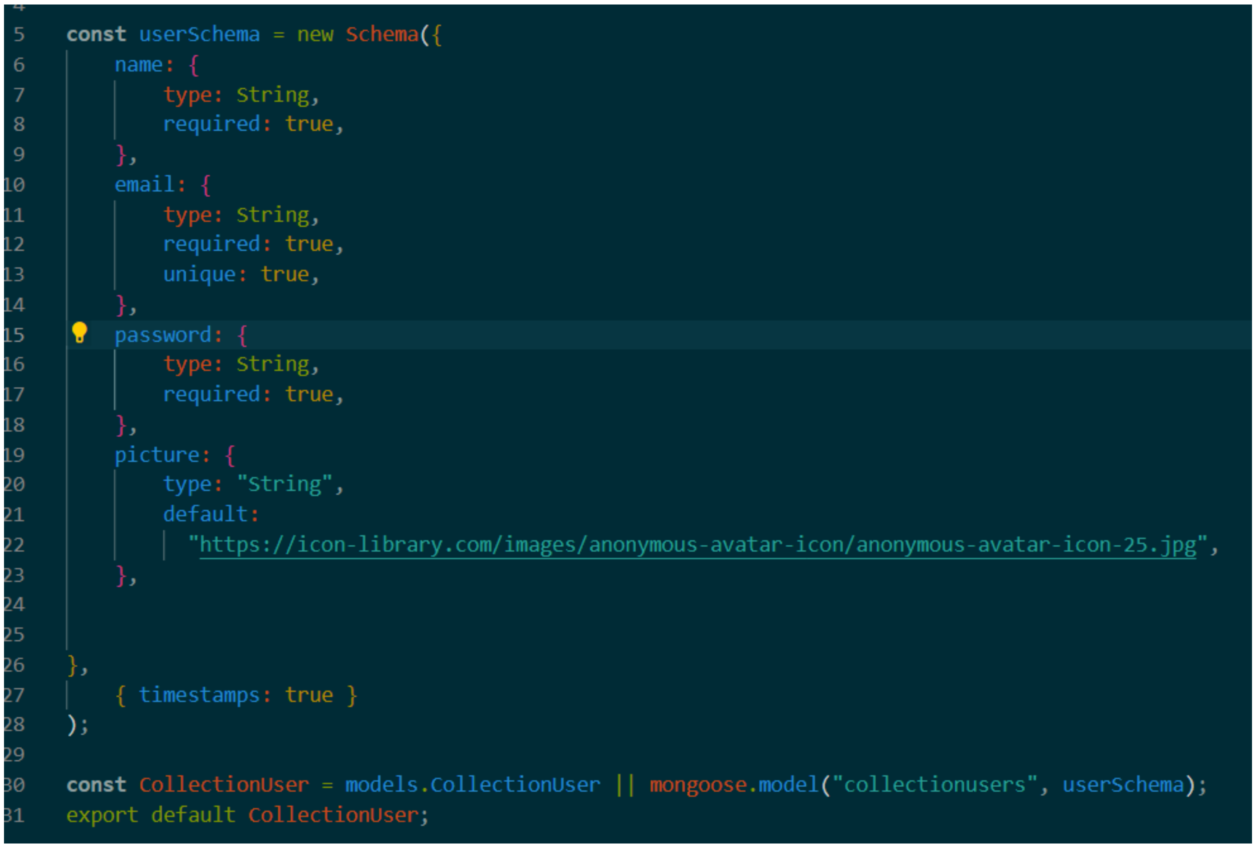
I use mongo.connect to connect to the mongodb compass URI that I stored in the .env folder like I did for the JWT token.

If it connects, I will get a connected to MongoDB message if it fails to connect to the MongoDB URI then I will get an error message.

8.3 Models

In MongoDB, data modelling refers to the process of designing and creating the structure of documents and collections that will be stored in the database. [16]

I have 4 models that I will break down:

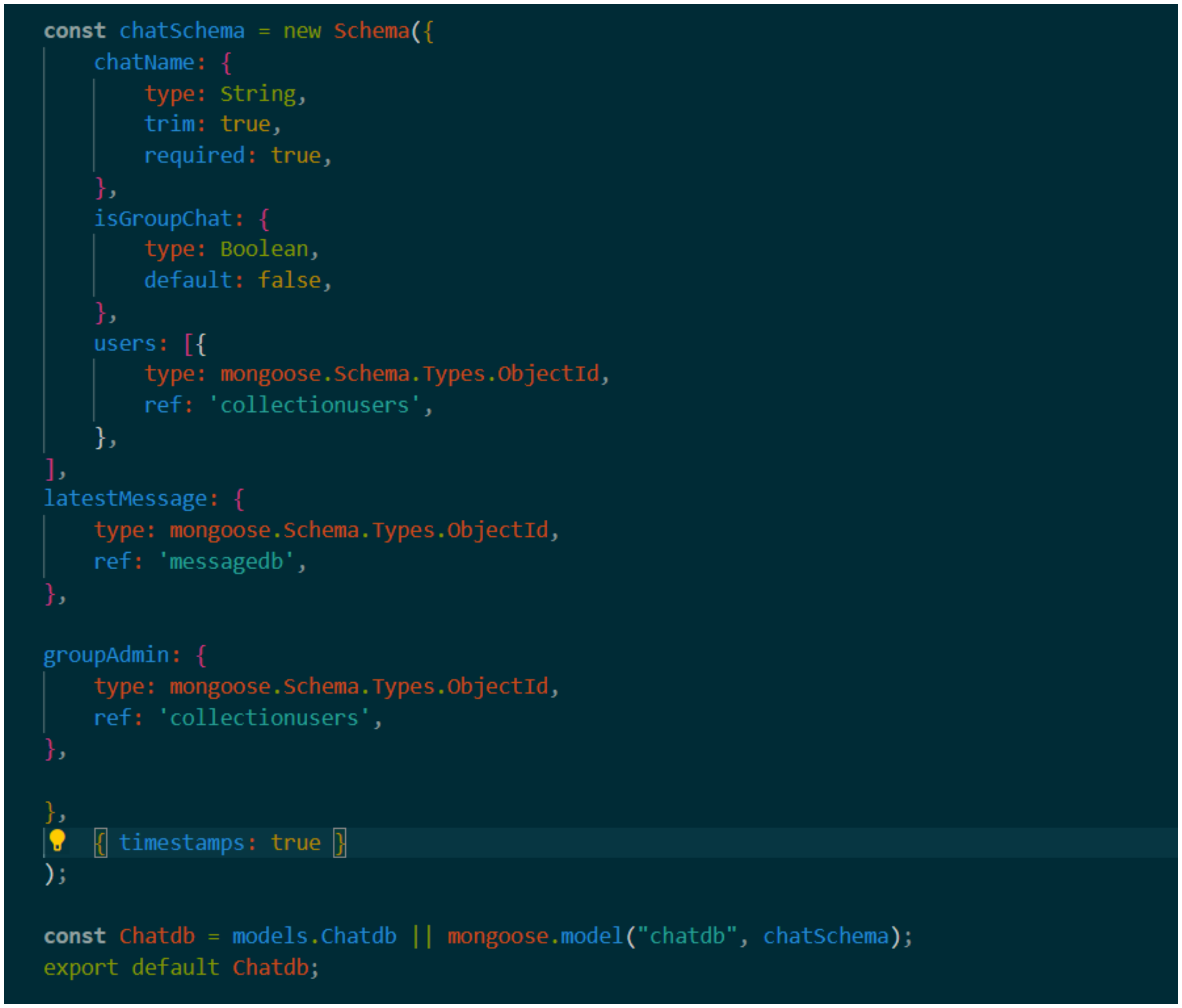


```
const userSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    picture: {
        type: "String",
        default:
          "https://icon-library.com/images/anonymous-avatar-icon/anonymous-avatar-icon-25.jpg",
    },


},
    { timestamps: true }
);

const CollectionUser = models.CollectionUser || mongoose.model("collectionusers", userSchema);
export default CollectionUser;
```

This code is the objects that will store in the database related to the user.

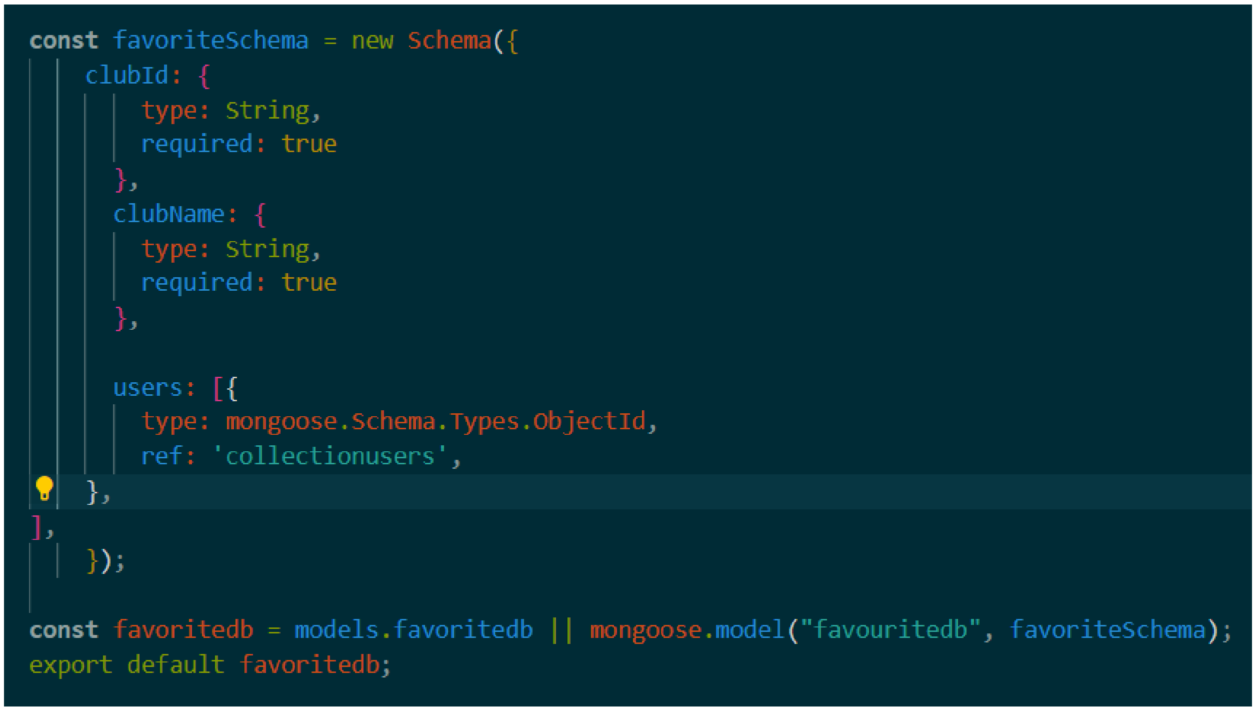
I will be storing the users name, email, password, picture, and timestamps and then it will create a new collection that will store that information and will allow you to reuse the information in that collection.

Next is the chat model:

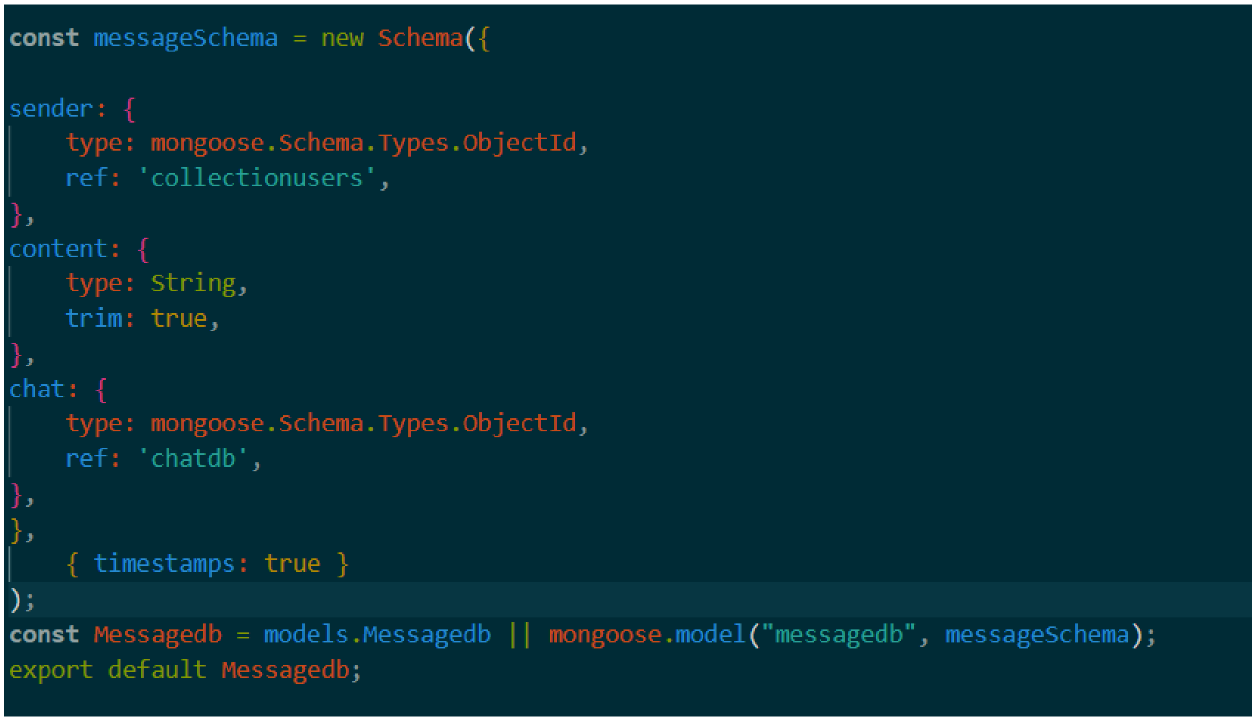


```
const chatSchema = new Schema({
    chatName: {
        type: String,
        trim: true,
        required: true,
    },
    isGroupChat: {
        type: Boolean,
        default: false,
    },
    users: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'collectionusers',
    },
],
latestMessage: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'messagedb',
},

groupAdmin: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'collectionusers',
},

},
    { timestamps: true }
);

const Chatdb = models.Chatdb || mongoose.model("chatdb", chatSchema);
export default Chatdb;
```

In the chat model I will be creating the chatName and isGroupChat objects and the users, latestMessage and groupAdmin will be imported from the other collections like the message collection and the user collection. Then it will store this all in the chat collection so I can use these objects in the backend code.



```
const favoriteSchema = new Schema({
    clubId: {
        type: String,
        required: true
      },
      clubName: {
        type: String,
        required: true
      },

      users: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'collectionusers',
    },
],
    });

const favoritedb = models.favoritedb || mongoose.model("favouritedb", favoriteSchema);
export default favoritedb;
```



```
const messageSchema = new Schema({

sender: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'collectionusers',
},
content: {
    type: String,
    trim: true,
},
chat: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'chatdb',
},
},
    { timestamps: true }
);
const Messagedb = models.Messagedb || mongoose.model("messagedb", messageSchema);
export default Messagedb;
```

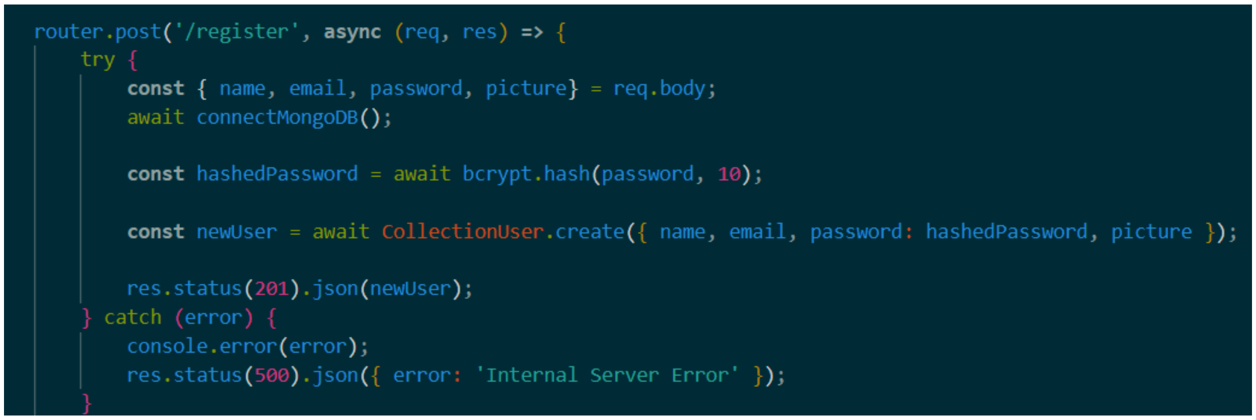
Then there is the favourite model and the messages model that also just store objects to the Database like the other 2 models I have explained.

8.4 Routes

Your back end uses its "routes" to decide which function will handle that HTTP request. Your back end "routes" are just a configuration that lets you define how requests map to functions in your back-end code [17].

I used routes to communicate the frontend and the backend so the frontend can pass data back and forth like in the frontend I get information from the API that's in the backend from the route and allow the frontend to send information like a user's name back to the backend.

8.4.1 Registration



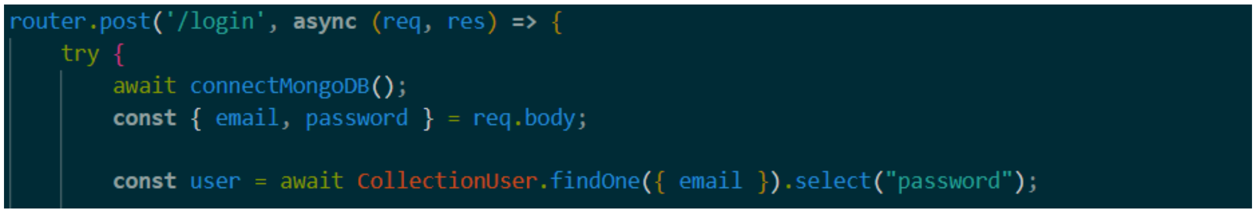
```
router.post('/register', async (req, res) => {
    try {
        const { name, email, password, picture} = req.body;
        await connectMongoDB();

        const hashedPassword = await bcrypt.hash(password, 10);

        const newUser = await CollectionUser.create({ name, email, password: hashedPassword, picture });

        res.status(201).json(newUser);
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
```

This is the code I'm using to create a user using the /register route that I called earlier in the frontend, I then define the variables that the user needs to send for him to create an account.

I then try to connect to MongoDB using connectMongoDB(); function. Next it hashes the password using bcrypt which a valuable tool to use to hash and store passwords [18]. Using the password the user created with 10 salt rounds.

I then created new objects in the collection for users filling in the information that was entered by the user filling the information.

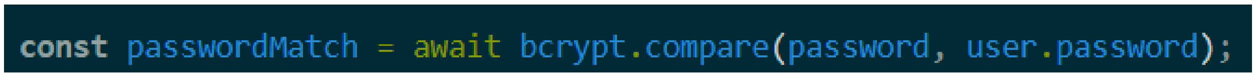
8.4.2 Login



```
router.post('/login', async (req, res) => {
    try {
        await connectMongoDB();
        const { email, password } = req.body;

        const user = await CollectionUser.findOne({ email }).select("password");
```

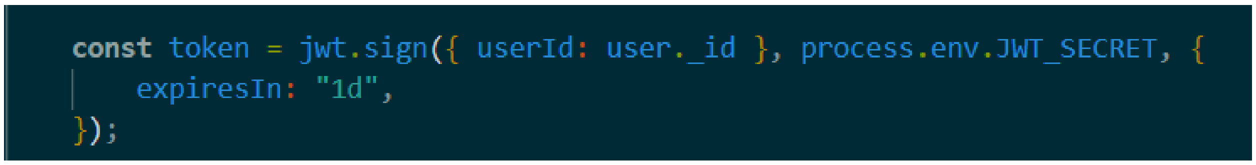
In the login code I also send the email and password through the request body to request that information from the frontend.

I then check the user collection to try and find an email that matches the one the user entered.



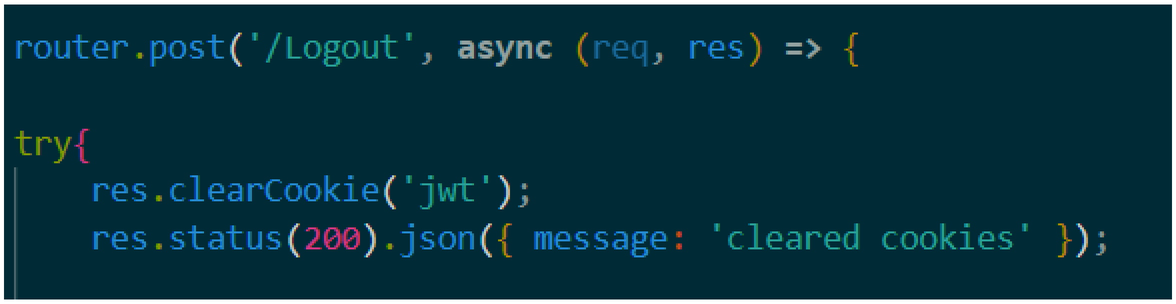
```
const passwordMatch = await bcrypt.compare(password, user.password);
```

Then checks if the passwords match using the password the user entered and the hashed password using bcrypt.compare.



```
    const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, {
        expiresIn: "1d",
    });
```

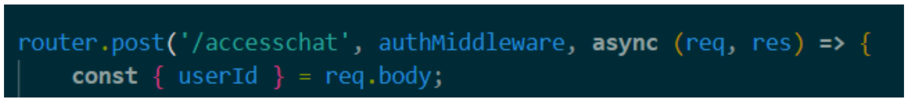
I then create the jwt token that I will use to authenticate the users. Using jwt.sign to sign token to the userId linking the token to the account, using the token key which is stored in the .env file. Then I make it, so the token expires after a day.



```
router.post('/Logout', async (req, res) => {

try{
    res.clearCookie('jwt');
    res.status(200).json({ message: 'cleared cookies' });
```

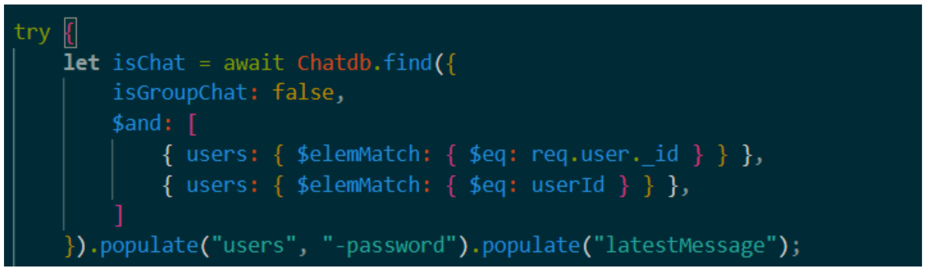
This is the logout page which clears the cookies which means that the user will not have an account/token/user stored in the cookies which means the user is logged out.

8.4.3 Chat



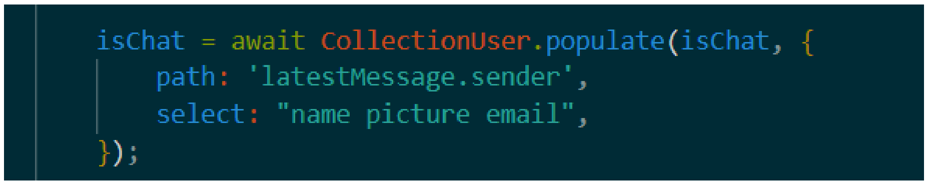
```
router.post('/accesschat', authMiddleware, async (req, res) => {
    const { userId } = req.body;
```

This is the code shows the route that allows the user to create or select chats with other users this route is protected by the middleware. It also requests the userId from the frontend.



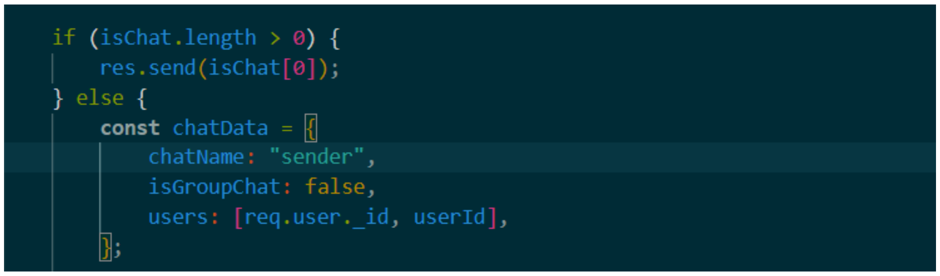
```
try {
    let isChat = await Chatdb.find({
        isGroupChat: false,
        $and: [
            { users: { $elemMatch: { $eq: req.user._id } } },
            { users: { $elemMatch: { $eq: userId } } },
        ]
    }).populate("users", "-password").populate("latestMessage");
```

Then I create a variable to find if it is not a group chat and then it will populate both the userId of the other user and the logged in user with information users without the password and the users' latest message.



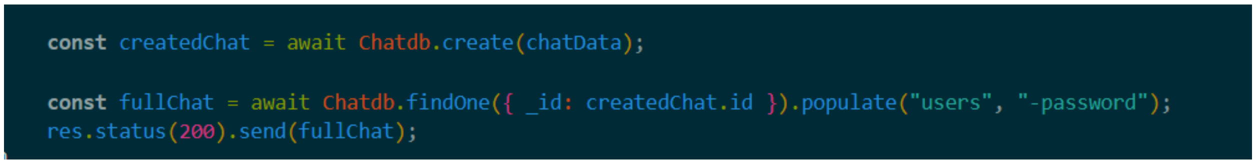
```
    isChat = await CollectionUser.populate(isChat, {
        path: 'latestMessage.sender',
        select: "name picture email",
    });
```

This code will populate ischat with the latest message from the sender and will also use bring over the other users/sender's, name picture and email.



```
    if (isChat.length > 0) {
        res.send(isChat[0]);
    } else {
        const chatData = {
            chatName: "sender",
            isGroupChat: false,
            users: [req.user._id, userId],
        };
```

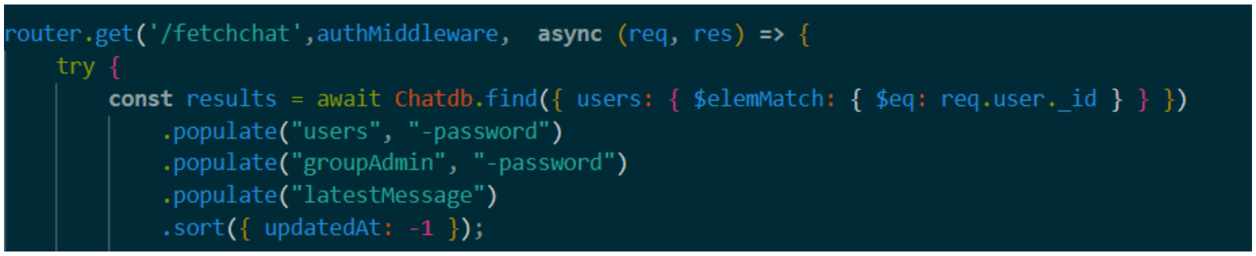
In this code it checks if there is anything in the isChat if there is then it will return that chat and if there is not it will store the chatName, if it a group chat and the 2 users in the chat in the chatData variable.



```
const createdChat = await Chatdb.create(chatData);

const fullChat = await Chatdb.findOne({ _id: createdChat.id }).populate("users", "-password");
res.status(200).send(fullChat);
```

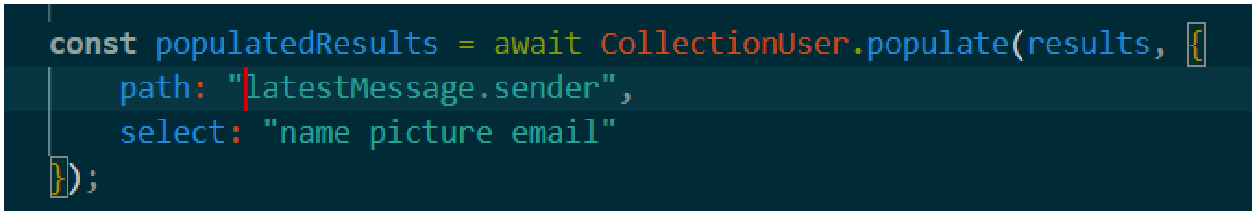
I will then create a new object in the chatDB collection using the information we stored in the chatData variable.

Then it will check the collection to find the id of the createdchat if it exists and then populate it with the users and not include the password in it.



```
router.get('/fetchchat',authMiddleware,  async (req, res) => {
    try {
        const results = await Chatdb.find({ users: { $elemMatch: { $eq: req.user._id } } })
            .populate("users", "-password")
            .populate("groupAdmin", "-password")
            .populate("latestMessage")
            .sort({ updatedAt: -1 });
```

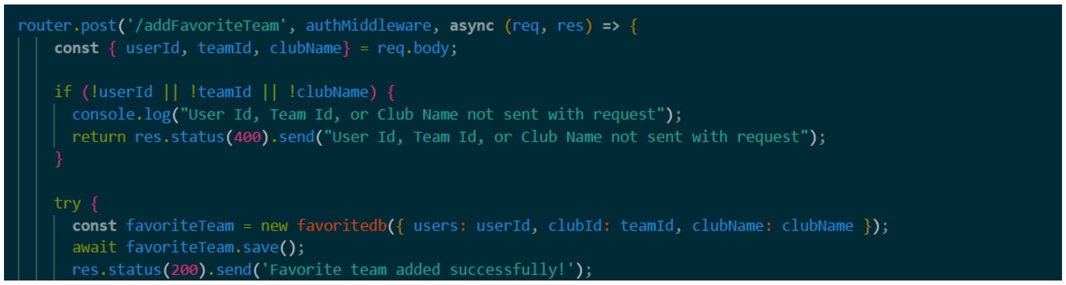
I use this get route to fetch the chats of the chats the user may have with the other users to do that it finds the user logged in and will populate the user's details, if it is a group chat or not, the latest message sent. Then use .sort to keep the chats updated.



```
const populatedResults = await CollectionUser.populate(results, {
    path: "latestMessage.sender",
    select: "name picture email"
});
```

This does the same thing as the chatData code I have explained earlier just populates the variable populatedResults with the sender's information.

8.4.4 Favourite

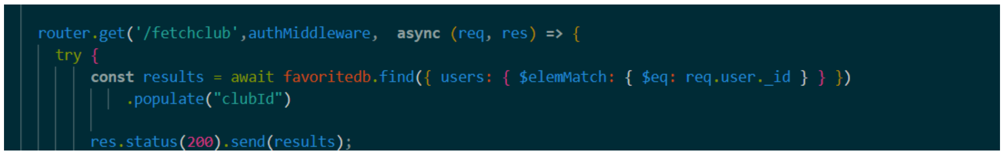


```
router.post('/addFavoriteTeam', authMiddleware, async (req, res) => {
    const { userId, teamId, clubName} = req.body;

    if (!userId || !teamId || !clubName) {
      console.log("User Id, Team Id, or Club Name not sent with request");
      return res.status(400).send("User Id, Team Id, or Club Name not sent with request");
    }

    try {
      const favoriteTeam = new favoritedb({ users: userId, clubId: teamId, clubName: clubName });
      await favoriteTeam.save();
      res.status(200).send('Favorite team added successfully!');
```

This here is the code that allows the user to store their favourite team in the database and connect it to a user and protect that information with a middleware.

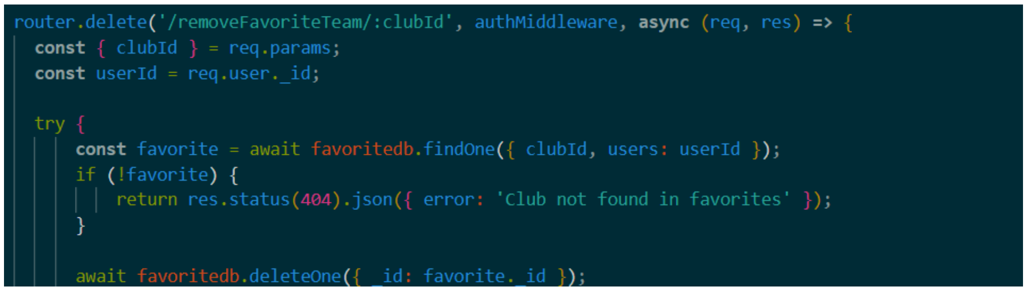
So, it requests the userId, teamId, and the clubname which is provided by the frontend then checks if it is missing any of the 3 is missing if it is missing it will send an error message.

Then it will create a new object in the favourite collection to store the 3 variables I got from the req.body.



```
  router.get('/fetchclub',authMiddleware,  async (req, res) => {
    try {
      const results = await favoritedb.find({ users: { $elemMatch: { $eq: req.user._id } } })
          .populate("clubId")

      res.status(200).send(results);
```

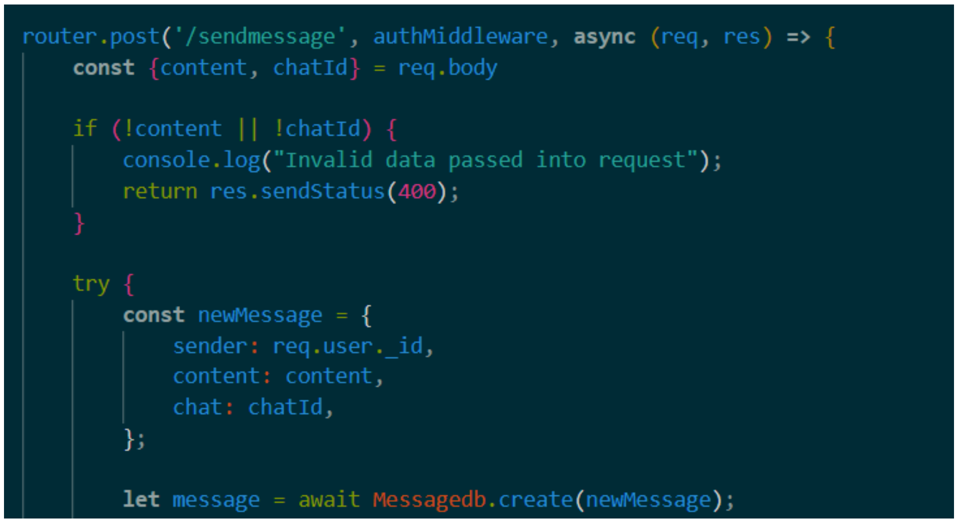
I then fetch all the clubs that I have saved in the database collection for the favourite. It checks the user that is logged in Id then populates that users clubId to then get the team by its Id.



```
router.delete('/removeFavoriteTeam/:clubId', authMiddleware, async (req, res) => {
  const { clubId } = req.params;
  const userId = req.user._id;

  try {
    const favorite = await favoritedb.findOne({ clubId, users: userId });
    if (!favorite) {
      return res.status(404).json({ error: 'Club not found in favorites' });
    }

    await favoritedb.deleteOne({ _id: favorite._id });
```

This is the code to delete a favourite team you don’t like anymore so you put the cludId in the params and define the logged in users so it checks for the clubId that was stored with that userId then it will delete that club from the users’ object.

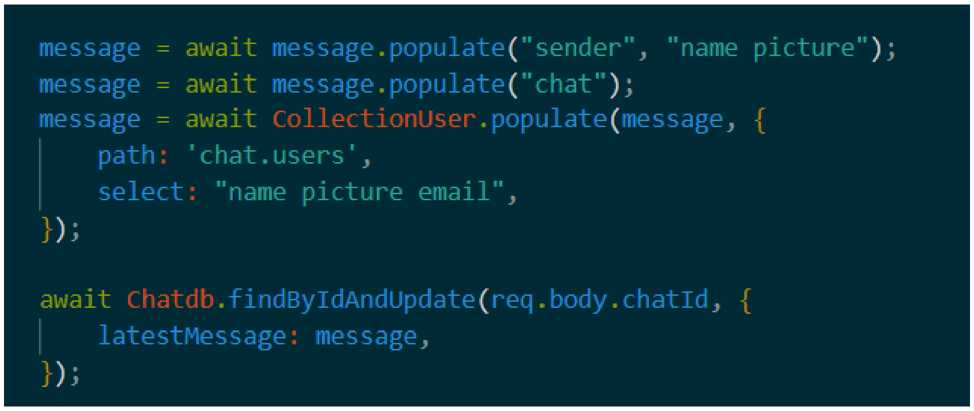
8.4.5 Messages



```
router.post('/sendmessage', authMiddleware, async (req, res) => {
    const {content, chatId} = req.body

    if (!content || !chatId) {
        console.log("Invalid data passed into request");
        return res.sendStatus(400);
    }

    try {
        const newMessage = {
            sender: req.user._id,
            content: content,
            chat: chatId,
        };

        let message = await Messagedb.create(newMessage);
```

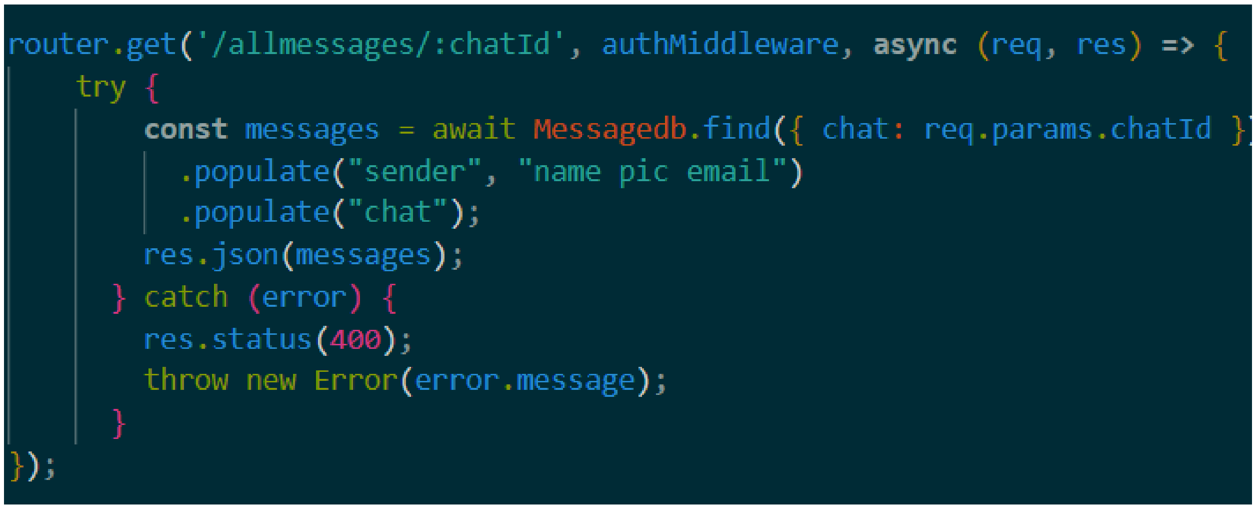
This the code I used to make a message. I request the content of the message and the chatId from the frontend.

Then I create a new message filling in the objects that I defined in the message model and storing it to the variable newMessage. I will then create a new document using the information in the newMessage variable.



```
message = await message.populate("sender", "name picture");
message = await message.populate("chat");
message = await CollectionUser.populate(message, {
    path: 'chat.users',
    select: "name picture email",
});

await Chatdb.findByIdAndUpdate(req.body.chatId, {
    latestMessage: message,
});
```

I then use the message variable that was used to create the document in the message collection and will populate the message with the sender's name, and picture from the other collections. It will populate the chat too. It will populate the users in the chat and take populate the name picture and email of the users. Then it will update the latest message

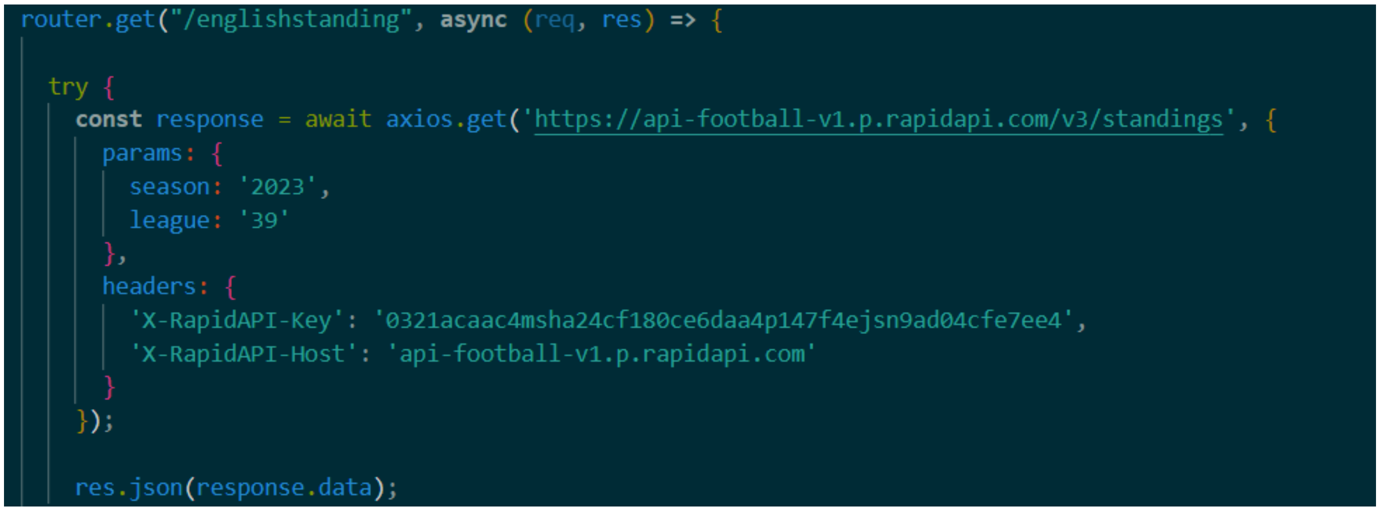


```
router.get('/allmessages/:chatId', authMiddleware, async (req, res) => {
    try {
        const messages = await Messagedb.find({ chat: req.params.chatId }
          .populate("sender", "name pic email")
          .populate("chat");
        res.json(messages);
      } catch (error) {
        res.status(400);
        throw new Error(error.message);
      }
});
```

This is the code I use to fetch the messages that are related to the chatId. And will populate the messages again like I explained before.

8.4.6 Table

This will be how I called the API from RapidAPI to get the information for the league table that is up to date I will only be explain 1 because all the other league table RapidAPI routes will be very similar.



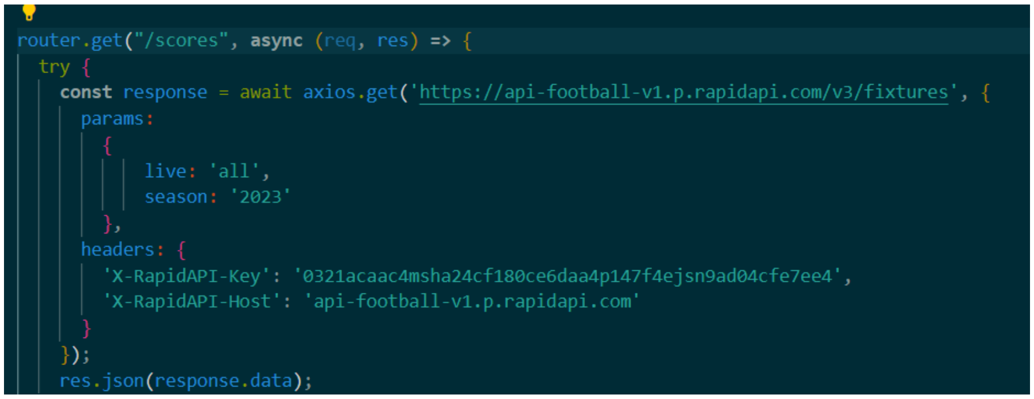
```
router.get("/englishstanding", async (req, res) => {

  try {
    const response = await axios.get('https://api-football-v1.p.rapidapi.com/v3/standings', {
      params: {
        season: '2023',
        league: '39'
      },
      headers: {
        'X-RapidAPI-Key': '0321acaac4msha24cf180ce6daa4p147f4ejsn9ad04cfe7ee4',
        'X-RapidAPI-Host': 'api-football-v1.p.rapidapi.com'
      }
    });

    res.json(response.data);
```

This is the how I used the API to get the live up to date premier league table.

I used axios.get which I explained earlier to get the information from the API endpoint using the params to define which season you want and what league you would like to see (league = 38 is the premier league). To show the league table you will just need to call the routes endpoint.

I did this for the top 5 leagues in football which is the Premier League, La Liga, Serie A, Bundesliga, and Ligue 1.

8.4.7 Fixtures

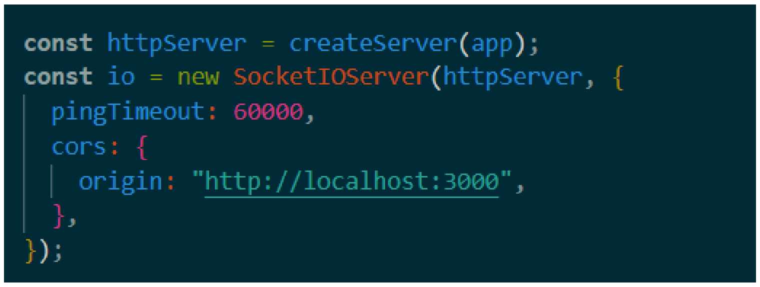


```
router.get("/scores", async (req, res) => {
  try {
    const response = await axios.get('https://api-football-v1.p.rapidapi.com/v3/fixtures', {
      params:
        {
          live: 'all',
          season: '2023'
        },
      headers: {
        'X-RapidAPI-Key': '0321acaac4msha24cf180ce6daa4p147f4ejsn9ad04cfe7ee4',
        'X-RapidAPI-Host': 'api-football-v1.p.rapidapi.com'
      }
    });
    res.json(response.data);
```

The Routes to get the fixtures is the same so I will show you an example so you can see the similarities of the fixtures code and table code the difference is just the params that I'm sending to the API to filter the information being sent to the website I do this to get the live fixtures, the past fixtures, and the fixtures to come.

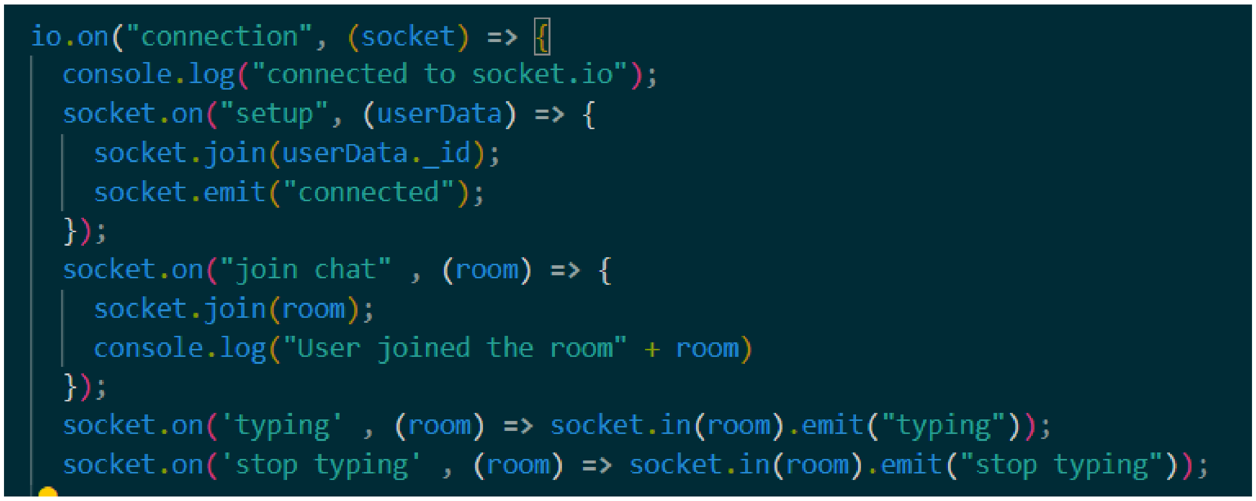
8.5 Server

In the server file I have a lot of important information that is the backbone of my website ill be explaining the important features found in my server starting with socket.io configuration:



```
const httpServer = createServer(app);
const io = new SocketIOServer(httpServer, {
  pingTimeout: 60000,
  cors: {
    origin: "http://localhost:3000",
  },
});
```

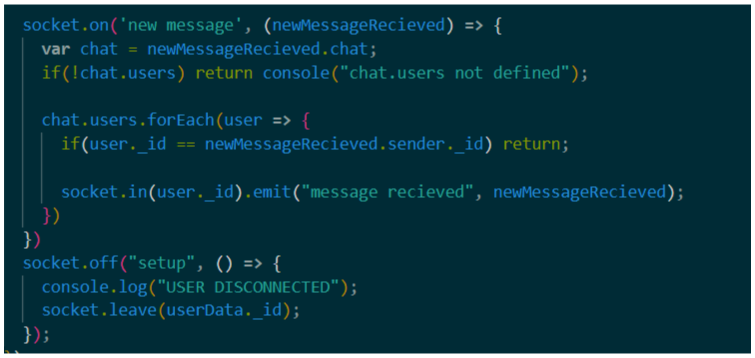
This code creates a http server for the socket.io to use as the socket server. It then uses that server to make the chat real time and store the server in the variable io. Then uses cors to connect the frontend port to the socket server.



```
io.on("connection", (socket) => {
  console.log("connected to socket.io");
  socket.on("setup", (userData) => {
    socket.join(userData._id);
    socket.emit("connected");
  });
  socket.on("join chat" , (room) => {
    socket.join(room);
    console.log("User joined the room" + room)
  });
  socket.on('typing' , (room) => socket.in(room).emit("typing"));
  socket.on('stop typing' , (room) => socket.in(room).emit("stop typing"));
```

Using the io variable that stores the socket server will wrap around the different events I explained throughout the frontend check the frontend for event explanations.

But when the user connects to the socket server you will get a message saying connected to socket.io, if the setup event is called it will go into that room using the userData Id provided so since the socket is in that room it then knows when someone is in a room so if a user joins the chat room it will print user joined the room and the room he joined. Then it also knows when the user is typing or not.



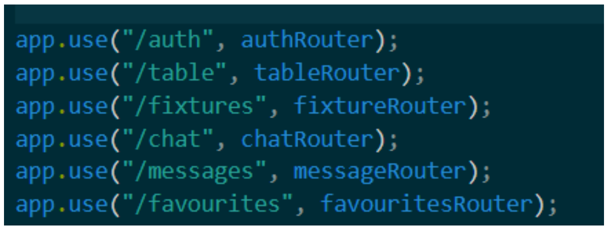
```
socket.on('new message', (newMessageRecieved) => {
  var chat = newMessageRecieved.chat;
  if(!chat.users) return console("chat.users not defined");

  chat.users.forEach(user => {
    if(user._id == newMessageRecieved.sender._id) return;

    socket.in(user._id).emit("message recieved", newMessageRecieved);
  })
})
socket.off("setup", () => {
  console.log("USER DISCONNECTED");
  socket.leave(userData._id);
});
```

This is the code to make the messages the user sends to be real time using socket.io.

So, when a user sends a message it will tell the socket sever through the event. Then it checks if the userId is the same as the id of the user sending the new message if it is then it returns but if its not the event message received will run and that will send the message to the user intended.

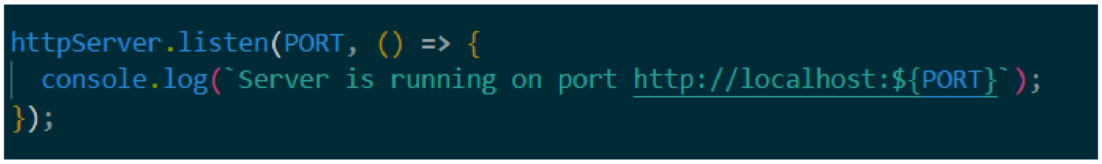
Finally, socket.off happens when a socket is disconnected from the setup that was turned on earlier and when that happens it will print User disconnected and leave the private room that the user had entered.

8.5.1 Server Routes



```
app.use("/auth", authRouter);
app.use("/table", tableRouter);
app.use("/fixtures", fixtureRouter);
app.use("/chat", chatRouter);
app.use("/messages", messageRouter);
app.use("/favourites", favouritesRouter);
```

This code uses app.use to make sure that all the routes is handled by a router, for example /auth is handled by authRouter which makes it able to use the endpoints in the routes codes and communicate with the frontend.



```
httpServer.listen(PORT, () => {
  console.log(`Server is running on port http://localhost:${PORT}`);
});
```

This code is how I make the server connect the port that is stored in the .env file and if it connects it will console and tell you the server is running.

9 Ethics

It was a major factor in while creating my website so have a high ethical standard throughout my website so to do that, I made sure that data security is a priority of knowing hearing all the stories and news about data breaches, data leaks and the selling of user's information from websites like Facebook said here [18] In the 2010s, personal data belonging to millions of Facebook users was collected without their consent by British consulting firm Cambridge Analytica, predominantly to be used for political advertising[18].

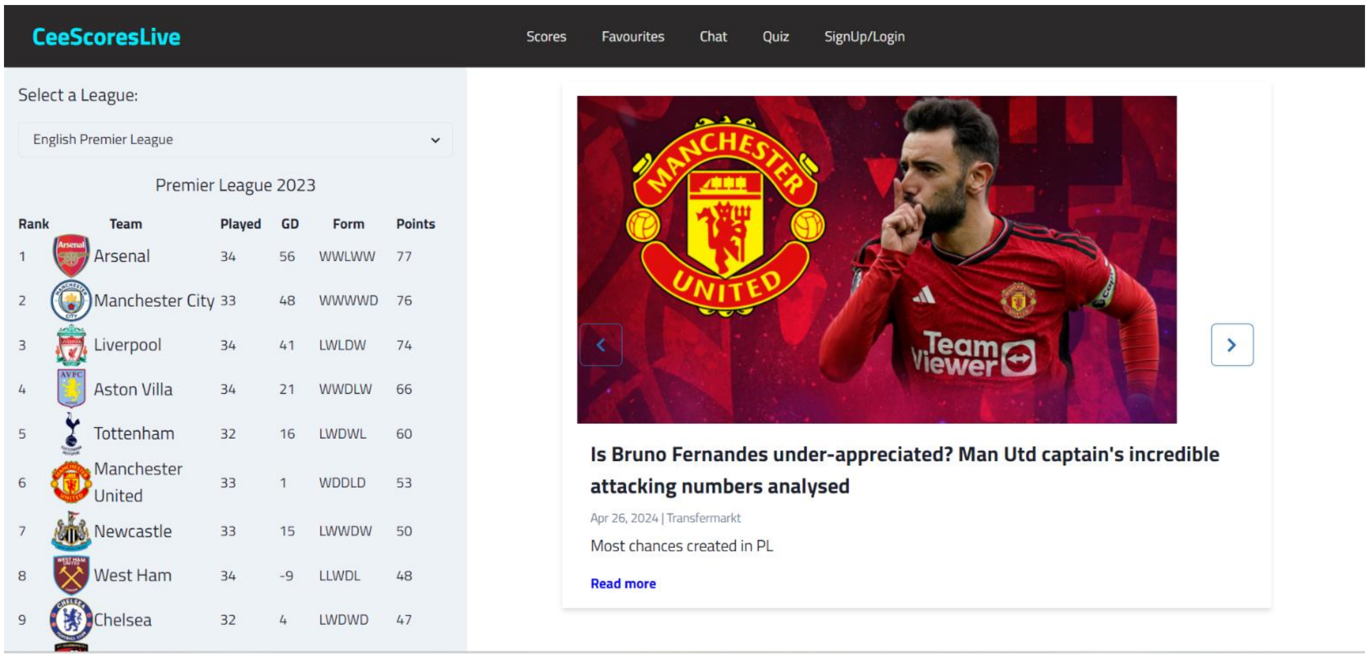
So, in my website I went through a lot to make sure that user's information was secure so it would not be able to be access by just anyone the first thing that I did was make sure the user's password was hashed so that only the user will be able to know the password cause even in the database the password will be hashed.

Next, I made a JWT token to each user and used an authentication middleware so before you can on the user the with that token can access the information that the user has created like the chats or the favourite team, he had favourited so not anyone can access the information making sure the user is protected.

Those are the 2 keyways I have made the data of the user secure and protected from other users trying to access his account data.

10 Conclusion

Making this project, I believe I have completed everything that I had set out to do when I started this journey of creating a web application from learning how code using frontend like React Js using styling Ui like chakra UI and CSS, learning backend like Node Js, Express Js, MongoDB, even authentication like JWT and API calling, things I never had a clue about at the start of the school term for year 4. I am very happy with how the projected turned out will continue to develop CeeScoresLive in the future.



Premier League 2023

Rank	Team	Played	GD	Form	Points
1	Arsenal	34	56	WWLWW	77
2	Manchester City	33	48	WWWWD	76
3	Liverpool	34	41	LWLDW	74
4	Aston Villa	34	21	WWDLW	66
5	Tottenham	32	16	LWDWL	60
6	Manchester United	33	1	WDDLD	53
7	Newcastle	33	15	LWWDW	50
8	West Ham	34	-9	LLWDL	48
9	Chelsea	32	4	LWDWD	47

Is Bruno Fernandes under-appreciated? Man Utd captain's incredible attacking numbers analysed

Apr 26, 2024 | Transfermarkt

Most chances created in PL

11 References

[1] React, [Online]

https://blog.hubspot.com/website/react-js#:~:text=js%3F-,React.,reusable%20piece%20of%20HTML%20code

[2] Chakra UI, [Online]

https://v2.chakra-ui.com/ , [Online]

[3] Node, [Online]

https://nodejs.org/en

[4] Express, [Online]

https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js#:~:text=StackExplore%20Program-,What%20Is%20Express%20JS%3F,helps%20manage%20servers%20and%20routes.

[5] Express, [Online]

https://expressjs.com/

[6] MongoDB, [Online]
https://www.mongodb.com/docs/compass/current/#:~:text=MongoDB%20Compass%20is%20a%20powerful,Download%20Compass

[7] RapidAPI, [Online]

https://apidog.com/blog/what-is-rapidapi-and-how-to-use-it/#:~:text=RapidAPI%20is%20a%20platform%20that,manage%20APIs%20from%20different%20providers.

[8] JWT, [Online]

https://blog.logrocket.com/jwt-authentication-best-practices/#:~:text=The%20server%20generates%20both%20an,stored%20in%20a%20secure%20cookie.

[9] Socket.Io, [Online]
https://socket.io/docs/v3/#:~:text=Socket.IO%20is%20a%20library,be%20also%20run%20from%20Node.

[10] Trello, [Online]

https://trello.com/tour

[11] Kanban, [Online]

https://www.wrike.com/kanban-guide/what-is-kanban/

[12] UseContext, [Online]
https://www.w3schools.com/react/react_usecontext.asp#:~:text=React%20Context%20is%20a%20way,easily%20than%20with%20useState%20alone.

[13] Axios, [Online]

https://www.shecodes.io/athena/172357-what-is-axios-a-javascript-library-for-making-http-requests#:~:text=Axios%20supports%20features%20such%20as,more%20information%20about%20Axios%20here.

[14] UseEffect, [Online]
https://www.geeksforgeeks.org/reactjs-useeffect-hook/#:~:text=React%20useEffect%20hook%20handles%20the,array%20is%20modified%20or%20updated.

[15] Chakra UI, [Online]

https://chakra-ui.com/docs/components

[16] MongoDB Models, [Online]

https://medium.com/@skhans/a-comprehensive-guide-to-data-modeling-in-mongodb-b63b2df9d9dd#:~:text=In%20MongoDB%2C%20data%20modeling%20refers,the%20application%20and%20its%20requirements.

[17] Routes, [Online]
https://www.reddit.com/r/learnprogramming/comments/mriadi/who_routes_who_frontend_or_backend/#:~:text=Your%20back%20end%20uses%20its,everything%20happens%20inside%20your%20browser.

[18] FaceBook data selling, [Online]
https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal#:~:text=In%20the%202010s%2C%20personal%20data,be%20used%20for%20political%20advertising.

[20] Real time chat, [Online]

https://youtu.be/fH8VIb8exdA?si=DPvpXDvCX2Jvnez0

[21] Quiz, [Online]

https://youtu.be/LF2FiUhV84I?si=HxAszoaFa4zQljbM