**JWT Authentication**

**Tip**: Using JWT for authentication

**Tip:** How JWT is wired up in MernStackProject 2022  
**Tip**: Bypassing the HEADERS ARE ALREADY SENT error in the console [ERR\_HTTP\_HEADERS\_SENT]  
**Tip**: Working with the Javascript Local Storage object (its changed a lot)

**Tip**: Bypassing the HEADERS ARE ALREADY SENT error in the console [ERR\_HTTP\_HEADERS\_SENT]  
  
This happens whenever you have a function that has res.status more than once. The way to fix this is to place a return statement on at least one res.status:  
Text

Description automatically generated

**Tip:** How JWT is wired up in MernStackProject 2022  
The two packages we are using for JWT are:  
  
**npm i jsonwebtoken**  
<https://www.npmjs.com/package/jsonwebtoken>

**npm i bcryptjs**  
<https://www.npmjs.com/package/bcryptjs>

Text

Description automatically generated  
  
Text

Description automatically generated

**/models/User.js**  
Using the user model in models – This is our userschema that we will use to send data over to our database once that person registers from the **Register** screen.  
  
Text

Description automatically generated  
  
**Registering the user (/client/components/auth/Register.js**Text

Description automatically generated  
Calls the route below  
In the user route  
/routes/api/users.js  
What this route (endpoint: POST) does:  
Takes in the user credentials: Name,Email,Password from the Register.js component  
It makes a call to MongoDB to see whether the user exists  
If user does not exist,  
He encrypts the password, and creates a new user  
When the user record is saved, it returns the payload, with that payload, he creates a JWT token (see the code below)  
Text

Description automatically generated  
  
  
Complete code below:  
Text

Description automatically generated  
  
Text

Description automatically generated  
  
Text

Description automatically generated  
  
To test the endpoint: (This is in Users & Auth in POSTMAN)  
Graphical user interface, text, application, email

Description automatically generated  
  
<http://localhost:5000/api/users>  
  
Payload:  
{

  "name":"Lionel Jones",

  "email":"ljones876@gmail.com",

  "password":"lionPeace123"

}

The response will be the jwt token  
Diagram, text

Description automatically generated  
  
https://jwt.io/  
You can paste in your token and get the decrypted value  
Graphical user interface, text, website

Description automatically generated

**MIDDLEWARE  
Next he creates a folder to intercept requests (/middleware/auth.js)**Graphical user interface, text, website

Description automatically generatedText

Description automatically generatedThis is the method that intercepts our request (middleware) and (decrypts) the token  
As you can see:  
It takes the token from the header of the request  
Check to see if there is a token in the header  
If a token, we decrypt it.   
Then (remember this code in /api/users -> register user):  
Text

Description automatically generated  
  
Compare that to this code:  
Text

Description automatically generated  
That user object is stored in jwt’s database and we retrieve the user object:  
req.user = decoded.user. WE WILL BE USING THE id:user.id ALL OVER OUR CODE WHEN WE ACCESS OUR PROTECTED ROUTES (ROUTES THAT REQUIRE AUTHENTICATION)  
  
The message you see above:  
Diagram

Description automatically generated  
  
When we access a protected resource, we will get a 401 response, ‘No token, authorization denied’  
  
To test out using the middleware:  
He uses:  
/**routes**/auth.js -- (this route **uses** the middleware for protected routes   
/**middleware**/auth.js) -- (this is our **middleware**)  
  
Text

Description automatically generated  
  
in the /routes/auth  
We make a reference to the /middleware/auth  
  
On routes that we use that we want protected, we add the auth reference to the request  
Text

Description automatically generated  
  
The way he tests:  
Graphical user interface, text, application, email

Description automatically generated  
  
We make call to the auth route and send a token in the header named:  
x-auth-token  
  
Remember this code:

Text

Description automatically generated  
It checks the header for that value to grab the token that will be passed back to our route (auth) to decode.  
  
What he is doing with the /api/auth (GET)  
He is:  
Validating the token using middleware that intercepts the request  
If the token is valid (he grabs the token data from the jwt database that we wrote to when we registered)  
If it valid (passes the middleware)  
He returns all of the user object information (less the password). He will then (later on) write this user information to REDUX.  
  
END OF MIDDLEWARE WIREUP  
  
Authenticating the user and getting the token back  
This route is located in the **/api/auth.js** (**POST**) as well  
  
Text

Description automatically generated  
  
This route takes a email and password  
It searches mongoDB by doing a search for a user based on the email field  
He decrypts the and compares the password sent in with the password stored in mongo db  
If everything is matched (meaning the email and password is correct), he writes a new token to jwt and returns the a new token.  
  
  
Text

Description automatically generated  
  
  
To test:  
Graphical user interface, text, application, email

Description automatically generated  
All of these routes for authentication are located:  
**/api/auth** and **api/users** route(s)  
Graphical user interface, application

Description automatically generated with medium confidence  
  
  
  
<http://localhost:5000/api/users>  
  
Graphical user interface, text, application

Description automatically generated  
  
  
  
  
<http://localhost:5000/api/auth>  
  
Graphical user interface, text, application, email

Description automatically generated  
  
  
  
  
<http://localhost:5000/api/auth>

Graphical user interface, text, application, email

Description automatically generated  
  
The register form  
<https://www.udemy.com/course/mern-stack-front-to-back/learn/lecture/14555458#overview>  
  
Graphical user interface, application

Description automatically generated  
  
12345678  
  
This submits form data  
Text

Description automatically generated  
  
To the auth.js action /actions/auth.js ->Register function  
Text

Description automatically generated  
  
This will create a user record, and then create a JWT token via the service.   
<http://localhost:5500/api/users>  
This function on the service, creates the user record, and also the JWT token  
Then when it is a success, it calls REDUX via   
Text

Description automatically generated  
  
So when you go to your reducer:  
Text

Description automatically generated  
  
That is where it is setting the javascript localstorage with your JWT token  
It is also setting the payload (which is the token) as well in state (the /api/users endpoint returns a token as the response)  
Text

Description automatically generated  
  
  
When this is set, back in the register function, it calls the loaduser() function  
actions/auth.js -> loadUser()  
  
Text

Description automatically generated  
  
  
Text

Description automatically generated  
  
This method:  
checks local storage for a token, if it exists  
It calls the:  
setAuthtoken method  
/utils/setAuthToken.js  
Text

Description automatically generated  
  
The setauthtoken sets the token value on the header via axios  
Text

Description automatically generated  
  
  
  
This is needed because the line in loadUser()

Text

Description automatically generated  
  
Needs to have header an entry for the token value (which was set by the setAuthToken method  
<http://localhost:5500/api/auth>  
This endpoint (has to have a token value when called)  
  
Then when successful, it calls REDUX to DISPATCH USER\_LOADED  
Text

Description automatically generated  
  
Text

Description automatically generated  
  
This set’s our redux’s store’s state property of   
isAuthenticated = true. Along with the user payload, The payload is:  
Text

Description automatically generated  
  
  
  
  
  
  
I am also going to watch redux  
Performing the test:  
A computer screen capture

Description automatically generated with medium confidence  
  
It wrote the record and created the token  
A screenshot of a computer

Description automatically generated  
  
But the javascript localstorage.setitem is not working:

A screenshot of a computer

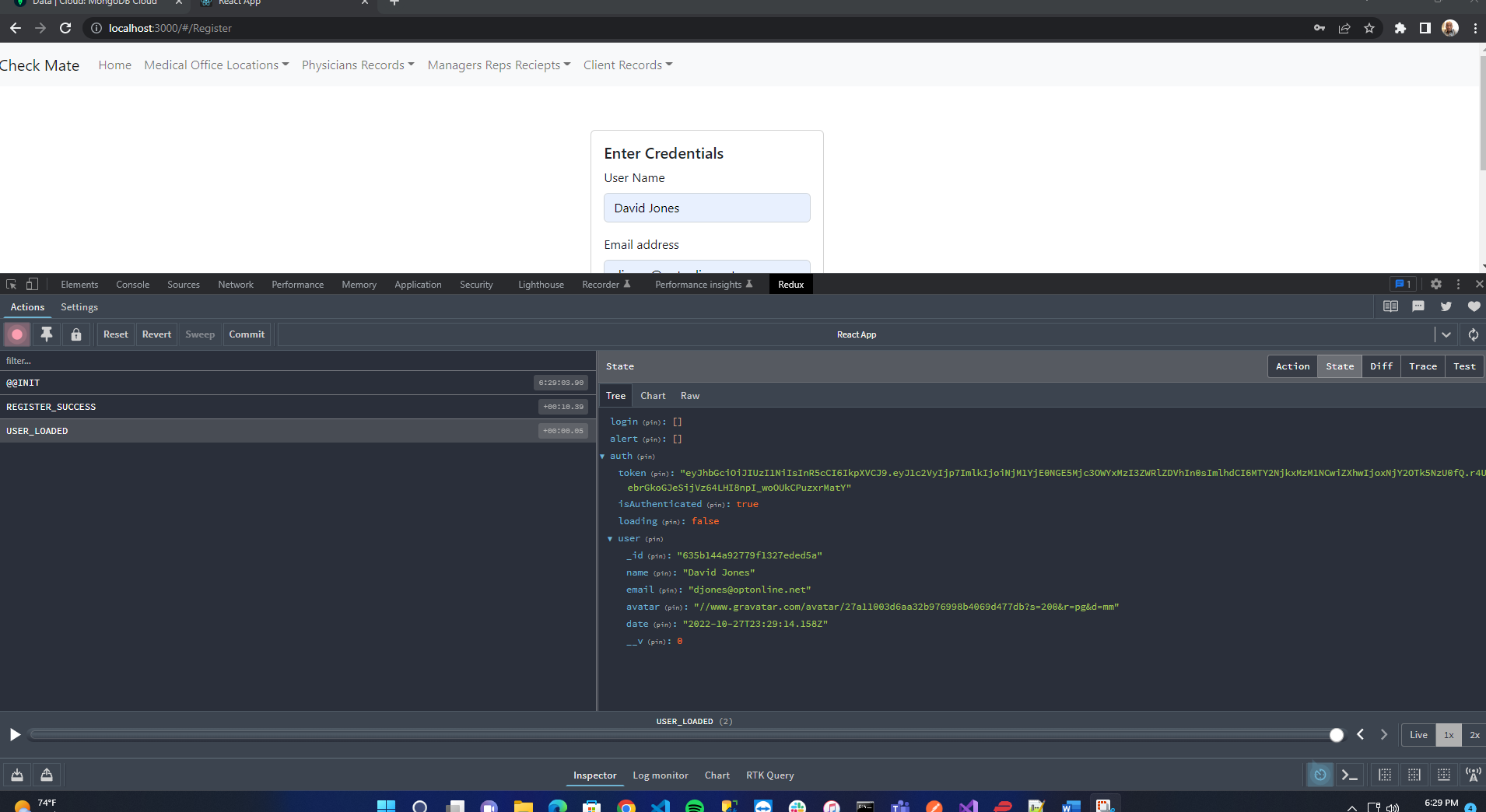
Description automatically generated with medium confidence  
  
A screenshot of a computer

Description automatically generated  
  
I found the issue:  
<https://stackoverflow.com/questions/23805377/localstorage-getitem-logsobject-object>

Graphical user interface, text

Description automatically generated

**(Tip**: Working with the Javascript Local Storage object (its changed a lot))  
  
I had to change the code in:  
LoadUser() - this is where it was setting the header, but I had to grab the value of the token using JSON.PARSE and the key – the old way does not work, but this way doesText

Description automatically generated  
  
And as you see the entire process above works:  
  
  
Text

Description automatically generated  
  
You see what it writes  
Text

Description automatically generated  
  
Graphical user interface, application

Description automatically generated

The login screen:

Graphical user interface

Description automatically generated  
This screen is pretty straightforward, just like register. It calls a different type, but the end result is the same, LoadUser is called just like with register to grab the token from local storage and authenticate and pass back the user payload

Text

Description automatically generated

Calls the /auth end point  
<http://localhost:5000/api/auth>  
passes in the email and password to find the user in mongodb  
Then it creates a new jwttoken

Text

Description automatically generated

Text

Description automatically generated

Then it calls the LOGIN\_SUCESS ACTION WITH THIS REDUCER

Text

Description automatically generated  
Text

Description automatically generated

Brings over the same payload as register and stores it in the store

Then calls this action with loaduser() to authorize (just like the last step with register)

Text

Description automatically generated  
  
User loaded also brings stores the user payload in state

Text

Description automatically generated

**And we good**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated  
 **Tip**: Using JWT for authentication  
https://jwt.io/  
Graphical user interface, text, website

Description automatically generated  
  
Explains what each part of the encoded parts of the token mean  
Wiring it up in our project:  
In our users route

Text

Description automatically generated  
When we write the record, we get a token  
Text

Description automatically generated  
  
Graphical user interface, text, application, email

Description automatically generated  
  
To check your token, go to  
https://jwt.io/  
And paste your token and you can see the encode and decode  
Graphical user interface, text, application, email

Description automatically generated  
  
Graphical user interface, text, application, email

Description automatically generated