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:

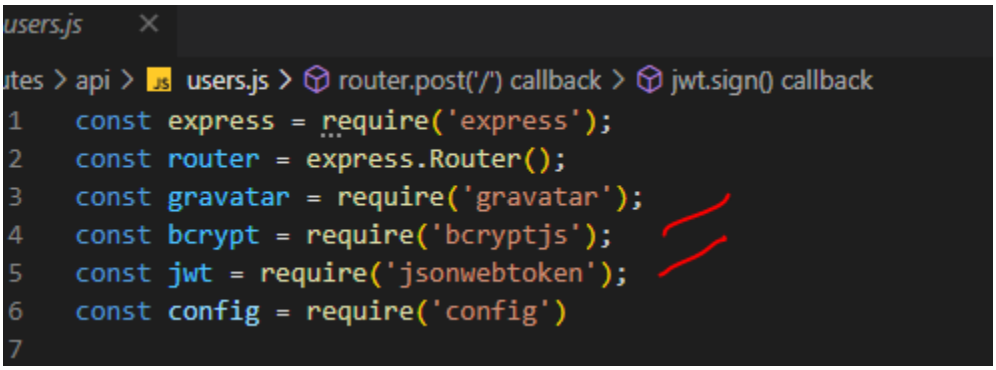
```
//See if user Exists
let user = await User.findOne({email});  //do a mongodb search query on em
if(user) {
    return res.status(400).json({errors:[{msg:'User already exits'}]});
}
```

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

```
    "url": "https://github.com/lionel5116/Me
  },
  "homepage": "https://github.com/lionel5116
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "config": "^3.3.7",
    "express": "^4.18.1",
    "express-validator": "^6.14.1",
    "gravatar": "^1.8.2",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^6.3.4",
    "request": "^2.88.2"
  },
  "devDependencies": {
    "concurrently": "^7.2.1",
    "nodemon": "^2.0.16"
  }
}
```



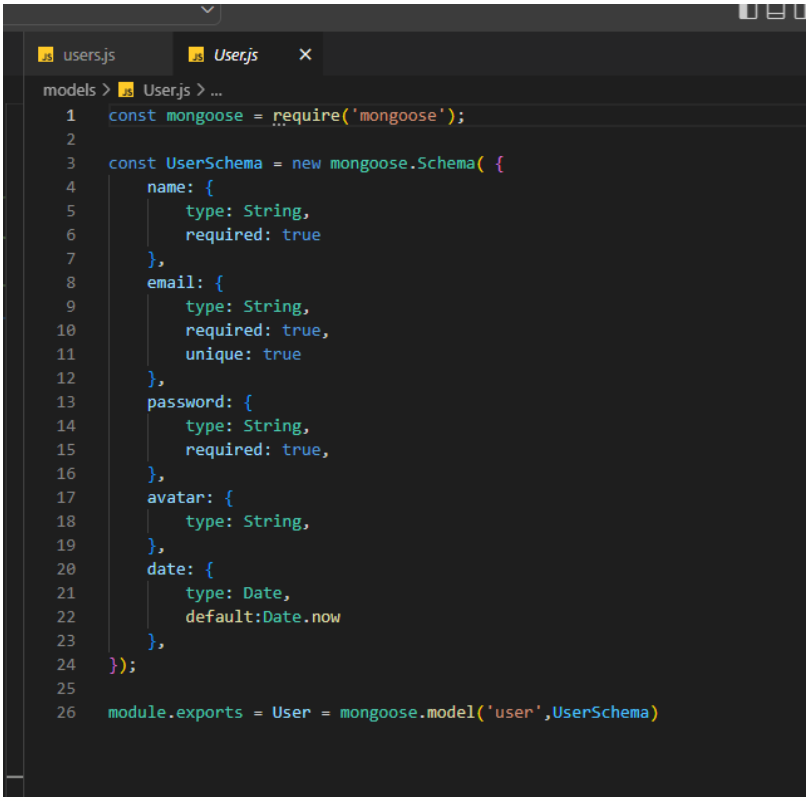
users.js

utes > api > users.js > router.post('/') callback > jwt.sign() callback

```
const express = require('express');
const router = express.Router();
const gravatar = require('gravatar');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const config = require('config')

```

/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.



Registering the user (/client/components/auth/Register.js

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)

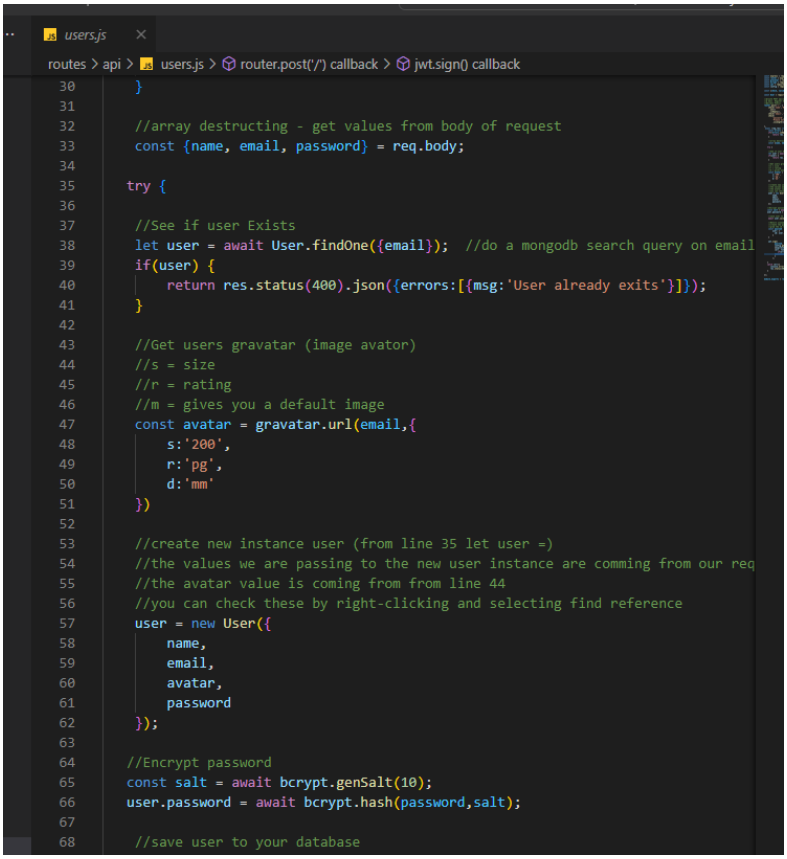
```
    await user.save();

    //Return jsonwebtoken..
    //create the payload to send to the jwt wire-up
    //grab the id returned from the instered record _id = id (mong
    const payload    = {
        user: {
          id: user.id,
        }
    }

    jwt.sign(
          payload,
          config.get('jwtSecretToken'),
          {expiresIn:'24h'},
          (err, token) => {
            if(err) throw err;
            res.json({token})
          }
        );

  }
```

Complete code below:

users.js

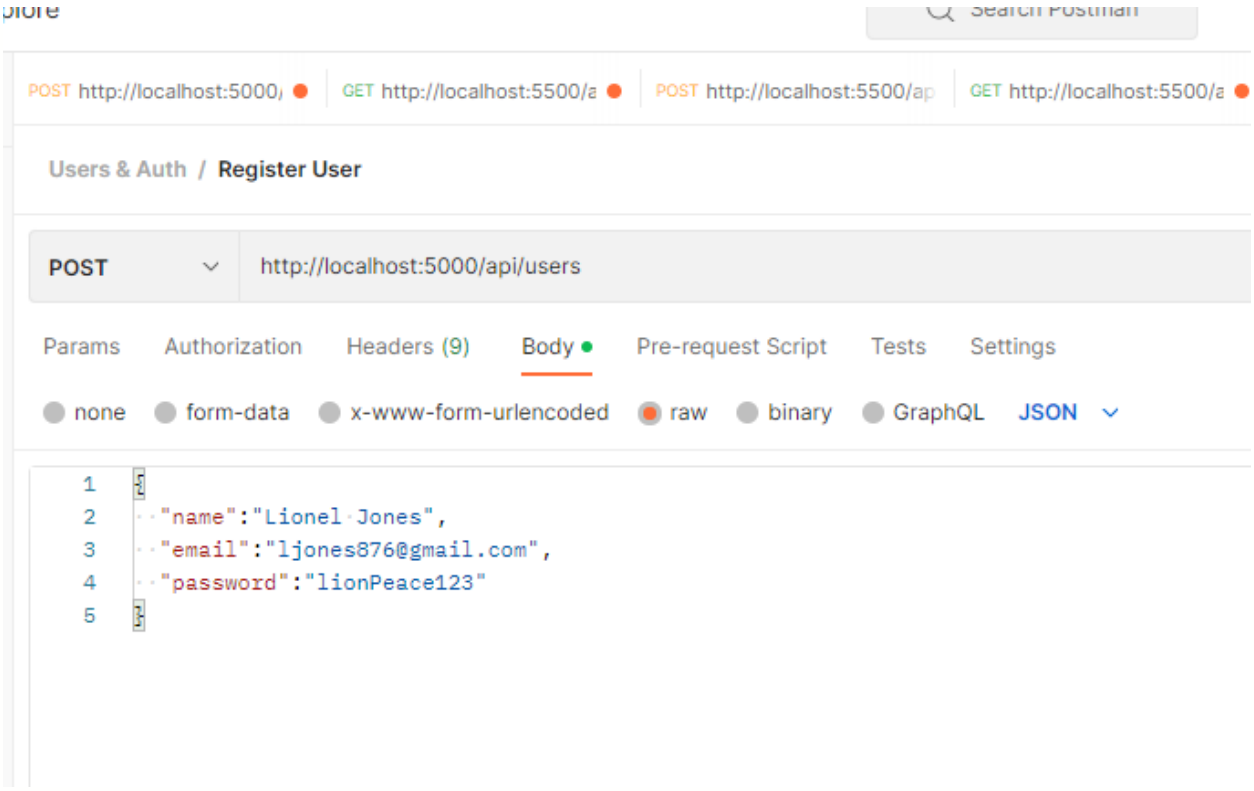
routes > api > users.js > router.post('/') callback > jwt.sign() callback

```
1   const express = require('express');
2   const router = express.Router();
3   const gravatar = require('gravatar');
4   const bcrypt = require('bcryptjs');
5   const jwt = require('jsonwebtoken');
6   const config = require('config')
7
8   const {check, validationResult} = require('express-validator')
9
10  const User = require('../../models/User')
11
12  //@route POST api/users
13  //@ desc  Regiter User
14  //@access Public
15  router.post('/', [
16      check('name','Name is required')
17       .not()
18       .isEmpty(),
19      check('email','Please include a valid email').isEmail(),
20      check(
21          'password',
22          'Please enter a password with 6 or more characters')
23          .isLength({min:6})
24
25  ],
26   async (req,res) => {
27      const errors = validationResult(req);
28      if(!errors.isEmpty()){
29          return res.status(400).json({errors:errors.array()});
30      }
31
32      //array destructing - get values from body of request
33      const {name, email, password} = req.body;
34
35      try {
```


users.js
routes > api > users.js > router.post('/') callback > jwt.sign() callback
30 }
31
32 //array destructing - get values from body of request
33 const {name, email, password} = req.body;
34
35 try {
36
37 //See if user Exists
38 let user = await User.findOne({email}); //do a mongodb search query on email
39 if(user) {
40 return res.status(400).json({errors:[{msg:'User already exits'}]});
41 }
42
43 //Get users gravatar (image avator)
44 //s = size
45 //r = rating
46 //m = gives you a default image
47 const avatar = gravatar.url(email,{
48 s:'200',
49 r:'pg',
50 d:'mm'
51 })
52
53 //create new instance user (from line 35 let user =)
54 //the values we are passing to the new user instance are comming from our req
55 //the avatar value is coming from from line 44
56 //you can check these by right-clicking and selecting find reference
57 user = new User({
58 name,
59 email,
60 avatar,
61 password
62 });
63
64 //Encrypt password
65 const salt = await bcrypt.genSalt(10);
66 user.password = await bcrypt.hash(password,salt);
67
68 //save user to your database

```
    //save user to your database
    await user.save();

    //Return jsonwebtoken..
    //create the payload to send to the jwt wire-up
    //grab the id returned from the instered record _id = id (mongoose allows you
    const payload   = {
        user: {
          id: user.id,
        }
    }

    jwt.sign(
          payload,
          config.get('jwtSecretToken'),
          {expiresIn:'24h'},
          (err, token) => {
            if(err) throw err;
            res.json({token})
          }
        );

   }
   catch (err){
      console.error(err.message);
      res.status(500).send("Server error")
   }

});

module.exports = router;
```

To test the endpoint: (This is in Users & Auth in POSTMAN)



http://localhost:5000/api/users

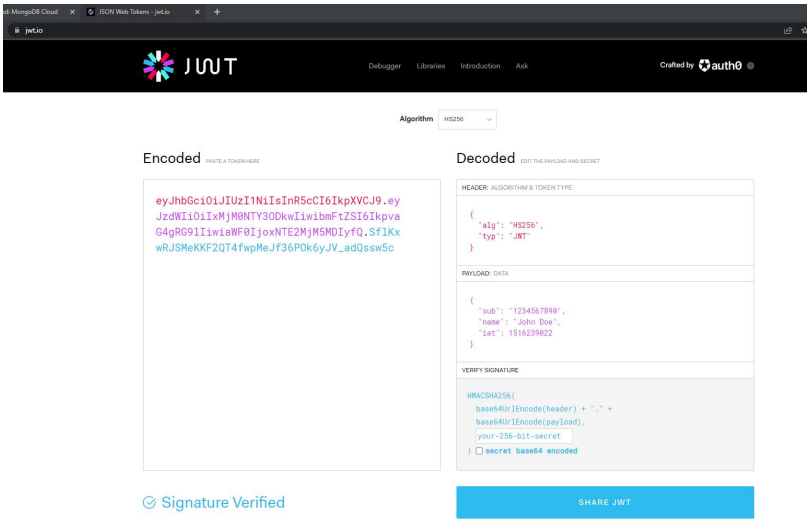
Payload:

```
{
  "name":"Lionel Jones",
  "email":"ljones876@gmail.com",
  "password":"lionPeace123"
}
```

The response will be the jwt token

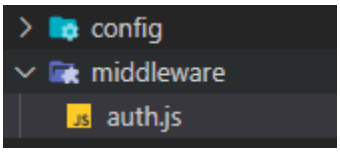
```
    jwt.sign(
        payload,
        config.get('jwtSecretToken'),
        {expiresIn:'24h'},
        (err, token) => {
            if(err) throw err;
            res.json({token})
        }
    );

}
catch (err){
```

https://jwt.io/
You can paste in your token and get the decrypted value



MIDDLEWARE

Next he creates a folder to intercept requests (/middleware/auth.js)



auth.js

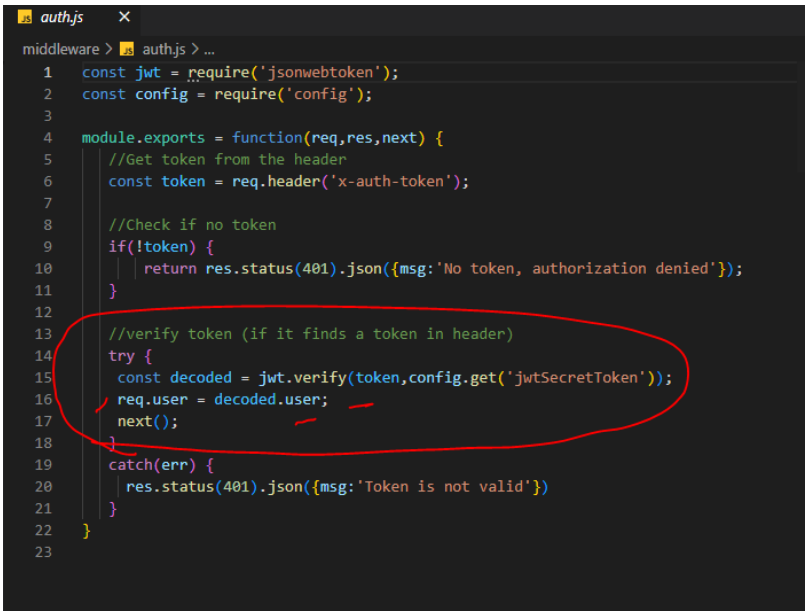
middleware > auth.js > ...

```
1   const jwt = require('jsonwebtoken');
2   const config = require('config');
3
4   module.exports = function(req,res,next) {
5       //Get token from the header
6       const token = req.header('x-auth-token');
7
8       //Check if no token
9       if(!token) {
10          return res.status(401).json({msg:'No token, authorization denied'});
11      }
12
13      //verify token (if it finds a token in header)
14      try {
15       const decoded = jwt.verify(token,config.get('jwtSecretToken'));
16       req.user = decoded.user;
17       next();
18      }
19      catch(err) {
20        res.status(401).json({msg:'Token is not valid'})
21      }
22  }
23
```

This 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):

```
//grab the id returned from the instered re
const payload    = {
    user: {
      id: user.id,
    }
}

jwt.sign(
      payload,
      config.get('jwtSecretToken'),
      {expiresIn:'24h'},
      (err, token) => {
        if(err) throw err;
        res.json({token})
      }
    );
```

Compare that to this code:



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:

```
const token = req.header('x-auth-token');

//Check if no token
if(!token) {
    return res.status(401).json({msg:'No token, authorization denied'});
}

//verify token (if it finds a token in header)
try {
```

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**)

auth.js

```
const express = require('express');
const router = express.Router();
const auth = require('../../middleware/auth')
const User = require('../../models/User')
const jwt = require('jsonwebtoken');
const config = require('config')
const bcrypt = require('bcryptjs');
const {check, validationResult} = require('express-validator')

//@route GET api/auth
//@ desc  Test route
//@access Public
router.get('/', auth, async (req,res) => {
  try {
   const user = await User.findById(req.user.id).select('-password');
   res.json(user)
  }
  catch(err){
  console.err(err.message);
   res.status(500).send('Server error!!!!')
  }
});


//@route POST api/auth
//@ desc Authenticate User & get token
//@access Public
```

middleware/auth.js

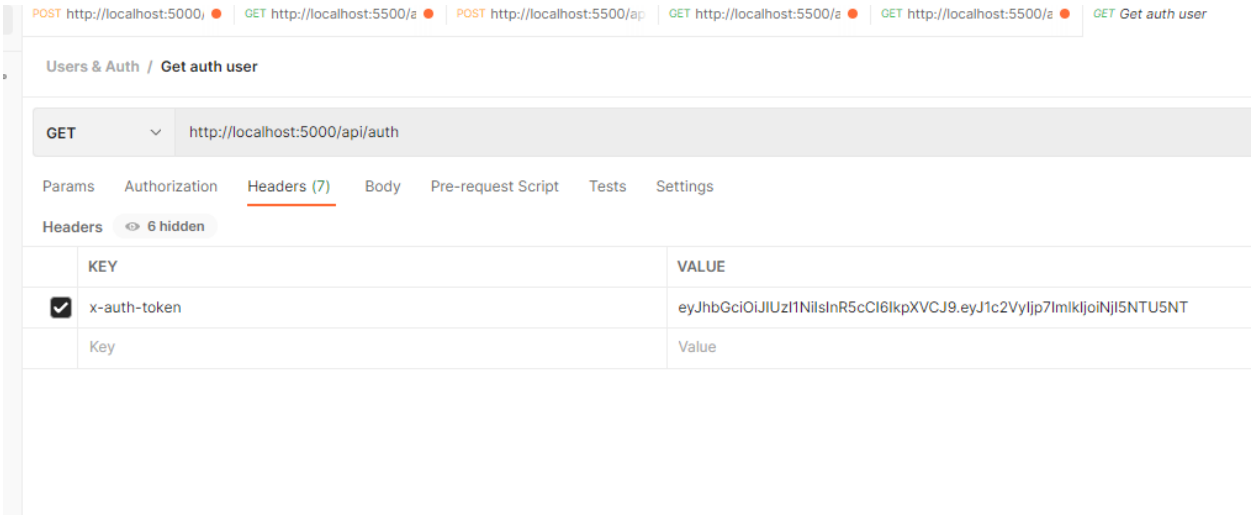
```
const jwt = require('jsonwebtoken');
const config = require('config');

module.exports = function(req,res,next) {
    //Get token from the header
    const token = req.header('x-auth-token');

    //Check if no token
    if(!token) {
        return res.status(401).json({msg:'No token, authorization denied'});
    }

    //verify token (if it finds a token in header)
    try {
     const decoded = jwt.verify(token,config.get('jwtSecretToken'));
     req.user = decoded.user;
     next();
    }
    catch(err) {
     res.status(401).json({msg:'Token is not valid'})
    }
}

```

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

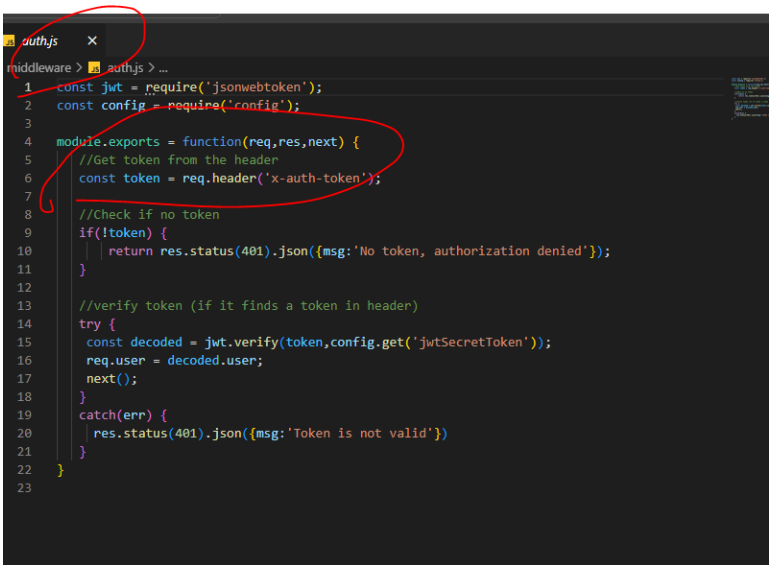
```
const express = require('express');
const router = express.Router();
const auth = require('../../middleware/auth')
const User = require('../../models/User')
const jwt = require('jsonwebtoken');
const config = require('config')
const bcrypt = require('bcryptjs');
const {check, validationResult} = require('express-validator')

//@route GET api/auth
//@ desc  Test route
//@access Public
router.get('/', auth, async (req,res) => {
  try {
    const user = await User.findById(req.user.id).select('-password');
    res.json(user)
  }
  catch(err){
  console.err(err.message);
    res.status(500).send('Server error!!!!')
  }
});


//@route POST api/auth
```

The way he tests:



We make call to the auth route and send a token in the header named:
x-auth-token

Remember this code:



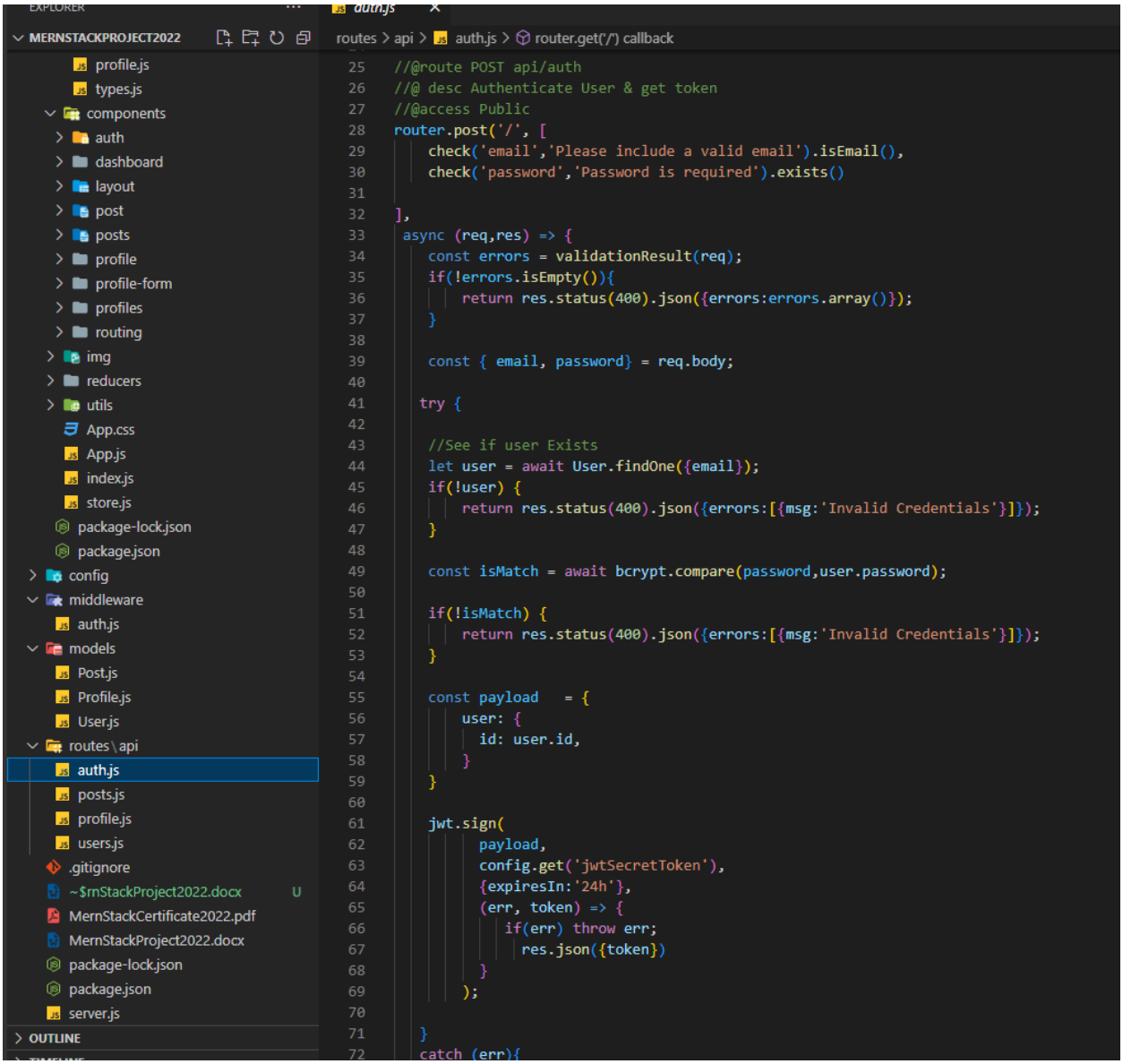
It checks the header for that value to grab the token that will be passed back to our route (auth) to decode.

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

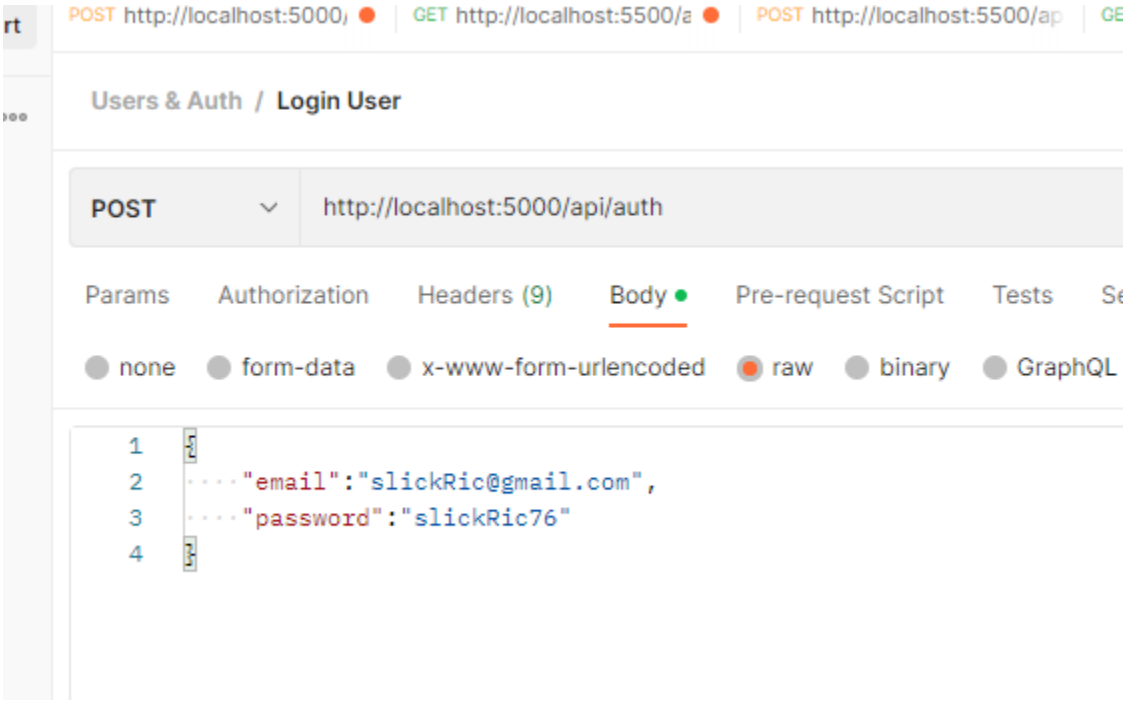
This route is located in the **/api/auth.js** (**POST**) as well



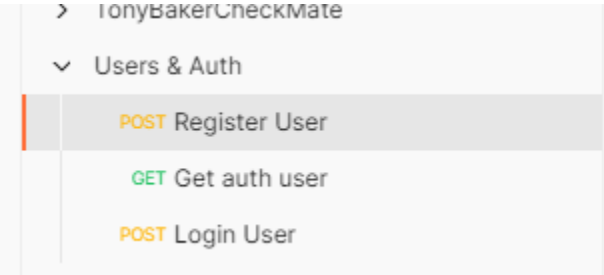
This route takes a email and password
It searches mongoDB by doing a search for a user based on the email field
He decrypts the and compares the password sent in with the password stored in mongo db
If everything is matched (meaning the email and password is correct), he writes a new token to jwt and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

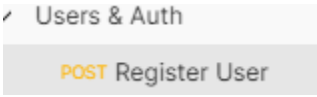
```
const isMatch = await bcrypt.compare(password,user.password);

if(!isMatch) {
    return res.status(400).json({errors:[{msg:'Invalid Credentials'}]});
}

const payload    = {
    user: {
      id: user.id,
    }
}

jwt.sign(
      payload,
      config.get('jwtSecretToken'),
      {expiresIn:'24h'},
      (err, token) => {
        if(err) throw err;
          res.json({token})
      }
    );

}
catch (err){
```

To test:

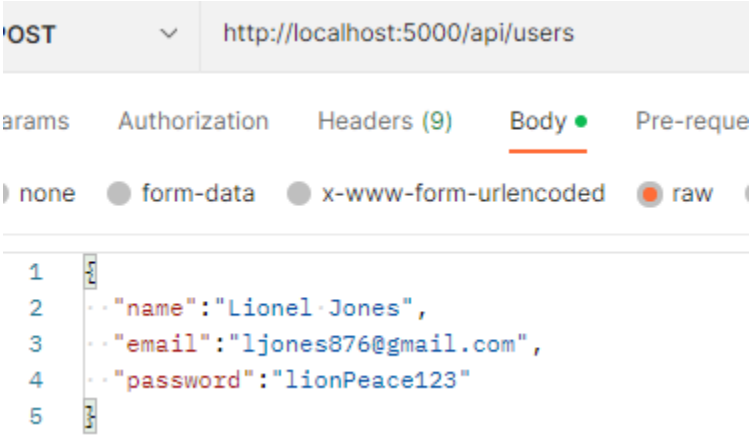


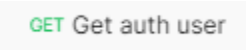
All of these routes for authentication are located:
/api/auth and **api/users** route(s)



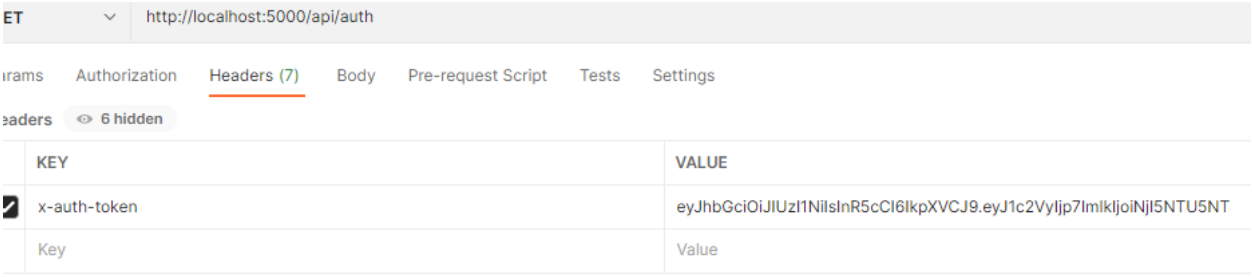


http://localhost:5000/api/users



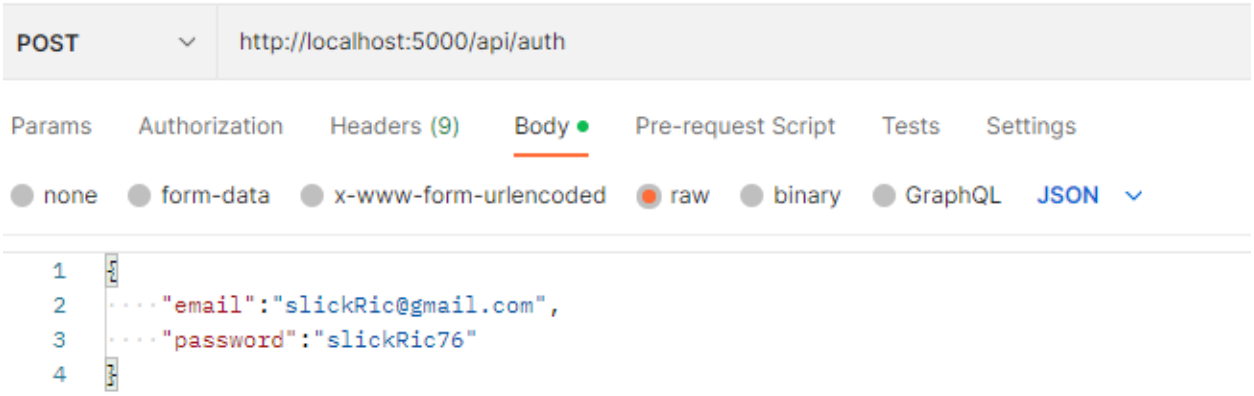


http://localhost:5000/api/auth

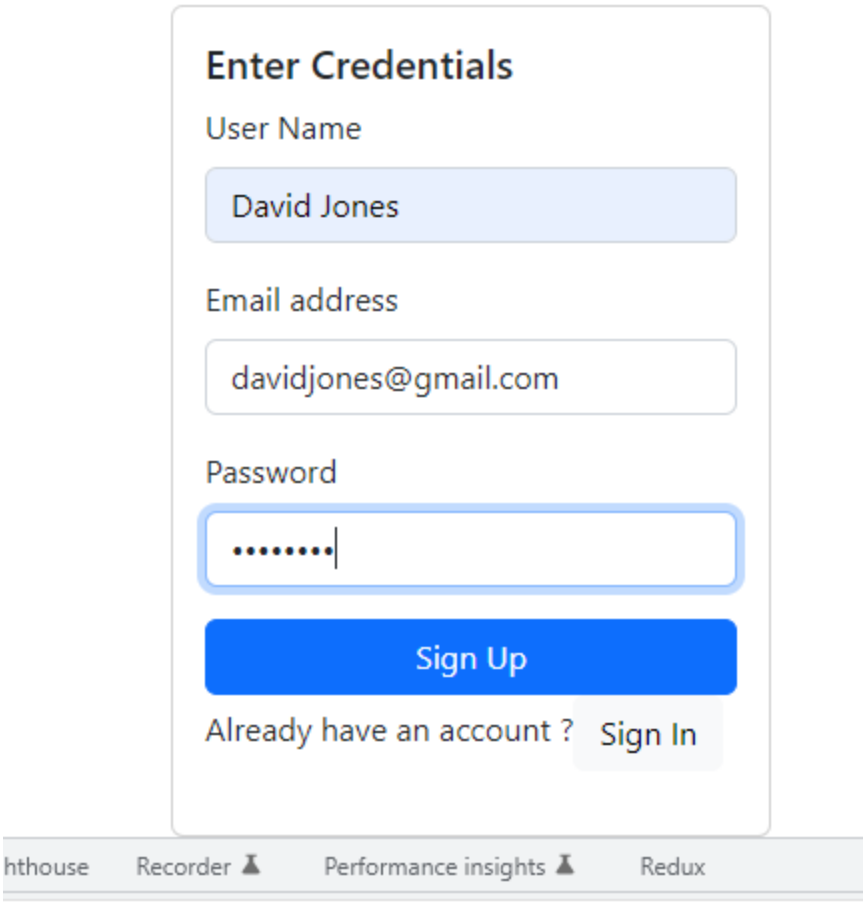


POST Login User

http://localhost:5000/api/auth

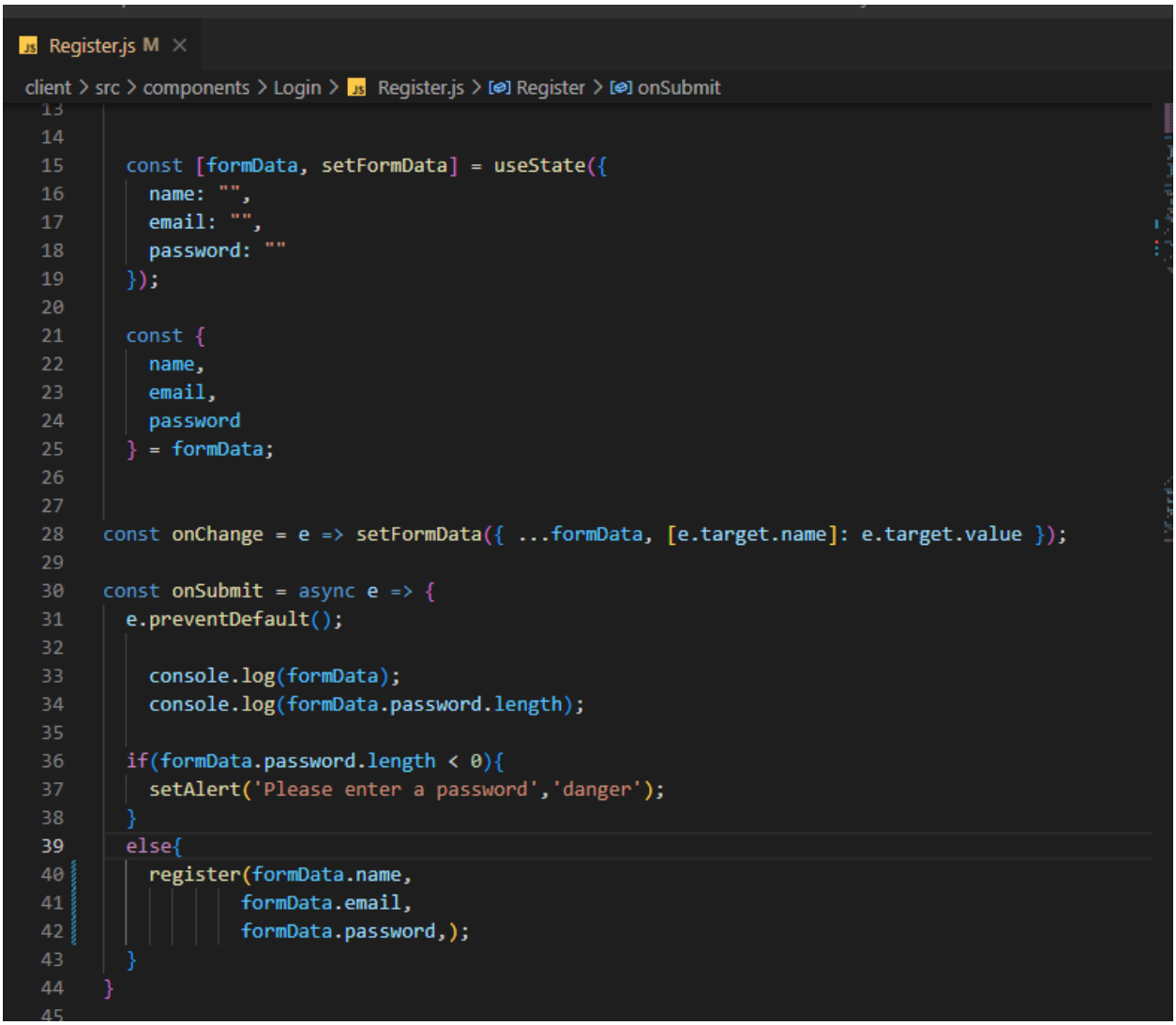


The register form
https://www.udemy.com/course/mern-stack-front-to-back/learn/lecture/14555458#overview



12345678

This submits form data



To the auth.js action /actions/auth.js ->Register function

```
//Register User
export const register = (name,email,password) => async dispatch => {
    const config = {
        headers: {
            'Content-Type':'application/json'
        }}

    const body = JSON.stringify({name,email,password});
    console.log(body);
        Jones, 59 minutes ago • Version 2 Complete. …
    let serviceUrl = "";
    serviceUrl = process.env.REACT_APP_SERVICE_URL + '/users'


    try {
        const res = await axios.post(serviceUrl,body,config);
        dispatch({
          type: REGISTER_SUCCESS,
          payload:res.data
        });

        dispatch(loadUser());

    } catch (err) {
        const errors = err.response.data.errors;
        if(errors) {
          errors.forEach(error => dispatch(setAlert(error.msg,'danger')));
        }
        dispatch({
            type: REGISTER_FAIL
          })
    }
};
```

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

```
try {
    const res = await axios.post(serviceUrl,body,config);
    dispatch({
      type: REGISTER_SUCCESS,
      payload:res.data
    });
```

So when you go to your reducer:

```
    user:null
}

export default function authReducer (state = initialState,action) {
    const {type, payload} = action

    switch(type) {
     case USER_LOADED:
       return {
           ...state,
           isAuthenticated:true,
           loading:false,
           user:payload
       }
     case REGISTER_SUCCESS:
         localStorage.setItem('token',payload.token)
         return {
             ...state,
             ...payload,
             isAuthenticated: true,
             loading:false
         }
     case LOGIN_SUCCESS:
         localStorage.setItem('token',payload.token)
         return {
             ...state,
```

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)

```
    //save user to your database
    await user.save();


    //Return jsonwebtoken.. WE ARE SIGNING OUR TOKEN PASSING IN OUR ID FROM
    //create the payload to send to the jwt wire-up
    //grab the id returned from the instered record _id = id (mongoose allo
    const payload   = {
        user: {
          id: user.id,
        }
    }

    //AFTER WE SIGN, WE SEND THE JWT TOKEN BACK AS A RESPONSE
    jwt.sign(
        payload,
        config.get('jwtSecretToken'),
        {expiresIn:'24h'},
        (err, token) => {
          if(err) throw err;
            res.json({token}) //AFTER WE SIGN, WE SEND THE JWT TOKEN BAC
        }
      );


  }
  catch (err){
     console.error(err.message);
     res.status(500).send("Server error")
  }
```

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

```
44
45  //Register User
46  export const register = (name,email,password) => async dispatch => {
47      const config = {
48          headers: {
49              'Content-Type':'application/json'
50          }}
51
52      const body = JSON.stringify({name,email,password});
53      console.log(body);
54
55      let serviceUrl = "";
56      //http://localhost:5500/api/users
57      serviceUrl = process.env.REACT_APP_SERVICE_URL + '/users'
58
59
60      try {
61          const res = await axios.post(serviceUrl,body,config);
62          dispatch({
63            type: REGISTER_SUCCESS,
64            payload:res.data
65          });
66
67          dispatch(loadUser());
68
69      } catch (err) {
```

```
//Load User
export const loadUser = () => async dispatch => {

   if (localStorage.getItem('token') !== null) {
      console.log(`Token exists`);
      setAuthToken(localStorage.token)      Jones, 1 hour ago • Version 2 Complete.
      console.log(localStorage.token);
   } else {
      console.log(`Token does not exist`);
   }


   let serviceUrl = "";
   serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'


   try {
    const res = await axios.get(serviceUrl);

    dispatch( {
        type: USER_LOADED,
        payload: res.data
    });
   } catch (error) {
      dispatch({
         type: AUTH_ERROR
      })
   }
}
```

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

auth.js M | setAuthToken.js

client > src > components > utils > setAuthToken.js > setAuthToken

p00149021@houstonisd.org Mag17615@7, 2 days ago | 1 author (p00149021@houstonisd.org Mag17615@7)

```
import axios from "axios";

const setAuthToken = token => {
    if(token) {
        axios.defaults.headers.common['x-auth-token'] = token;
    }
    else {
        delete axios.defaults.headers.common['x-auth-token']
    }

}

export default setAuthToken
```

The setauthtoken sets the token value on the header via axios

```
if(token) {
    axios.defaults.headers.common['x-auth-token'] = token;
}
else {
```

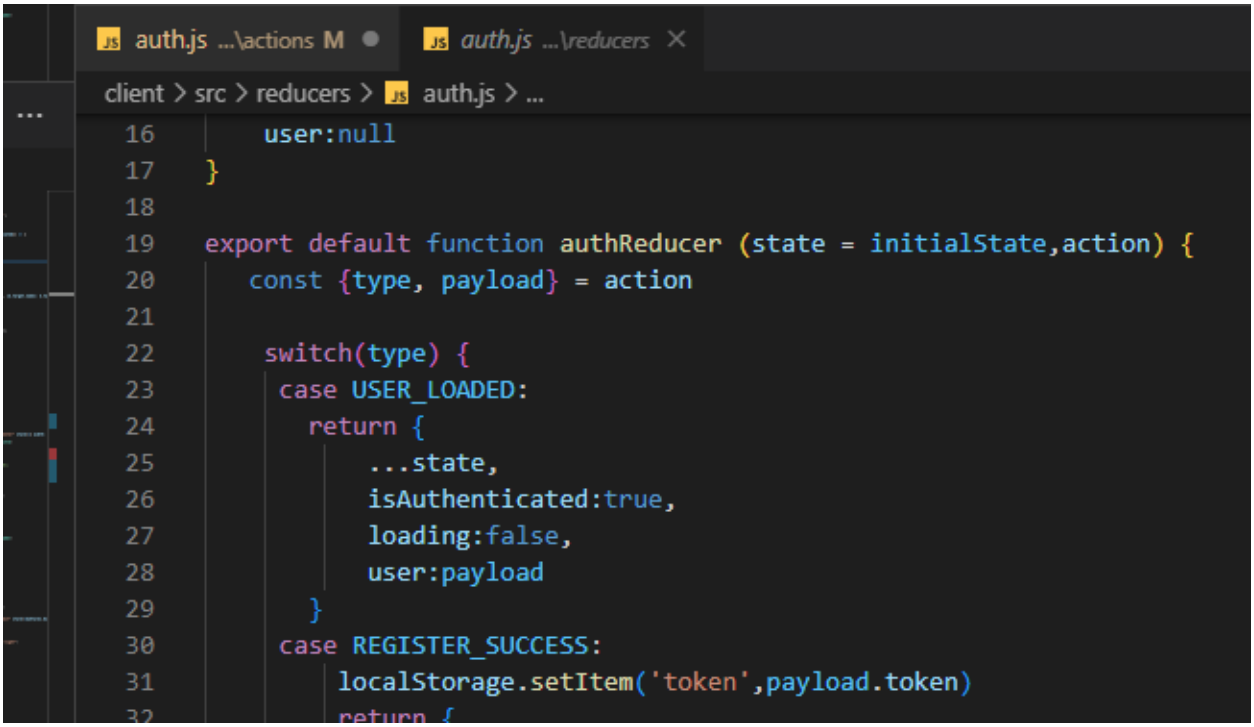
This is needed because the line in loadUser()

```
    let serviceUrl = "";
    serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'


    try {
     const res = await axios.get(serviceUrl);

     dispatch( {
         type: USER_LOADED,
         payload: res.data
     });
    } catch (error) {
        dispatch({
            type: AUTH_ERROR
        })
    }
}
```

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

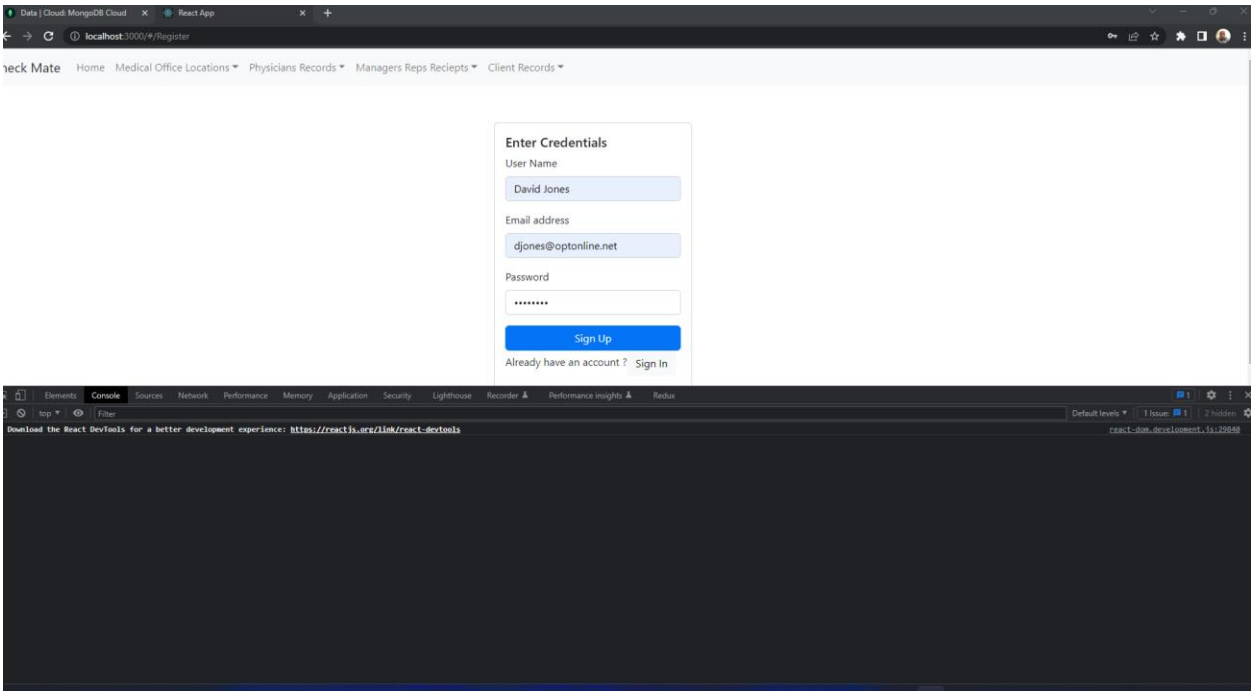
```
    const res = await axios.get(servic

    dispatch( {
        type: USER_LOADED,
        payload: res.data
    });
   } catch (error) {
       dispatch({
           type: AUTH_ERROR
       })
   }
}
```



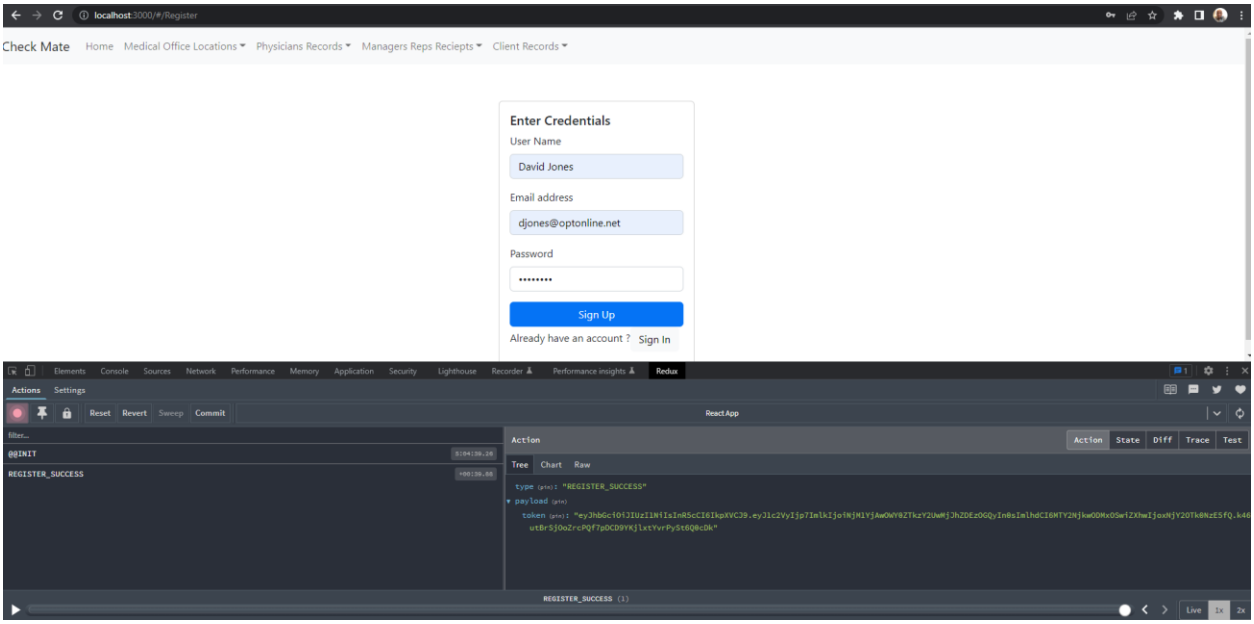
This set's our redux's store's state property of
isAuthenticated = true. Along with the user payload, The payload is:

```
    }

    //we grab the userID from the MongoDB user to sign with a new token
    const payload   = {
        user: {
          id: user.id,
        }
    }

    //sign and create a new token with the user found
    jwt.sign(
          payload,
          config.get('jwtSecretToken'),
          {expiresIn:'24h'},
          (err, token) => {
             if(err) throw err;
               res.json({token})
          }
        );

  }
  catch (err){
     console.error(err.message);
     res.status(500).send("Server error")
  }

});
```

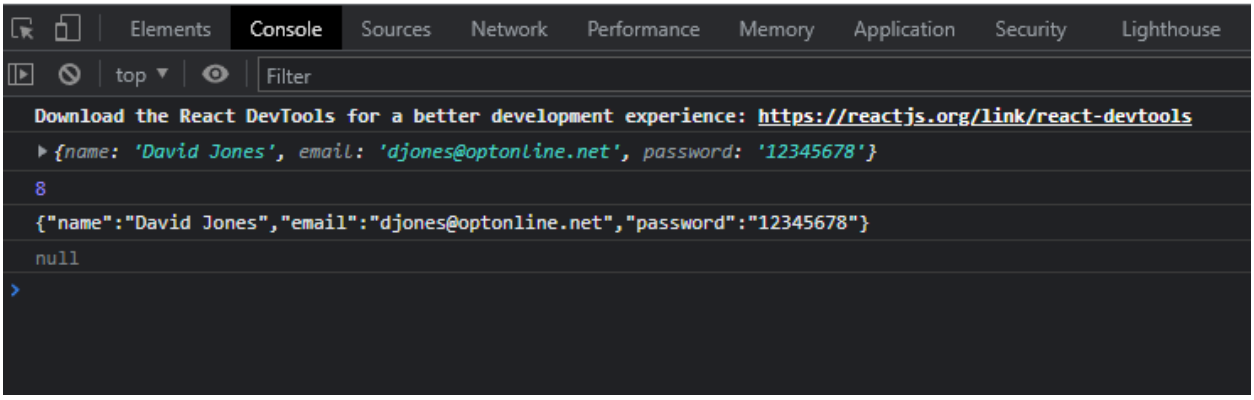
I am also going to watch redux
Performing the test:

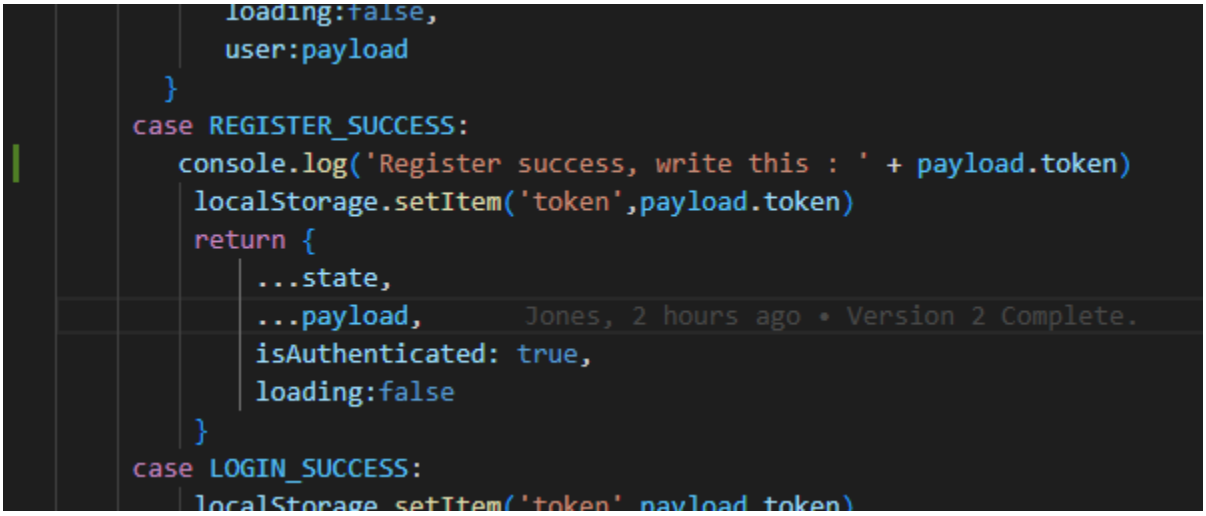


It wrote the record and created the token



But the javascript localstorage.setitem is not working:





I found the issue:
https://stackoverflow.com/questions/23805377/localstorage-getitem-logsobject-object

43

Local storage only supports `string` datatype. So you have to

1. Convert it to String before saving to LocalStorage

```
localStorage.setItem('key', JSON.stringify(data));
```

2. Convert back to JS object, reading from LocalStorage

```
data = JSON.parse(localStorage.getItem('key')); //forgot to close
```

In case of your code, it should be -

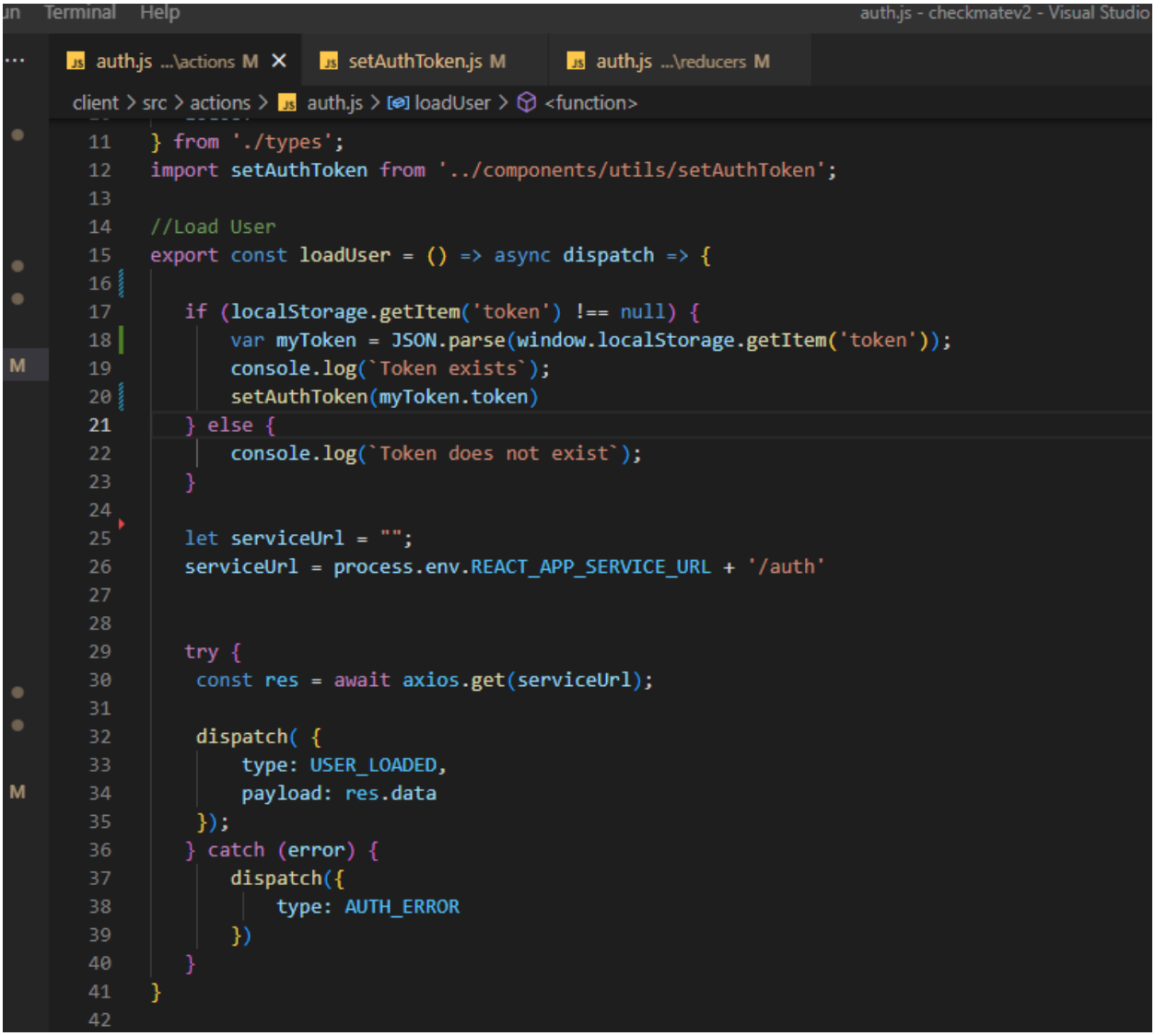
```
var widgets = JSON.parse(localStorage.getItem('widgets'));
```

and

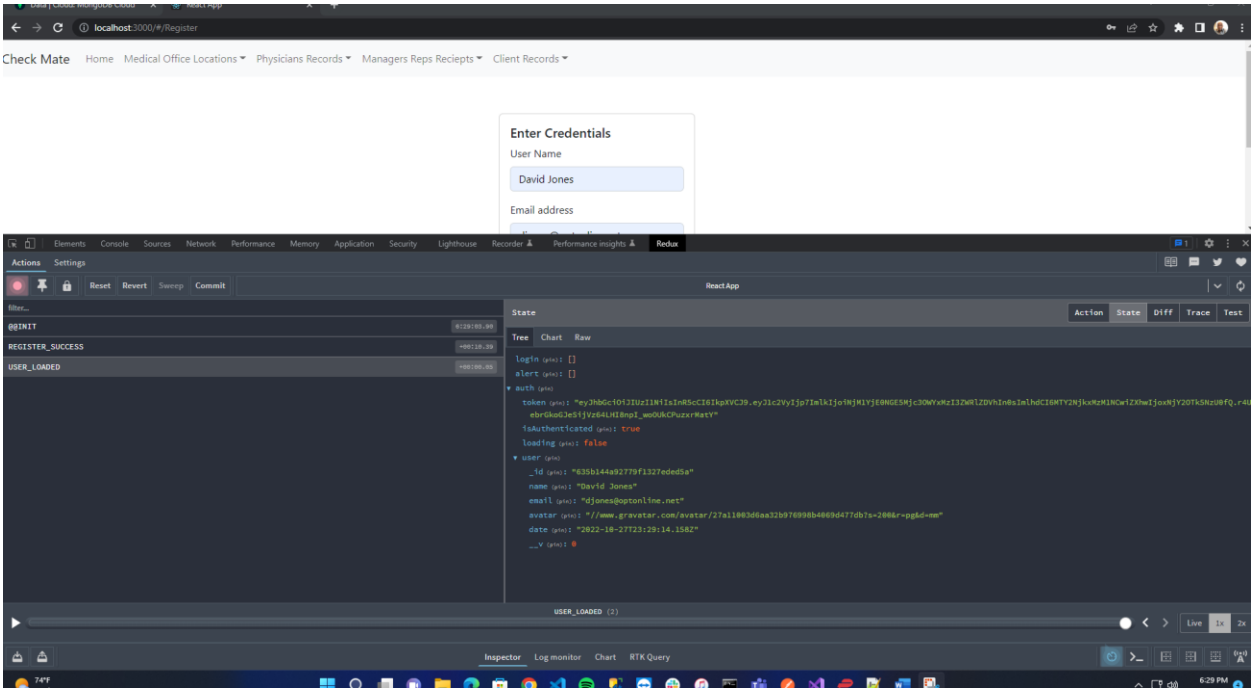
```
localStorage.setItem('widgets', JSON.stringify(widgets));
```

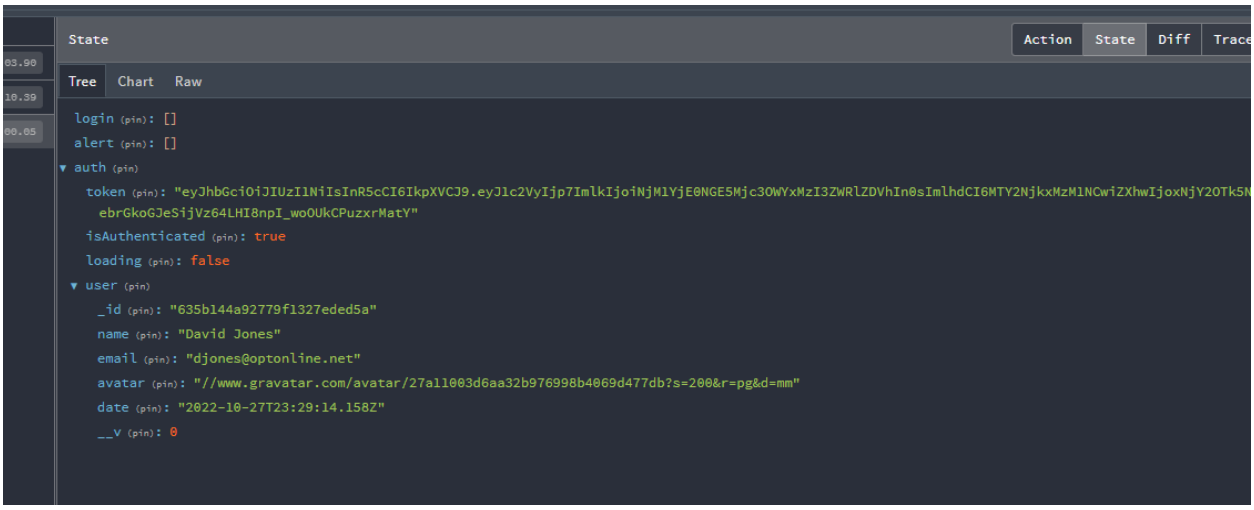
(Tip: Working with the Javascript Local Storage object (its changed a lot))

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 does



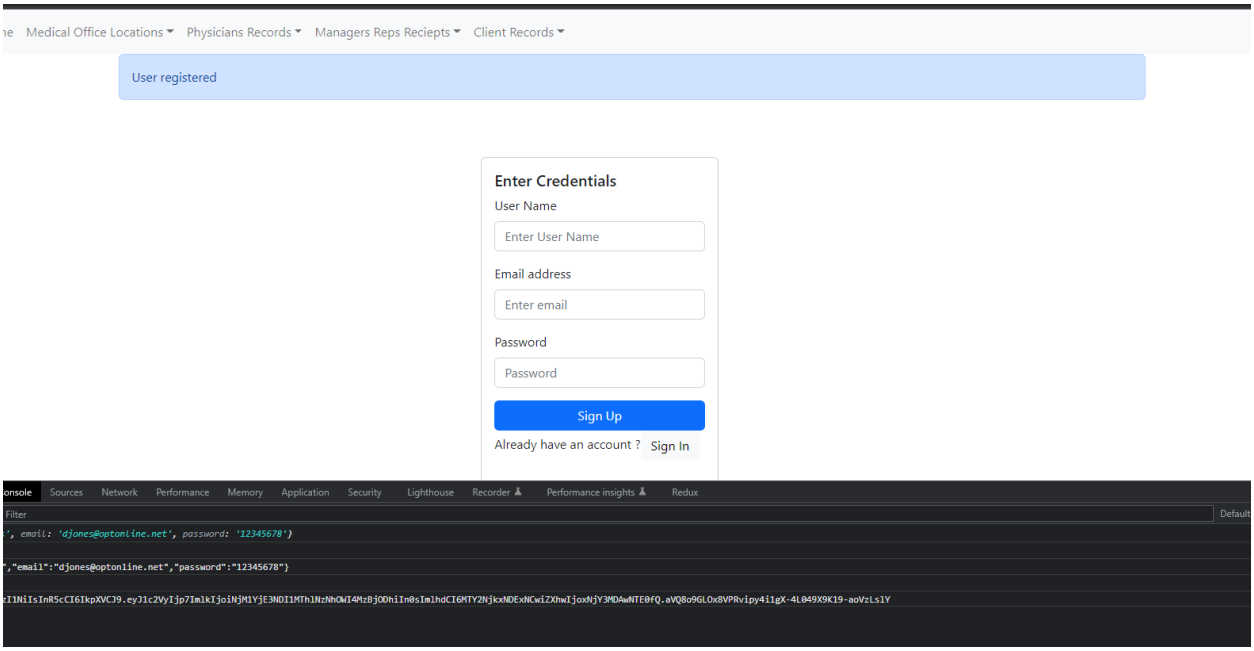
And as you see the entire process above works:



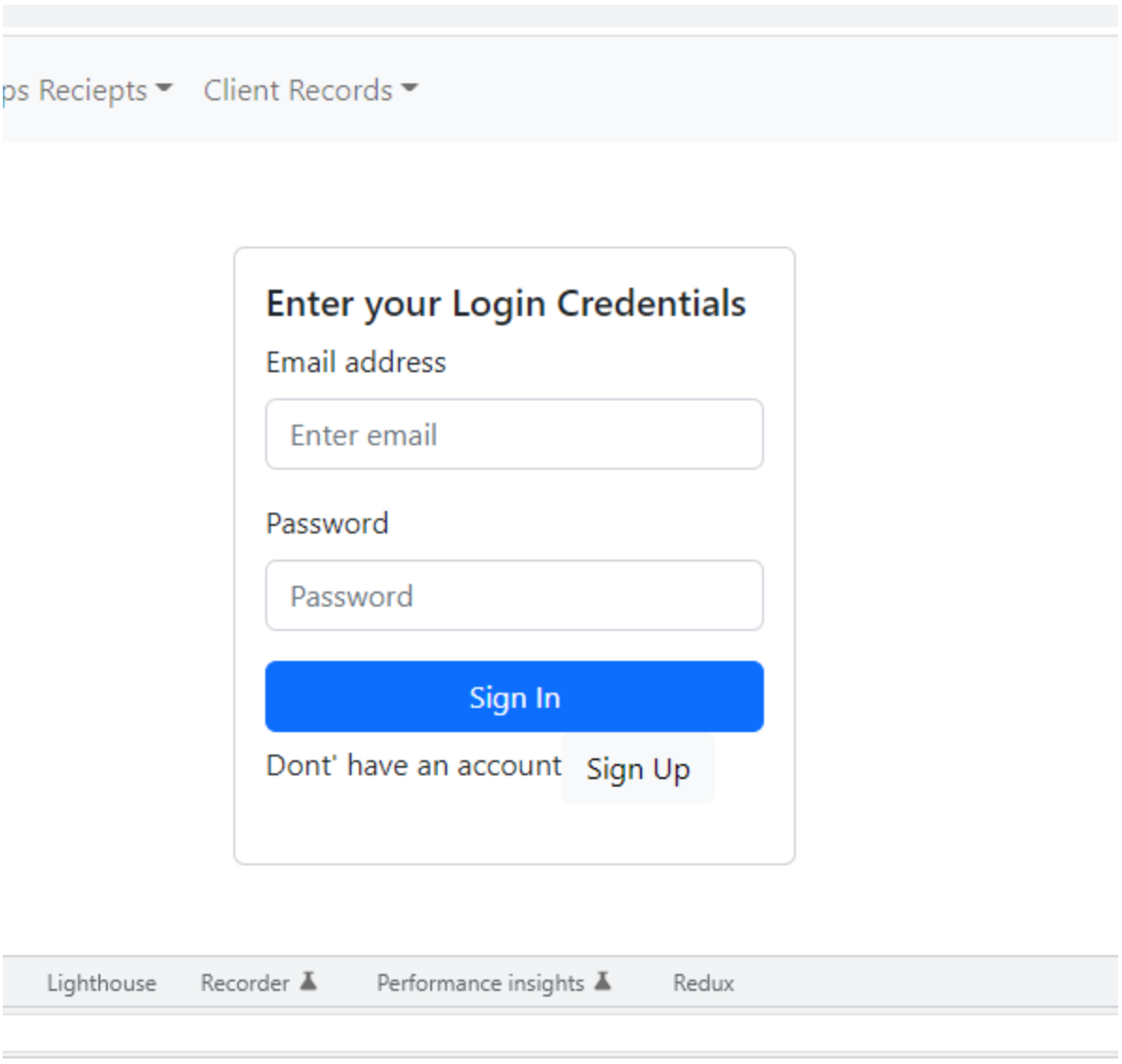


You see what it writes

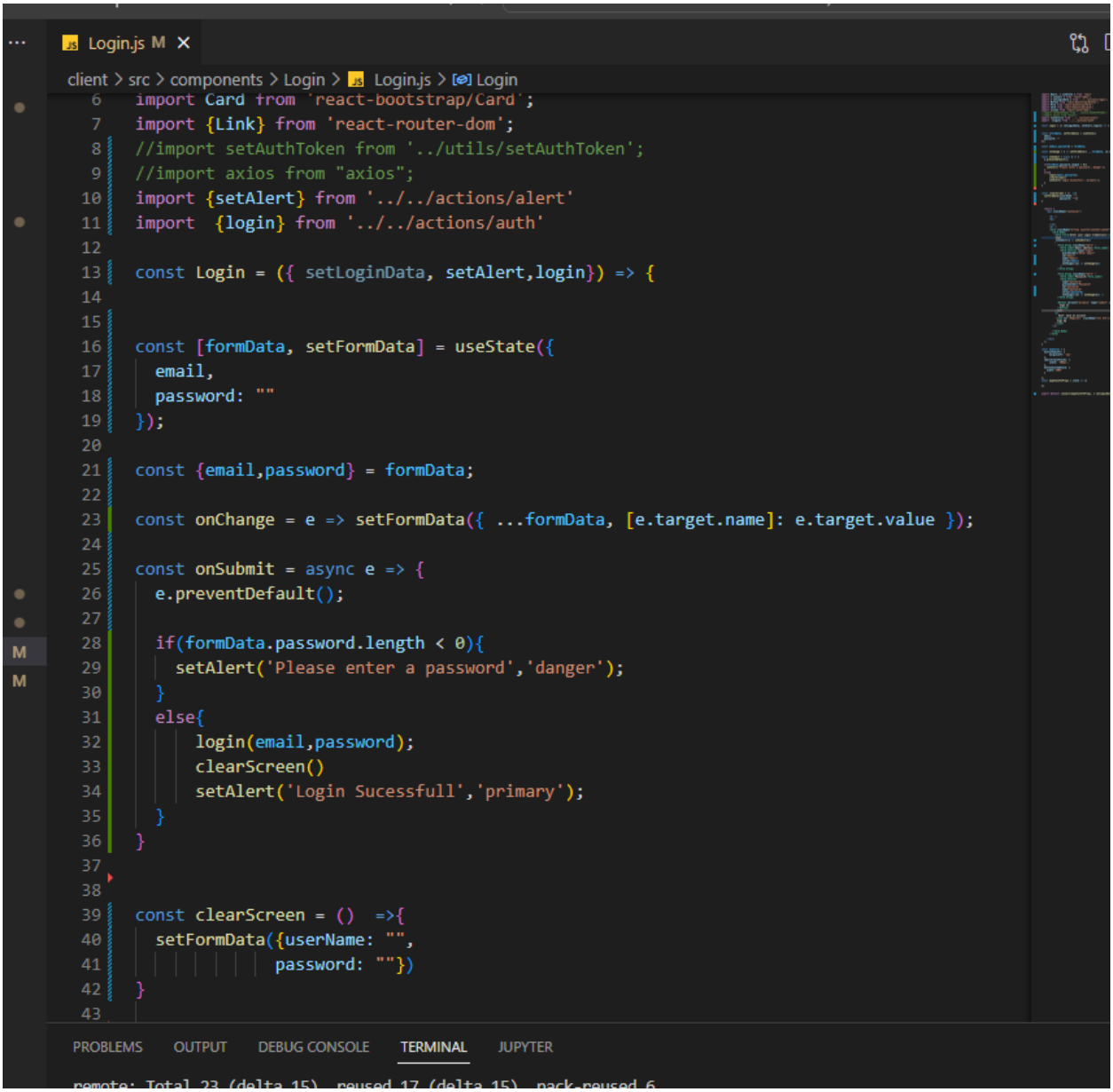
```
        loading:false,
        user:payload
    }        Jones, 3 hours ago • Version 2 Complete.
case REGISTER_SUCCESS:
    //localStorage.setItem('token',payload.token)
    return {
        ...state,
        ...payload,
        isAuthenticated: true,
        loading:false
    }
case LOGIN_SUCCESS:
```



The login screen:



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

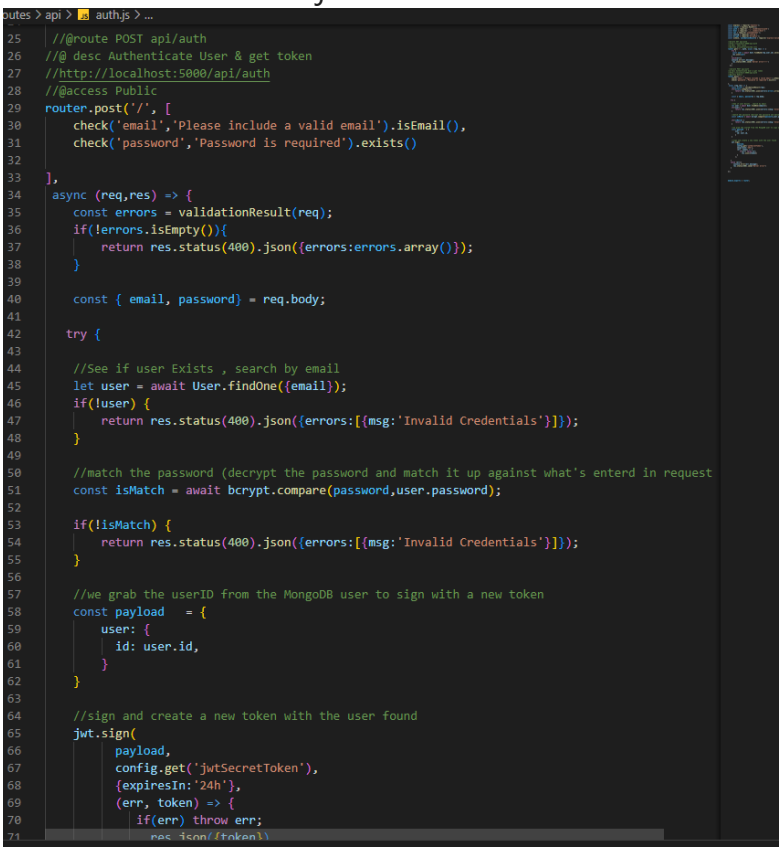

Login.js M
client > src > components > Login > Login.js > Login
import Card from 'react-bootstrap/Card';
import {Link} from 'react-router-dom';
//import setAuthToken from '../utils/setAuthToken';
//import axios from "axios";
import {setAlert} from '../../actions/alert'
import {login} from '../../actions/auth'
const Login = ({ setLoginData, setAlert,login}) => {
const [formData, setFormData] = useState({
email,
password: ""
});
const {email,password} = formData;
const onChange = e => setFormData({ ...formData, [e.target.name]: e.target.value });
const onSubmit = async e => {
e.preventDefault();
if(formData.password.length < 0){
setAlert('Please enter a password','danger');
}
else{
login(email,password);
clearScreen()
setAlert('Login Sucessfull','primary');
}
}
const clearScreen = () =>{
setFormData({userName: "",
password: ""})
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER

Calls the /auth end point

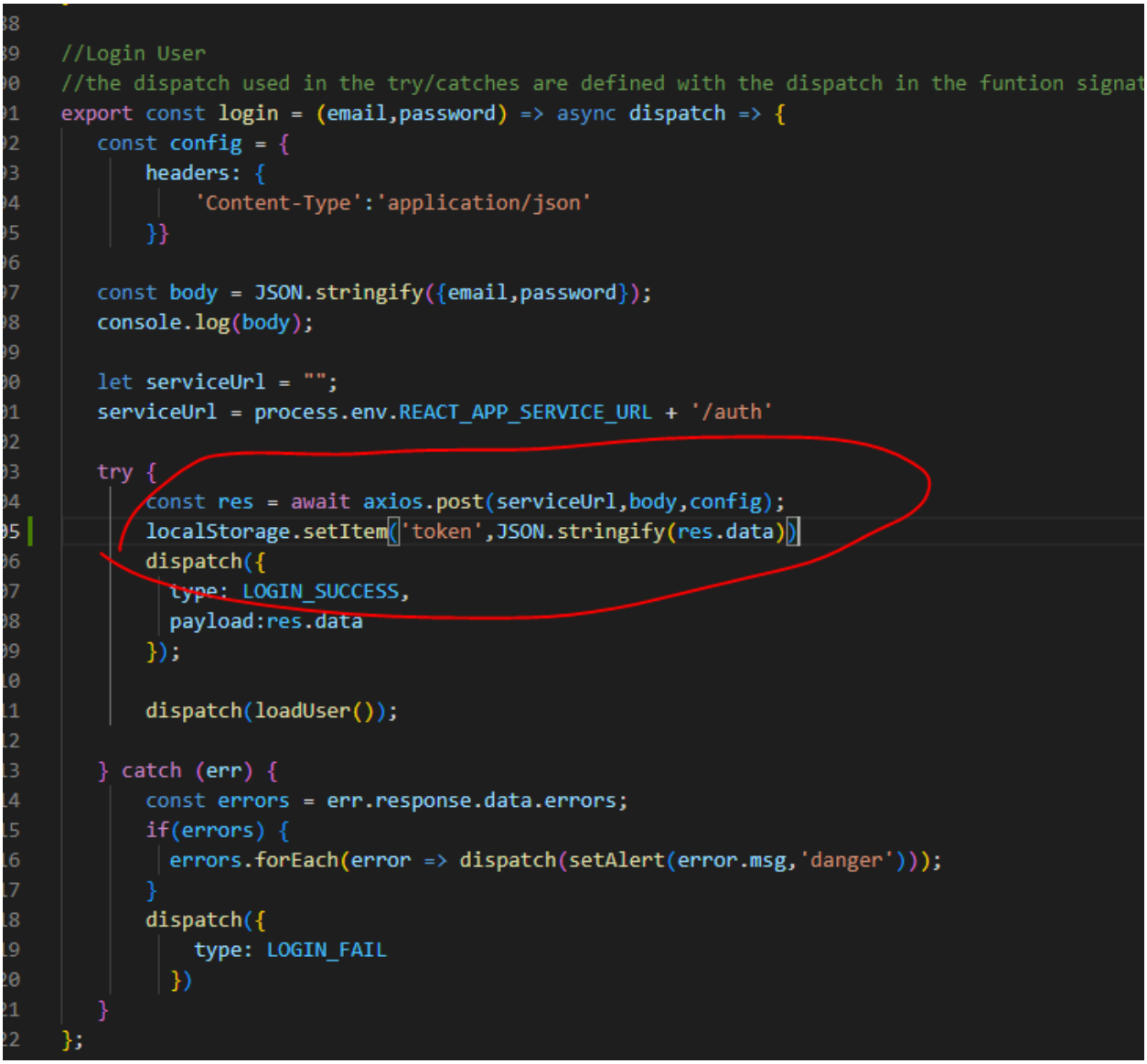
http://localhost:5000/api/auth

passes in the email and password to find the user in mongodb

Then it creates a new jwttoken



```
outes > api > auth.js > ...
25   //@route POST api/auth
26   //@ desc Authenticate User & get token
27   //http://localhost:5000/api/auth
28   //@access Public
29   router.post('/', [
30       check('email','Please include a valid email').isEmail(),
31       check('password','Password is required').exists()
32
33   ],
34    async (req,res) => {
35       const errors = validationResult(req);
36       if(!errors.isEmpty()){
37           return res.status(400).json({errors:errors.array()});
38       }
39
40       const { email, password} = req.body;
41
42      try {
43
44       //See if user Exists , search by email
45       let user = await User.findOne({email});
46       if(!user) {
47           return res.status(400).json({errors:[{msg:'Invalid Credentials'}]});
48       }
49
50       //match the password (decrypt the password and match it up against what's enterd in request
51       const isMatch = await bcrypt.compare(password,user.password);
52
53       if(!isMatch) {
54           return res.status(400).json({errors:[{msg:'Invalid Credentials'}]});
55       }
56
57       //we grab the userID from the MongoDB user to sign with a new token
58       const payload   = {
59           user: {
60             id: user.id,
61           }
62       }
63
64       //sign and create a new token with the user found
65       jwt.sign(
66             payload,
67             config.get('jwtSecretToken'),
68             {expiresIn:'24h'},
69             (err, token) => {
70                if(err) throw err;
71                  res.json({token})
```



```
//Login User
//the dispatch used in the try/catches are defined with the dispatch in the funtion signat
export const login = (email,password) => async dispatch => {
    const config = {
        headers: {
            'Content-Type':'application/json'
        }}

    const body = JSON.stringify({email,password});
    console.log(body);

    let serviceUrl = "";
    serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'

    try {
        const res = await axios.post(serviceUrl,body,config);
        localStorage.setItem('token',JSON.stringify(res.data))
        dispatch({
          type: LOGIN_SUCCESS,
          payload:res.data
        });

        dispatch(loadUser());

    } catch (err) {
        const errors = err.response.data.errors;
        if(errors) {
          errors.forEach(error => dispatch(setAlert(error.msg,'danger')));
        }
        dispatch({
            type: LOGIN_FAIL
          })
    }
};
```

Then it calls the LOGIN_SUCESS ACTION WITH THIS REDUCER

```
try {
    const res = await axios.post(serviceUrl,body,config);
    dispatch({
      type: LOGIN_SUCCESS,
      payload:res.data
    });

    dispatch(loadUser());
```

```
    }
case LOGIN_SUCCESS:
    localStorage.setItem('token',payload.token)
    return {
        ...state,
        ...payload,
        isAuthenticated: true,
        loading:false
    }
```

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

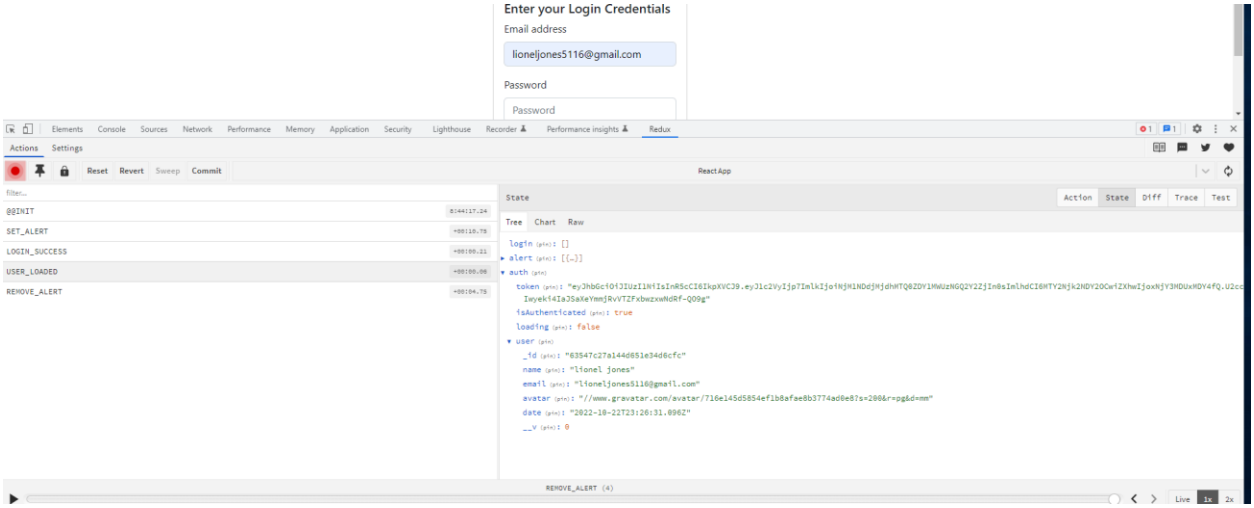
Then calls this action with loaduser() to authorize (just like the last step with register)

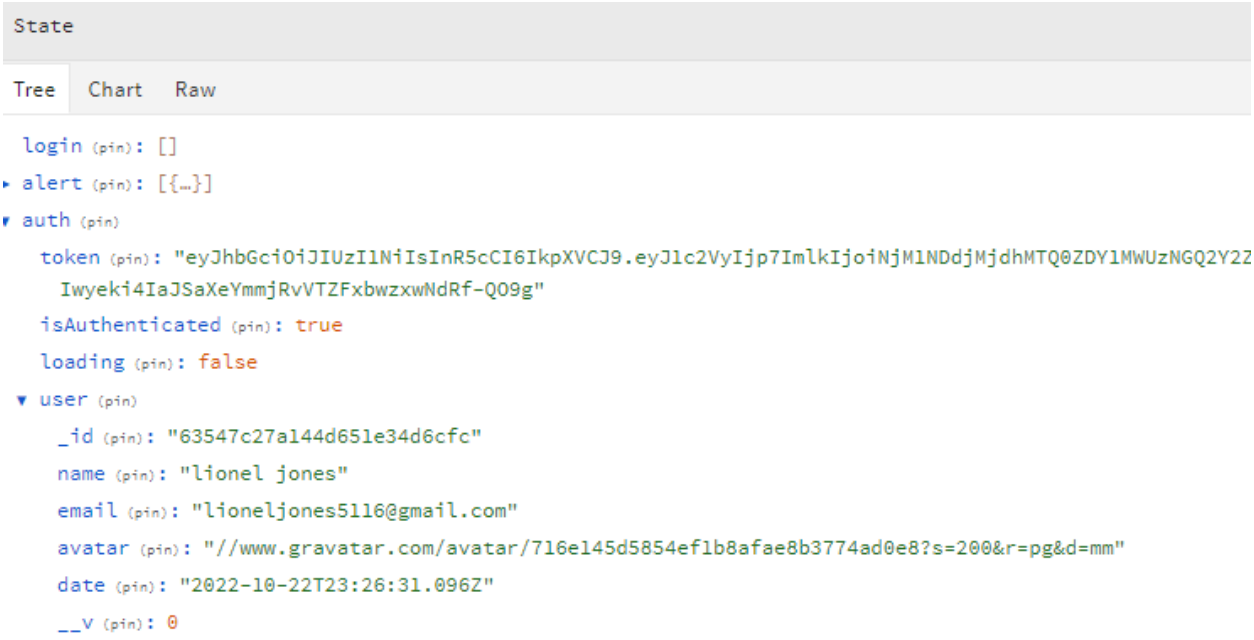
```
//Load User
export const loadUser = () => async dispatch => {

    if (localStorage.getItem('token') !== null) {
        var myToken = JSON.parse(window.localStorage.getItem('token'));
        console.log(`Token exists`);
        setAuthToken(myToken.token)
    } else {
        console.log(`Token does not exist`);
    }

    let serviceUrl = "";
    serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'


    try {
     const res = await axios.get(serviceUrl);
     setAlert('User registered','primary');
     dispatch( {
         type: USER_LOADED,
         payload: res.data
     });
    } catch (error) {
        dispatch({
            type: AUTH_ERROR
        })
    }
}
```

User loaded also brings stores the user payload in state

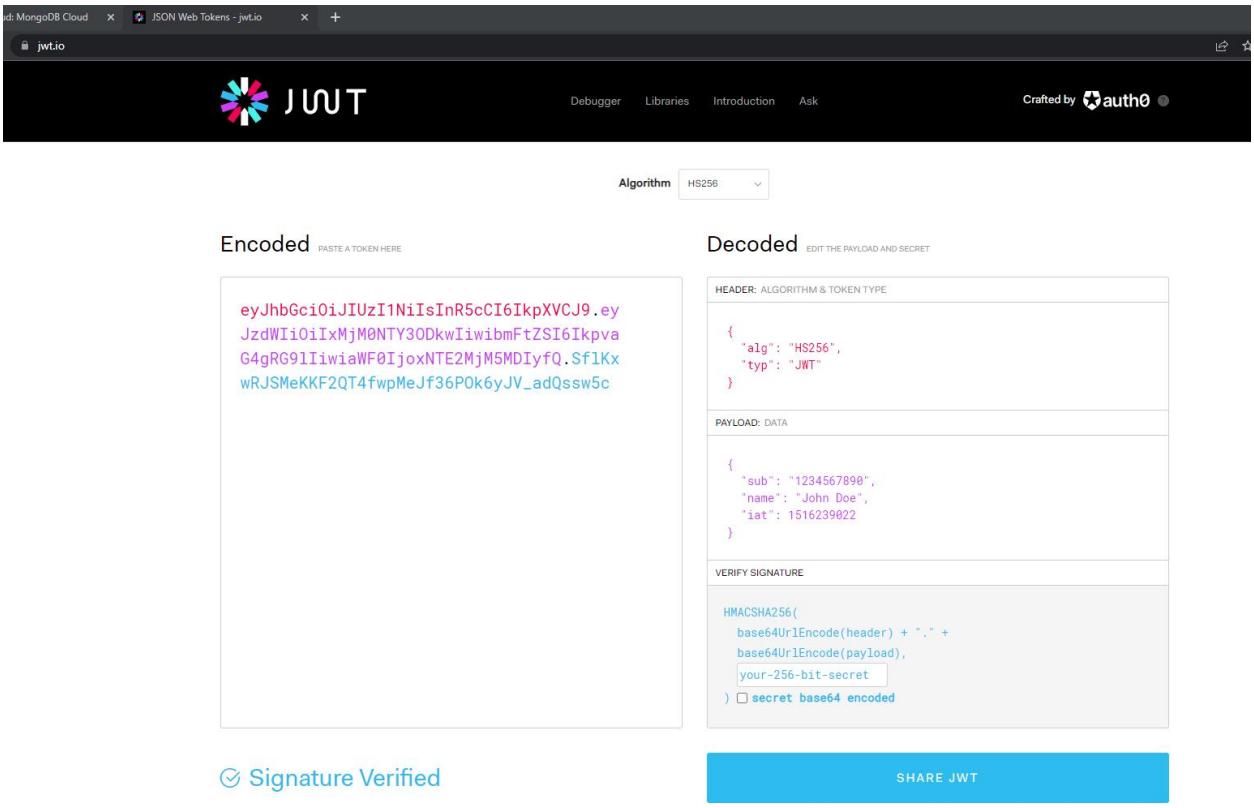
```
const {type, payload} = action

switch(type) {
 case USER_LOADED:
   return {
        ...state,
        isAuthenticated:true,
        loading:false,
        user:payload
    }
```

And we good





Tip: Using JWT for authentication
https://jwt.io/

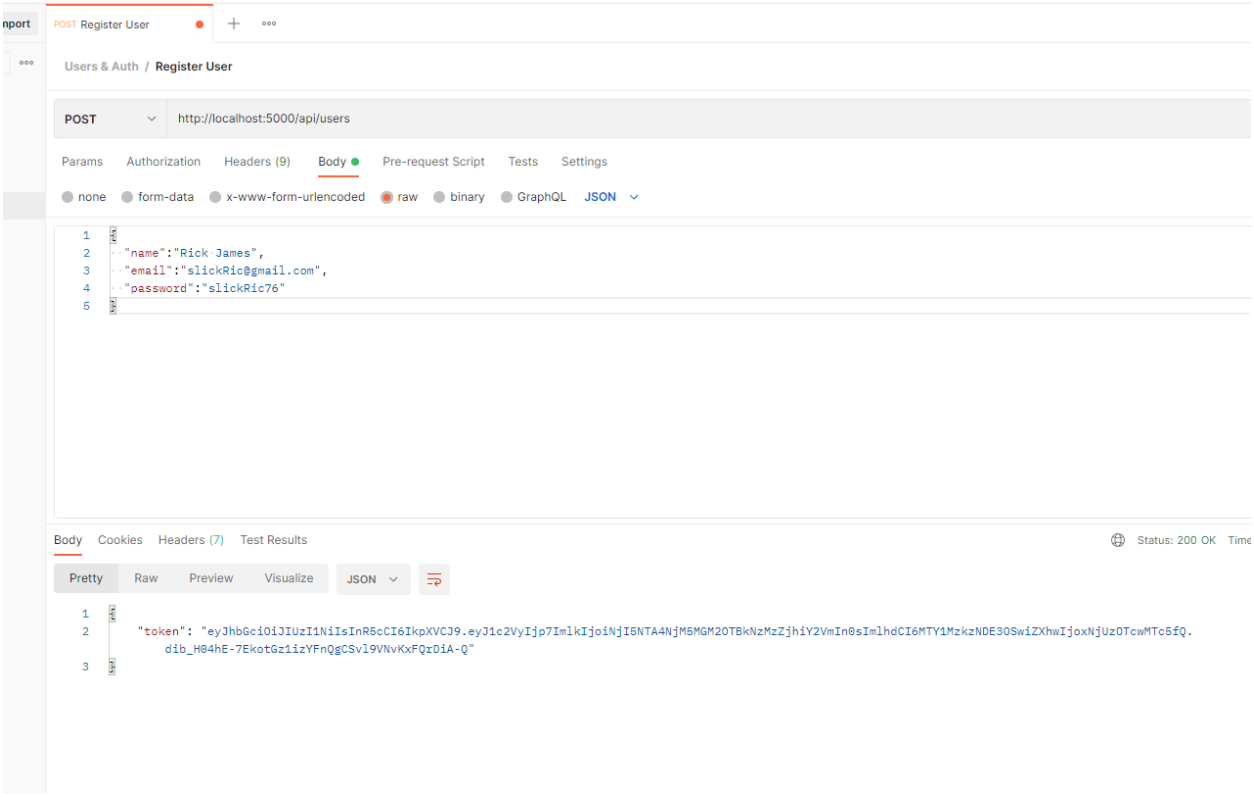


Explains what each part of the encoded parts of the token mean
Wiring it up in our project:
In our users route

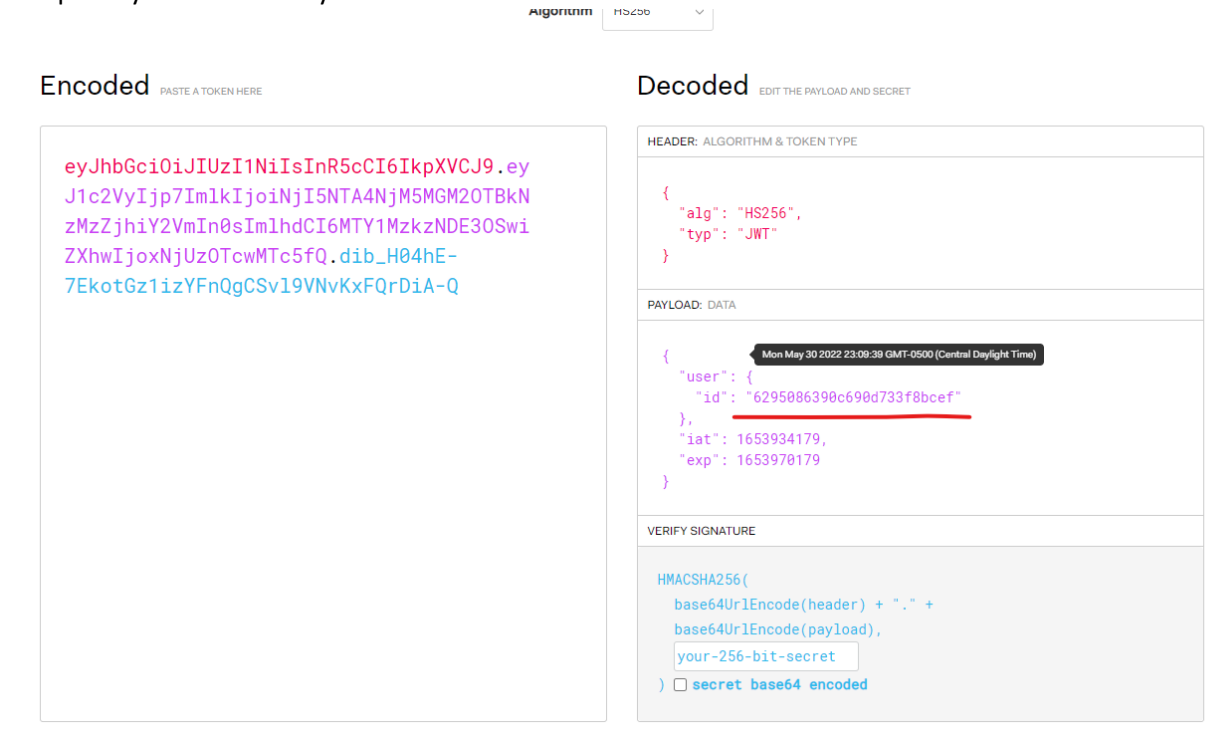
```
const jwt = require('jsonwebtoken');
const config = require('config')
```

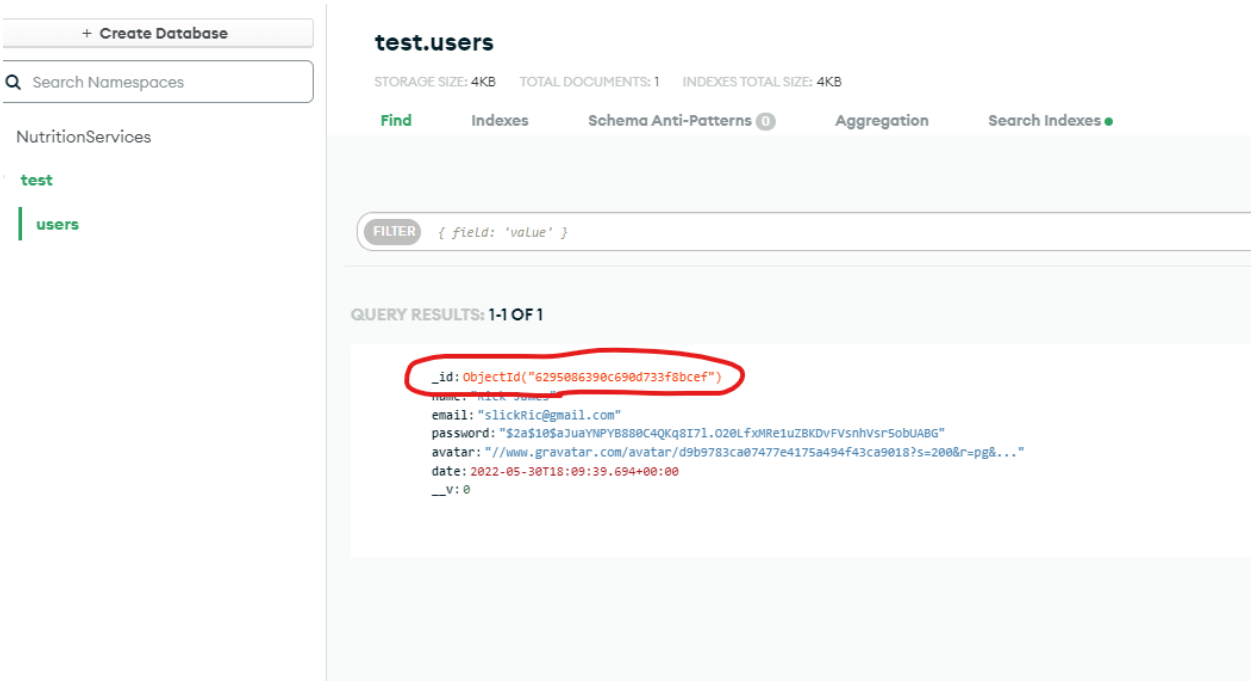
When we write the record, we get a token

```
//Return jsonwebtoken..
//create the payload to send to the jwt wire-up
//grab the id returned from the instered record _id = id (mongoose allows you to leave the _ off of the _id field)
const payload   = {
    user: {
      id: user.id,
    }
}

jwt.sign(
      payload,
      config.get('jwtSecretToken'),
      {expiresIn:36000},
      (err, token) => {
        if(err) throw err;
          res.json({token})
      }
    );
```



To check your token, go to
https://jwt.io/
And paste your token and you can see the encode and decode




+ Create Database
Search Namespaces
NutritionServices
test
users
test.users
STORAGE SIZE: 4KB
TOTAL DOCUMENTS: 1
INDEXES TOTAL SIZE: 4KB
Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
FILTER
{ field: 'value' }
QUERY RESULTS: 1-1 OF 1
_id: ObjectId("6295086390c690d733f8bcef")
email: "slickRic@gmail.com"
password: "$2a$10$aJuaYNPYB880C4QKq8I7l.O20LfxMRe1uZBKDvFVsnhVsr5obUABG"
avatar: "//www.gravatar.com/avatar/d9b9783ca07477e4175a494f43ca9018?s=200&r=pg&..."
date: 2022-05-30T18:09:39.694+00:00
__v: 0