

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 email
if(user) {
  return res.status(400).json({errors:[{msg:'User already exists'}}});
}
```

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>

```

20   "url": "https://github.com/lionel5116/Me
21 },
22   "homepage": "https://github.com/lionel5116
23   "dependencies": {
24     "bcryptjs": "^2.4.3",
25     "config": "^3.3.7",
26     "express": "^4.18.1",
27     "express-validator": "^6.14.1",
28     "gravatar": "^1.8.2",
29     "jsonwebtoken": "^8.5.1",
30     "mongoose": "^6.3.4",
31     "request": "^2.88.2"
32   },
33   "devDependencies": {
34     "concurrently": "^7.2.1",
35     "nodemon": "^2.0.16"
36   }
37 }

```

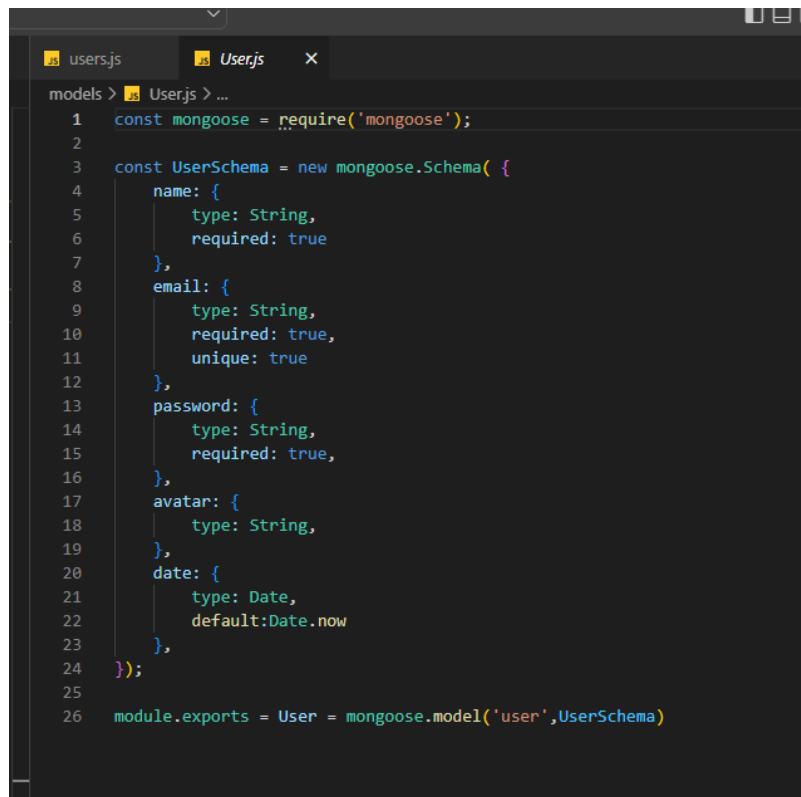
```

users.js  X
ites > 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

```

/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.



```
1  const mongoose = require('mongoose');
2
3  const UserSchema = new mongoose.Schema( {
4    name: {
5      type: String,
6      required: true
7    },
8    email: {
9      type: String,
10     required: true,
11     unique: true
12   },
13   password: {
14     type: String,
15     required: true,
16   },
17   avatar: {
18     type: String,
19   },
20   date: {
21     type: Date,
22     default: Date.now
23   },
24 });
25
26 module.exports = User = mongoose.model('user', UserSchema)
```

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

```
Registerjs X
client > src > components > auth > Registerjs > ...
41   <Fragment>
42     <section className='container'>
43       <h1 className='large text-primary'>Sign Up</h1>
44       <p className='lead'>
45         <i className='fas fa-user'></i> Create Your Account
46       </p>
47       <form className='form' onSubmit={e => onSubmit(e)}>
48         <div className='form-group'>
49           <input
50             type='text'
51             placeholder='Name'
52             name='name'
53             value={name}
54             required
55             onChange={e => onChange(e)}
56           />
57         </div>
58         <div className='form-group'>
59           <input
60             type='email'
61             placeholder='Email Address'
62             name='email'
63             required
64             value={email}
65             onChange={e => onChange(e)}
66           />
67           <small className='form-text'>
68             This site uses Gravatar so if you want a profile image, use a
69             Gravatar email
70           </small>
71         </div>
72         <div className='form-group'>
73           <input
74             type='password'
75             placeholder='Password'
76             name='password'
77             minLength='6'
78             value={password}
79             onChange={e => onChange(e)}
80           />
81         </div>
82         <div className='form-group'>
```

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 Register 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 async (req, res) => {
26   const errors = validationResult(req);
27   if(!errors.isEmpty()){
28     return res.status(400).json({errors:errors.array()});
29   }
30
31   //array destructuring - get values from body of request
32   const {name, email, password} = req.body;
33
34   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 exists'}}]);
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 coming 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)

more

Search Postman

POST http://localhost:5000/ ● GET http://localhost:5500/a ● POST http://localhost:5500/ap ● GET http://localhost:5500/a ●

Users & Auth / Register User

POST http://localhost:5000/api/users

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

```
1 {  
2   "name": "Lionel Jones",  
3   "email": "ljones876@gmail.com",  
4   "password": "lionPeace123"  
5 }
```

<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){

```

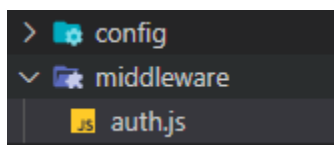
<https://jwt.io/>

You can paste in your token and get the decrypted value

The screenshot shows the JWT.io website interface. At the top, there's a navigation bar with the JWT logo and links for Debugger, Libraries, Introduction, and Auth. Below the navigation bar, there's a dropdown menu for the Algorithm, currently set to HS256. The main area is divided into two panels: 'Encoded' and 'Decoded'. The 'Encoded' panel contains a long, colorful string representing an encoded JWT token. The 'Decoded' panel shows the decoded structure of the token, including the header, payload, and signature. The header is an object with 'alg' set to 'HS256' and 'typ' set to 'JWT'. The payload is an object with 'sub' set to '1234567890', 'name' set to 'John Doe', and 'iat' set to '1516239022'. The signature is a long string of characters. Below the decoded structure, there's a 'VERIFY SIGNATURE' section with a checkbox for 'secret base64 encoded' and a 'SHARE JWT' button.

MIDDLEWARE

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



```
auth.js X
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:

```

authjs
middleware > authjs > ...
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

```

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**)

The screenshot shows two code editors side-by-side. The left editor, titled 'authjs', contains the route definition for GET /api/auth. It imports express, router, and the auth middleware. It defines a GET route that uses the auth middleware and an async function to find a user by ID and return their details. The right editor, titled 'authjs' (likely 'middleware/auth.js'), contains the middleware implementation. It imports jwt and config, and exports a function that checks for a token in the request header. If no token is present, it returns a 401 status with a message. If a token is present, it verifies the token using jwt.verify and calls next() to proceed to the route handler. If the token is invalid, it returns a 401 status with a message.

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

```

1  const express = require('express');
2  const router = express.Router();
3  const auth = require('../middleware/auth')
4  const User = require('../models/User')
5  const jwt = require('jsonwebtoken');
6  const config = require('config')
7  const bcrypt = require('bcryptjs');
8  const {check, validationResult} = require('express-validator')
9
10 //@route GET api/auth
11 //@ desc Test route
12 //@access Public
13 router.get('/', auth, async (req,res) => {
14   try {
15     const user = await User.findById(req.user.id).select('-password');
16     res.json(user)
17   }
18   catch(err){
19     console.err(err.message);
20     res.status(500).send('Server error!!!!')
21   }
22 });
23
24 //@route POST api/auth

```

The way he tests:

POST http://localhost:5000/ • GET http://localhost:5500/a • POST http://localhost:5500/api • GET http://localhost:5500/a • GET http://localhost:5500/a • GET Get auth user

Users & Auth / Get auth user

GET http://localhost:5000/api/auth

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers 6 hidden

KEY	VALUE
<input checked="" type="checkbox"/> x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm97ImkiOiNjI5NTU5NT
Key	Value

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

Remember this code:

```

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

```

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)

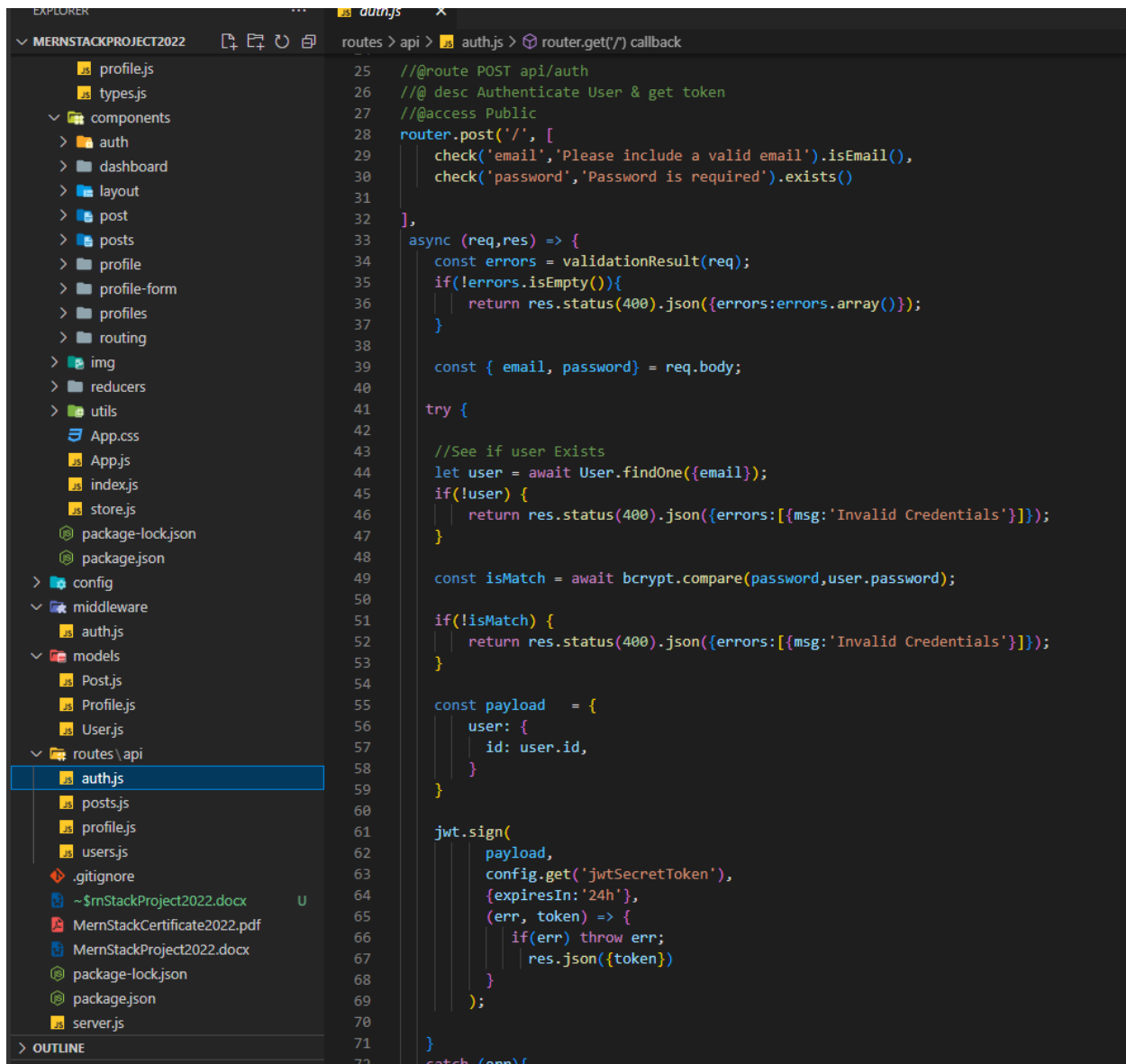
If it valid (passes the middleware)

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



```
25 // @route POST api/auth
26 // @desc Authenticate User & get token
27 // @access Public
28 router.post('/', [
29   check('email', 'Please include a valid email').isEmail(),
30   check('password', 'Password is required').exists()
31 ],
32 ),
33 async (req, res) => {
34   const errors = validationResult(req);
35   if (!errors.isEmpty()) {
36     return res.status(400).json({ errors: errors.array() });
37   }
38
39   const { email, password } = req.body;
40
41   try {
42     // See if user exists
43     let user = await User.findOne({ email });
44     if (!user) {
45       return res.status(400).json({ errors: [{ msg: 'Invalid Credentials' }] });
46     }
47
48     const isMatch = await bcrypt.compare(password, user.password);
49
50     if (!isMatch) {
51       return res.status(400).json({ errors: [{ msg: 'Invalid Credentials' }] });
52     }
53
54     const payload = {
55       user: {
56         id: user.id,
57       }
58     };
59
60     jwt.sign(
61       payload,
62       config.get('jwtSecretToken'),
63       { expiresIn: '24h' },
64       (err, token) => {
65         if (err) throw err;
66         res.json({ token });
67       }
68     );
69   } catch (err) {
```

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:

rt POST http://localhost:5000/ GET http://localhost:5500/a POST http://localhost:5500/ap GE

100 Users & Auth / Login User

POST http://localhost:5000/api/auth

Params Authorization Headers (9) Body Pre-request Script Tests Se

none form-data x-www-form-urlencoded raw binary GraphQL

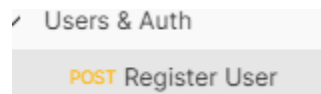
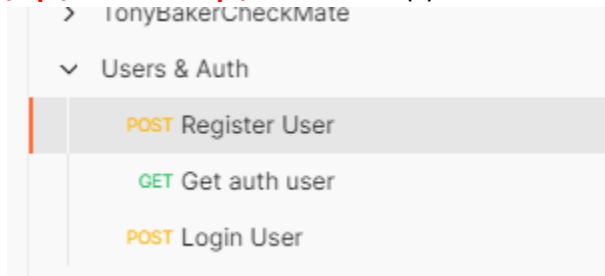
```

1 {
2   ... "email": "slickRic@gmail.com",
3   ... "password": "slickRic76"
4 }

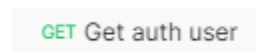
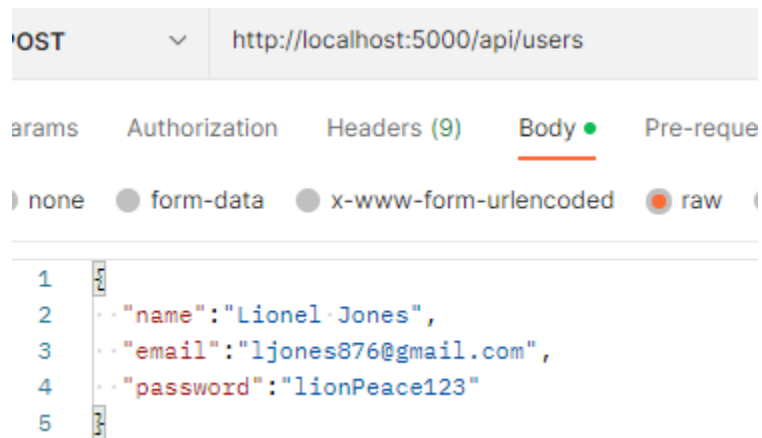
```


All of these routes for authentication are located:

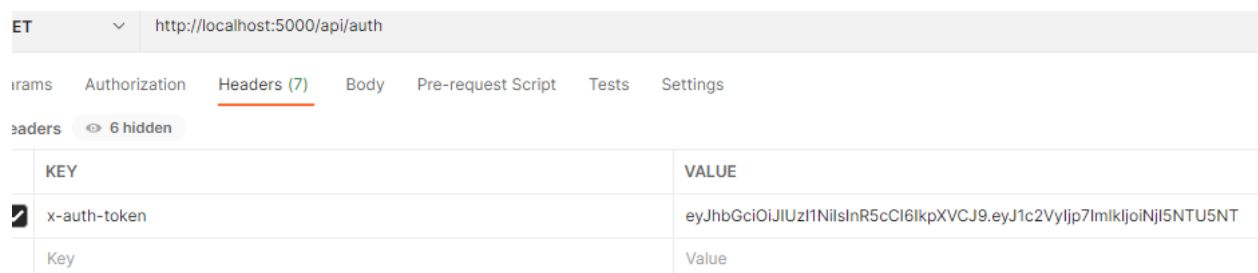
/api/auth and **/api/users** route(s)



<http://localhost:5000/api/users>



<http://localhost:5000/api/auth>



POST Login User

<http://localhost:5000/api/auth>

POST ▼ http://localhost:5000/api/auth

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

```
1 {  
2   ... "email": "slickRic@gmail.com",  
3   ... "password": "slickRic76"  
4 }
```

The register form

<https://www.udemy.com/course/mern-stack-front-to-back/learn/lecture/14555458#overview>

Enter Credentials

User Name

David Jones

Email address

davidjones@gmail.com

Password

.....

Sign Up

Already have an account ? [Sign In](#)

hthouse Recorder  Performance insights  Redux

12345678

This submits form data

```
Register.js M X
client > src > components > Login > Register.js > Register > onSubmit
13
14
15 const [formData, setFormData] = useState({
16   name: "",
17   email: "",
18   password: ""
19 });
20
21 const {
22   name,
23   email,
24   password
25 } = formData;
26
27
28 const onChange = e => setFormData({ ...formData, [e.target.name]: e.target.value });
29
30 const onSubmit = async e => {
31   e.preventDefault();
32
33   console.log(formData);
34   console.log(formData.password.length);
35
36   if(formData.password.length < 0){
37     showAlert('Please enter a password','danger');
38   }
39   else{
40     register(formData.name,
41               formData.email,
42               formData.password,);
43   }
44 }
45
```

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);

  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:

```

16   user:null
17 }
18
19 export default function authReducer (state = initialState,action) {
20   const {type, payload} = action
21
22   switch(type) {
23     case USER_LOADED:
24       return {
25         ...state,
26         isAuthenticated:true,
27         loading:false,
28         user:payload
29       }
30     case REGISTER_SUCCESS:
31       localStorage.setItem('token',payload.token)
32       return {
33         ...state,
34         ...payload,
35         isAuthenticated: true,
36         loading:false
37       }
38     case LOGIN_SUCCESS:
39       localStorage.setItem('token',payload.token)
40       return {
41         ...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     dispatch(loadUser());
67   } catch (err) {
68     console.log(err);
69   }
70 }
```

```

13
14 //Load User
15 export const loadUser = () => async dispatch => {
16
17   if (localStorage.getItem('token') !== null) {
18     console.log('Token exists');
19     setAuthToken(localStorage.token)
20     console.log(localStorage.token);
21   } else {
22     console.log('Token does not exist');
23   }
24
25
26   let serviceUrl = "";
27   serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'
28
29
30   try {
31     const res = await axios.get(serviceUrl);
32
33     dispatch( {
34       type: USER_LOADED,
35       payload: res.data
36     });
37   } catch (error) {
38     dispatch({
39       type: AUTH_ERROR
40     })
41   }
42 }

```

This method:

checks local storage for a token, if it exists

It calls the:

setAuth token method

/utils/setAuthToken.js

```

client > src > components > utils > setAuthToken.js > setAuthToken
p00149021@houstonisd.org Mag17615@7, 2 days ago | 1 author (p00149021@houstonisd.org Mag17615@7)
1 import axios from "axios";
2
3 const setAuthToken = token => {
4   if(token) {
5     axios.defaults.headers.common['x-auth-token'] = token;
6   }
7   else {
8     delete axios.defaults.headers.common['x-auth-token']
9   }
10 }
11
12
13 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()

```
5
6
7   let serviceUrl = "";
8   serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'
9
10
11  try {
12    const res = await axios.get(serviceUrl);
13
14    dispatch( {
15      type: USER_LOADED,
16      payload: res.data
17    });
18  } catch (error) {
19    dispatch({
20      type: AUTH_ERROR
21    })
22  }
23 }
```

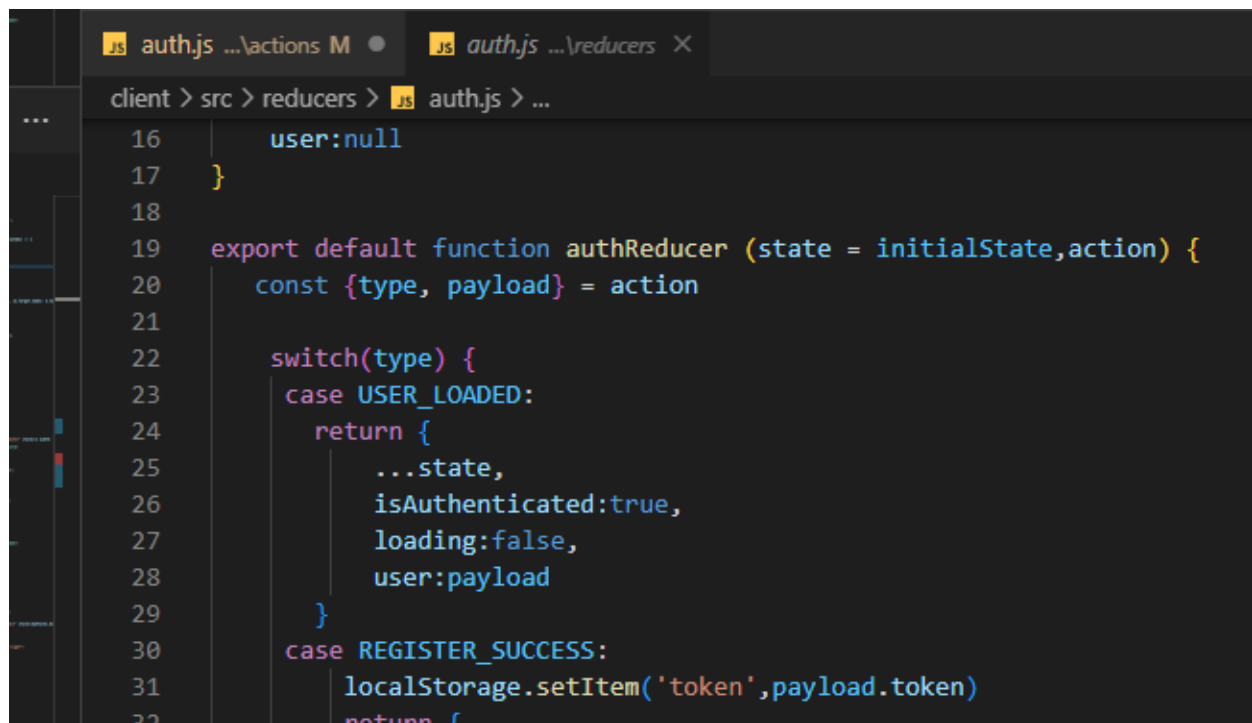
Needs to have header an entry for the token value (which was set by the setAuthToken method
<http://localhost:5500/api/auth>

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(serviceUrl);

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

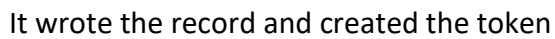


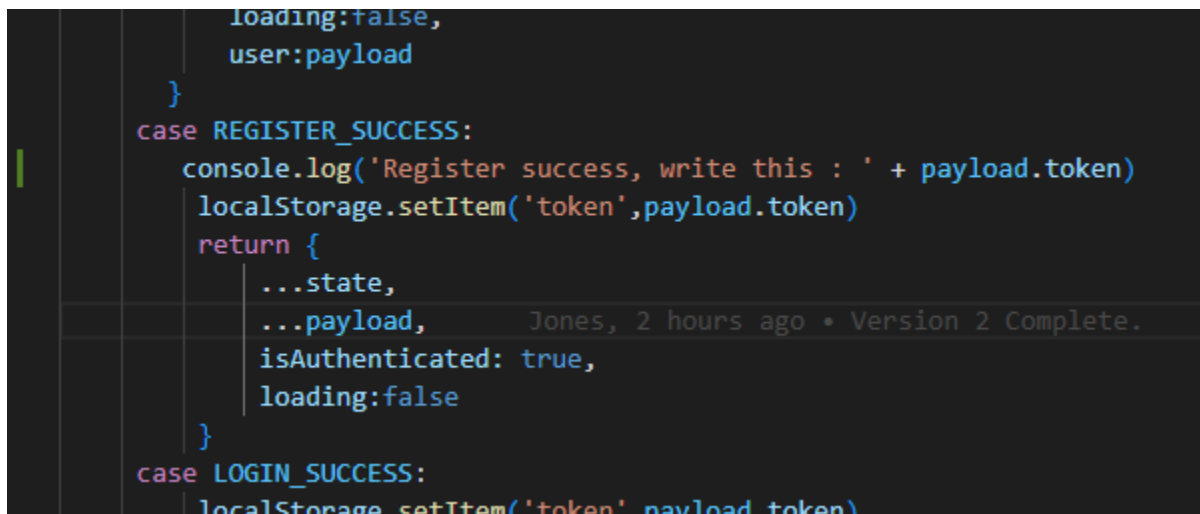
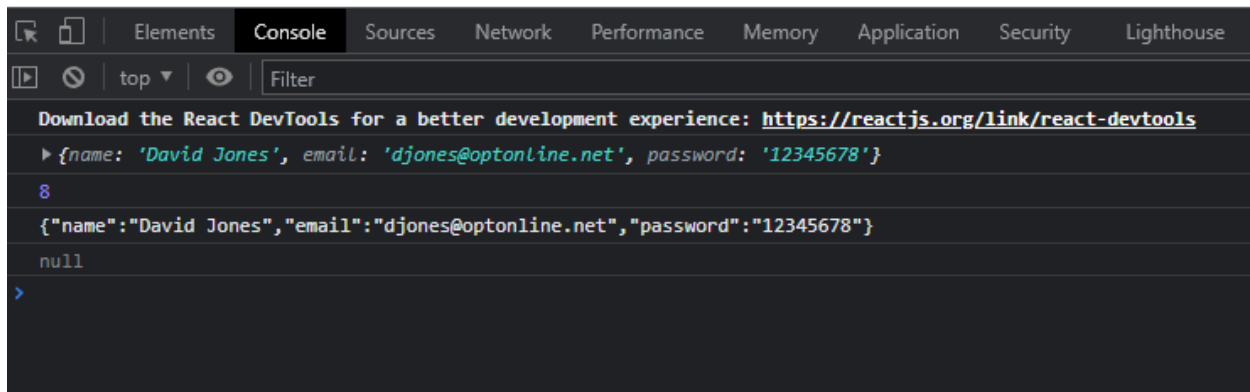
```
16     user:null
17   }
18
19   export default function authReducer (state = initialState,action) {
20     const {type, payload} = action
21
22     switch(type) {
23       case USER_LOADED:
24         return {
25           ...state,
26           isAuthenticated:true,
27           loading:false,
28           user:payload
29         }
30       case REGISTER_SUCCESS:
31         localStorage.setItem('token',payload.token)
32         return {
```

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

```
5     }
6
7     //we grab the userID from the MongoDB user to sign with a new token
8     const payload = {
9       user: {
10         id: user.id,
11       }
12     }
13
14     //sign and create a new token with the user found
15     jwt.sign(
16       payload,
17       config.get('jwtSecretToken'),
18       {expiresIn: '24h'},
19       (err, token) => {
20         if(err) throw err;
21         res.json({token})
22       }
23     );
24
25   }
26   catch (err){
27     console.error(err.message);
28     res.status(500).send("Server error")
29   }
30
31 });
32
```

I am also going to watch redux
Performing the test:





I found the issue:

<https://stackoverflow.com/questions/23805377/localstorage-getitem-logsobject-object>

nal

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'));
```

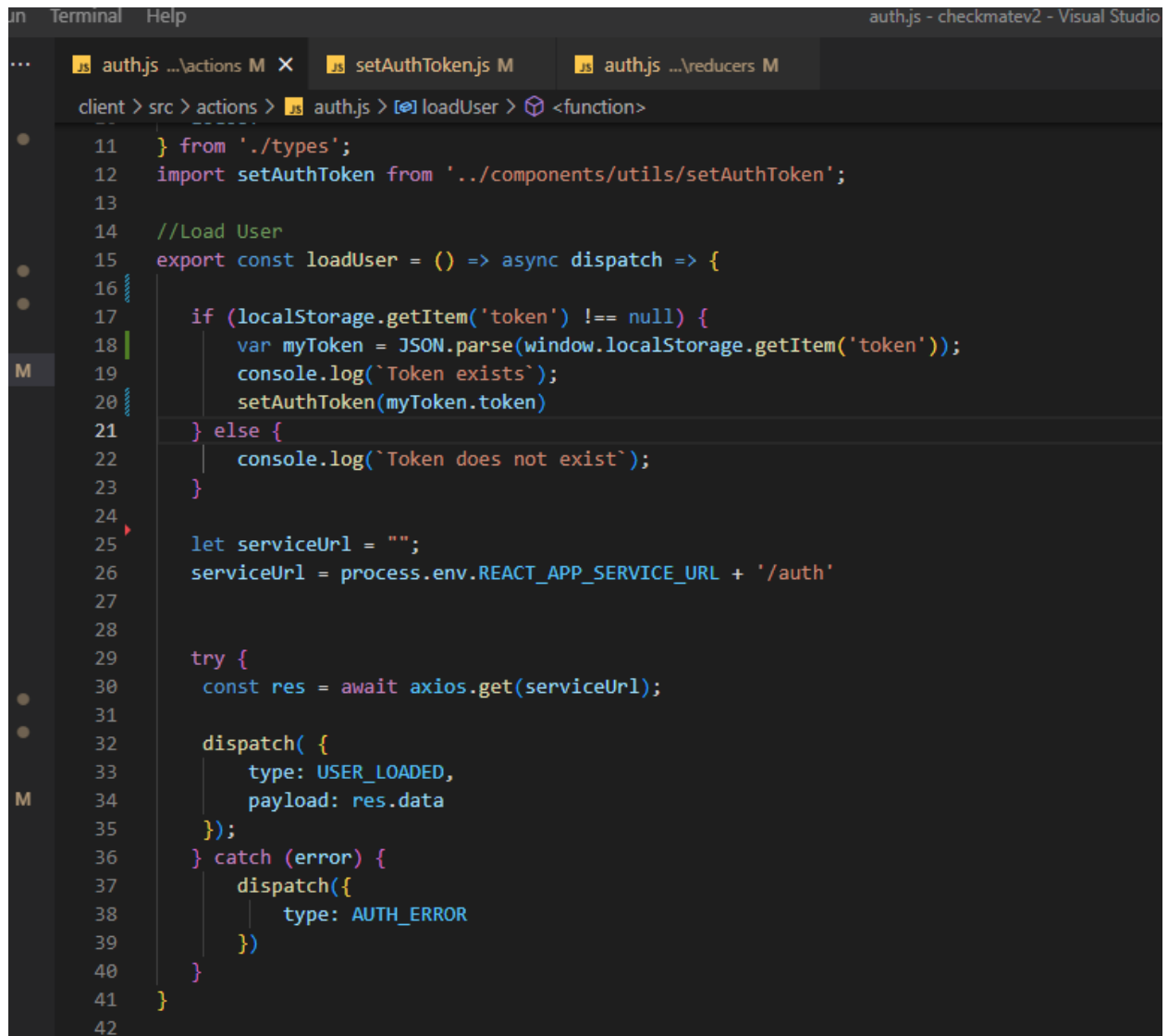
and

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

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

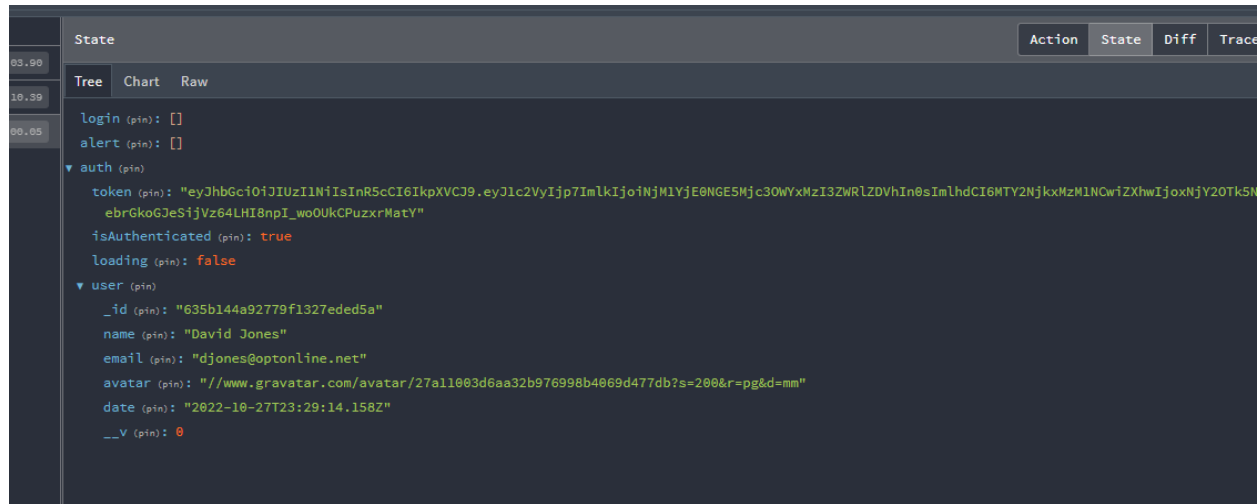
I had to change the code in:

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

The image shows a Visual Studio Code editor window with the file explorer on the left and the editor on the right. The editor is open to a file named 'auth.js' in the 'actions' directory. The code is a JavaScript function 'loadUser' that checks for a token in local storage, sets the auth token, and makes an axios GET request to a service URL. The code is as follows:

```
11 } from './types';
12 import setAuthToken from '../components/utils/setAuthToken';
13
14 //Load User
15 export const loadUser = () => async dispatch => {
16
17   if (localStorage.getItem('token') !== null) {
18     var myToken = JSON.parse(window.localStorage.getItem('token'));
19     console.log(`Token exists`);
20     setAuthToken(myToken.token)
21   } else {
22     console.log(`Token does not exist`);
23   }
24
25   let serviceUrl = "";
26   serviceUrl = process.env.REACT_APP_SERVICE_URL + '/auth'
27
28
29   try {
30     const res = await axios.get(serviceUrl);
31
32     dispatch( {
33       type: USER_LOADED,
34       payload: res.data
35     });
36   } catch (error) {
37     dispatch({
38       type: AUTH_ERROR
39     })
40   }
41 }
42
```

And as you see the entire process above works:




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

Medical Office Locations Physicians Records Managers Reps Receipts Client Records

User registered

Enter Credentials

User Name

Enter User Name

Email address

Enter email

Password

Password

Sign Up

Already have an account ? Sign In

```
onsole Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights Redux
Filter
', email: 'djones@optonline.net', password: '12345678')
", "email": "djones@optonline.net", "password": "12345678"
cIINIIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjpbImkiIjo1bmJMYVJE3NDI1MTkzNzNhbmI4t8BJ00hiIn8sIm1hdCI6MTY2ZjcxNDExNCwiZXhwIjo0NjY3MDAwNTE8FQ. aVQ8o9GLOx8VPRvipy41lgX-4L049X9K19-aoVzLs1Y
```

Tip: Using JWT for authentication

<https://jwt.io/>

The screenshot shows the JWT.io website interface. At the top, there's a navigation bar with the JWT logo, links for Debugger, Libraries, Introduction, and Ask, and a note 'Crafted by auth0'. Below the navigation bar, there's a section for encoding and decoding JWTs. The 'Encoded' section on the left has a text area containing a JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c`. The 'Decoded' section on the right shows the token's structure:
- **HEADER:** `{ "alg": "HS256", "typ": "JWT" }`
- **PAYLOAD:** `{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }`
- **VERIFY SIGNATURE:** `HMACSHA256(base64UrlEncode(header) + ".", base64UrlEncode(payload), your-256-bit-secret)`
Below the encoded token, there's a 'Signature Verified' status. At the bottom right, there's a 'SHARE JWT' button.

Explains what each part of the encoded parts of the token mean

Wiring it up in our project:

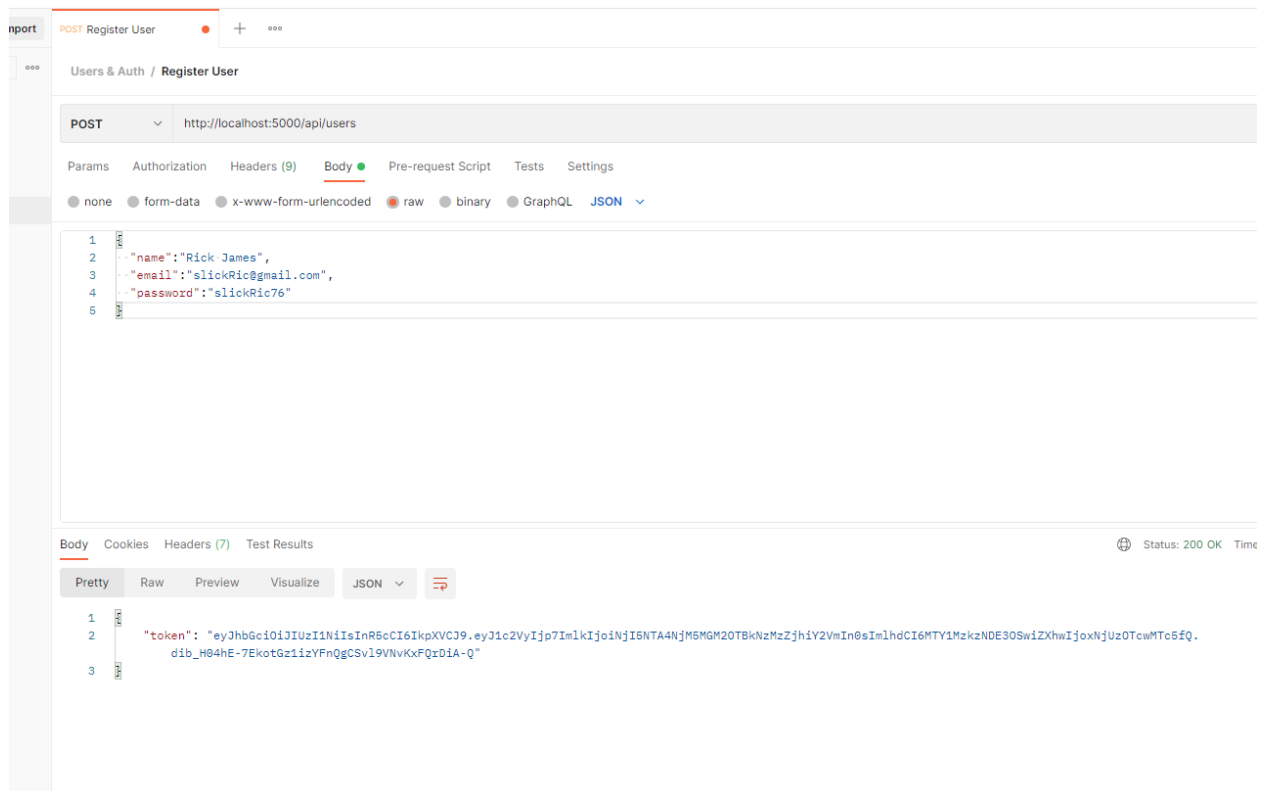
In our users route

```
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

Algorithm HS256

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7Im1kIjoiaW5NTA4NjM5MGY2OTBkNzZzZjhiY2VmIn0sIm1hdCI6MTY1MzkzNDE3OSwiZXhwIjoxNjUzOTcwMTc5fQ.dib_H04hE-7EkotGz1izYFnQgCSv19VNvKxQrDiA-Q
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "user": {    "id": "6295086390c690d733f8bcef"  },  "iat": 1653934179,  "exp": 1653970179}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret)
```

☐ secret base64 encoded

+ 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"),  "name": "Rick Sanchez",  "email": "slickRic@gmail.com",  "password": "$2a$10$a3uaYNPY8880C4QKq8I71.020LfxHRe1uZBKDVfVsnhVsr50bUABG",  "avatar": "https://www.gravatar.com/avatar/d9b9783ca07477e4175a494f43ca9018?s=200&r=pg&...",  "date": 2022-05-30T18:09:39.694+00:00,  "__v": 0}
```