Tips:  
**Tip**: Running both Server.js and React command with one command (**npm run dev**)  
**Tip**: Run commands using script  
**Tip**: Setting up middleware using express.js  
**Tip**: Checking for Validation of payload  
**Tip**: Exporting and importing modules Node.js  
**Tip**: Creating a new user record (document) using mongoose DB  
**Tip**: Using JWT for authentication  
**Tip**: Creating a protected route with your JWT token (**more on middleware**)  
**Tip**: Logging in with email address and password an validating against our token   
**Tip**: Using POSTMAN  
**Tip**: When to use AUTH middleware and not use  
**Tip**: Getting a new token for my user (Test User – lioneljones)  
**Tip**: Routing with Express.js  
**Tip**: Registering an application on GitHub for development  
**Tip**: The way routing works in this version of reactjs  
**Tip**: How to create the (correct) functional component using react tools vs code  
**Tip**: Writing to our mongoDB database with a two project application (Reactjs and NodeExpress), using a proxy as well  
**Tip**: Wiring up Redux and DevTools  
**Tip**: How he laid out REDUX (actions,Reducers,Constants)  
**Tip**: Adding a section html tag to your APP.js file  
**Tip**: state.filter() finally explained  
**Tip**: Client Side Validation using HTML5  
**Tip**: Using Javascript localStorage object  
**Tip**: Working with JWT and the token (when a user logs back in after being authenticated)

MernStackProject2022  
This coarse project is from Udemy Course  
<https://www.udemy.com/course/mern-stack-front-to-back/>

Tools  
React Developer Tools  
<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi/related?hl=en>  
  
Redux Developer Tools  
<https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd/related?hl=en>  
  
After you install them, add them pin them  
A screenshot of a computer screen

Description automatically generated with medium confidence  
  
Mongo DB Atlas  
<https://cloud.mongodb.com/v2/624348b28c94b80563a3c46e#clusters>  
PE  
BFLU@  
  
Mongoose Connector  
<https://mongoosejs.com/>  
<https://www.npmjs.com/package/mongoose>  
  
Git repo  
<https://github.com/lionel5116/MernStackProject2022.git>  
  
echo "# MernStackProject2022" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/lionel5116/MernStackProject2022.git

git push -u origin main

git add .  
git add -A  
git commit -m "first commit"  
git push -u origin main  
  
**Dependencies**  
npm i express express-validator bcryptjs config gravatar jsonwebtoken mongoose request

Below is for allowing use to run react and express.js at the same time

npm i -D nodemon concurrently  
  
We added this to our package.json file (script)

Text

Description automatically generated  
  
To run our server, type  
npm run server (as shown in our script above)  
  
Text

Description automatically generated  
  
Go to Postman  
Graphical user interface, text, application, email

Description automatically generated  
  
Connecting to the MongoDB Database:  
Graphical user interface, text, application, email

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

Description automatically generated  
  
mongodb+srv://lionel5116:<password>@cluster0.jwcnt.mongodb.net/?retryWrites=true&w=majority  
  
For special characters in your password

https://www.mongodb.com/docs/atlas/troubleshoot-connection/#special-characters-in-connection-string-password

**mongodb+srv://lionel5116:Mag17615%40@cluster0.jwcnt.mongodb.net/?retryWrites=true&w=majority**

See project for initial wireup of connection  
And run it:  
A screenshot of a computer

Description automatically generated with medium confidence  
  
And that’s all for the initial setup  
  
Creating the React APP  
npx create-react-app client  
  
And now we have our react app  
Graphical user interface, text, application, chat or text message

Description automatically generated  
  
**Tip**: Running both Server.js and React command with one command  
To run both the node server and react at the same time with one command:  
Graphical user interface, text, application

Description automatically generated  
  
In your root package.json file add:  
Text

Description automatically generated  
  
Then run the following command: (at the root application folder)  
npm run dev  
  
Text

Description automatically generated  
  
And it works!!!!

Next cd into your client folder and install the following dependencies  
npm i axios react-router-dom redux react-redux redux-thunk redux-devtools-extension moment react-moment  
  
**Tip**: Adding a proxy  
Add this to your package.json file in the client  
Text

Description automatically generated  
  
**Tip**: Run commands using script  
Remember in our script entry in package.json  
Text

Description automatically generated  
  
Our command to run our app always starts:  
npm run <script command >  
So to run our server (just the node.js)  
npm run server (as opposed to node server.js)  
Node looks at the “main”: “server.js” tag to determine node.js app’s entry point  
Running this command utilizes “nodemon” with watches for changes in our code, this way when we make a change, node recompiles

**Tip**: Setting up middleware using express.js  
Text

Description automatically generated  
  
**Middleware**:

Middleware functions are functions that have access to the request object ( req ), the response object ( res ), and the next middleware function in the application's request-response cycle.  
Middleware checks for validity of a post’s payload. Normally used for authentication  
  
**Tip**: Checking for Validation of payload  
In addition to added your middleware code (shown in the tip above)  
Graphical user interface, text

Description automatically generated  
  
Then when perform a post that is invalid:  
Graphical user interface, text, application, email

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

Description automatically generated

**Tip**: Exporting and importing modules Node.js  
A screenshot of a computer

Description automatically generated with medium confidence  
When you hover over the import using the require .. keyword

A screenshot of a computer

Description automatically generated with medium confidence  
  
Notice how **internally** it uses the **import** keyword (just like in react/angular)

**Tip**: Creating a new user record (document) using mongoose DB  
The standard way that most developers create documents in a mongoDB is to use mongoose. The first thing you do is create a schema:  
Text

Description automatically generated  
  
Next grab an import  
A screenshot of a computer

Description automatically generated with medium confidence  
  
Then as show below  
We get the request body from the POST  
We create an instance of the user schema, set the values  
Then to write to our database, we use the command  
<await> user.save() (user.save is a promise so we use the await keyword)

Text

Description automatically generated

So before saving:  
We only have one database (this was a database I created for another project)  
Graphical user interface, email

Description automatically generated  
  
After we do our post  
Graphical user interface, text, application, email

Description automatically generated  
  
MongoDb creates a database called test, then it creates create’s the user document:

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

**Tip**: Creating a protected route with your JWT token (more on middleware)  
  
**Middleware**:

Middleware functions are functions that have **access to the request object** ( ***req*** ), ***the response object*** ( res ), and the next middleware function in the application's **request-response** cycle.  
**Middleware checks for validity of a post’s payload**. **Normally used for authentication**  
  
Below we created a middleware folder and a .js file to handle decoding our jwt token

Then in our auth.js route, we protect the route with the auth.js middleware code  
A screenshot of a computer

Description automatically generated with medium confidence  
To test it out, we grab the token that we created when we created the first user  
Graphical user interface, text, application

Description automatically generated  
  
We add a header with key that we entered in our middleware and cut and paste the jwt token in the value, when we hit the URL, we get Auth route  
  
If we change the token we get:

Graphical user interface, text, application, email

Description automatically generated  
  
Getting the user information

If you change the code in the auth.js route

Text

Description automatically generated

Working with middleware and passing in the token in the get request, the middleware will decode the token and grab the user from the response and return the user object that is tied to the token (see the code below) …(sweeetttttt!!!). This is classic “MIDDLEWARE………….”  
Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated  
  
**Tip**: Logging in with email address and password an validating against our token  
In the auth.js file, we add another route (POST) to pass in a payload with our email address and password to authenticate, see the auth.js POST route  
  
Text

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

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

Description automatically generated  
  
**Tip**: Using POSTMAN  
Postman is good for saving all of your previous posts / requests for testing  
Graphical user interface, text, application, email

Description automatically generated  
  
You can also use presets for things that you enter a lot

Graphical user interface, text, application, email

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

Description automatically generated  
  
**Tip**: When to use AUTH middleware and not use  
Whenever you are using a Token (a protected route), you have to make sure you include your “auth” middleware in your route (this is typically for what we call a “private” ->protected route  
Text

Description automatically generated  
//For public routes (non-protected), where we are not using a token, we don’t need to pass in middleware  
Text

Description automatically generated  
  
A screenshot of a computer

Description automatically generated with medium confidence

**Tip**: Getting a new token for my user (Test User – lioneljones)  
  
My Credentials I used to create my user  
{

  "name":"Lionel Jones",

  "email":"ljones876@gmail.com",

  "password":"lionPeace123"

}

Go to POSTMAN

{

    "email":"ljones876@gmail.com",

    "password":"lionPeace123"

}

Graphical user interface, text, application, email

Description automatically generated  
  
It will return a token  
Graphical user interface, text, application, email

Description automatically generated  
  
To check for user with Token  
Graphical user interface, text, application, email

Description automatically generated

**Tip**: Routing with Express.js  
<https://expressjs.com/en/guide/routing.html>

The way routing works in express.js  
Routing is part of the express.js package  
Text

Description automatically generated  
  
First you create a .js file (i:e **Profile.js**)  
You include these require(imports)

Text

Description automatically generated  
You create your default route:  
A screenshot of a computer

Description automatically generated with medium confidence  
  
Then export the route

A picture containing text

Description automatically generated

Notice how we don’t export the name of the file, just the convention as shown above to create the expose the .js file as a route in application. And based on the /<.js filename>, we determine how to reference the URL. /api/<.js filename>/

By default, when you browse the url, you include the /api/  
Then the name of the .js file (without the of course) to complete the route as shown below:  
<http://localhost:5000/api/profile>

Graphical user interface, text, application, email

Description automatically generated

**Tip**: Registering an application on GitHub for development  
<https://github.com/settings/developers>

Go to the link above to register an application to use in your custom react app.  
A screenshot of a computer

Description automatically generated with medium confidence  
  
We then use this in our custom application to return all of a user’s github repos   
Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Tip**: The way routing works in this version of reactjs  
Text

Description automatically generated

**Tip**: How to create the (correct) functional component using react tools vs code

**RAFCE – but use RCFE for default components**  
Text

Description automatically generated

**Tip**: Writing to our mongoDB database with a two project application (Reactjs and NodeExpress), using a proxy as well  
Notice how we have “2” projects

A screenshot of a computer

Description automatically generated with medium confidence  
  
Our Client (our React Application) and the rest is our Node.js project

In our package.json file we have a “proxy” entry

Text

Description automatically generated  
  
When we run

A screenshot of a computer

Description automatically generated with medium confidence  
It fires up both projects  
So when we write the code as shown below:

Text

Description automatically generated  
  
Since we have a proxy of <http://...:500>.., it knows to look at the package.json file for the endpoint

When we submit

Graphical user interface, text, application, email

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

Description automatically generated

It wrote the record

**Tip**: Wiring up Redux and DevTools

<https://github.com/zalmoxisus/redux-devtools-extension>

We wired up REDUX devtools to work with Chrome using and npm package that we installed earlier.   
<https://www.npmjs.com/package/redux-devtools-extension>

­­ Graphical user interface, application

Description automatically generated

The way that we wired it up was when we created our store:

Text

Description automatically generated  
Now when we look at our dev-tools and select REDUX, you see that the tools are working (It’s much cleaner than the documentation given by github). (The old way of wiring it up).  
A picture containing text, screenshot, indoor, electronics

Description automatically generated  
  
  
The way he wired up redux:  
He wired up the first part (**the same way as traditional**)  
Create a store.js file, create reducers, pass reducers to the store, in the app.js file he imports the provider and wraps all of the content inside of it and set’s the store, Then  
He created an types.js file (really for CONSTANTS)  
Graphical user interface, text, application

Description automatically generated  
Then he created a reducer  
Text

Description automatically generated  
  
Then he created an actions folder, inside he creates his actions that he dispatches on the store  
A screenshot of a computer

Description automatically generated with medium confidence  
  
These two files work together:  
When he dispatches an action, he set’s the action type, the reducer receives the type from the dispatched action and passes along the payload to set the state.  
  
In our component(i:e register)  
We import the action (ie: setAlert)  
We pass down props to our component (the same as map state to props)

Text

Description automatically generated  
At the export of the component, we wrap our component with connect:

Text

Description automatically generated  
Then we dispatch an action  
Text

Description automatically generated  
  
If we look at our REDUX tools:  
A screenshot of a computer

Description automatically generated  
It shows our global state tree and actions dispatched

The alert state you see below (is our reducer)  
Graphical user interface, application

Description automatically generated  
Comes from our reducer .js name  
A screenshot of a computer

Description automatically generated  
  
  
  
  
A screenshot of a computer

Description automatically generated with medium confidence  
  
  
The way he wired up alert was to create an alert component that tapped into the redux store:  
Text

Description automatically generated  
Then in our App component:  
Text

Description automatically generated  
  
This alert component can be accessed from anywhere in our app, just like our navigation, and when we write to the store, it will return information from our store from the alert reducer.  
Graphical user interface, text, application, email

Description automatically generated  
  
Sweet!!!!  
He also wired up the alert to disappear after 5 seconds  
A screenshot of a computer

Description automatically generated with medium confidence  
He dispatches another action to send in a different payload ID and filters (removes) where the ids do not match in the state  
A screenshot of a computer

Description automatically generated with medium confidence  
So the alert does not need a close button.

**Tip**: Adding a section html tag to your APP.js file

Text

Description automatically generated  
  
This was needed for components that you could not add a section to (Alert was erroring out with the key needed for the map function), so I added a section html tag and placed the component inside of it to get the component to render in a container.

**Tip**: state.filter() finally explained  
The filter method here is just the standard javascript.filter method  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter>  
The filter() method creates a new array with all elements that pass the test implemented by the provided function.  
A screenshot of a computer

Description automatically generated with medium confidence  
  
So here in the reducer  
Text

Description automatically generated  
  
When he calls this method, it passes in the exact uuid in the payload as what is passed into state, to alert.id != payload = false , because they are the same, therefore returning an empty state.  
Text

Description automatically generated  
Example here in javascript  
Text

Description automatically generated  
The statement proves false, therefore returning an empty array.  
  
**Tip**: Client Side Validation using HTML5  
Text

Description automatically generated

**Tip**: Using Javascript localStorage object

You can use Javascript’s built-in localStorage object to store information

Text

Description automatically generated

**Tip**: How he laid out REDUX (actions,Reducers,Constants)  
The way that he laid out REDUX is the same way that our projects at HISD (the five apps) are laid out.  
The neat thing is, for each component, he named them the same in the actions,reducers and component folders.  
A screen shot of a computer

Description automatically generated with low confidence  
  
This makes it easy find the appropriate reducer and action from a project standpoint.

For his actions:  
He places all of the method calls (endpoints) inside of an action and calls them from the UI component.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Then in the action, he dispatches the payload (calls the reducer) to pass in the payload and store it into the store.  
Text

Description automatically generated  
He also stores the payload in javascripts localStorage object as well  
Text

Description automatically generated

**Tip**: Working with JWT and the token (when a user logs back in after being authenticated)

A screenshot of a computer

Description automatically generated with medium confidence  
  
The way authorization works:  
When the user is registered, we write the token to javascript local storage in the reducer (auth reducer)  
When the application refreshes, we place a call to an action called loadUser, what loadUser does is check javascript’s local storage for a key ‘token”, if the key exists, we write it to our axios default header that we set up in a file called utils.js.

When a call to axios the /api/auth endpoint that has the token written to our header for authorization. When this happens, the user is authorized.

A screenshot of a computer

Description automatically generated  
  
One thing that I did do is place the time to expire to ‘24h’

Text

Description automatically generated  
  
A screenshot of a computer

Description automatically generated