**React JS Chat Application Notes**

* <https://www.youtube.com/watch?v=Bv9Js3QLOLY>
* We will learn about, routing, social authentication, firebase, react context, rest api’s, environmental variables
* We will also use Chat Engine

**Starter Code**

* Go to <https://github.com/adrianhajdin/unichat-course>
* Click code and download as a zip
* Put zip on desktop and extract it
* Go inside the extracted folder and bring out the unichat-course-master folder which is what we will work in
* Open the folder in vs code
* Run npm install (this will install all the necessary dependencies)
* Text

  Description automatically generated
* When you get to here, run npm start
* Text

  Description automatically generated
* We should see an empty html page
* Go to the files and this is the quick rundown:
  + index.js is very basic and doesn’t have much and all it does it render the App component (which we have to make) and import the index.css file
  + App.js has some basic imports from react-router-dom. It also has imports we will need in the future that are currently commented out.

**Creating Login**

* Go to App.js
* Inside the return, we have a div which has a router component. Inside the Router, we have a Switch component which tells react to render one of the Route components we can see. Thus, we will either render the login form or the chats component
* Uncomment <Route path="/" component={Login} />
* Now, if we have a path of ‘/’, then react will render the login component
* To use the login component, uncomment its import and create a new Login.js file inside the components folder
* Go to Login.js and import React from ‘react’ and {GoogleOutlined, FacebookOutlined} from '@ant-design/icons'
* In the return of the Login function, we have the div which renders a header, and two divs.
* The first of the two divs renders a GoogleOutlined component which allows users to sign in via their google account
* The second of the two divs renders a FacebookOutlined component which allows users to sign in via their facebook account

Text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Firebase

* Now we want to create our firebase, connecting it to our chat, and making our app work
* Go to <https://firebase.google.com/>
* Create a new project, disable google analytics, and create the project
* Click the settings icon beside the project overview tab and click project settings
* Scroll down and click the </> icon
* Name the app name unichat and toggle on ‘set Firebase Hosting’ and click register app
* Keep clicking next and click continue to console
* Still in project settings, scroll down and find the sdk setup and config tab
* Click the config toggle and copy the code
* Graphical user interface, text, application

  Description automatically generated
* Inside the src folder, we will create a firebase.js file
* import firebase from "firebase/app" which is inside our node modules which we installed using npm install
* import "firebase/auth"; (just do it)
* Now copy paste the config code from firebase into the file and make it look like the following
* Now, we are exporting the auth created by firebase that includes all of our apiKeys, authDomains, etc.
* Text

  Description automatically generated
* Now we want to add Facebook and Google into our Firebase application
* Go to Firebase and Click the build tab and click authentication
* Click get started
* Click Google, click enable, and enter any of our own google emails, then click save
* Click Facebook, and copy that OAuth link
* Go to <https://developers.facebook.com/> and click get started (which uses your facebook account)
* Now on the top right, click my apps and click the green create app button
* Click consumer and next
* Name it unichatmessenger123 and create app
* Click setup on the facebook login icon which should take you to
* Graphical user interface

  Description automatically generated
* Click Web, enter <http://localhost:3000/> for now as the url and click save and continue and keep clicking next until we are done
* Now on the left, under the facebook login tab, click settings
* Paste the OAuth link into Valid OAuth Redirect URIs
* Click save changes
* Now, go to settings, click basic and copy the app ID and the app secret
* Now go back to firebase and paste the app ID and the app secret and click save
* Now, we should have google and facebook login enabled
* Go to Login.js, and import the following
* import "firebase/app";
* import { auth } from "../firebase"; this is the auth configuration object that we exported from the firebase.js file
* import firebase from 'firebase/app';
* Go to the two divs which act as buttons for the user to click to login
* Inside the div, create onClick={() => auth.signInWithRedirect(new firebase.auth.GoogleAuthProvider())} for google and a similar one for facebook

Text

Description automatically generated

**Context**

* We will now go to src, create a new folder called contexts, create a new file inside there called AuthContext.js
* In this file, we will manage the context of the application by importing useContext from ‘react’
* Recall that context allows us to manage application wide state
* As well, we will also import useState from ‘react’ which allows us to store/change variables that can change what is displayed
* We will also import useEffect from ‘react’ which allows us to run some code if a certain dependencies are changed.
* We will also import useHistory from ‘react-route-dom’.
  + Aside:
  + Recall that we used <Switch>, <Route>, and <Link> in the past to try and implement routing.
  + Whenever we used those routers, React creates a history object for us to use.
  + We can access this history object through props.history since it is a default prop so all components have this prop.
  + We can also use history.push to push a new entry onto the history stack.
  + For example,
  + Text

    Description automatically generated
  + props.history.push('/home') would route the user to the component associated with the /home url.
  + However, if we try to pass props into the Login component, it would overwrite the default props containing the history object. Another problem is if we want to access this history object from outside the component which requires us to make our own history object.
  + Thus, if we are trying to use history from within a component, we can use the useHistory hook.
  + The useHistory hook ships with React Router and lets us access the state of the router to navigate from inside our components.
  + Hooks must be used inside of a component. That’s just how they work
  + Graphical user interface, text, application, email

    Description automatically generated
  + In this example we used a literal button to push /home to the history stack.
  + push(path, [state]) is a function that pushes a new entry onto the history stack
* We also will import the auth object we created in firebase.js
  + Just to refresh the memory the auth object is the following
  + Text

    Description automatically generated
* Now, we have imported everything we need
* Now, we will create our context object by writing the following:



* Now, we will create a function that will export our entire context, allowing other components to use them
* To do so, we write the following:
* 
* From other files, if we import useAuth, we can call useAuth which is a function that will execute the useContext function. Recall useContext is basically just returns its parameter, which is a context object. Thus, we pass in the AuthContext context object we just created a parameter to useContext which is being exported. So in other components, when they use useAuth, it is basically the same as using AuthContext since useAuth calls the useContext method and gets the AuthContext object. TLDR, when we use useAuth in our other files, we will have access to AuthContext.
* Now, we will create and export a const AuthProvider that has a value of a function. This function has a parameter of {children}. Recall, every component receives the props.children prop by default and props.children holds the content that is passed between the opening and closing of a component tag. All of react components has {children} within the props and in this case, we are going to destructor {children}. Most likely, whenever we AuthProvider, we are going to use react children. children is going to render all of the JSX code that we pass into the AuthProvider.
* Inside the function, we are going to create some state
* Firstly, we will create a const that manages if the app is in a loading state or not
* Initially, loading will be set to true since that is what a web app does when it starts, it loads the content onto the screen
* Next, we are going to create a const that manages the user
* Initially, we don’t have a user since the user has to log in so user will be set to an empty object
* 
* Now, we will create a const called history which allows us to use the useHistory hook that we imported, allowing us to renavigate in the future
* 
* Now, we will set up the useEffect.
* useEffect is a function that accepts another callback function as its first parameter, and dependencies array as its second. When the dependencies array changes, the useEffect function will re-execute the first function.
* In the first parameter which is the callback function, we are going to call auth.onAuthStateChanged which is a function from the auth object we imported from our firebase.js file we created. The onAuthStateChanged function takes a callback function as its parameter. Whenever the state of the auth changes, we will call this callback function which is the parameter to the onAuthStateChanged function. This callback function has a parameter of user which stores the data of a user. Once we get the data, we can change the state. We can now change the state user to have a value of user which is the parameter of the callback function. As well, since we now have the user data, that means we have retrieved the data and no longer loading so we will set the state loading to false. Now, we are ready to redirect our user to the chat route. To do so, we use the history const we created before and use history.push(‘/chats’) which will redirect the user to the /chats route.
* We want to make our dependency array be [users, history]. The user changes by adding a new user and the history changes by renavigating to a different route. Whenever, the user or history changes we will re-execute the callback function for useEffect.
* Text

  Description automatically generated
* Now, we will create a new constant called value which will be equal to the state user
* Recall that AuthProvider is equal to a function and we have yet to make that function return something
* Thus, we will return an AuthContext.Provider component. Recall that Provider is a component that is built inside the context object that we named AuthContext. Thus, we access it via AuthContext.Provider. This Provider component needs to be wrapped around all the components that are interested in interacting with the context.
* We will pass in the const value to the Provider component as a prop and this prop has a name of value. Recall that const value is equal to our user state. Thus, whenever the user state changes, it updates the const value. This updated const value will be passed into the Provider component as a prop. And passing this updated value will cause all the components that are listing to be updated.
* In between the AuthContext component tags, we will add {!loading && children} which will show the children if the web app is not loading.
* Text

  Description automatically generated
* Now, we will review the code in AuthContext.js
* We are creating a context and storing it in AuthContext
* We are creating a function called useAuth which allows us to grab that context
* We are also setting the initial loading/user sets
* Then, we are grabbing the user information from the firebase authentication and passing that user information into the state
* Then we redirect the user to /chats
* Now, we can use the AuthProvider function in index.js and further explore the use of the {children}.
* Text

  Description automatically generated
* Now, we will go to App.js
* We want to import AuthProvider which is the function that returns the AuthContext.Provider tag. Thus, we uncomment the import
* As well, we should uncomment the uses of AuthProvider in the code, since we want to actually use it
* This portrays what react context is. React context is one big object that contains all the data, in this case user data, and wraps all the other components. This AuthProvider component handles the entire application’s state.
* Currently, we only have the login component, but we will also need to create the chat component so uncomment that line along with the import for the Chat component at the top of App.js
* Text

  Description automatically generated
* Now, we will go create the chat component

**Chats Component**

* We are going to create the chats component
* In the components folder, create a new file called Chats.js
* Inside Chats.js, we will import React from ‘react’;
* Now, if we go to our development server, we see the following

Graphical user interface, text, application, chat or text message

Description automatically generated

* However, if we try to manually go to localhost:3000, it’ll automatically redirect us back to localhost:3000/chats
* To resolve this issue, go to AuthContext.js
* The reason we can’t leave localhost:3000/chats is because with our history object, we are only pushing to ‘/chats’, no matter if we have the user or not.
* To resolve the issue, we can initially set the value of the user state to be null
* Text

  Description automatically generated
* Then in the useEffect, we can the condition that if we have the state user data, only then will we redirect to ‘/chats’.
* Text

  Description automatically generated
* Now, if we try to go to localhost:3000, we get the sign in/welcome page which is what we want. (note if it doesn’t work, we have to go to firebase, the project, and authentication, and make sure the all users are deleted)
* Now, we go back to Chat.js
* We will import {useHistory} and {ChatEngine} and {auth}
* Text

  Description automatically generated
* Now, write the following return JSX code
* Text

  Description automatically generated
* The ChatEngine component requires a few props. One of which is the height. Another prop is the projectId.
* We can find the projectId inside the ChatEngine website
* Go to <https://chatengine.io/>
* Sign up and then log in
* Create a new project and name it unichat
* Graphical user interface, application

  Description automatically generated
* Copy paste the projectId into Chat.js
* It is not a good idea to keep sensitive information in the public so we will put it into environment variables later
* Another prop the ChatEngine component needs isa userName and a userSecret, which for now, we will just have a placeholder of ‘.’ meaning we will have to grab an actual username and usersecret later on.
* Text

  Description automatically generated
* Right now, our logout div doesn’t do anything so we want it to call a function when we Logout. We will call that function handleLogout
* Now, we will write the code for handleLogout
* Text

  Description automatically generated
* We will create handleLogout as an asynchronous function which will execute the auth.signOut() function, but before that even completes, it goes and redirects the user to the sign in page which has a path of ‘/’. This redirection occurs using the history object that we have to define (shown right above the async function).
* Text

  Description automatically generated
* Now, when we save and reload our project, we can sign in with google, and it will redirect us to /chats. We see the following page that tells us the credentials are incorrect. This is fine for now since we didn’t actually grab the real google login info yet.
* Graphical user interface, text, application

  Description automatically generated
* Now, we will use the data that is coming from the social firebase login and pair it with and API call to ChatEngine to create users
* Go to Chat.js
* Import useAuth
* Text

  Description automatically generated
* Recall that we created useAuth and it simply allows us to access the context object
* To access the context object, we can call the useAuth function
* Graphical user interface, text

  Description automatically generated
* For now, we’ll print user just to see what is going on
* Notice when we log in and go to /chats and then inspect the page, we see hella erros
* As well, when we try to log in with facebook, we can’t seem to log in, conclusion, fk facebook
* To solve this, go to firebase, authentication,
* Go to facebook for develops, go to our app, Facebook Login (on the right), Settings

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

* Save our changes
* Click settings (big tab) and click basic
* Graphical user interface, application, Teams

  Description automatically generated

BACK TO THE CODE