**Dev-Connector Frontend**

* Npx create-react-app client
* In package.json add scripts:

„client“: „npm start –prefix client“,

„dev“: „concurrently \“npm run server\“ \“npm run client\““

* Cd client
* Npm i

Axios - to make http request,

React-router-dom,

Redux,

React-redux,

Redux-thunk – middleware that allows us to make async request in our actions

Redux-devtools-extension

Moment – date and time library,

React-moment -that we can use moment in a component

* Ad proxy in package.json. „proxy“: <http://localhost:5000>
* In src delete: App.test, index.css, logo.svg reportWebVitals.
* Add css file from cousrse source
* In source create folder 'components' , in components create another folder called 'layout', in layout create file 'Navbar.js', 'Landing.js'
* **React Router Setup** in App.js import Router, Route, Routes and Switch. Wrap everything in 'Router'. Replace 'Landing' component with 'Route'.
* const App = () =>
* <Router>
* <Fragment>
* <Navbar />
* <Routes>
* <Route exact path="/" element={<Landing />} />
* </Routes>
* </Fragment>
* </Router>
* export default App;
* Inside 'components' folder create 'auth' folder. Inside 'auth' folder create Login.js and Register.js components.
* Make 'Login' and 'Registe' buttons work. in Navbar.js import 'Link'

 return (

    <nav className="navbar bg-dark">

      <h1>

        <Link to="/"><i className="fas fa-code"></i> DevConnector</Link>

      </h1>

      <ul>

        <li><a href="!#">Developers</a></li>

        <li><Link to="/register">Register</Link></li>

        <li><Link to="/login">Login</Link></li>

      </ul>

    </nav>

  )

* Make 'Landing' component buttons work.
* **Register form & use state Hook**
* Request example
* if (password !== password2)
* {
* console.log('Passwords do not match')
* } else
* {
* const newUser = {
* name,
* email,
* password
* }
* try
* {
* const config = {
* headers: {
* 'Content-Type': 'Application/json'
* }
* }
* const body = JSON.stringify(newUser);
* const res = await axios.post('/api/users', body, config)
* console.log(res.data);
* } catch (err)
* {
* console.error(err.response.data);
* }
* }
* Redux. . .we call an action, dispatch the action to a reducer and reducer decides what to do with that state, deleting a post, updating a profile, etc. Then its gonna send that state back to components, any of them that need it.
* We're gonna have auth reducer that will handle everything that will have to do with authentication. Basically on every single load of the App component, we wanna check for a user, we wanna load a user, we wanna hit our backend api/ and see if we're logged in.
* **Creating a Redux store**  ...inside src create file 'store.js'. Inside src create folder 'reducers'. Inside reducers create file 'index.js' ..thats our root reducer.
* **Alert reducer** inside reducers create a file alert.js and import it in index.js (root reducer). Its just a function that takes a piece of state (any state that has to do with alerts) and action. An action is gonna get dispatched from action file. Inside src create folder called 'actions' inside it create file 'types.js' which will hold all of our variables. Inside 'actions' folder create file 'alert.js'. Open terminal and cd into client. Npm i uuid.
* **Alert component & action call** we wanna call alert action, good place to do it is in Register.js when passwords don't match. Wee need to connect that component to redux. We do it through 'connect'. Import connect and our setAlert action. In 'component/layout' folder create component 'Alert.js'
* **Auth reducer & Register action** in 'reducers' folder create file 'auth.js'. Now we're working with http request, we need to deal with the backend. We make axios request to the backend in our actions file. We're gonna make the request, get the response. If its a succesfull response, we're gonna dispatch REGISTER\_SUCCESS if its a fail we dispatch REGISTER\_FAIL. In actions folder create file 'auth.js'.
* **Load user and set auth token** Here we wanna handle the process of taking token that we have stored, sending it to the backend for validation and then loading the user. We want that to happen every single time the main App component is loaded. In src create folder 'utils' and inside a file 'sestAuthToken.js.
* **User Login**

1. Create action: actions/auth.js
2. Create types LOGIN\_FAIL, LOGIN\_SUCCESS
3. Go into reducer and handle what happens when either of these fire off
4. We have our action, we have the reducer functionality, now we need a way to fire off that action, that comes from component. Component/auth/Login.js

* Now, what should happen is:it will fire off 'login', get the email and password we passed in, and then we'll get the body stringify it, send it along to the route /api/auth, dispatch LOGIN\_SUCCESS
* Inside our register and login form, we wannna make it so that when we're authenticated it redirects us. In login component we need to bring in our state, because it has 'isAuthenticated'. Create mapStateToProps.
* import { useNavigate } from 'react-router-dom';
* //Redirect if loged in
* const navigate = useNavigate();
* if (isAuthenticated)
* {
* return navigate('/dashboard')
* }
* **Logout and navbar links:**

1. Create LOGOUT in types.
2. Bring in LOGOUT in actions/auth.js
3. Bring in LOGOUT in reducers/auth.js
4. In component/layout/Navbar.js bring in connect, PropTypes, logout.

* **Protected route for Dashboard:** in components create folder 'dashboard' and in that folder a file 'Dashboard.js'. This is where we're going to fetch all our data using an action and then bring it in from redux state and pass it down to other components. We don't wanna be able to acces dashboard page when loged out, so we need to create a private component. In components create new folder 'routing' and inside a file 'PrivateRoute.js'. Add new route in App.js. As soon as dashboard load, we need to pull in the current user's profile. In order to do that, we need profile in our state, which means we need profile reducer, profile actions file.
* **Profile Reducer & Get Current Profile lection 45:** We need to make request to our backend so we can send that data down throught the state to components.

1. In reducers folder create file profile.js and add it into main reducer reducers/index.js
2. Add GET\_PROFILE and PROFILE\_ERROR types.
3. In actions folder create file 'profile.js'
4. Handle the two types in reducer
5. Call getCurrentProfile action function in Dashboard component