Today I am working on the creation of login’s in my application, and designing it for a react sense. I was investigating the use of JSON web tokens, as I want to ensure only logged in users have access to Sonrasc. Then, the users who login will have the option to create User accounts for their business so other people can log on and create invoices for the business.

Posting a list of links for my own reference. I’m using the ‘Beer locker’ for most of my research and following the react issues on Github to keep track of what people are saying.

<https://stormpath.com/blog/build-a-react-app-with-user-authentication/>

<https://github.com/mjrussell/redux-auth-wrapper>

<https://github.com/joshgeller/react-redux-jwt-auth-example>

<https://github.com/reactjs/redux/issues/297>

<https://github.com/auth0/react-browserify-spa-seed>

<https://auth0.com/blog/2016/01/04/secure-your-react-and-redux-app-with-jwt-authentication/>

<http://www.dampmann.com/2014/09/nodejs-mongodb-jwt-bcrypt-and.html>

<http://scottksmith.com/blog/2014/05/29/beer-locker-building-a-restful-api-with-node-passport>

<https://auth0.com/blog/2016/01/04/secure-your-react-and-redux-app-with-jwt-authentication/>

<https://scotch.io/tutorials/easy-node-authentication-setup-and-local>

Today I am working on functionality that a user can register. This involves utilising the new routes I added for entering users into the database using their Schema. It also involves not storing their passwords directly, so I have to hash their passwords with a salt so there is more security.

**General Flow**

The overall flow goes something like this:

1. The log in form dispatches an action creator which triggers a POST to the server
2. The server validates login credentials and returns a valid JWT or 401 Unauthorized response as needed
3. The original action creator parses the server response and dispatches success or failure actions accordingly
4. Success actions trigger an update of the auth state, passing along the token and any decoded data from the JWT payload
5. A higher-order authentication component receives the new auth state as props
6. If authentication was successful, the higher-order component renders its child component and passes the auth props down to it
7. Before mounting, the child fetches data from the server using the token it received from its parent wrapper

Taking a look at the code should make this more clear!