##### REDUX REGULAR

##### 

[REDUX flow](#_db81zb76eert)

[REDUX STRUCTURE](#_q442jgcup1k9)

[The Root Reducer](#_bvm653jyz62e)

[The connect function](#_1fob9te)

[Container vs. Presentational Components](#_keg5xg467m79)

[mapStateToProps()](#_ex91qmlle1uf)

[mapStateToProps, mapDispatchToProps and connect](#_kn9w2d8xf8ky)

[Making a Reducer](#_plj8llcxktwv)

[Combine reducer](#_j8kp48t2ugel)

[Creating a STORE](#_618u2j4mczly)

[Make components](#_of7qxaio9nk3)

[Provider](#_7czflpd4ktgc)

[Create a container folder](#_or0z5ilb1pyv)

[Actions](#_3vgeocmrgx65)

[Map state to props() code](#_w3f43zyv4lre)

[Passing Data to Component](#_povbb29cmhbr)

[Action and Action Creators code](#_o4deh522ipz9)

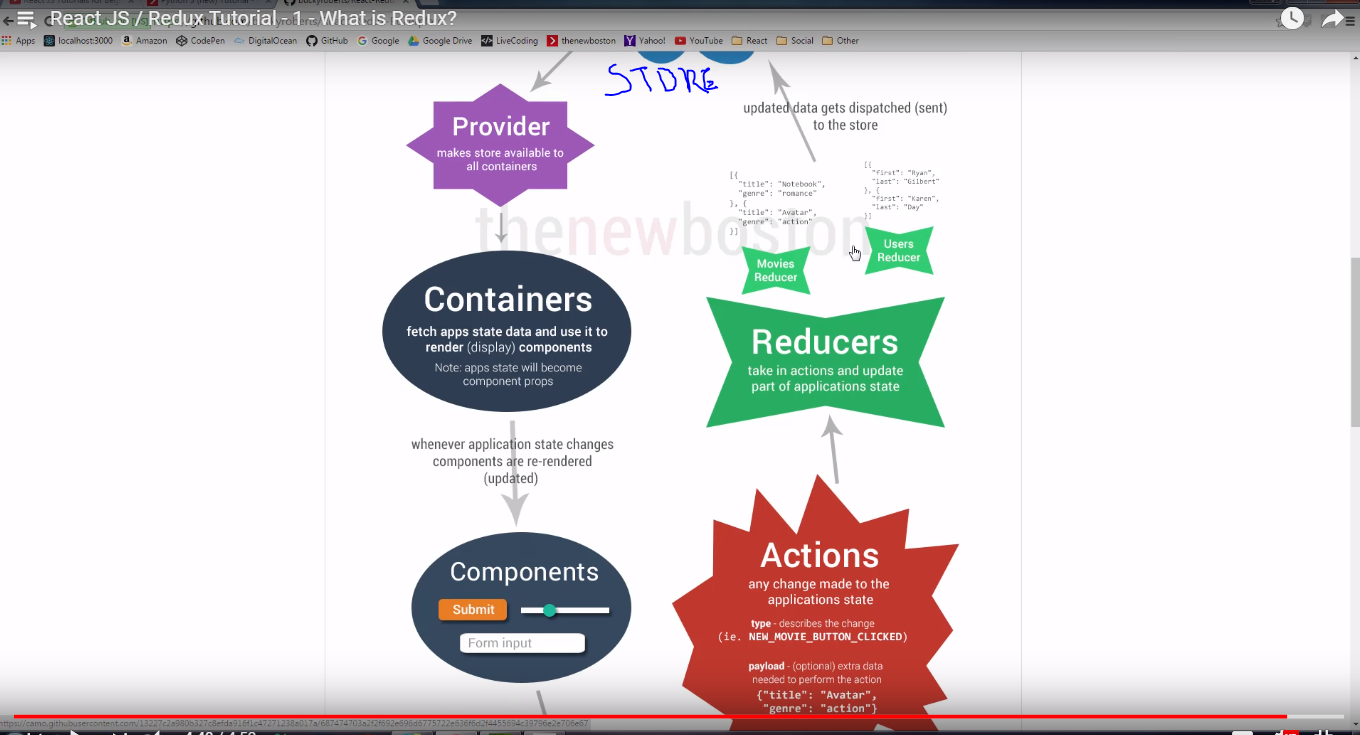
[Reducers connection with state](#_gbm9hr4tjgl9)

[Import in app.js (user details)](#_1o6opguqj56i)

##### 

##### 

##### REDUX flow





##### REDUX STRUCTURE

npm install --save react-redux

**Application File Structure**

The file structure of our React Redux application is pretty simple.

├── package.json  
├── src  
│ ├── actions  
│ ├── components  
│ ├── index.html  
│ ├── index.js  
│ ├── reducers  
│ └── store  
├── tools  
└── webpack.config.js

We have a package.json since we'll use npm to manage dependencies.

The webpack.config.js file will use webpack to bundle our app for the browser.

The tools directory will house some of our supporting code, like the configuration of our server using Express.

The src directory is where the magic happens, it contains the meat of our React Redux application.

Actions  
Actions are JSON objects that contain information about changes that need to be made to state. They can be dispatched by various parts of your application, and they are received by the store.  
  
Actions are produced by functions called action creators.  
  
Store  
  
The store holds the whole state our application. It can dispatch actions and it receives actions that are dispatched to it. However, the store doesn't want to handle the dispatched actions and actually enact changes to state. For that, we use reducers.  
  
Reducers  
  
The store passes dispatched actions to reducers, which receive the actions and make the appropriate changes to state.  
  
Components  
  
Components are the presentation, or view, layer of our application. Certain components, the parent or container components, will be connected to our store. Those components will be alerted whenever a change to state has been made by a reducer. Those components will then re-render, with new properties as dictated by the changes made to application state.  
  
index.html and index.js  
  
The index.html file provides the initial HTML for our app and includes a script tag that links to the bundled version of our app that webpack will create for us.  
  
The index.js file is the entry point for our React app. It will render the absolutely most top-level parent component and attach it to an element on our index.html page.

You should be able to run npm install and then start the server via npm start.

##### The Root Reducer

We'll use Redux's combineReducers function. This is a helper function that [turns an object whose values are different reducing functions into a single reducing function you can pass to createStore.](http://redux.js.org/docs/api/combineReducers.html)

##### The connect function

The connect function is provided by Redux. It subscribes our container component to the store, so that it will be alerted when state changes.

Not every component will be connected, or subscribed, to the store. Only container, or "stateful", components will be connected to the store.

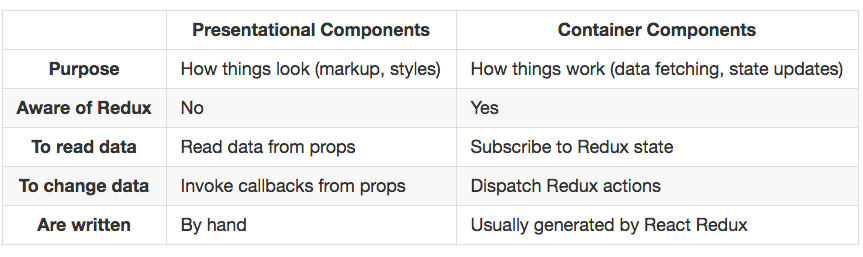
##### Container vs. Presentational Components

A container component is a component that is connected to the store and aware of application state and changes to that state. It takes that state and passes aspects of it to presentational components as props.

Container components have functions that dispatch actions that enact our earlier flow of action dispatch -> API call -> action dispatch -> store -> reducer -> component re-render. Container components will pass such functions into presentational components as props.

Presentational components are not aware of the store or our application state. They know about their own props. They respond to user actions by invoking callback functions that their container component passed them.

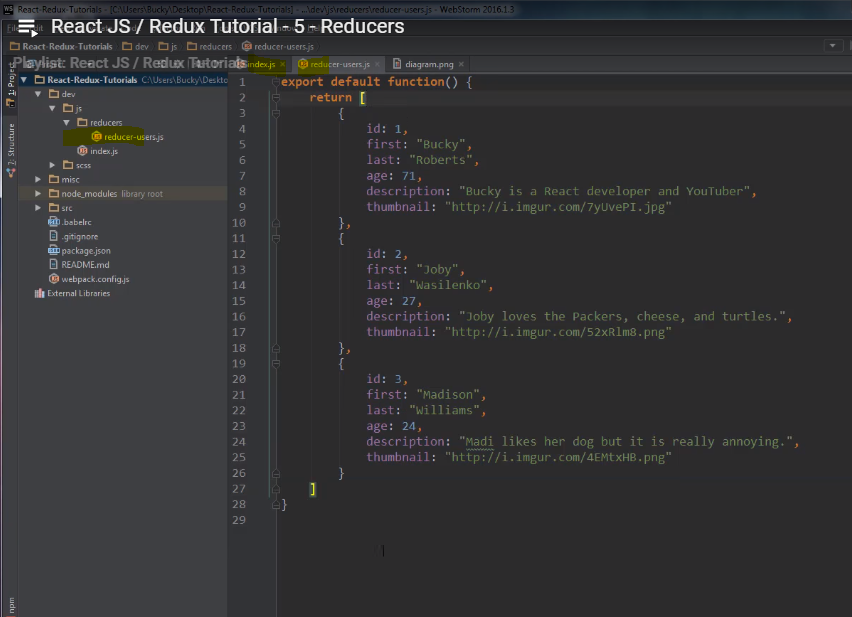
These differences will become clearer when we build out our create, edit and delete cat features.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mapStateToProps() is a utility which helps your component get updated state(which is updated by some other components),  mapDispatchToProps() is a utility which will help your component to fire an action event (dispatching action which may cause change of application state)   |  |  |  | | --- | --- | --- | | [share](https://stackoverflow.com/a/42572170)[improve this answer](https://stackoverflow.com/posts/42572170/edit) | [edited Aug 20 at 14:01](https://stackoverflow.com/posts/42572170/revisions) | answered Mar 3 at 6:32    [Saisurya Kattamuri](https://stackoverflow.com/users/2648817/saisurya-kattamuri)  **301**210 | |  |
|  | add a comment |

|  |  |
| --- | --- |
| up vote1down vote | mapStateToProps, mapDispatchToProps and connect from react-redux library provides a convenient way to access your state and dispatch function of your store. So basically connect is a higher order component, you can also thing as a wrapper if this make sense for you. So every time your state is changed mapStateToProps will be called with your new state and subsequently as you props update component will run render function to render your component in browser. mapDispatchToProps also stores key-values on the props of your component, usually they take a form of a function. In such way you can trigger state change from your component onClick, onChange events. |

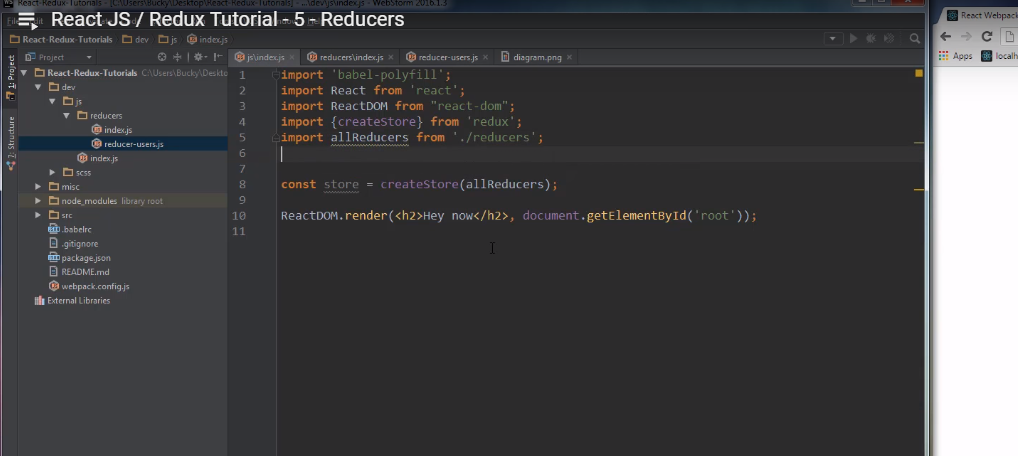
##### Making a Reducer



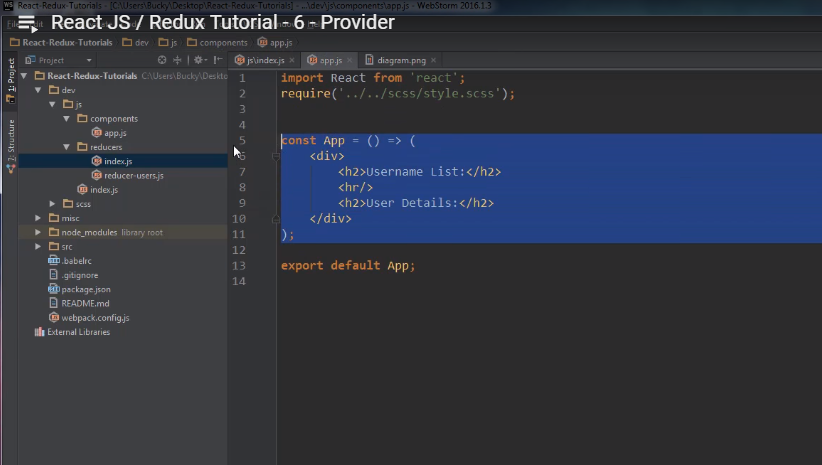
##### Combine reducer

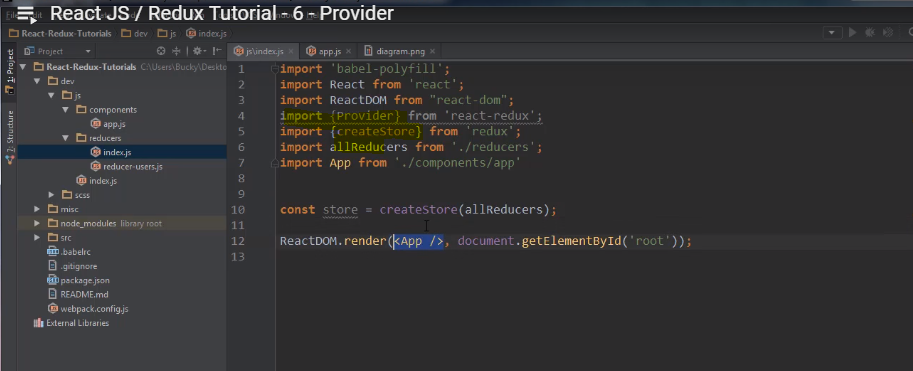
##### 

##### Creating a STORE



##### Make components

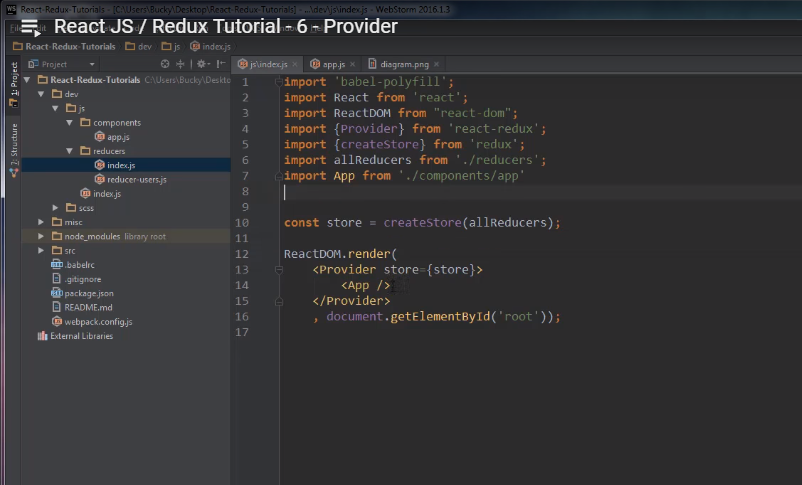




##### 

##### Provider

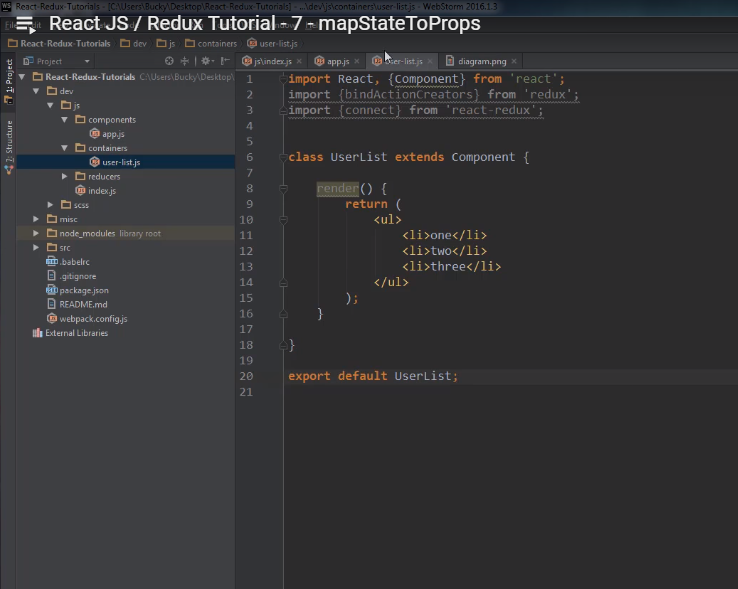
<Provider store={store}> </Provider> is a new provider component



##### Create a container folder

Container -

User-list.js



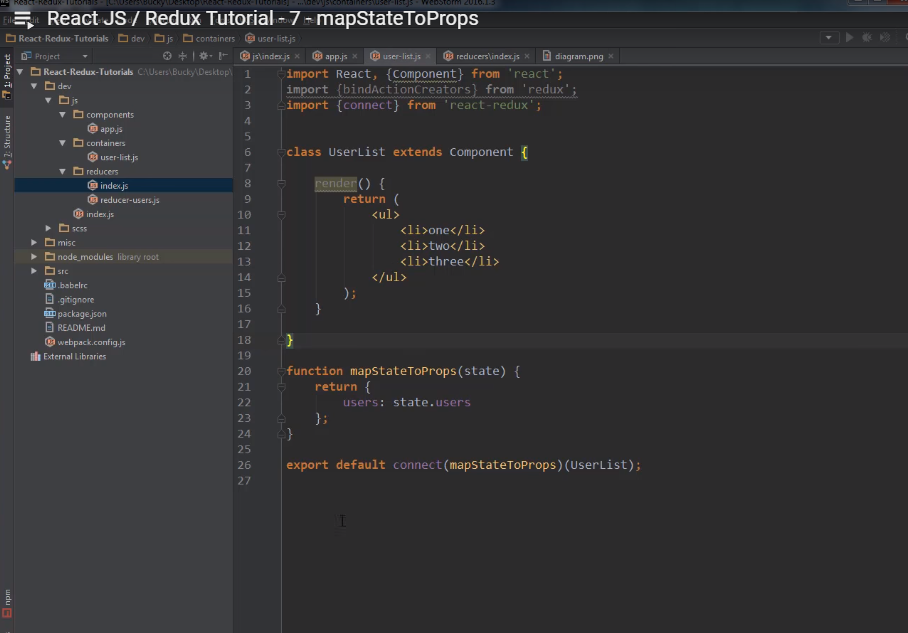
##### 

##### **Actions**

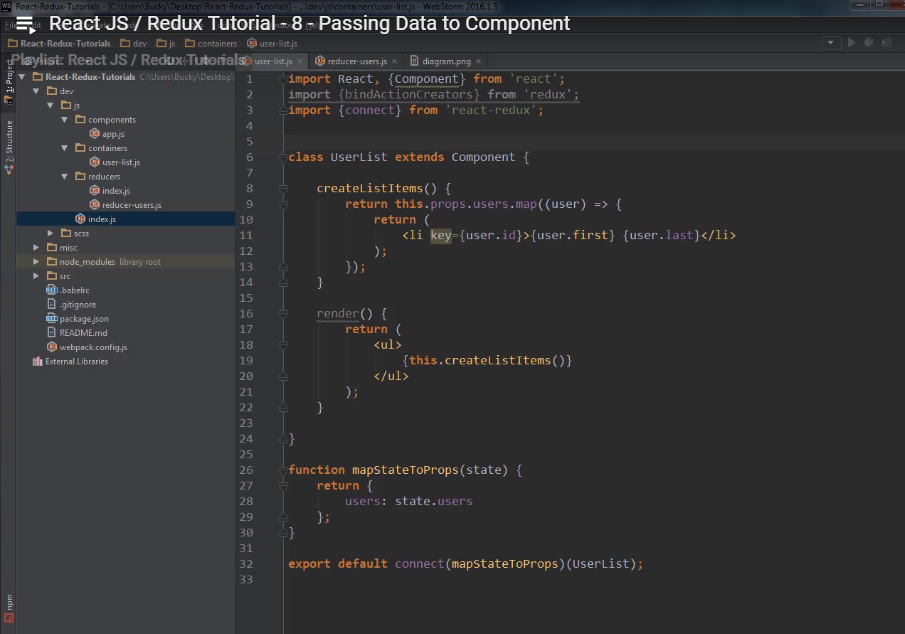
is a function

##### 

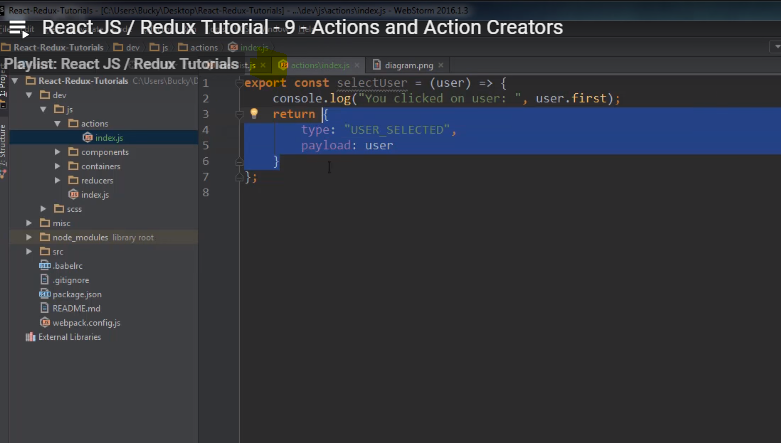
##### Map state to props() code

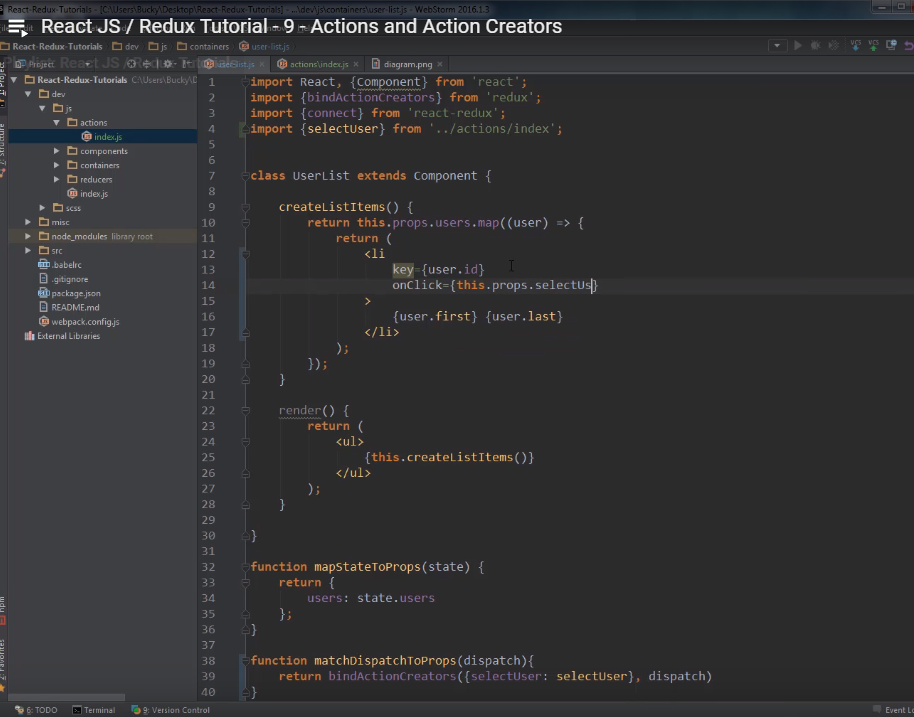


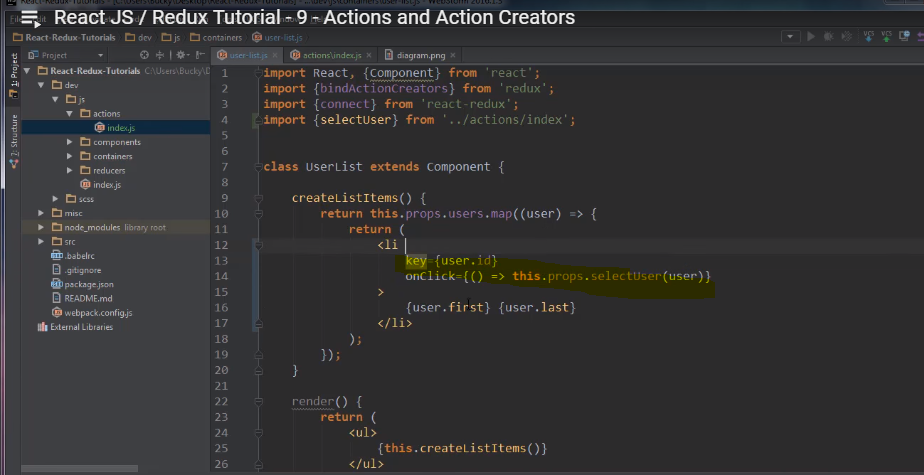
##### Passing Data to Component



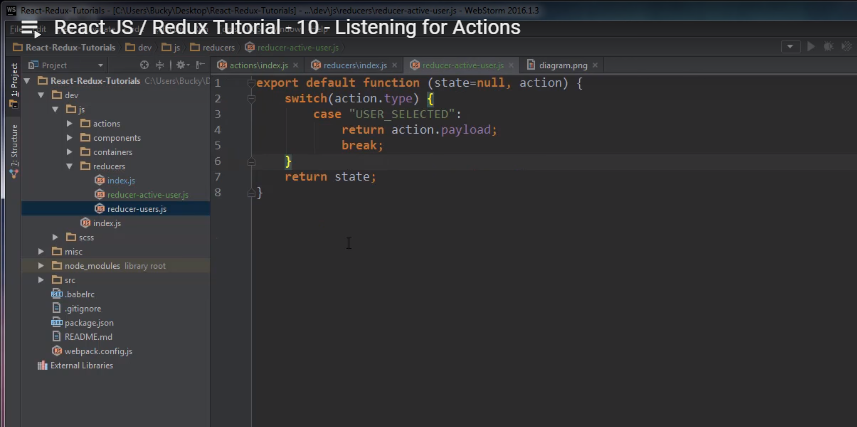
##### Action and Action Creators code

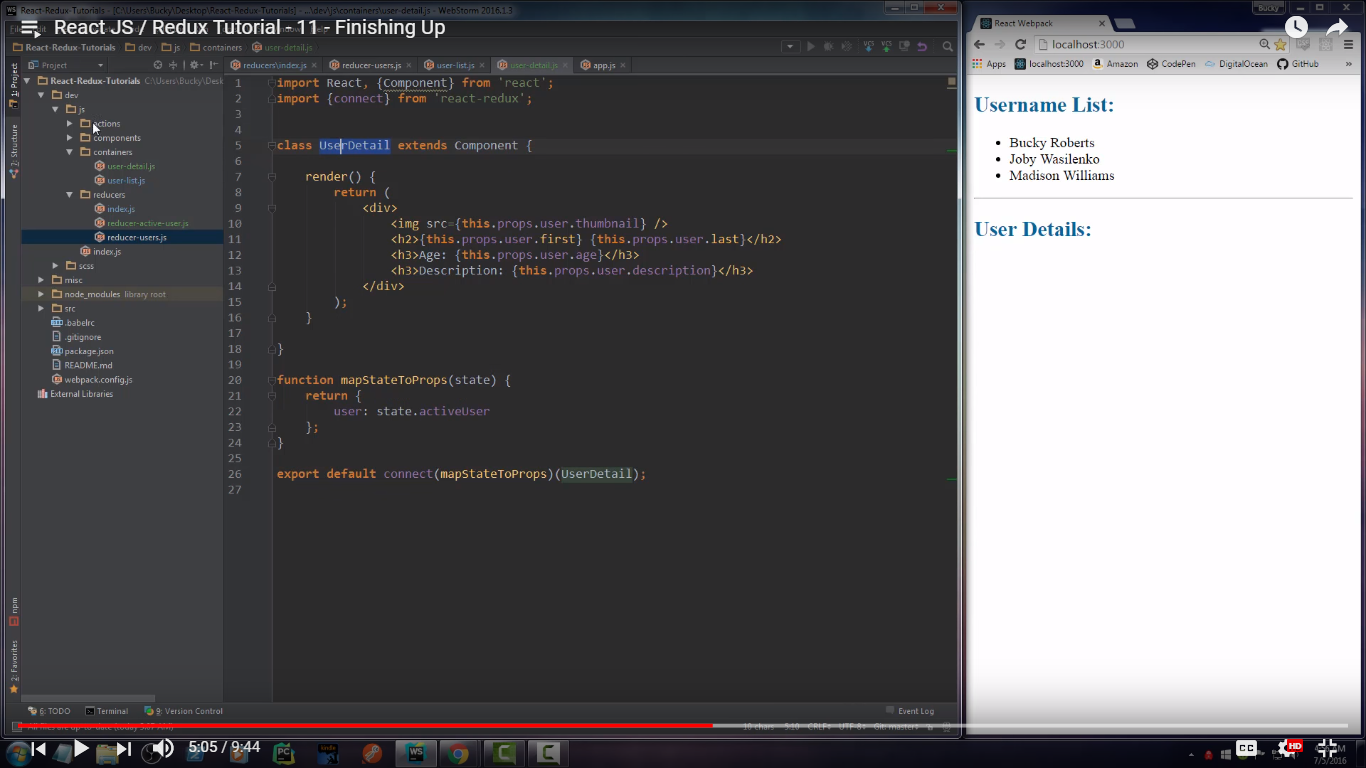




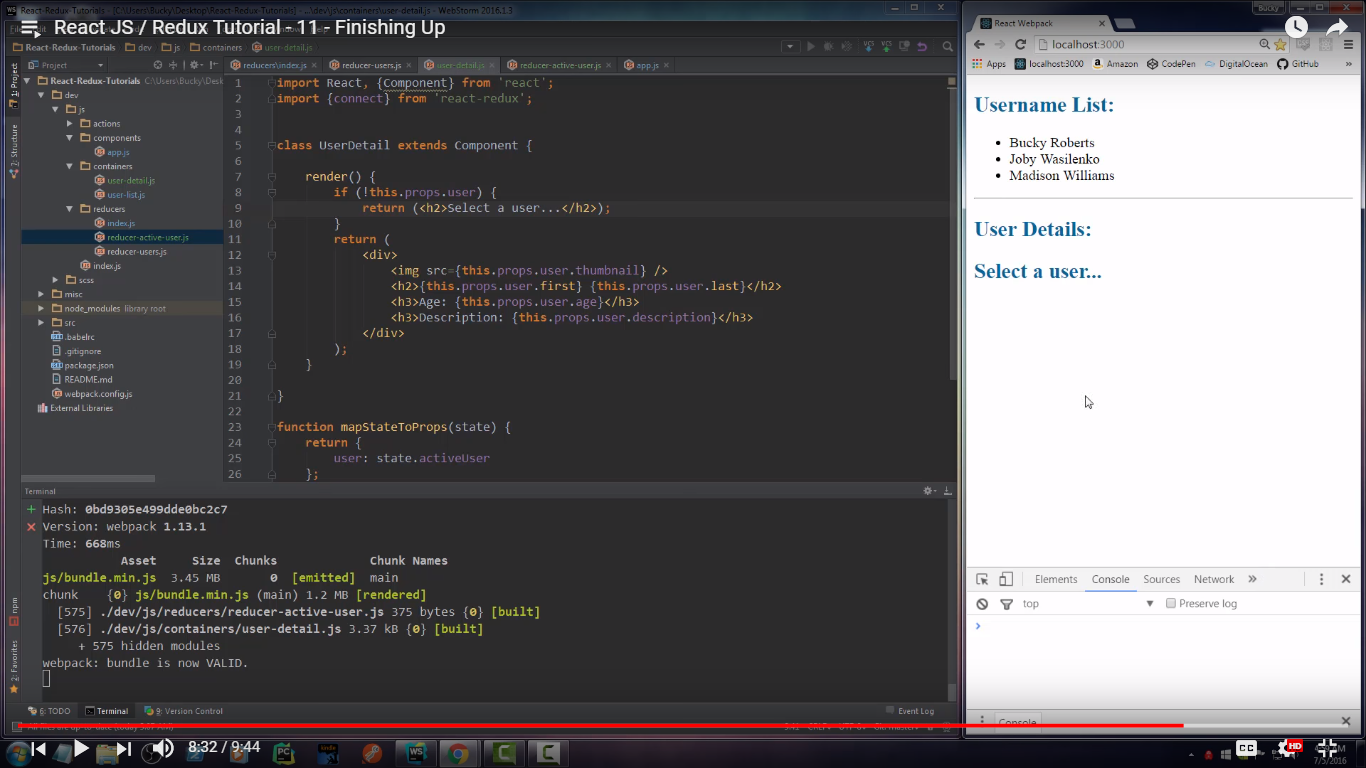


##### Reducers connection with state

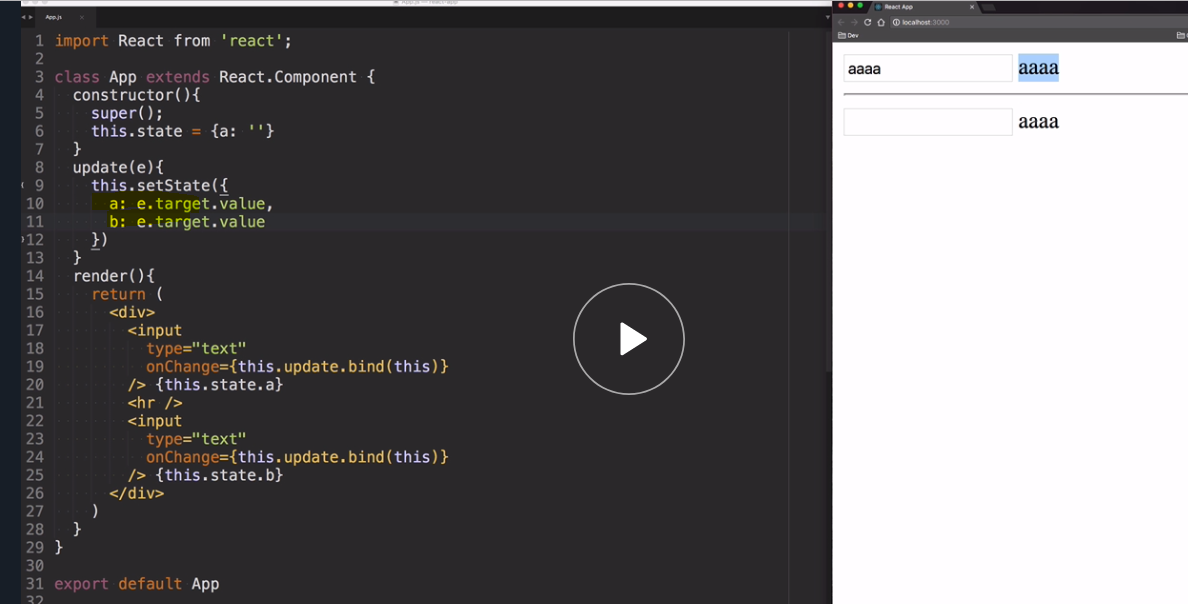


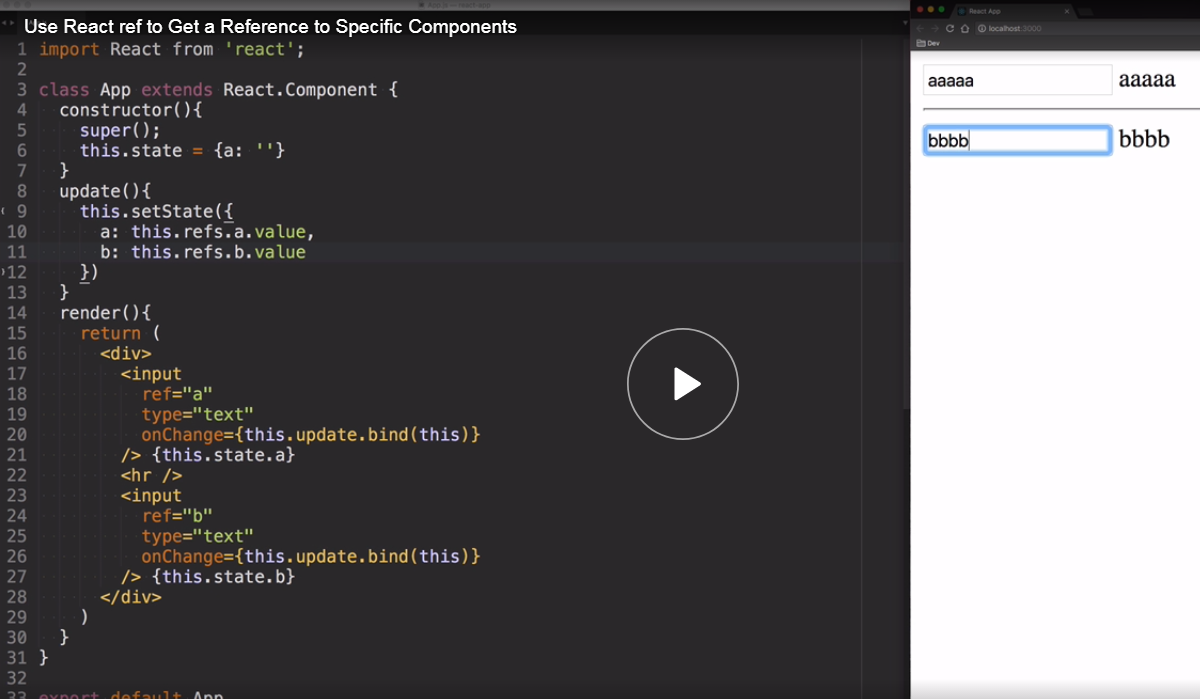


##### Import in app.js (user details)

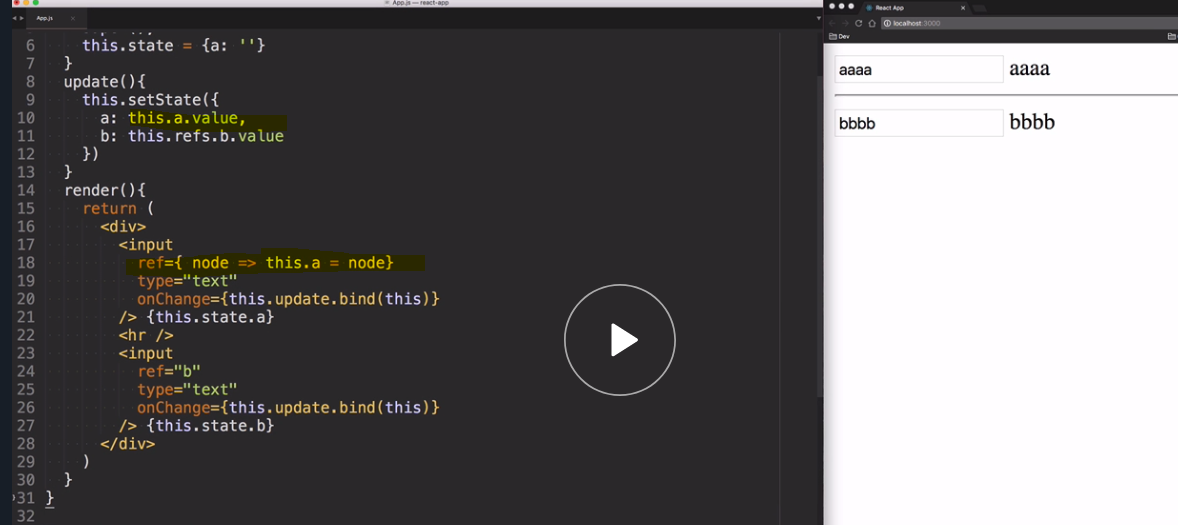


**REFS**

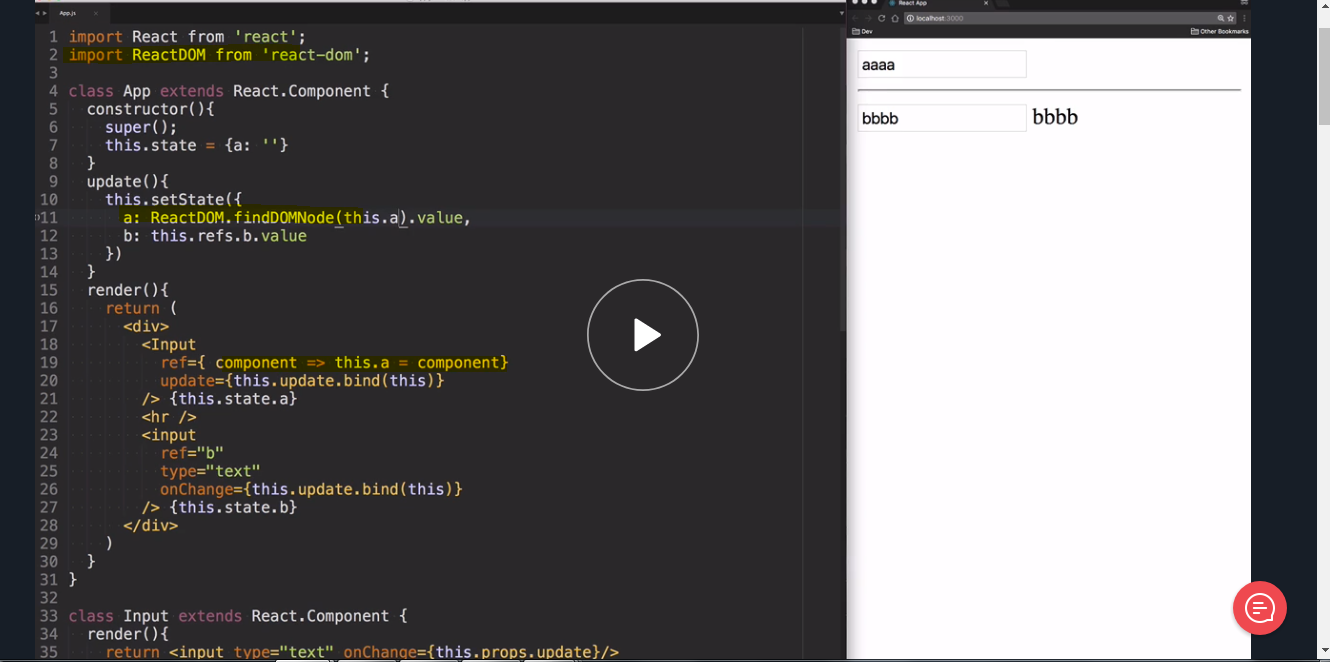
****

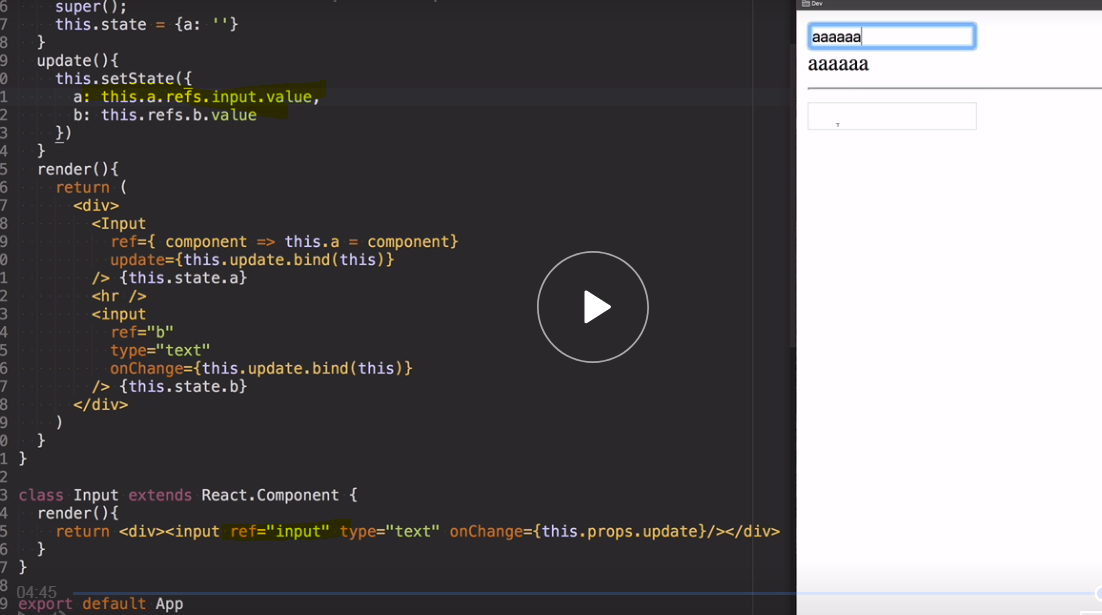


**REF A NODE**



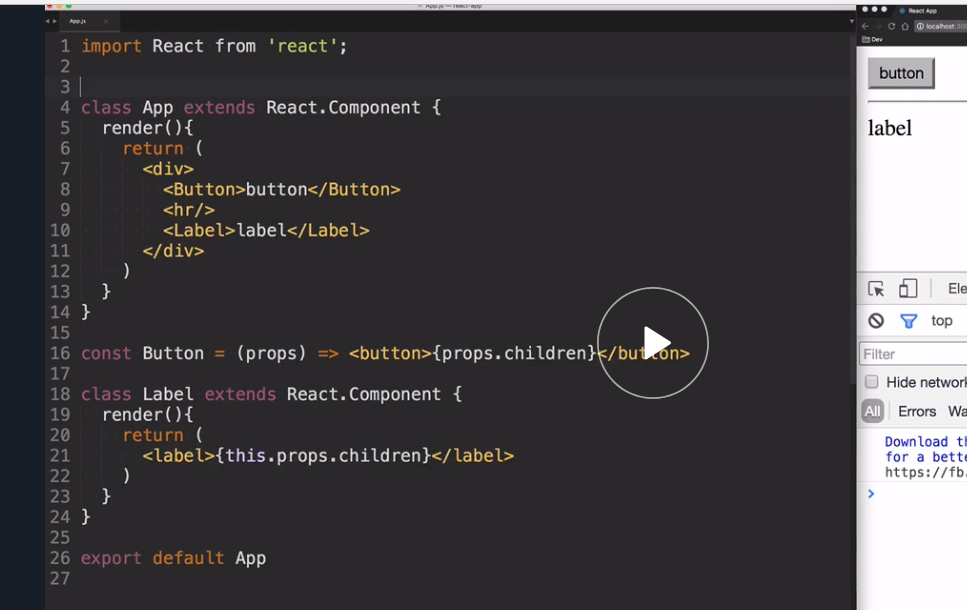
***REFERENCE INSTANCE OF ANOTHER COMPONENT***

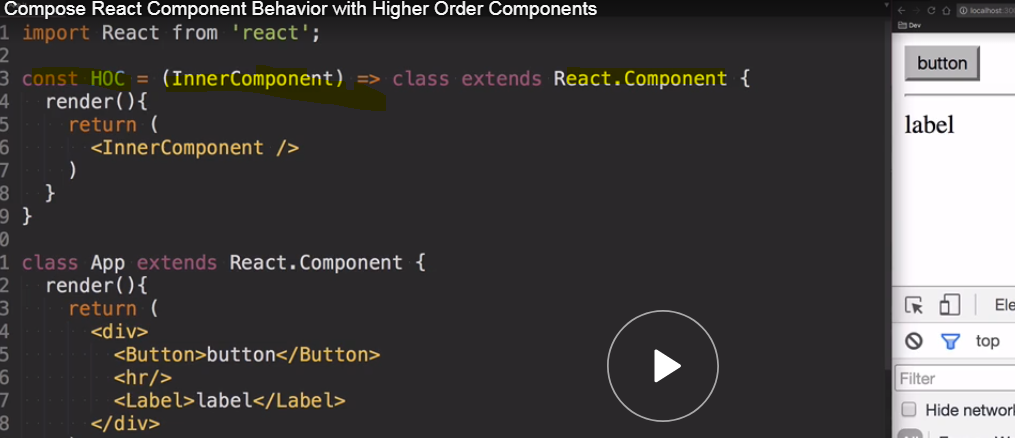
******

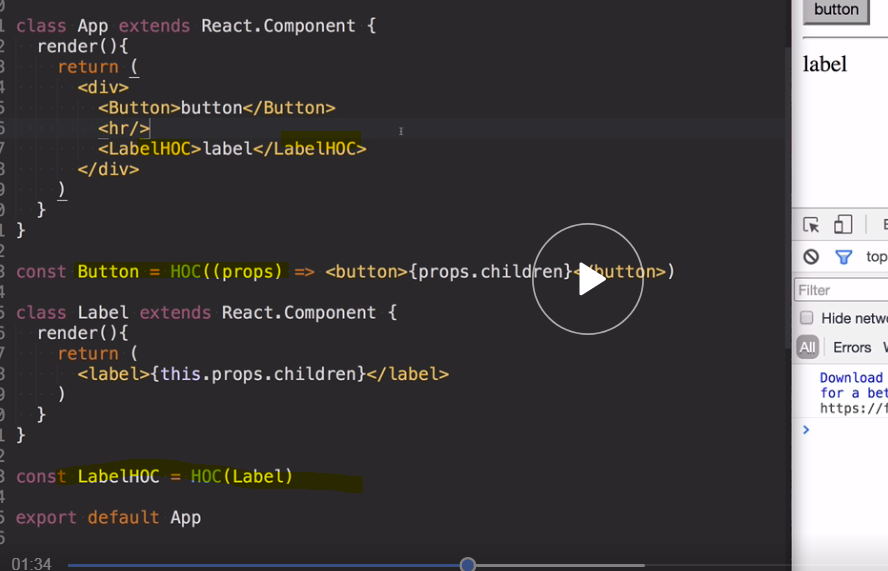
******

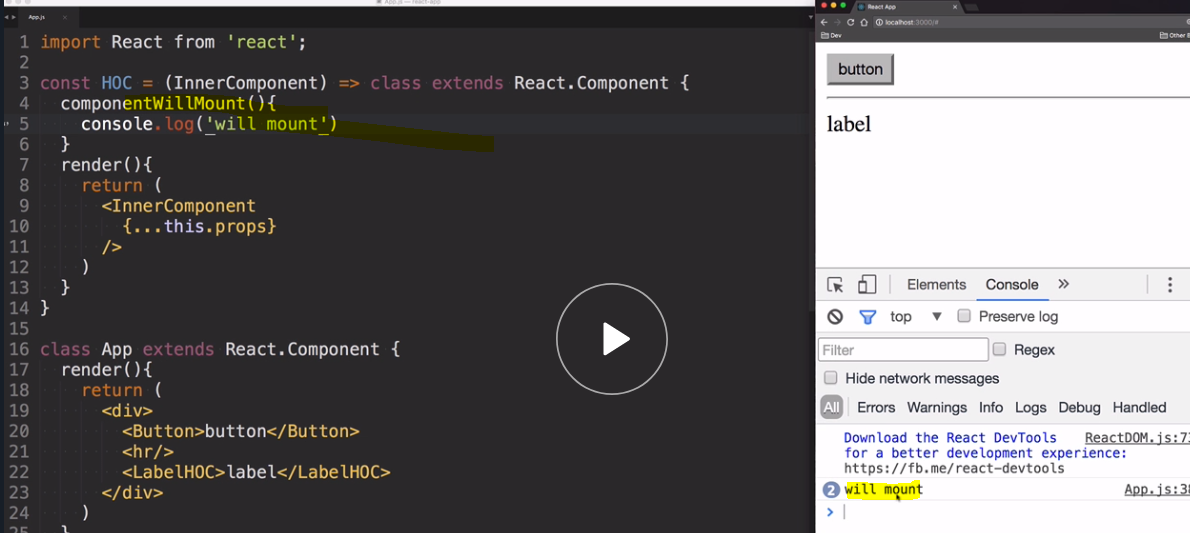
***HOC***

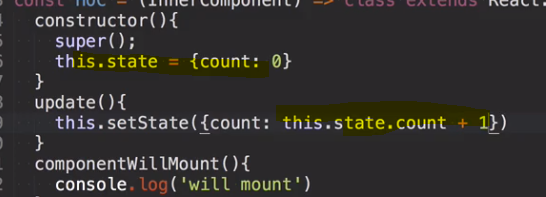
Purpose of HOC is to share common functionality or info among multiple components, sole functionality is to take in a component and return a component

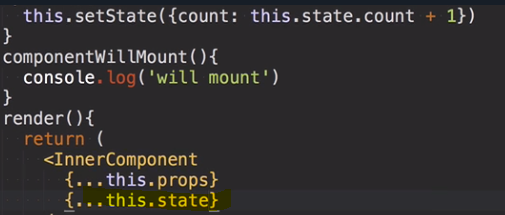
******

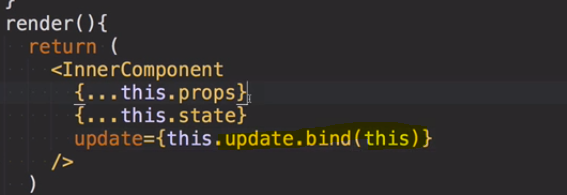
******

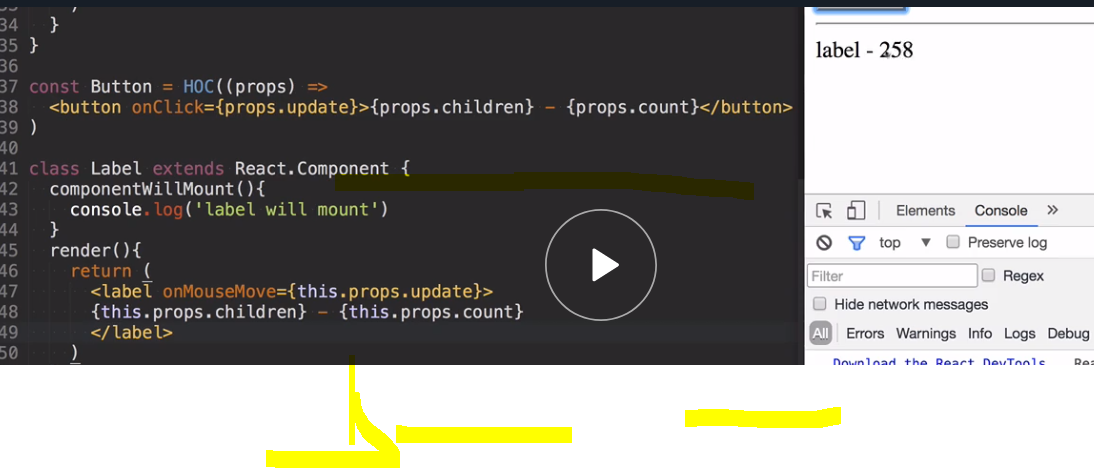
******

******

******

******

******

******