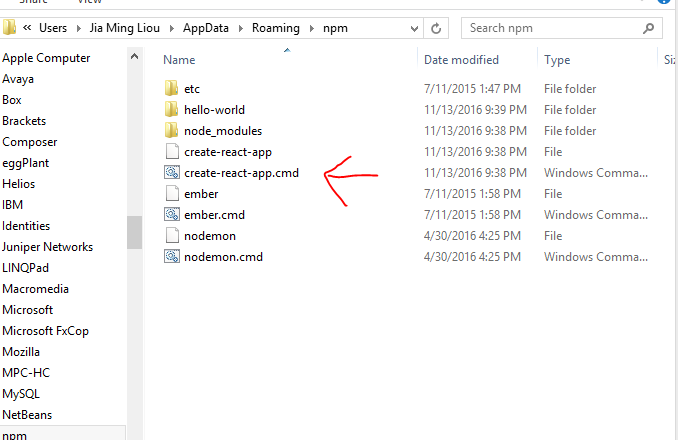
Create an instruction

npm install -g create-react-app

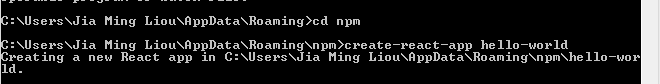
Create an app with ReactJS

* + - 1. Go to folder with the créate-react-app.cmd



* + - 1. Then create the app with

create-react-app hello-world



Start localhost

* + - 1. Go to the app folder e.g hello-world

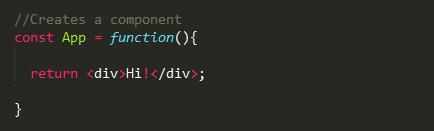
cd hello-world

* + - 1. Start the app and then it would start the browser

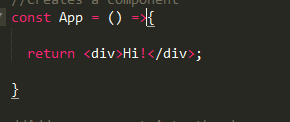
npm start

Create a component

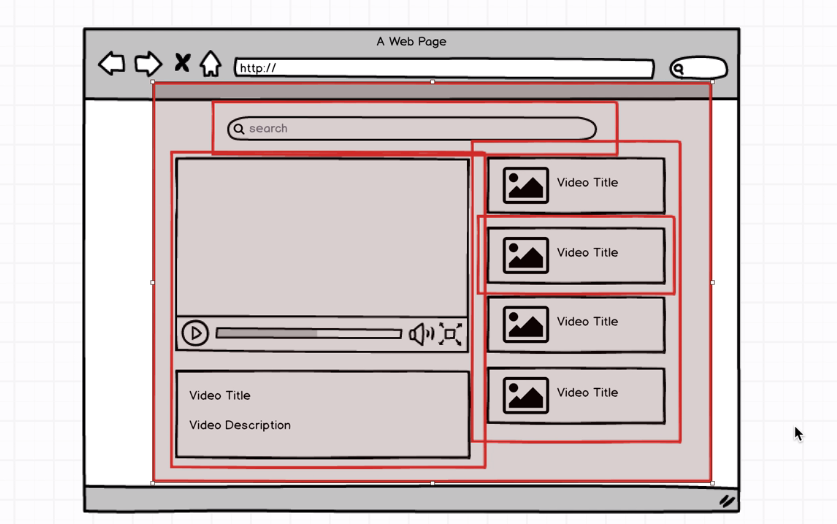
This is a functional component



OR

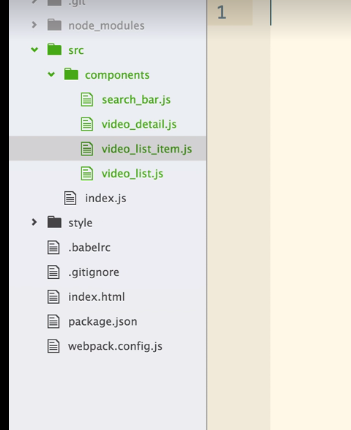


Divide the page into components



Each red box is a component and there is a main component that hosts the rest of the boxes.

In the source folder create a folder for components and create the components

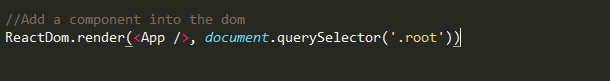


Add component into the website

QuerySelector has to be a class

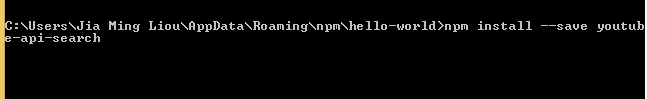
Index file



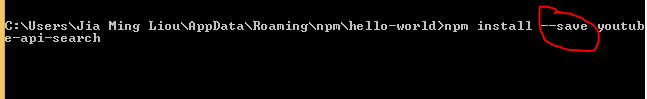
App.js

Download and install packages

e.g Youtube

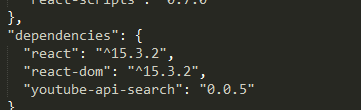


This –save will save it into the package.json

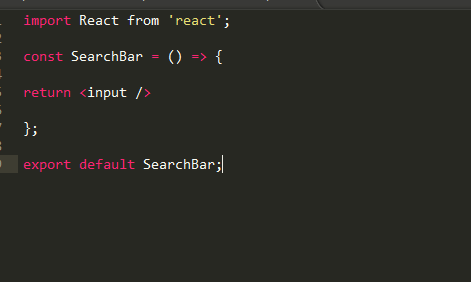


After installing it

This is the package.json



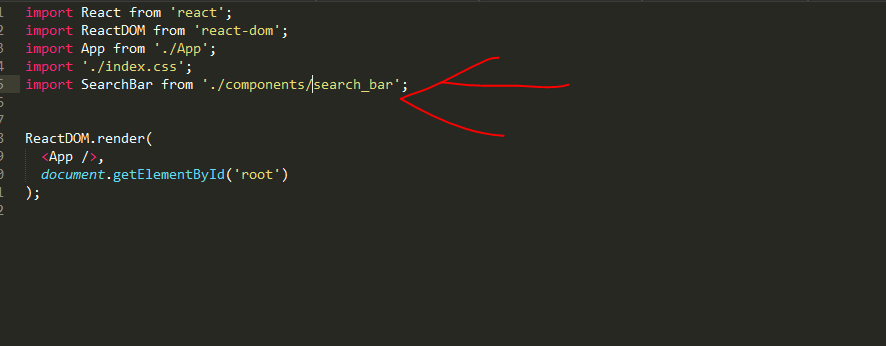
Make function of a file available



This will let the “import SearchBar” be available and only import the function

When it is a library that we didn't write it is like import React from ‘react’

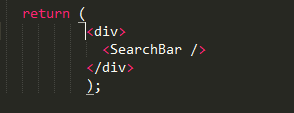
But when we import a file that we created like search bar we need to write a relative path.



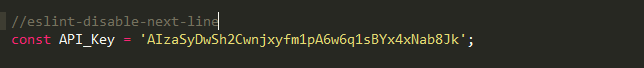
Notes:

Always import React from 'react'; for the files that will have jsx

When we return a multiline we need to put the statement inside a parethesis.



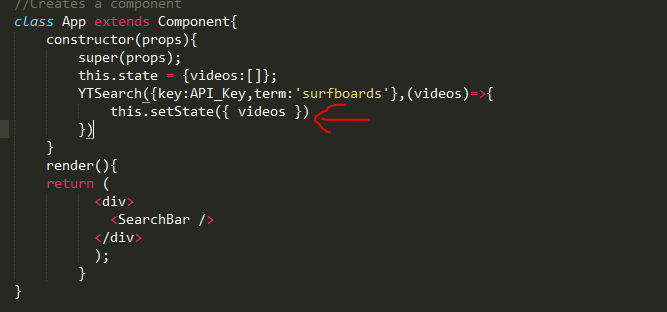
If get a warning we can use, to ignore the warning



When referencing javascript inside jsx we use {} to wrap it e.g.

{this.state.term}

ES6 if lets say for we want to set a state after we have done an ajax call and if the name of the state and name of the response are the same we can use

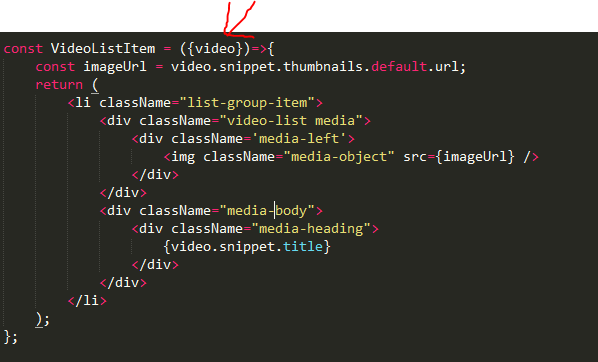


This is like YITSearch({},function(videos){

this.setState({videos:videos})

})

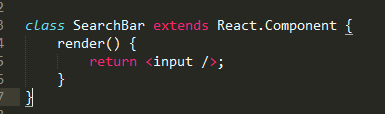
In this case normally it would say props but if we want to use it inside the function as const video = props.video we can use ES6 and replace the props with ({video})



Downward data flow

Most parent component should be fetching data (api,flex,redux)

Class Components

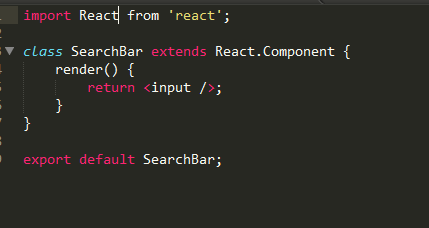


Using ES6

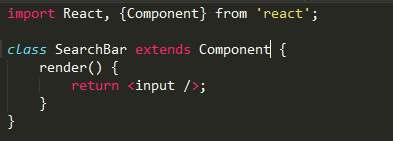
Instead of using

* import React from ‘react’;
* import React, {Component} from ‘react’;

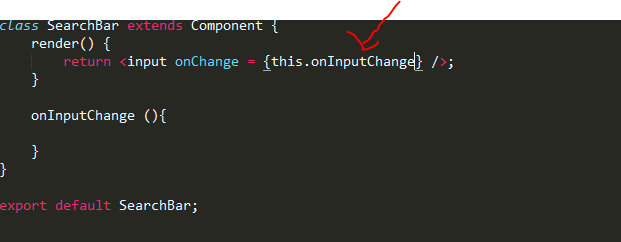
Before



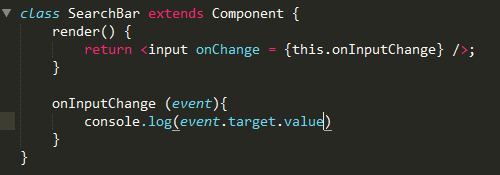
After



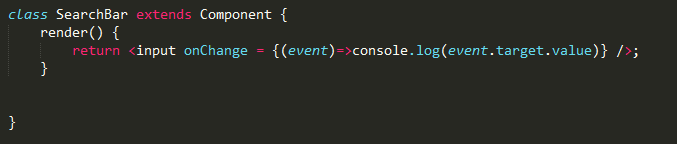
Event Handler



Get the value of input

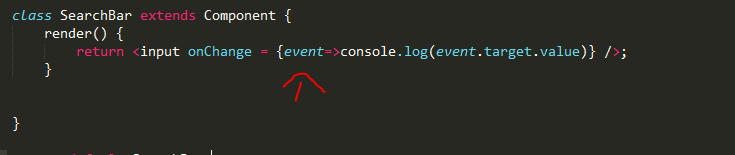


After ES6



OR

No parenthesis

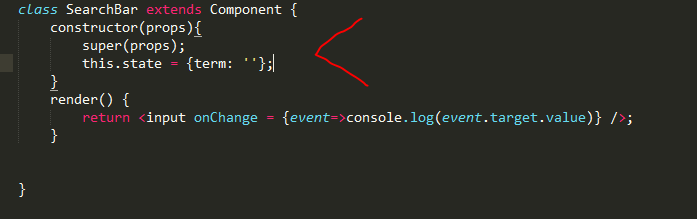


State

Record in react to user events

*Functional components doesn't have state only class components*

Declare a state

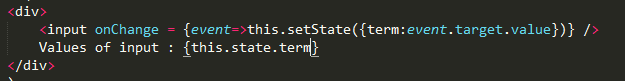


Change a state

*Do not use this.state.term to change the state e.g this.state.term = …*



When referencing the values of state we can use this.state.term



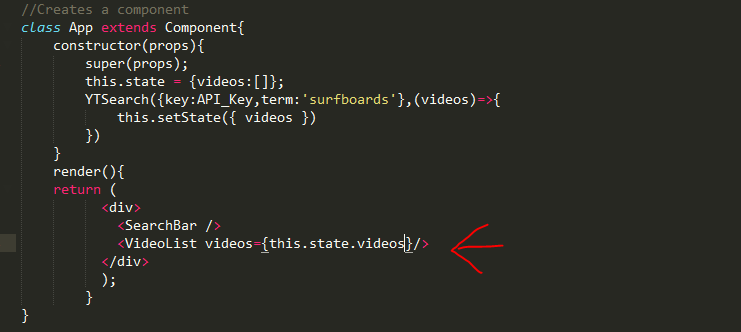
Google API

Console.developers.google.com

Passing data from parent to child

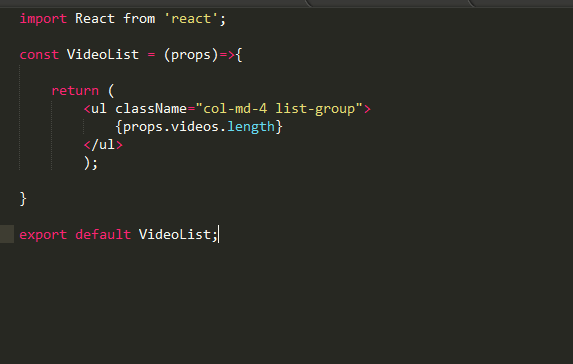
Index to video list

Index



Video\_List.js

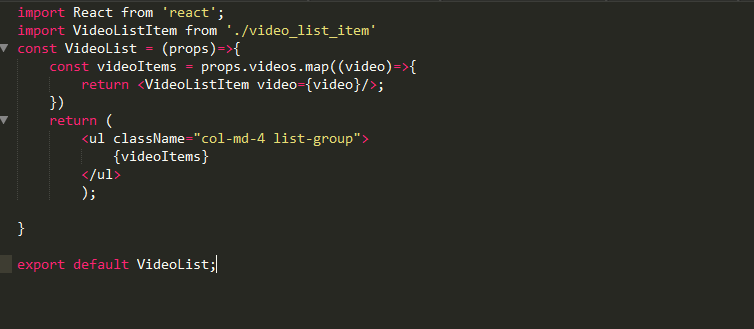
Since it is a functional component the argument arrives as object called props



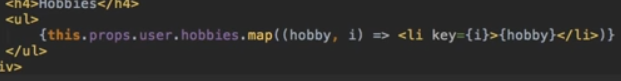
Rendering a list into a page

We are getting the list of videos from index.js so now we use a loop(map) to make them an item of <VideoListItem>, in this case we pass the video to video list js and in this file it takes it and make it an element and then returns it, we then save the return item in a const and then call it from the <ul>

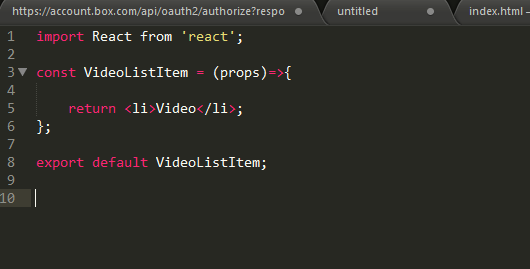
VideoList



OR



VideoListItem

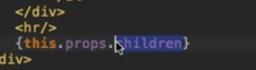


Another way to pass information from one component to another

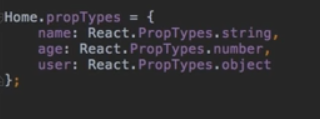
We have a home component and we will like to pass <p> element.



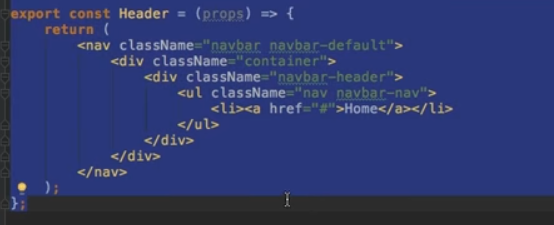
In the home component we use



Checking props type



Stateless components



Load boostrap

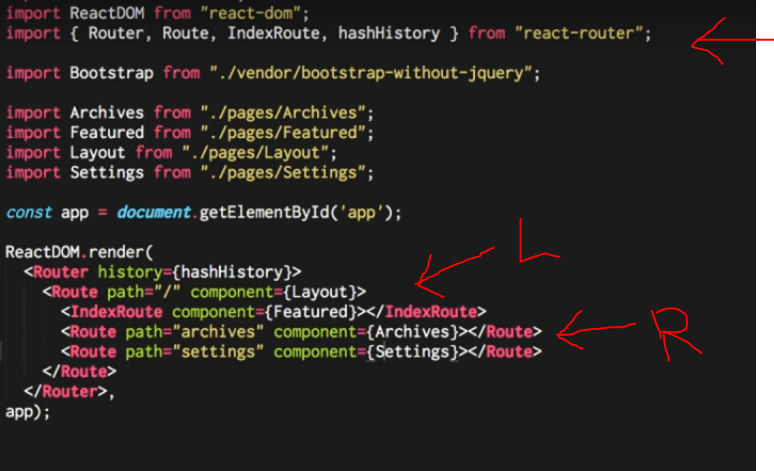
npm install react react-dom bootstrap react-bootstrap babel-preset-react --save

import Bootstrap from 'bootstrap/dist/css/bootstrap.css';

Router

Npm I react-router –save

Npm I history@1 –save

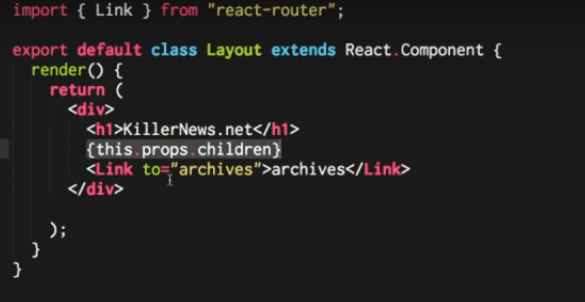


*Instead of normally rendering a layout we will have to render the router*

*L: This is the index page.*

*R: This should be the rest of the paths*

*This would be the layout setting.*



*{this.props.children} is where we would want to render the content of the other routes.*

*Link tag will be the button that will let us move through the rest of the routes.*

*There are many ways to do the routing*

*-instead of Link we can use a function and it can be history.replaceState or history.pushState, but they do different things when using the back button.*

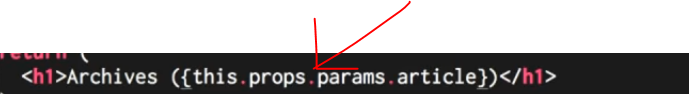




*To add a params*

*In case we want for it to be optional it will be path=”archives(/:article)”*

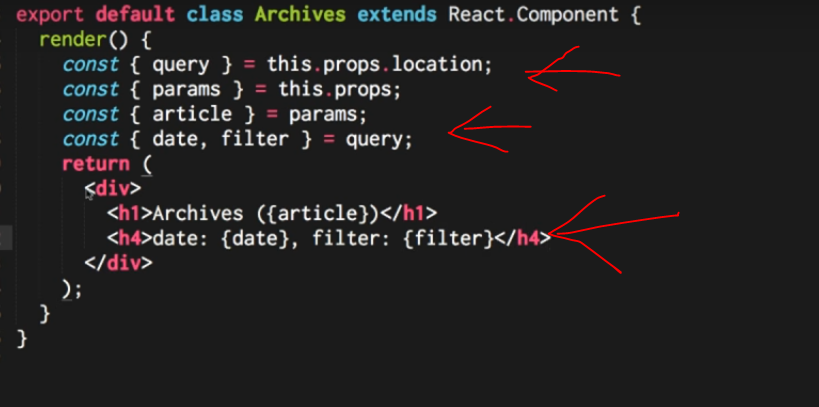
*To access the params given*



*When adding queries*

*E.g :*

*url/archives?date=xxx&filter=xxx*



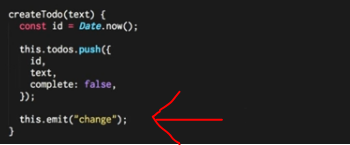
Flux

Emitter

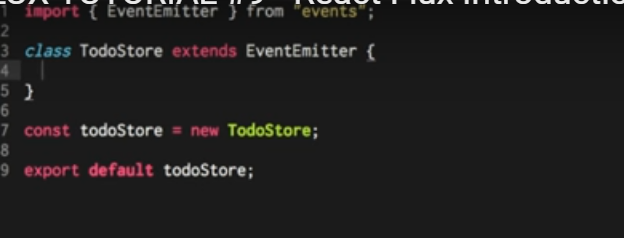
This will make that when we do something like create we can change the state with the componentWillMount function



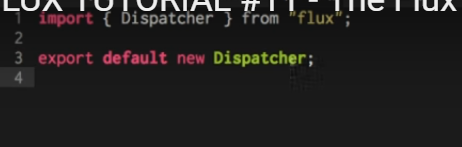
*This is how it is used*



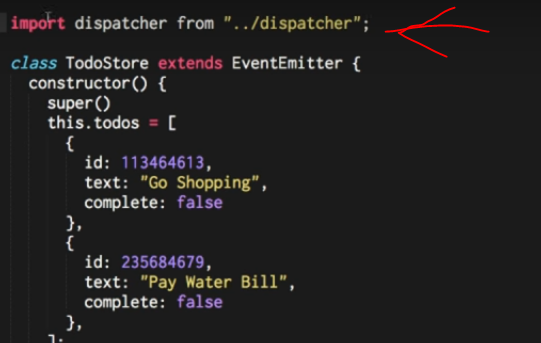
Store



Dispatcher



Register our store as a listener



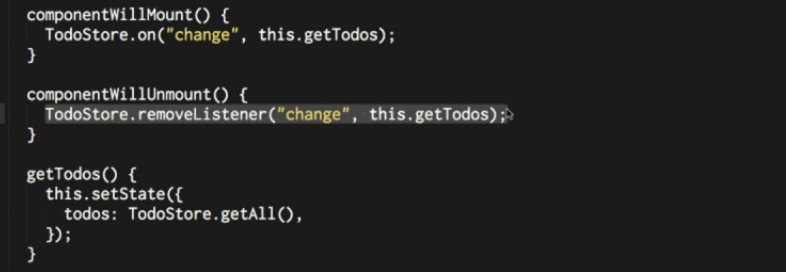


*We will do a method that will handle all the actions labeled in the Store.*

Actions



Memory issues

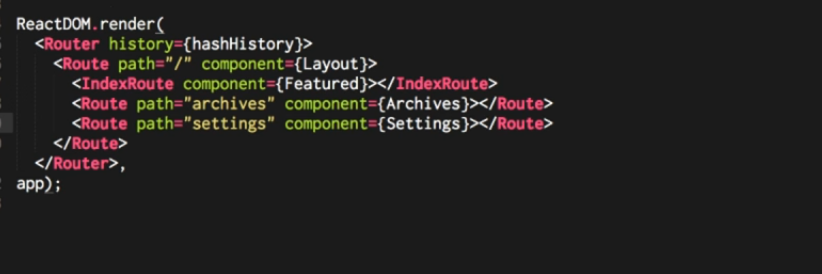


Router process

* In client.js we need to import react-router, also in here we will need to import all the other paths to use it

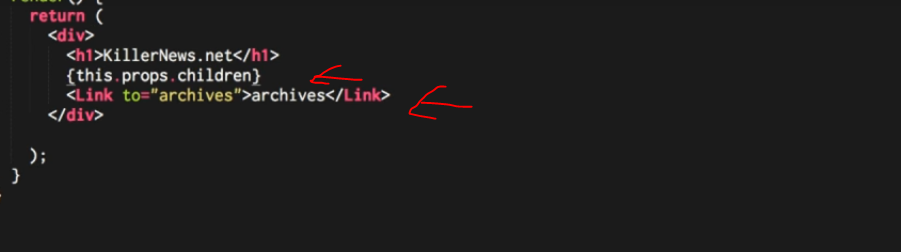


* Instead of rendering layout we will render Router with the different paths



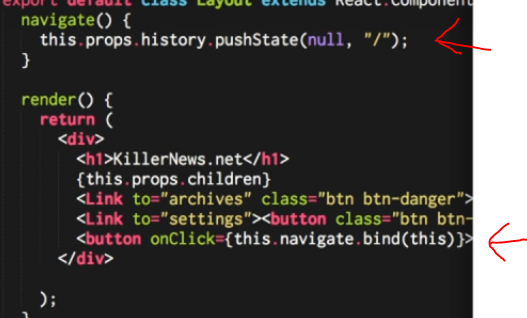
* In layout.js we need to import link from react router and use it like this, and also for rendering the component meaning other paths we use {this.props.children}

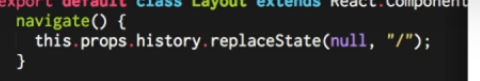




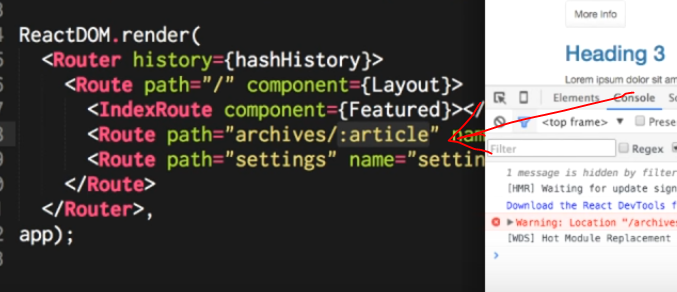
Different ways of navigating

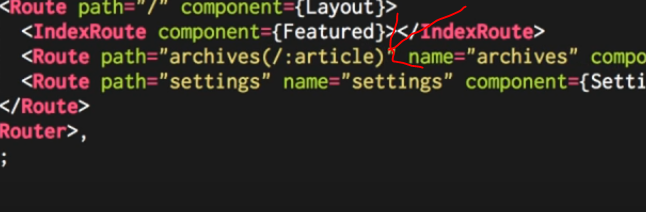
-We do a function call navigate and use the following, using the pushState we get the back button, if we use replaceState there is no back button





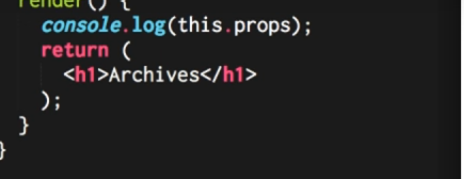
* In case we need to use id in the url , but doing this it will need to be matching always so there should be always id, if we want it optional since the other way will give us an error we add parenthesis

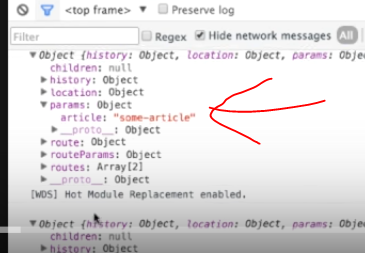




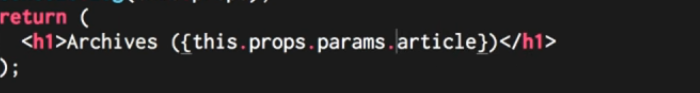
e.g if we use xxxxx/archives/xxxx

* this will be the result (this is in archives)

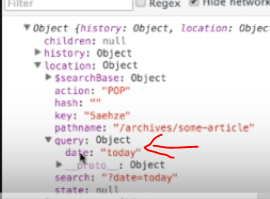




* This will be how we use it.

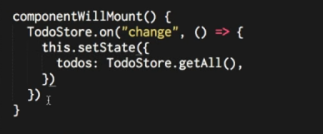


* In case we have an url xxxx/archives/xxxx?date=xxx

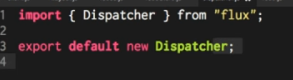


Flux process

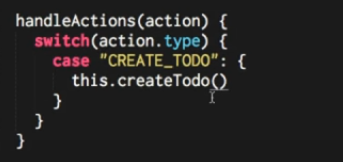
* Create the store (e.g CreateTodo,getAll,etc), we will need to import this in the Todo.js (page with the todos),
  + - * In todostore if we want to expose this globally and run it in the console we need to do a window todoStore = todoStore and then we can use it in the console log like todoStore.createTodo(‘xxxx’)
  + In the CreateTodo we have to emit a change
* Create the componentwillmount function in the todo.js



* Create a dispatcher, import this dispatcher in the TodoStore



* Go to the todostore.js and register the dispatcher
  + Create a handle method that will handle all the actions





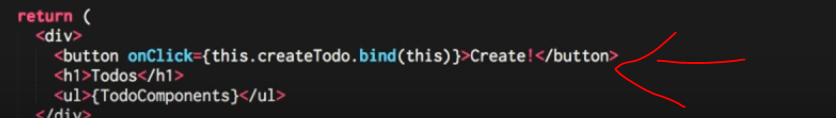
* Create Actions, import dispatcher file in action.js, also import this action file into Todo.js



Import this in Todo.js the \* will import all the exports



We create a button that will create a todo, we will need to create a function that creates the todo in the Todo.js and this will call the TodoAction create





Steps to create app

Create Client.js

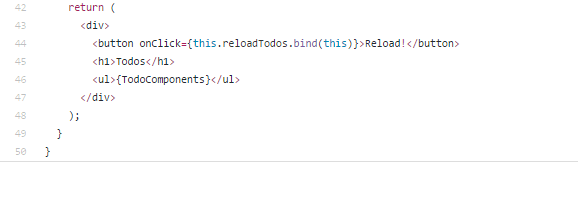


Create Layout



Create the rest of routes

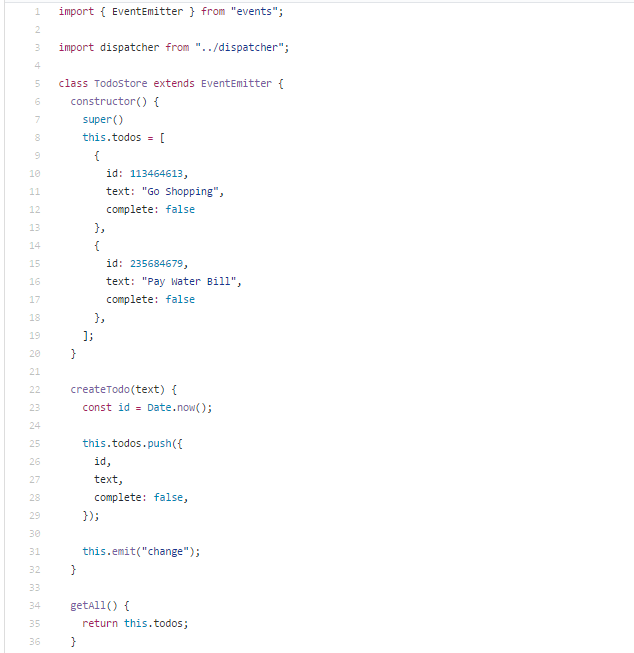




Create Todo component



Create Store





Create Actions



Uninstall npm

Npm uinstall xxx –save

Redux

Npm I redux --save