

Course Started on July 1, 2021

To Learn in this module

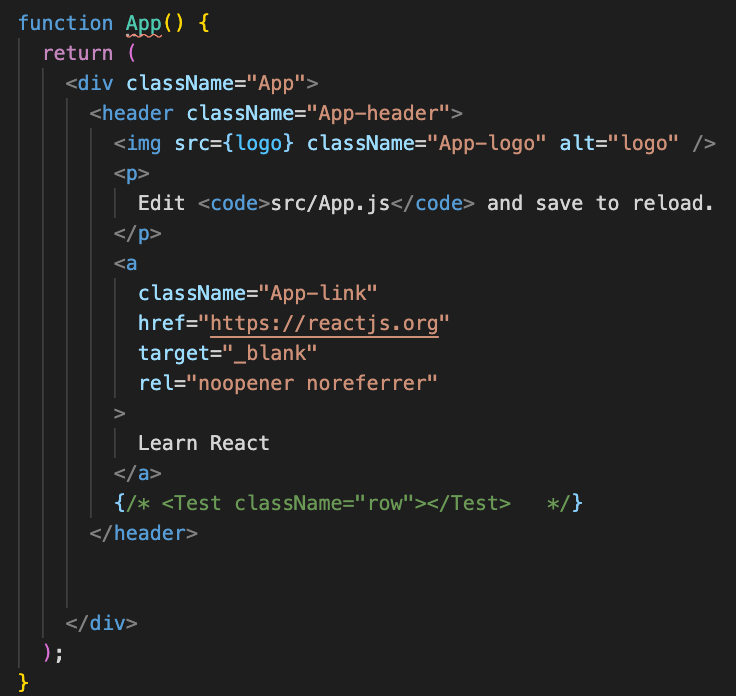
1. React Basic

Section 2

* 1. Creates a virtual DOM to minimize the actions in the actual DOM. As movements in DOM is an extensive operation that affects the performance of the web.
  2. React is a declarative which means that everything is declared via JSX and changes via state or data. State is normally an object that serves as a source of information to be pass on display in the browser via JSX thru various functions.
  3. One-way downward data flow. There’s no way a data will go up. These restrictions improve the performance of the web app.
  4. React principle is to concentrate in the creation of components to build an app. A component is more manageable, scalable and easy to debug.

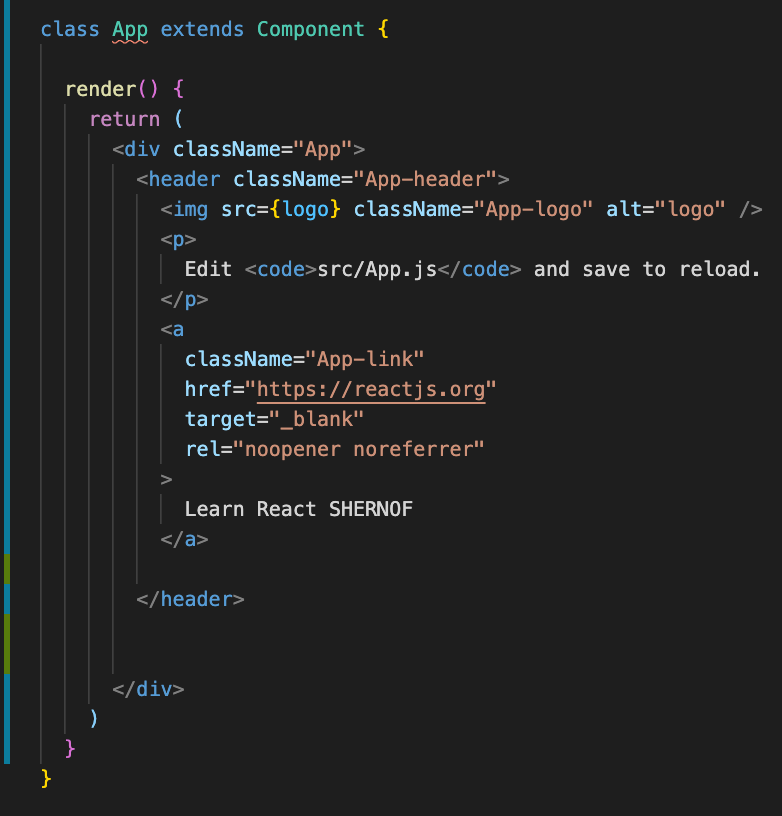
Section 3

1. Babel and Webpack are responsible to translate react code to older version of JS and HTML that browser understand. Create-react-app automatically configure this, so do not eject if necessary.
2. Create React App 2
   1. React-scripts makes not to worry about the config of babel and webpack.
   2. Public folder is the location when for the final build html file (index.html). these are the files that browser understands
   3. Src folder is the workbench. This is the location of the files we’re going to work with.
   4. Eject if you don’t want to use the auto config babel and webpack
3. React Project Startup
   1. A build folder is created when npm build is run. This contains the minified version of the files from public. This contains the files for deployment.
   2. Babel makes sure that the JSX will be understood by different browsers, similar to what jQuery does.
   3. Webpack is a modular binder, it allows to build modular codes.
   4. React library allows to write JSX (JS and HTML) codes
   5. ReactDOM interacts with the DOM. It grabs the #root and insert the <App /> into it.
4. Do not eject
5. Hooks vs Classes
6. Class Components
   1. Import { Component } (React.Component) in order to write class components, then change the function to class then extends components. The Component enables the access to render() method and the render() returns the html inside the previous function.

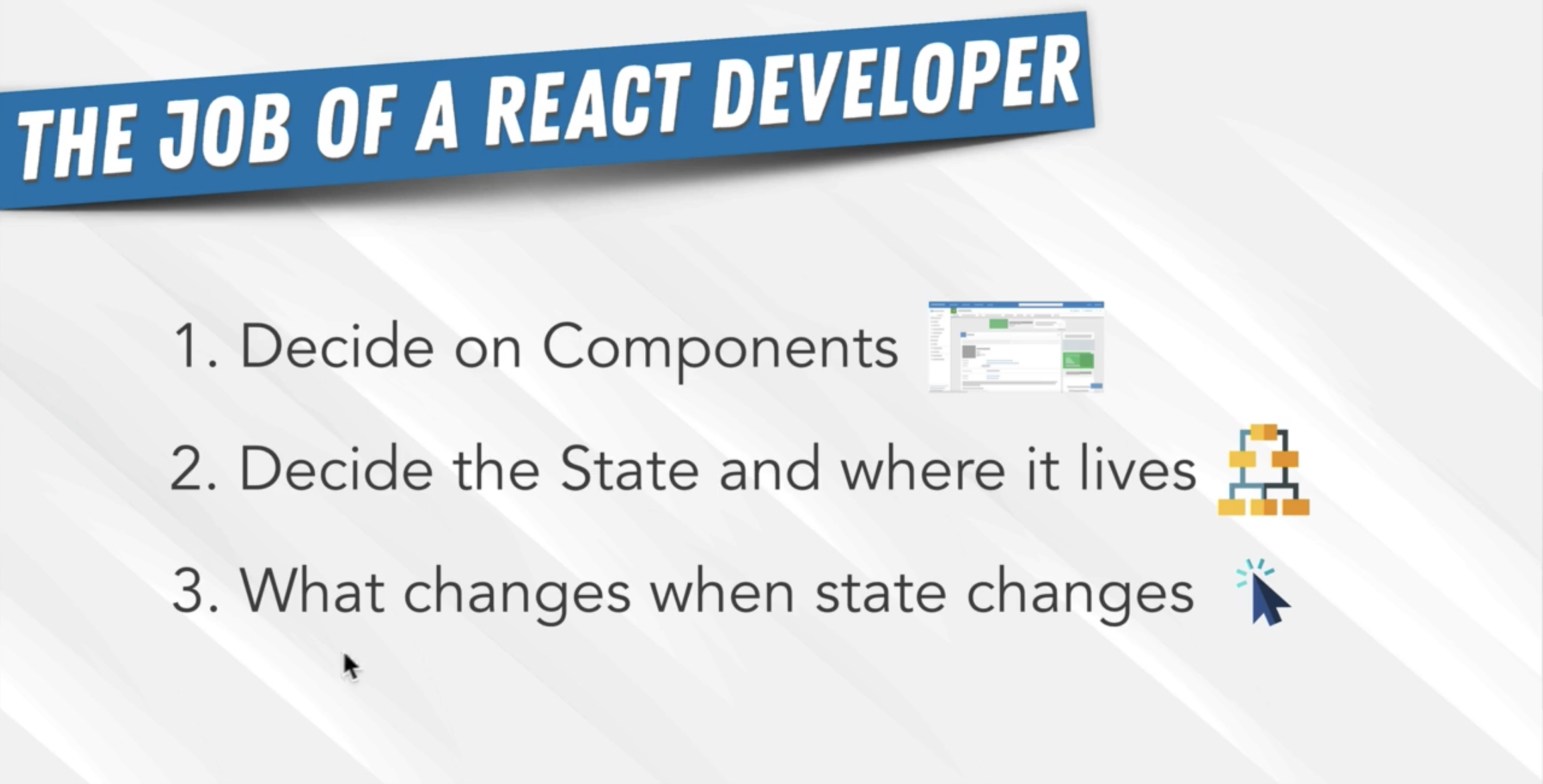




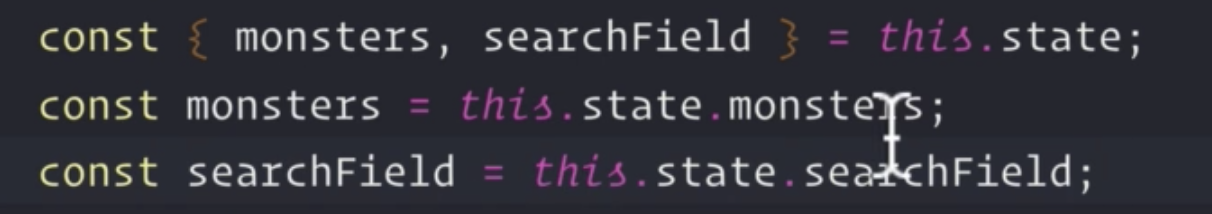
* 1. A class component gives an access to “state”.
  2. State = data; dynamic data. It’s a JS objects with properties that can be access at any point inside the class.
     1. This is done by calling the constructor(){} keyword and
     2. From here we can call super() from constructor(){} and it helps in “this” and gives access to “this.state”.
     3. This.state property now can be set to something.
  3. The Component also gives access to “setState”
     1. setState let’s the state inside the constructor to be modified.
     2. At this point we have now the access for the html to the methods such as “onCLick” which takes a function that takes the setState which takes the object that contains the properties that needs to be change in the state as well as the new state value.
     3. This gives the control of what I want to display in my components

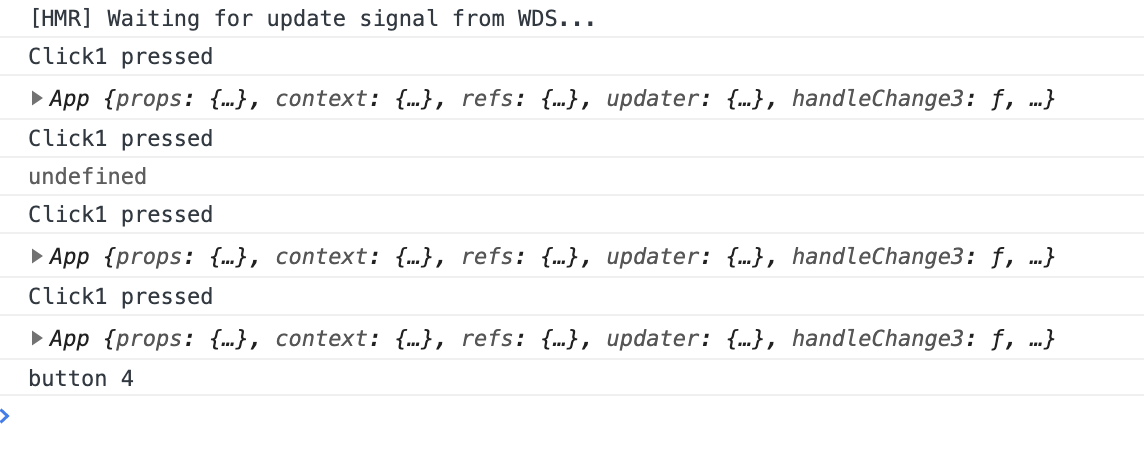
1. Thinking JSX
   1. JSX consists of HTML tag inside the JS
   2. {} means javascript expression
   3. In React, it is not allowed to modify state without calling the setState() method. This is to comply with the unilateral data flow of react.
   4. Data flow = state >>> render >>> setState >>> re-render.
   5. We can use whatever html attributes in react such as src etc.
   6. JSX mimic HTML to create a virtual DOM.
2. Dynamic Content
   1. Start to create the Robots App
   2. This use the map() method to iterate the function to each of the properties in the this.state object
   3. Key = { angel.\_id } is necessary for react to know what element needs to be updated and re-render if there’s a call for setState or an update. Key = {angel.\_id} is always call upon using map()
   4. Learn here how to get data from array and display it inside the App
   5. 
3. See map @ 349
4. Single Page application
   1. Render() is built in inside the React Component library
5. Fetching content
   1. This lesson tackles on how the App calls the backend and put the data into state and then later be use during the rendering of the component.
   2. Lifecycle are methods that are called in different of which a component gets rendered
   3. This lesson used the componentDidMount() to fetch data from an api (<https://jsonplaceholder.typicode.com/users>). It then return a promise which takes a function .then(response => response.json()) and convert it to a json data, then converted the hardcoded objects of the “angels” array into the data from the api, in this case “name” was displayed in the browser.
   4. 
   5. Main lesson: use of componentDidMount(){} and utilization of API.
6. Learn more of the Promise in the bonus video
7. Architecting the App
   1. Be organize in files. At the end of the day it will still be converted by Babel and Webpack in the build stage.
8. Card List Components
   1. Props is the data received from state, Props is the data passed from state to component.
   2. Props has a property of children.
   3. Data from state is passed as props into the component inside the main component which is the App and receive as an argument or parameter in the child component file let’s say card-list.component.jsx.
   4. Props is inside the element while children is in between element bracket
   5. Children is a data in between an element tag.
   6. Summary >>> Use this.state.angels as a props in App.js in the <Cardlist angels={this.state.angels} /> and pass it on as a props to the Cardlist.jsx. Cardlist.jsx receives this as a props, thus the first map result of this.state.angel becomes props.angels in Cardlist.jsx
9. String interpolation
10. Card Component
    1. Learnt here how to use props as an incoming data from Cardlist component
    2. The output ‘angel’ from the map result was then pass to Cards as a props <Cards angel={angel}, the data then become props.angel.name as a sample.
    3. Import the Cards.jsx into Cardlist.jsx
    4. Notice here the data transition especially the ‘s’ in angels.
       1. From App.js = this.state.angels. it was mapped thus it resulted to angel. This becomes a props in <Cardlist angels={this.state.angels}/>
       2. In Cardlist.js = props was received thus this.state.angels >>> props.angels in CardList.js. This was passed on to <Cards angel = {angel} /> as a result from mapping.
       3. Cards.js = angel = {angel} was received as a props thus resulted to props.angel
       4. See the transition of data as a result of state passing to props to different children.
          1. App.js = this.state.angels
          2. Cardlist.js = props.angels
          3. Cards.js = props.angel
11. Breaking into components



* 1. This provides flexibility to each components. Upon giving new props the component can display different information .
  2. It resulted to performance improvement.
  3. Separation of concerns
  4. Easier to test and debug

1. State vs Props
   1. State is the main data that change during user interaction. Normally in the parent component. State can be put anywhere though.
   2. Props is the received state data. This becomes attribute of the component see 35.d.iv.
   3. This follows the unidirectional data flow
   4. Note: components main function is to render.
   5. Classes vs Hooks
      1. To create component using classes, it will give the liberty to use the setState functions which is a part of the react.component.
2. Search Field State
   1. Learn different attributes of <input /> and other elements
   2. Learn the events in react like onChange, onClick etc.
3. React Events
   1. React Events - <https://reactjs.org/docs/events.html#gatsby-focus-wrapper>
   2. HTML Events - <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events>
   3. HTML onchange event - <https://www.w3schools.com/jsref/event_onchange.asp>
4. Filter state
   1. Destructuring is pulling of properties from an object
   2. Object destructuring = <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment#object_destructuring>
   3. . First line (destructured) = 2nd and 3rd lines. Thus destructuring makes the coding simplier.
5. Review the filter() and includes() functions. Refer to Section 33.
   1. Here, input element was used with an attributes used were the placeholder, and type=’search’, (review different types of attributes).
   2. onChange( e => this.setState(searchField : e.target.value)). Used the onChange() method to define the function to run when there’s a change in input inside the input box. ‘e’ is the event and target.value is the native event of the browser. ‘searchField’ is a new state created by this function
   3. destructuring the this.state will result to const { angels, searchField } = this.state. This will make the initial ‘this.state.angels’ intact.
   4. const filteredAngels = angels.filter(angel => angel.name.toLowerCase().includes(searchField.toLowerCase())) >>> this variable filtered the this.angels that resulted to a new array angel and includes() an array method which use to check if the string value passed on (searchField) contains the value of the string called on in the input box.
   5. toLowerCAse() removes the sensitivity of the case.
   6. Use now the filteredAngels as the state to be use in the <Cardlist /> instead of the original this.state.angels. filteredAngels is where the input box is going to search base from the input and the DOM will re-render base from this function.
   7. 38-41 – learnt the use of filter() and include()s method
6. Search Box Component
   1. Functional component is enough if there’s no need for internal state or lifecycle methods.
   2. Here, we created the input box as a separate component named <SearchBox />.
   3. To be re-usable its attributes placeholder and onChange were destructured via { placeholder, handleChange }. This are the parameters that the SearchBox can now accept.
   4. <SearchBox type=’type’ placeholder={placeholder} onChange={handleChange} />
   5. Remove then the input box in the App.js and implement the <SearchBox placeholder=”search” handleChange={ e => this.setState(searchField: e.target.value) }/>
   6. This lesson is more on **how to create a functional component**.
7. Where to use the state?
   1. Lifting state up will make all the children components be able to access the state.
   2. Putting the state in the parent component is a good idea as all the children will have access to it.
   3. If there’s no much reqt for the info from the state of a parent component, then there’s a little need to put it up high in the chain.
8. Class Methods and Arrow Functions
   1. ‘this’ is a special keyword in javascript that references the context of where its being invoke.
   2. Use arrow function in creating methods inside a class.
   3. This lesson shows how to create a method inside the class component App
   4. 
   5. Thus <Searchbox placeholder=”cari” handleChange= { handleChange } /> from <Searchbox placeholder=”cari” handleChange= { e => this.setState({ searchField: e.target.value })}
9. Exercise Event Binding





Click1 – nothing happened as this.Change1() will just run automatically as it was immediately invoke when the script is loaded

It displayed the App(){} as the handleChange1{} is initially in the context of the App

Click2 – Displays ‘Click1 pressed’ as it just assigned handleChange1()

It displayed ‘undefined’ when ‘this’ was console log because handleClick1() was not binded to App

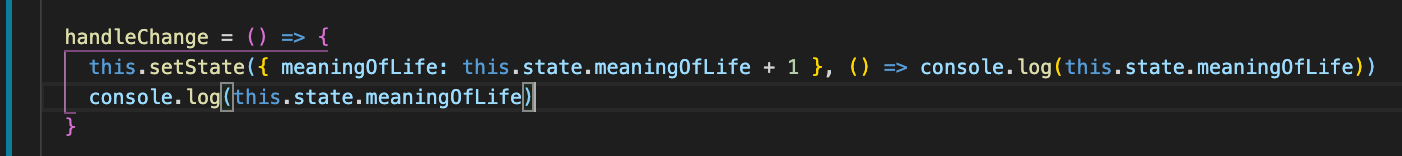
Click3 – it displayed App because handleChange2() was binded in the app, see this.handleChange2.bind(this)

Click4 – is a simplified version of handleChange2() using Arrow finction

See p8

1. Quick note

Use arrow functions on any class methods you define and aren't part of React (i.e. render(), componentDidMount()).

1. OptionalGIT + GITHUB
2. Connecting SSH to Github
   1. Create a repo in github and copy the ssh key
   2. Git remote add origin “ssh” at the terminal
   3. Npm or yarn add gh-pages
   4. At package.json
      1. Add “homepage”: <https://shernof.github.io/react-basic>
      2. Create the deploy scripts
      3. “predeploy”: “npm run build”
      4. “deploy”: “gh-pages -d build”
      5. Run npm run build in the terminal
      6. In settings, make sure that the Source is in github-pages branch
3. Deploying the app
4. NOTE: CORS Error
5. React and React DOM
6. Latest React Package Update
7. Revisiting DOM
8. Asynchronous setState
   1. Asynchronous JS doesn’t happened immediately. Use callback to let the code happen instantaneously
   2. 
   3. Above is the syntax to make the code instantaneous
   4. This is a bad practice especially if the App is getting big. No guarantee that the state will be updated due to the reason that JS is doing a batch setState update
   5. React rule for this are:
      1. Use object if no computation required for setState
      2. Use function and use the 2 parameters ‘prevState’ & ‘prevProps’. The prevState will ensure that before the update, the App has the latest state
      3. See below how the 2 parameters were used
      4. 
   6. Use props in the constructor and super(props) it’s a handy and good practice.
   7.  
   8. Above is the use of props in the constructor
   9. Summary:
      1. Use function and the 2 parameters prevState and prevProps if setState doesn’t involve computation
      2. Use object notation if setState doesn’t have any computation
      3. Use a 2nd function or a callback if there’s a need to use the updated state

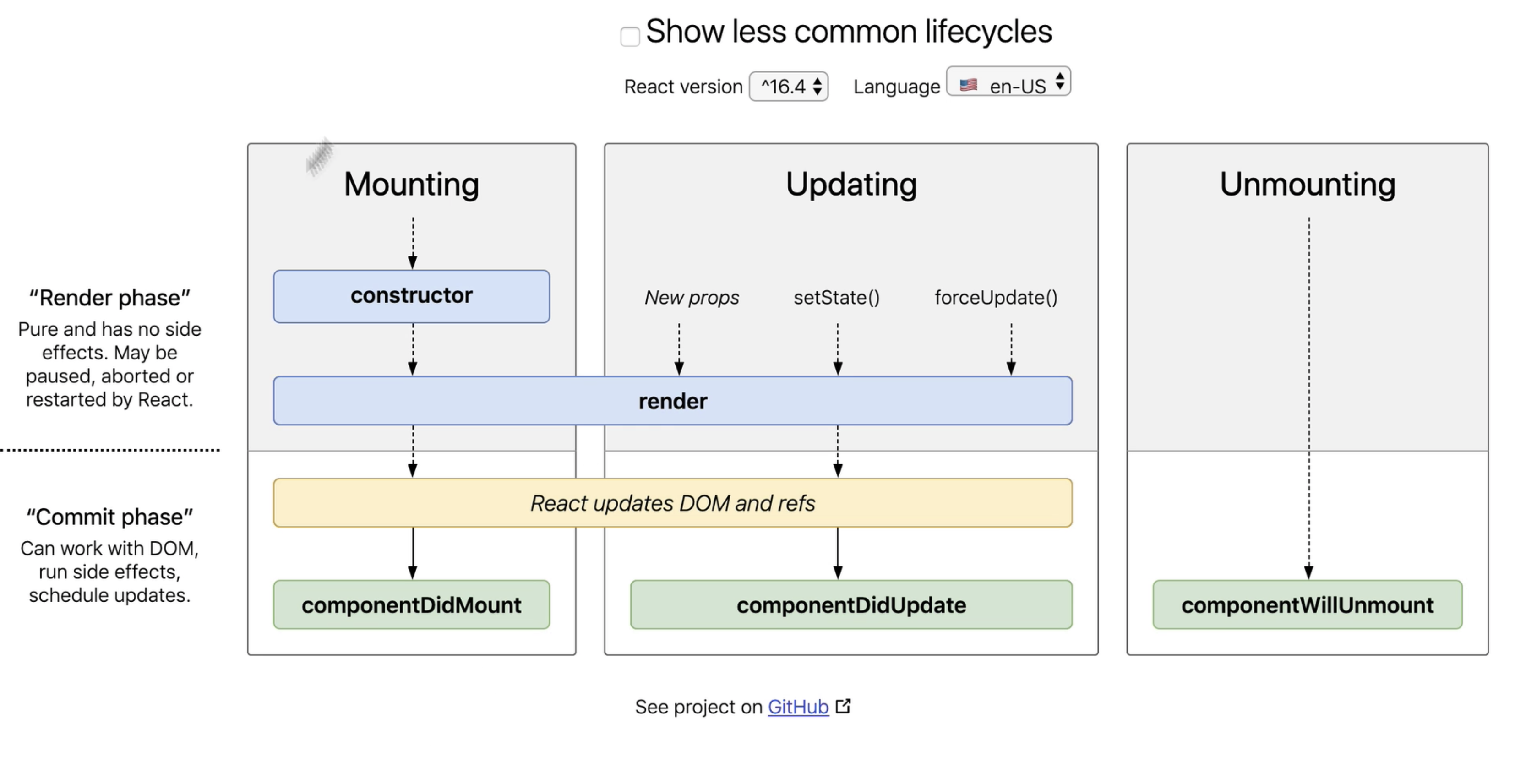
j. See p10

55. Introduction to Life Cycle Methods

a. use of bang operator (!); very useful for toggling the state with Boolean type

b. this lesson shows how to display and remove a component from the DOM by using the ternary operation

56. React Lifecycle Method – Mounting



a. State is located in the constructor so that it will initialize first prior to loading of all context of the class.

b.

57. React Lifecycle Method – Updating

a.

58. React Lifecycle Method – Unmounting

59. Section Review

60. Endorsement on linkedIn

61. Long Road ahead

62. Project overview

63. Github + project repo

64. Course guidelines

65. Quick note about github

66. Update CRA & React

**Section 2 – Create the Ecommerce web app**

67. E-commerce setup and sass

a. remove unnecessary files, remove service Workers code in index.js

b. clear App.css

c. install npm i node-sass

d. create the HomePage.jsx

68. Project files + modules

69. Project Component Architecture

70. CSS and scss files

71. Homepage and Directory Components

a. Create the MenuItem.jsx, make it dynamic by assigning a value of the title from destructured props ({ title }) which will come from <Directory />

b. Create the class <Directory />. Use the data provided temporarily (directory.data.js).

c. Using normal map = this.state.sections.map( section => (<Menu key={id} title={section.title}>))

d. Here, **destructuring the sections object was used**. this.state.sections.map(({title, imageUrl, id}) => <MenuItem key={id} title={title} image={imageUrl}). Notice the syntax is simpler. Pass this to <Menu> as props see ({ title, imageUrl, id })

e. export the <Directory> to <Homepage>

f. In react, it allows the JSX to use another property called “style” which is for css purposes. It takes an object that has properties as of css, takenote that it should be in camel case format. This allows the app to have a dynamic css options.

g. Implement this in <Menu> see style={{ backgroundImage: `url({${imageUrl}})` }} and add properties of the backgroundImage in the scss, see background-position and backgropund-size

h. React also has a feature to selectively apply a css property coming from data.js which is size in the className. Then style this size in the menu.scss. this is to achieve a larger images of the menus 4 & 5. **Notice the syntax from className=”menu-item ” to className={`${size} menu-item`}**

72. Styling Menu Items

a. add background-color: white and opacity: 0.7 in class content

b. Add opens-sans, add this in index.html and implement in App.css  
 c. use { title.toUpperCase }

d. create another div to include the image. This is so that the content div is not affected when the hover effect on the image takes place. Use overflow: hidden so that the image will just grow inside the div

73. Updated Google Fonts

74. Quick Note: React 17

75. Updating Packages to latest version

Section 5 – Master Project: React Router and Routing

76. Routing in React

a.

77. Routing the Project

a. npm I react-router-dom

b. import { BrowserRouter } from ‘react-router-dom’ in index.js and wrap the <App /> inside it. BrowserRouter is a component gives the functionality of routing to the component <App />

c. import { Route } from ‘react-router-dom’ in <App />. **Basic parts of <Route exact path=’/’ components={Homepage} >**.

d. exact displays only what is the exact value of path

e. path is the url

f. is the page to display

g. <Switch><Switch/> = it renders only the route that it sees first. If it sees a match then it will stop rendering the other

pages.

h. <Switch> + exact will ensure that the correct page will be displayed

78. React Router DOM

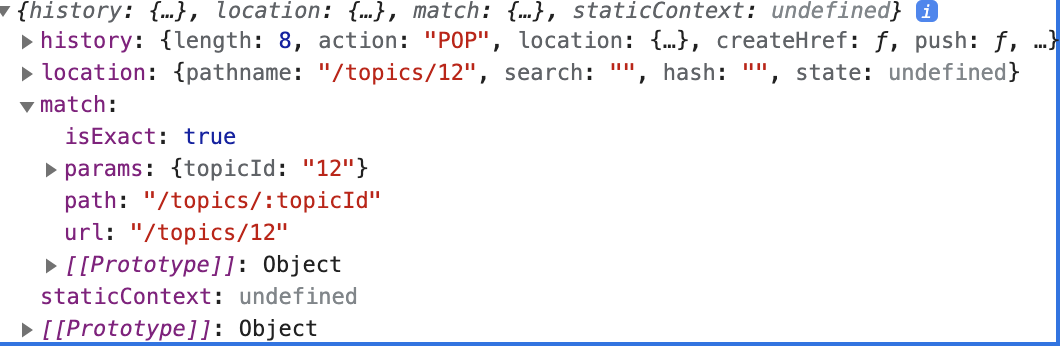
a. There are three properties returned back by <Route>; history, location and match see props in <Homepage>

b. Match has 4 parameters:

I. url = just the url

ii. path = defined url

iii. params is an object that contains the id. This will give now the topicId



iv. isTrue a Boolean to say if the path is correct

c. <Link> is similar to <a>; <Link to={`${props.match.url}/13`}> <<<< dynamic way of navigating

d. History – can be use in a method to direct to a component t render

e. Location.pathname

f. see routing app for the syntax

79. withRouter()

a. it solves the props tunnelling

b. this is a higher order type of function that takes a component as a parameter and return the modified version of the said

component. Once use, the location, history and match properties of Router can now be access.

c. Use it in <Menu>. Export default withRouter(Menu). Can use now the history and match.url in the onClick={() =>

history.push(`${match.url}${linkUrl}`)}. The linkUrl came from the static data in the <Directory>

Section 6 – Forms & Components

80. Shopping Data

81. Shop Page – Previews of collections

a. create the class comp <Shoppage>

b. import the SHOP\_DATA to <Shoppage>

c. after <Shoppage> creation import it in <App>; <Route path=’/shop’ component={Shoppage}>

d. Build the <CollectionPreview> component. **Items.map to get the title**.

e. in the <Shoppage> do collections.map and destructure the array (…otherCollectionProps). Don’t forget const

{ collections } = this.state to make the this.state intact. Take note the use of **items.filter().map()** to display 4 items only

f. watch out on how to handle the performance concern in items.filter().map() as this anonymous function will always be rendered if the <CollectionPreview> is rendered or re-rendered.

82. Collection Item

a. create the <CollectionItem>. Destructure and get the imageUrl, id, title, price which came from the state in <Shoppage>

b. import the <CollectionItem> to <Shoppage>

c. destructure the item (map) in <Shoppage> to be able to pass it as a props to < CollectionItem >; use spread operator

{ …otherItemProps }.

d. put the images folder in public

83. Header

a. <Link> is use here

b. take note on how to implement the <Logo>. Import { ReactComponent as Logo } from …..

c. Put the <Header> on top of <Switch> so that it’s always visible.

d. style the a{}

84. Logo

85. Forms in React

86. Sign in component

a. create the <SignInAndSignUp> page

b. build the <SignIn> class component

c. use the native <form>

d. create the handleSubmit and handleChange method for <form>

87. Form input component

a.

88. custom buttons

Section 7 : Firebase and user authentication

89. Section overview

90. Firebase introduction

a. Backend development is related to database, authentication and server.

b. A

91. Important Note

92. Adding project to db

a. Sign in to firebase using google acct

b. go to console >>> create and name a project

c. Configure the firebase. Click web </> in the project overview page >>> register the app (no need to setup the hosting) >>>

copy and paste the object from the script given by firebase in the firebase.utils.js

d. nom I firebase

93. Note about github

94. Google sign in authentication

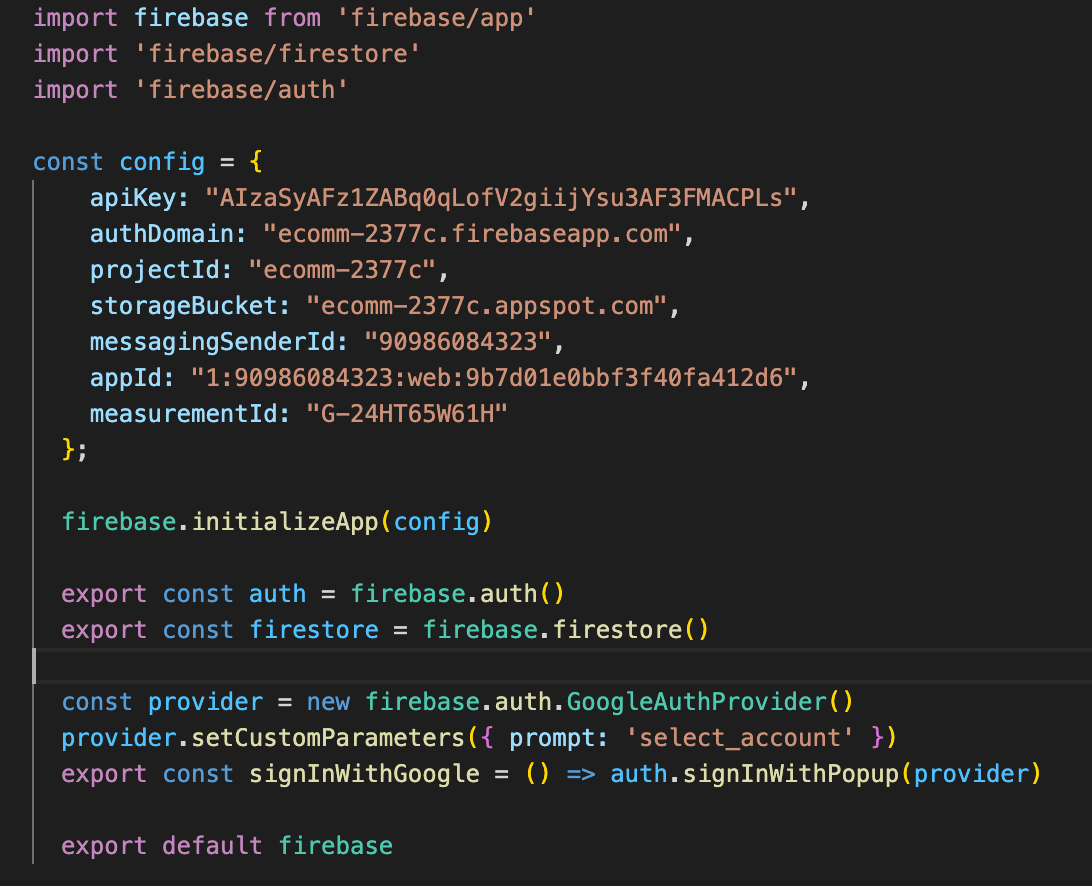
a. add src/firebase; configure the firbase.utils.js

b. At firebase;/authentication >>> sign in method. Enable the google

c. import “signInWithGoogle” in the <SignIn>

d. Create another <CustomButton onCLick={signInWithGoogle}

e. below is the firebase utils

f. 

95. Cloning from this point on

96. Google Sign In authentication 2

1. React Router see – Section 5 76 to 79
2. Firebase
3. Redux
4. Payment method
5. CSS in JS
6. Redux Saga
7. Hooks
8. Context API
9. GraphQl
10. Performance considerations
11. PWA
12. Jest Testing