**Introduction to React @ 19 June 2019**

Install React Developer Extension for Chrome browser (not sure whether this will work on production)

Be in the right folder on terminal

* sudo npm I -g create-react-app (similar to express-generator)
* create-react-app hello-react

we don’t use nodemon because the script in react is set up so that it has its own system of monitoring changes and reloading.

* Npm start or yarn start (this will open up in localhost:3000 in browser)

Leave the dev server and watch for any changes and will reload in the background.

Most of the work is done in the “src” folder

Entry point – index.js

**Dynamically create DOM elements**

In Index.js, add underneath the import

const element = React.createElement('h1', {

style: {

color: 'red'

}

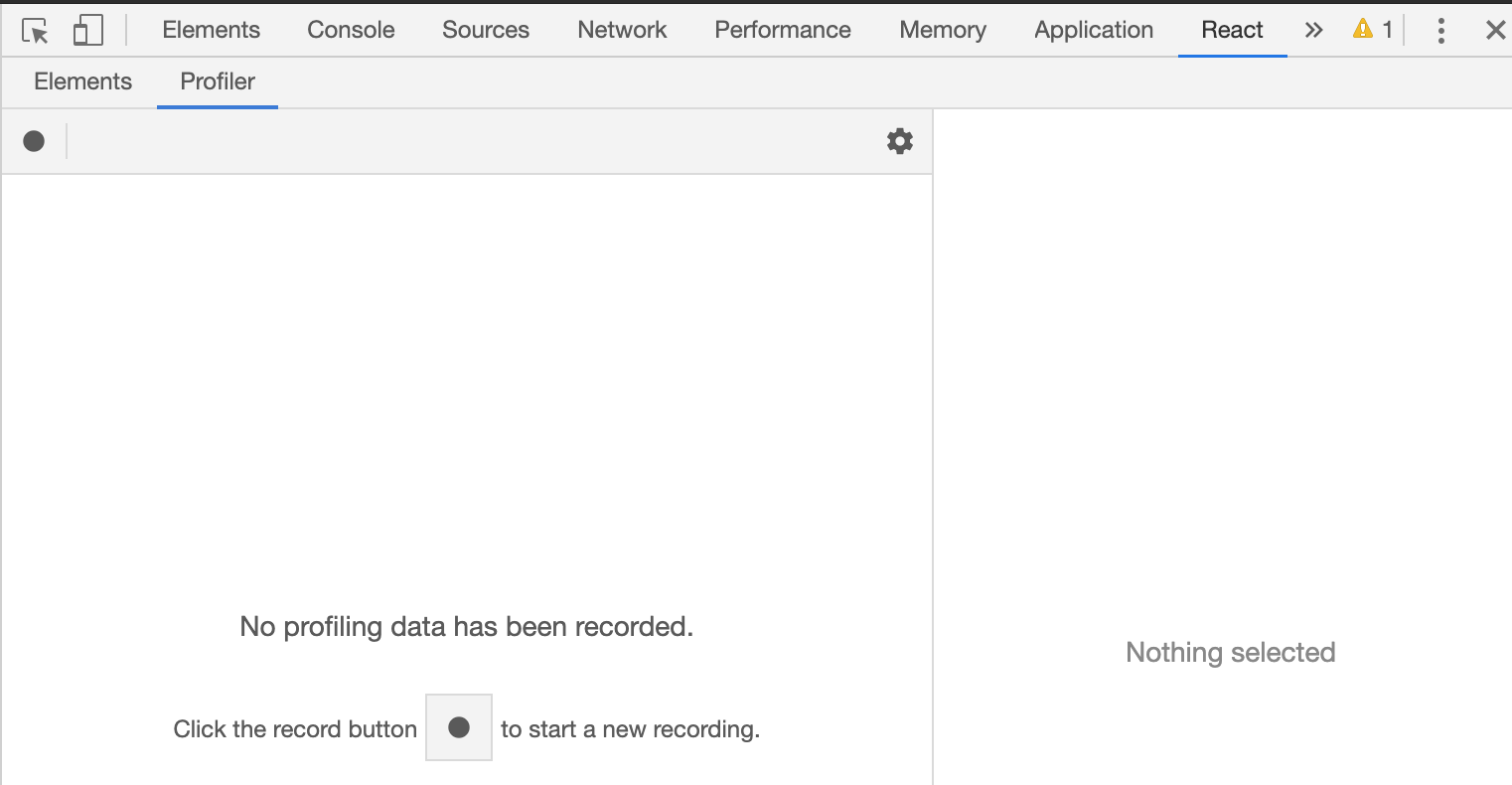
}, 'Hello World!')

This isn’t part of the DOM yet, just in memory. To see it on the page, need to tell React ot render it.

Add to index.js

ReactDOM.render(element, document.getElementById('root'))

Profiler to test performance issues, what is taking loading so long



You can change the value of the props in the Elements view of the React Dev Tools

You can nest react DOM create elements

const element = React.createElement('h1', {

style: {

color: 'red'

}

},

React.createElement('ul', {},

React.createElement('li', {}, 'Item 1'),

React.createElement('li', {}, 'Item 2')

)

)

It runs in the browser, but everything is inside a module so you can’t access it through console.

This is unmaintainable and bloated way of doing things. Write HTML that is integratable with javascript/react code 🡪 JSX (Javascript and XML)

**JSX**

const element2\_2 = <h2>Goodbye</h2> // JSX version of element2

There are no quotes around the html. You are not creating a string but an element. JSX creates an actual elements.

Nested html elements in JSX

const element3 = (

  <h1 style={{ color: "red", textDecoration: "underline" }}>

    <ul>

      <li>Item 1</li>

      <li>Item 2</li>

    </ul>

  </h1>

)

JSX is being compiled to straight Javascript via React. For this to happen, a server has “babel” is a transpiler. It takes whatever language you are working in (i.e. JSX) and compiles it down to javascript. Babel is configured when create-react-app runs in the terminal

Development builds is not minified. In production, the code will be minified.

In practice, we will put it out in its own file and import it into index.js. module.exports is common JS.

React uses ES6, which introduces its own method of doing modules. So instead of using ‘require’, we use import. (different syntax but same as const express = require(‘exprss’))

When importing multiple things

import { something1, something2 } from "something"

when exporting, just use the keyword ‘export’ in front of the thing we are exporting

**Make your own component (simple/functional component – the simpler way)**

In Index.js

function Project() {

  // return the JSX that we want to render

  return <h3>My Projects</h3>

}

Add the name of the function in a tag format in the ReactDom statement in index.js

ReactDOM.render(<Project />, document.getElementById("root"))

This is how React distinguishes between a HTML element and a component by looking at the first letter capitalisation.

**To modularise this**

Can put all components in a folder called “components”

Inside “src”, create fodler “components”

File name should be the same as the name of the component AND capitalised (“Project.jsx” or “Project.js”)

Add Project.jsx

import React from "react"

export function Project() {

  return <h3>My Projects</h3>

}

In index.js

import { Project } from "./components/Project"

You can many exports in a module but you can only have one default export.

export default function Project() {

  return <h3>My Projects</h3>

}

If you declare a function as a default export in a module then you don’t need to destructure when you import it

import Project from "./components/Project"

The project name is hardcoded, but prefer to specify a project name or title when you render the prompt.

In index.js

ReactDOM.render(<Project title='My awesome project' foo='bar' />, document.getElementById("root"))

In project.js, modify Project functional component

export function Project(props) {

  return <h3>{props.title}</h3>

}

ReactDOM.render can only render one element, so if you want to render multiple instances of a component, you need to wrap it in one parent element (i.e. a div)

ReactDOM.render(

  <div>

    <Project title='My awesome project' />

    <Project title='Project1' />

    <Project title='Project2' />

    <Project title='Project3' />

  </div>,

  document.getElementById("root")

)

Any component can only return a single element. So you have to render it inside a parent element (i.e. a div)

**Composing our app out of other components (encapsulating components inside another component)**

In index.js, upate the ReactDOM statement

ReactDOM.render(<App />, document.getElementById("root"))

In app.js

import { Project } from "./components/Project"

function App() {

  return (

    <div>

      <Project title='My awesome project' />

      <Project title='Project1' />

      <Project title='Project2' />

      <Project title='Project3' />

    </div>

  )

}

Index.js renders the App component, which in turns render the Project component