Announcements:

Welcome Wednesday at 2:30 mountain

Outcomes Thursday at 10am mtn

Week1 To Do List:

* Check out next.js
* Check out Rust

Colorful Comments:

* Red (!)
* Blue (?)
* Green (\*)
* Yellow (^)
* Pink (&)
* Purple (~)
* Mustard (TODO)

At 11 am we will have a live review session every Wednesday and Friday

Day 1:

React – a client side javascript library

Vanilla Javascript requires every action to be written out which can lead to lengthy code

React Framework – React is a component based JS framework. With react you develop your applications by creating re-usable components. Essentially react works at a higher level and do not have to dive into the nitty gritty parts of coding elements.

React JSX – JSX is a syntax extension of JavaScript for combining JS and html into a single document. JSX is more strict than html.

* There must always be a closing tag even on elements that don’t have a separate closing tag <img contents />
* Entire statement must be enclosed in a single parent tag
  + Can also use <> contents </>
* Reserved words cannot be used (ie. Class) so use className instead.

SPA’s – (Single Page Applications) are a webpage design where the contents of the page are being replaced rather than navigating to a new page. Essentially every change on the page is just replacing html instead of loading an entirely new page. We are manipulating the DOM

Some JS notes:

* New class syntax
* Spread and Rest operator
* Objects and Arrays are reference type

Day 2:

Components – React uses components to build pages. Each component will have its own html, css, and logic. The CSS is more of an afterthought.

React uses a declarative approach – you are defining the desired end/target state(s). React will figure out the DOM updates behind the scenes

React code is just Javascript with some special syntax for React

Index.js will hold the code which is executed initially upon page load

JSX syntax is simplified html – gets converted behind the scenes to code that works for the browser (look at sources in dev tools)

The App component will stay in the src directory, but all other components will be put in a components folder. With react we are basically making a component tree, components will be nested within each other and all are eventually nested in the App.js component.

Best practice to have app files be in upper case form where the first letter of every word is capitalized.

Steps:

1. Create a Component
2. Export Component
3. Import Component and use it

Props – allow for transfer of data between files

Card Components can be built to add repeated styling to multiple other components

JSX Notes:

* React used to need to be imported to every file that used JSX, no longer an issue with the dependencies included
* JSX code is just syntactic sugar, under the hood it gets transformed.

Day 3:

Goal – be able to define state easily

Event listeners can be added inline to JSX as attributes – They expect to be set to the value of a function, either inline or declared previously.

React will only run through code once on its own and will not register new data afterword’s.

State is a way for react to know that something has changed and component should be re-evaluated.

useState():

* Import useState via “import React, { useState } from ‘react’;”
* useState() must be called at the beginning of the component function with a default state value which would be the props value your are looking at.
* When called useState(props.value) returns an array of the value and a function which can be de-structured inline.
* The variable can be named anything and used inside of JSX code.
* The function should be setVariable and will be called inside the eventHandler function with the new value passed in.

State operates on a per component basis even if we have multiple of the same component

When adding event listeners with React the event is automatically captured and can be used in the callback function.

There are a few

Multiple states can be called for one component to track different things without interference

1. You can call useState for each value you want to watch
2. You can call useState just once and input an object with all values you want to watch

There are a couple of methods for keeping track of multiple states

1. The simplest is to just call useState() on each value you want to track.
   1. This method does not run into any issues with dependencies on previous state
2. Another way is to call useState() once, and input an object which contains all of the values you are tracking. Once declared there are two methods of tracking state in you handlers
   1. First, declare the set function and input an object.
      1. In the object use the spread operator to copy the original object
      2. Re-declare any values that need to be changed
   2. Second, declare the set function and input a new function
      1. Function parameter is prevState
      2. Function returns and object similar to the other option
         1. Spread out prevState, then re-declare any values
      3. This method should be used if your values depend on the previous state

Two way binding

* When gathering input data we can store that data to an object that we create in a submit handler.
* To reset the input values:
  + First, add a value attribute to each input element in the JSX
  + Second, at the end of the submitHandler, call all the set functions with an empty string, this will return all values to be blank.

Transferring Data to parent nodes

* Create a new Handler function inside the parent function – this will receive the desired data
  + Once received it can do something with that data, modify, sent to another component, and so on
* That function must be sent down to the child using props (good practice to use onPropName
  + Create a prop attribute that holds the Handler function you just created
* In the Child function we will receive a parameter of props ( the main function )
  + In the Childs Handler function that has the data, call the props function and input the data that needs to be sent.

Lifting State Up

* The need to move data to the first parent component that has access to both involved components
  + We lifted the state to the app
  + Now we can pass that state down the other side to create new cards

Controlled Components – a component that is fully controlled by its parent

Stateless, presentational, Dumb – components that are static

Stateful, smart – dynamic components

Day 4:

Adding a list

* At the component that is generating the JSX
  + Grab the props and use .map
  + For each item, plug it into the desired JSX to be returned
  + This will get the initial list to display, but will not update with changes
* At the parent of the JSX
  + Initialize state
  + Call useState with the initial data
  + In the newDataHandler function add the new data to the initial data
    - Make sure to use function format due to dependency on initial data
* Be sure to add the “key” prop to the .map JSX
  + Key should be set equal to a unique identifier
  + This allows React to render the list items in the correct locations with minimal action.

Methods can be used to filter or modify data as needed as long as all needed data is present in the given component – filtering by year

Ternary Expressions can be used in line to add conditional statements (ie. Checking length of array? Does data exist? )

JSX can be stored In variables outside of return. Using this with a conditional outside of return you can set a variable to whatever JSX will be the result. This leads to a much cleaner JSX snippet in the actual return and will usually be more readable.

Conditional Part can be moved to its own component. If this is done then they entire JSX return can be rewritten and helps simplify parent component

Making a form display Conditionally

* Initialize state on the parent component of the form and create a true/false state.
  + In this case we have isEditing and initially it is equal to false.
* Create two handlers
  + Start editing – sets value to true
  + Stop editing – sets value to false
* In the JSX put conditionals on the elements
  + Wrap in dynamic tags, and use condition && JSX text
* On the form JSX add a prop to pass the stopEditing function to the form
  + On the form component, add a button and onClick to call passed function

Making a Chart

* Created a component folder with two components, Chart and ChartBars
  + Chart is the container for the bars, we use props to receive the dataPoints from another new ExpensesChart component in the expenses folder.
* Expenses passes the filtered data to the ExpensesChart component
* ExpenesesChart turns that data into dataPoints and sends them to Chart
* Chart handles the data and creates ChartBar elements by mapping it.

**React Styling**

Dynamic Inline Styling:

* Initialize a state for valid vs invalid input
* In a JSX element the style attribute can be used. Set it equal to {} and it takes an object so final line will look like style={{color: ‘red’}}
  + A ternary can be used to define two different styles states
  + Style={{color: !isValid ? ‘red’ : ‘black’}}
* In the submit handler have the state trigger to false if conditions are not met – this will trigger that styling state
* In the input tracker, have a quick check if event.target.value has a length, if yes then set isValid back to true to reset syling
* No a great method, should avoid inline styling.

Dynamic Class:

* Create a class in CSS for the different style state (ie. Adding .invalid)
* In the JSX className us template literals inside a dynamic bracket (back tiks) to insert the constant class, and a conditional class based of the previously used isValid. If not, add class, else add empty string.

Css selectors are not scoped, can become issue on larger projects

Solutions

Styled

Package called “Styled Components”

This will be used on UI components that are focused only on styling things

* Helps to build components with styles only attached to said component
* Install Package
  + Npm install –save styled-components
* Essentially declare the component = styled.(component type)` insert css and adjust to package requirements `
* Can also be used on partial components, conditionals can still be added as a className prop
* Another way is to apply props to the JSX and use conditionals in the declared styling

Media query can also be used to adjust styling for screen size

@media (style-condition: true) {

Styles

}

Another option:

CSS Modules

* To utilize, just import your css files a little differently
  + Import styles (or “classes) from ‘./Button.css’
* Replace classNames with {styles.class}
  + Everything else is done like we did Dynamic Classes Previously

EOD NOTES:

Public folder:

Should replace README file that is created with react app with a README specific to the project you are working on

Replacing favicon.ico with a different image will change displayed image, just make sure image name is the same

Manifest place to declare information for wherever the code is being deployed – not really applicable to web applications

Robot.txt - declares rules for bots accessing your application

Src folder:

App.css can be deleted, it is just initial styling

App.js will be our main component

App.test.js – testing file for your app

Index.css – can realistically delete – basically just some reset css

Index.js – connecting to the index.html file for actually displaying components

reportWebVitals.js – performance tracking

setupTests.js – test setup file, bit difference than vanilla js

Re-rendering will target anything that contains the state that changed. One state can be used by multiple components. Only the components utilizing a state will be re-rendered.

Question: When you lift state, how do you access those values repeatedly in the parent component

State should be declared in the common parent of components

Pass the props down and set them in child components