React Training Materials

Monday, December 24, 2018

3:57 PM

The OneNote and Powerpoints are available at:

<https://hub.perficient.com/bu/DDC/COE/Shared%20Documents/Front%20End/Workshop%20Items/React/Week%201>

The repo is available here:

<https://github.com/jdbarras-prft/react-level-1>

Prerequisites - Send these instructions out before workshop

Thursday, January 24, 2019

10:23 PM

Get VS Code

Git Bash

Install Node

Day 1-2

Thursday, January 24, 2019

10:36 PM

Introductory Presentation

Wednesday, January 30, 2019

1:44 PM

At this point the slide show "1 - React Level 1 - Introduction.pptx" should be presented.

Code Along Instructions

Thursday, January 24, 2019

10:21 PM

* I have coded all of the examples ahead of time. I am going to write them again. This may seem silly, but I find that staring at already written code can be more difficult to conceptualize and follow the flow. I much prefer to write this together. The only reason I did it ahead was because with the pressure on I would have spent twice as long because of screw ups.
* Make a folder for this course 'react-ws'
* Make separate folders for each concept as we go through them. My examples will follow a similar structure. You will likely want to refer back to syntax. Keep it neat and orderly.
* Follow along with each coding example and the code-along. This is not a brown-bag. If you don't write the code it only feels like you're learning.

1 - React.createElement and ReactDOM

Thursday, January 24, 2019

10:36 PM

<Refer to examples/01-createElement/index.html>

React.createElement creates an object representation of a DOM node.

Lines 15 - 19 creates an object representation that can be seen by logging it to console.

Note the props and type on this object. This is a React element. It's what lives in the virtual DOM, it's light-weight, and it's cheap to create, destroy and manipulate.

Now we need a way to take a React element (or a whole tree of React elements) and render it to the DOM as actual DOM elements.

Up to this point we have used the React library. This library is responsible for building and manipulating components and elements. Next we will need ReactDOM, which is specific to making React components work with the browser DOM. The reason for this separation is to allow the core React library to be used in other environments, such as native mobile applications which use the React Native library. Think of ReactDOM and React Native as adapters that let you plug React into the environment you are developing for. In our case we use ReactDOM because we're making a web app.

The only method you are likely to use out of ReactDOM is ReactDOM.render(). This guy has a big job though. It is responsible for reconciling the virtual DOM and the real DOM and replacing only that which has changed.

2 - Basic Composition, Elements, and Components

Thursday, January 24, 2019

10:36 PM

<Refer to examples/02-basic-composition/index.html>

To be an effective React developer you will have to make use of composition. It is the overarching design principle of all React applications. The basic building block of React - the component - implies by its very name that it is meant to be part of a larger whole.

In this code example we are making use of a simple wrapper element to contain the other three. This is then being rendered to the DOM.

<Refer to examples/03-components/index.html>

A React Component is a function or class that optionally takes input (props) and returns a React Element. We can convert the three elements from the previous example to components.

3 - JSX and Props

Sunday, February 3, 2019

12:54 PM

These examples have served only to show what's happening behind the scenes. React wouldn't be a great choice if this is the way elements were written. So far, we haven't seen any of the benefits to React and the way it updates the DOM, but even with those benefits it would be difficult to use without JSX.

HTML is already a concise way to create DOM nodes for a webpage. JavaScript is a great way to add interactivity. To harness both, and get the benefits that come with React, we use JSX, which is a marriage of HTML and JS.

<Refer to examples/04-JSX-and-Props/jsx.html>

We add the Babel script and change our script type to 'text/babel' which makes Babel preprocess the JSX back into JavaScript. JSX will be turned into React.createElement calls just like we saw before.

Components referenced in JSX can receive parameters of any type called "props". They are passed in from the caller of the component using a syntax kind of like html attributes. This is how data flows in a React application. It is a unidirectional flow from parent to child. We will learn later about how to go the other direction, but for now understand that a child component can receive any number of named parameters using the JSX attribute-like syntax. These can be accessed inside of the component by using the props object.

Exercises (have students work these in real time. Answer questions. Go over solutions):

<https://codepen.io/collection/nLLxry/>

Day 3-4

Sunday, February 3, 2019

7:38 PM

4-Introduction to State

Sunday, February 3, 2019

7:39 PM

When creating a component there are two main considerations: The UI, and any state that the component will need to manage. As an example, we can take a look at the newsfeed on LinkedIn. The list of news articles could be state that is managed by the newsfeed component. The rest of the application does not need to know about this state, so it could be managed locally within the component and treated like a module.

The application as a whole can continue to grow in complexity, but with a component based approach, we only need to focus on the piece we are working on.

<Refer to examples/05-State/state1.html>

Now that we are using babel, we can use JSX and ES6. We will create a boilerplate App function using an ES6 arrow function.

We create an array of names called "friends". We will map over this array to make an unordered list of friends. When rendering a series of elements in a loop we need to provide a key attribute. React uses this key to know which element changed. This allows us to optimize the render to only the item that changed rather than re-rendering the entire list.

We are using props to render the list, but if we want to change the contents of the list based on user interaction or some other event, we need a way to change the array and notify React that it has changed, and needs to re-render. This requires state and the setState function.

Until recently, the only way to have local state was by using a class component (more on this later). So far we have only been using functional components.

<Refer to examples/05-State/state2.html>

Functional components can take an input (props) and render some UI. Class components do the same but can have state and have access to "lifecycle hooks", which will be covered later.

It is important to understand the difference between functional and class components. Only use classes when you need the extra features (state or lifecycle hooks). React is more optimized for functional components and they are easier to reason about.

Think of props the same way you would think about arguments to a function. State, however, is managed by the component, must be changed using setState(), and triggers a re-render.

<Continue through examples/05-State/state3.html through examples/05-State/state4.html>

Describe functionality of assignment <Solution in examples/05-State/state5.html>.

* Add Clear All button that removes all items from either list
* Add Completed section and button to move items in and out of this section

5-Lifecycle-Hooks

Sunday, July 7, 2019

9:35 PM

<Refer to 2 - React Level 1 - Lifecycle Hooks.pptx>

* + constructor(props) - Sets up component state… First thing to fire… This is also where you bind methods.
  + render - Describes the UI. Examines props and state and returns UI.
  + componentDidMount - Component has mounted to the DOM. This is when AJAX requests are usually made
  + getDerivedStateFromProps
  + shouldComponentUpdate(nextProps, nextState)
  + componentDidUpdate - Fires after every re-render (after every state change or prop change). Gets previous props and state as parameters. This can be used to see what updated and take some action
  + componentWillUnmount - Component will be removed from the DOM, use to remove listeners

6-Containers vs Presentational Components

Sunday, July 7, 2019

4:25 PM

The "container / presentational component" paradigm is a pattern that is common but not the only way.

Containers:

* Often (but not always) classes.
* Handles common state between child components
* Is the parent of similarly grouped child components (could be other containers or presentational components)
* Manages the state for itself and its children
* Connected to Redux if necessary

Presentational:

* Usually (but not always) functional components
* Not usually responsible for state (although with hooks, this is changing to some degree)
* Not usually connected to Redux
* Usually a "pure" function, which means given an input (parameters) it returns the same output.

Folder Structures

Can be grouped by type (container/presentational):

src/

containers/

WidgetA/

index.js

WidgetB/

index.js

presentational/

WidgetA/

Subcomponent1.js

Subcomponent2.js

WidgetB/

Subcomponent1.js

Subcomponent2.js

Subcomponent3.js

Or by functional area:

src/

WidgetA/

index.js

Subcomponent1.js

Subcomponent2.js

WidgetB/

index.js

Subcomponent1.js

Subcomponent2.js

Subcomponent3.js

Things to cover at the end

Sunday, July 7, 2019

4:20 PM

16.8 features… hooks, suspense api

classes are not necessary in most cases because:

useState hook

useEffect hook

If your project uses Redux then most state should be there instead.

Most lifecycle hooks can be avoided by using good design patterns.

Revamp

Monday, January 18, 2021

10:11 AM

* + Convert class based to functional comp (include component did mount)
  + Move CRA above state
  + Insert project architecture module right after CRA