**CS 628 Modern Full-Stack Development**

**HOS02: Server-Side Action: Node and NPM**

January 10, 2023 - Developed by Shingo Kise

School of Technology & Computing (STC)

City University of Seattle (CityU)

A blue and white flag

Description automatically generated with medium confidenceIcon

Description automatically generatedIcon

Description automatically generated

**Before You Start**

* This tutorial targets Windows users and MacOS users.
* There might be subtle discrepancies along the steps. Please use your best judgment while going through this cookbook-style tutorial to complete each step.
* For your working directory, use your course number. This tutorial may use a different course number as an example.
* The directory path shown in the screenshots may be different from yours.
* If you are not sure what to do or confused with any steps:
  + Consult the resources listed below.
  + If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Understand the fundamental structures of React
* Understand how to create components
* Understand properties (“props”)
* Understand how state works to update DOM
* Understand Styling

**Resources**

* Zammetti, F. (2022). *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, Python, Django, and Docker* (2nd ed.). Apress.
  + Ch. 3: 3. Client-Side Adventures: React
* React - <https://reactjs.org/>
* Visual Studio Code - <https://code.visualstudio.com/>

**Introduction**

In this course, as we learn “full-stack” development, we’ll be learning about coding clients as well as the server code they make use of to form a cohesive, whole application. In the next few chapters, we will focus on the development of the frontend side. In this chapter, we begin writing frontend side code and learn about fundamentals of React.

**React**

There are many framework/library/toolkit for building web-based client applications.

But, over the last couple of years, a few popular options have floated to the top of the pile and React is one of them.

Chart

Description automatically generated

2022 stackoverflow Developer Survey

**A Brief History of React**

React (sometimes referred to as React.js or ReactJS) is a product of Facebook. React is a library for building web-based user interfaces.

It all started back in around 2010 when Facebook developers began to run into a lot of issues with code maintenance. Possibly the big turning point in the history of React was in 2012 when Facebook started running into a lot of problems managing the ads displayed on the site. So, the engineers at Facebook started looking for a solution.

Today, many websites in a variety of industries are built with React. The most famous companies that use the framework include Meta, Netflix, Uber, Airbnb, and The New York Times.

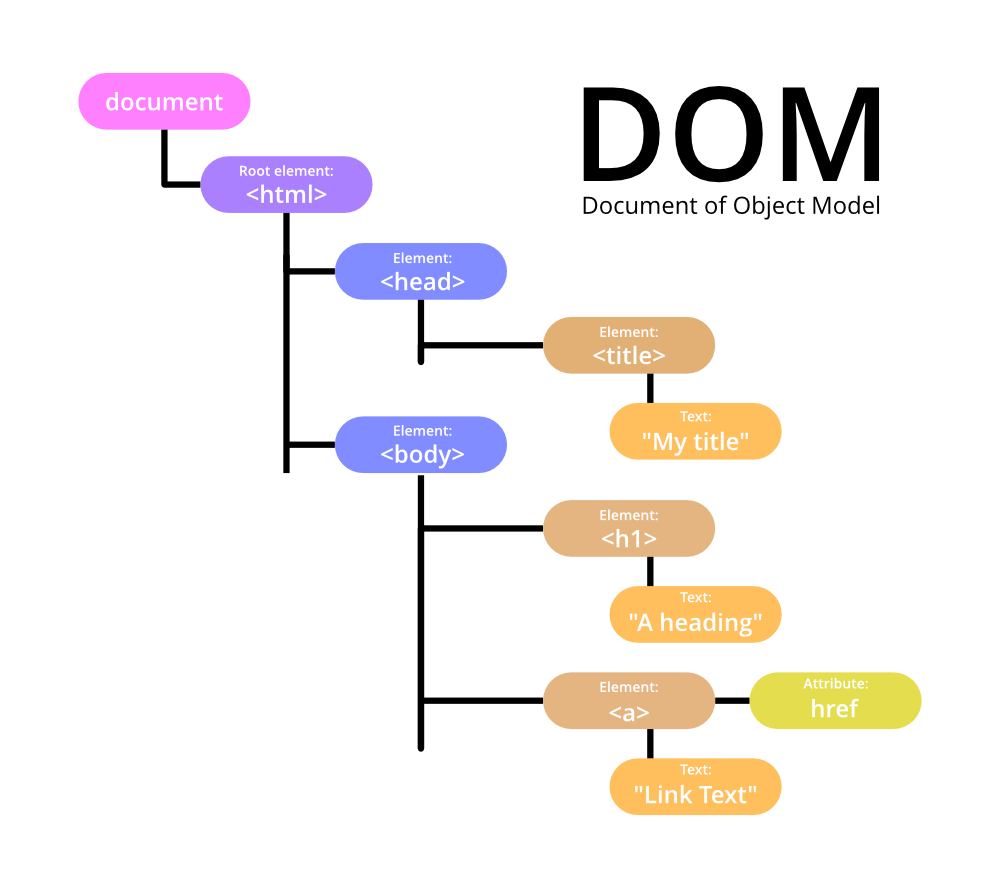
**What IS React?!**

At a very high level, the point of React is to make it easy to reason about the structure of your interface at any given moment in time. This is accomplished by way of *components*, which you can think of as self-contained pieces of the interface. With many of those components, a user interface can be built.

Key elements in React are components, props, state, style, and virtual-DOM.

**What is virtual-DOM?**

We will not directly use virtual-DOM but React will be using it extensively. DOM is the Document Object Model, which has the tree-like structure that the browser builds as it parses the HTML file. All the elements, denoted by tags, in the HTML, become nodes in this tree.

This DOM has a direct tie to what you see on the screen, and it offers an API to manipulate it with.

Typically, when you do something that makes a change to the page, whether it be as a result of user action or programmatically, the browser has to perform some relatively intensive and expensive work, primarily to render the entire screen to “repaint” the screen. All of this takes computing time and sometimes it impacts the user experience.

React uses the concept of a virtual DOM. This is, in essence, a secondary DOM that sits conceptually on top of the real DOM in memory. Rather than manipulate the real DOM directly, React updates the virtual DOM and then will intelligently figure out the changes and the least amount of real DOM work that can be done to accomplish the update. Most importantly, this allows React to batch up real DOM changes and apply them all in one go, which is much more efficient than doing each one individually. The result is better performance and user experience (no refreshing the page often) than can typically be achieved with direct DOM manipulation.

1. **The Real Star of the Show: Components**

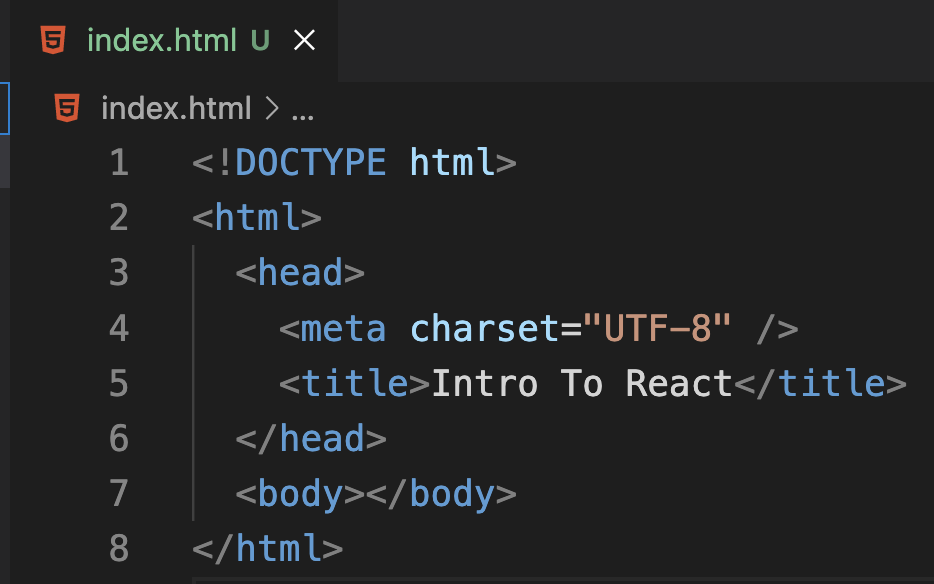
Let’s start things off by creating a HTML document.

Open VS code and then open the **CS628-HOS3/FirstnameLastname** folder

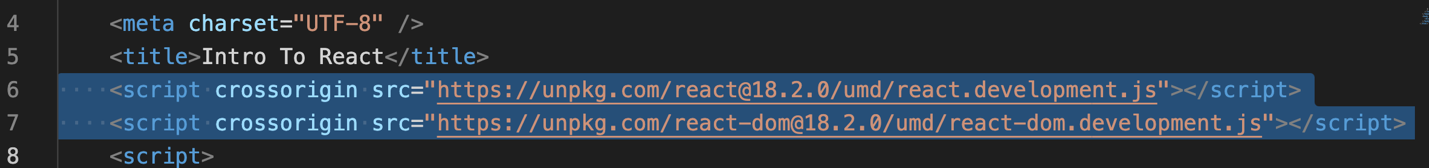
A screenshot of a cell phone

Description automatically generated

Create an HTML file named index.html and type the following code



Now, to it, let’s add two lines into the <head> of the document to bring React into the fold:

Here are the links:

<https://unpkg.com/react@16/umd/react.development.js>

<https://unpkg.com/react-dom@16/umd/react-dom.development.js>

These code uses a Content Delivery Network (CDN), rather than using ‘npm install’ to download the main React code (react.development.js) and react-dom package.

* React can talk to different renderers, which are the bits of code that produce the visual output.
* ReactDOM provides DOM specific methods that can be used at the top level of a web app to enable an efficient way of managing DOM elements of the web page.

Let’s add more scripts to it! (Line 14 – 55)

Modify your index.html file to match the screenshot below.



This function uses one of the most important things React offers, the React.createElement() method.

React.createElement(type, {props}, ...children);

This method will construct a new React element, which is the smallest building block of the visual interface of a React app. These building blocks called components allows us to build complex user interface (UI).

In this start() function, React.createElement() is used to build up a tree of elements, all of which are children of the top-level div element, which the variable topMostComponent holds a reference to. Each call to React.createElement() after that first one is creating a child of the element above it in the tree. Then, when that tree is all built up, this code executes (Line 44-45):

Text

Description automatically generated

The code is creating a root component and inserting it into the mainContainer <div> in the line 53. This is the uber-component that is the parent to all other React components on the page. Then, the topMostComponent, which is a tree of components, is rendered into that root component.

The render() method takes that built-up virtual DOM tree and renders it to the real DOM, and after the difference is applied, the final output will be shown on the browser.

Now Let’s run the code.

To run the code, since we only have a HTML file, let’s install Live-Server extension on VS code to run the code.

> Click the extension and search Live-Server and install

A screenshot of a computer

Description automatically generated with medium confidence

> If you successfully install it, you should see the Go Live option somewhere on your IDE.

(At the bottom right of IDE for my case)

Graphical user interface, application

Description automatically generated

If you click *Go Live*, a web page should open, and you can see the output like this.

Graphical user interface, text, application, email

Description automatically generated

***Please take a screenshot of your output.***

What we have so far is not what React applications usually written as. This is just an example in vanilla JavaScript. Instead, React uses something called JSX (JavaScript XML), and that makes it considerably easier to write React apps with. What we have so far is helpful to understand how React is doing things in the end, as when writing a React app with JSX, it will produce code similar to this.

Let’s modify the entire <script> section.

Modify your index.html file to match the screenshot below.

Text

Description automatically generated

In Line 11, React component is defined, which is why it extends the React.Component class. In React components, the most important is a render( ) method, which is the only thing your component is required. Itis responsible for returning several things, such as another React component and a React element, which are almost always what you would return.

The this.props (Line 15) member will be present on any component and React populates it when you use the component using whatever you pass as the second argument to React.createElement(), and the data in props is available as appropriate inside the component.

In Line 36 – 41, data is passed as attributes (title, href, and description) as part of the elements returned.

This time, we call React.createElement() but now we pass it the name of the React component as the first argument. React instantiates that component and passes the props to it, and whatever that component’s render() method returns effectively is inserted at that point in the tree.

***Open a new terminal, enter following command to commit the section 1 work to git.***

$ git add --all

$ git commit -m "Section 1"

**Components Need Info: Props**

Props, which is short for properties, is how information is passed between components.

For the simple HTML elements created earlier, the attribute ‘href’ for a link or the text inside an <h2> element are examples of props. For custom React components, whatever props a component needs can be defined. For the Bookmark component, that’s ‘title’, ‘href’, and ‘description’.

What’s important to realize about props is three things.

* + - 1. They are always passed down from a parent component to a child.
         * In any case, the source of the information is always the parent component.
      2. Props are only given to the child component when it’s being created.
      3. Props are immutable.
         * The way it works is that any time a change must occur to a component, React will re-render part of the DOM tree. That means that the component will need to be passed its props from the parent again (props will not “change”, but new props will get passed in when re-building the component).

**2. Components (Sometimes) Need Memory: State**

There are two types of data that serve to control components in some way: props, and state.

Changes to state do not cause React to destroy and recreate a component directly. It will change just the tiniest portion of the virtual DOM tree that the change demands, and then the minimum real DOM changes will result.

Let’s make some changes to our Bookmark component.

Modify your <script> code in the index.html file to match the screenshot below.

**Text

Description automatically generated**

In Line 17-21, you can see the constructor was added. The constructor is optional, but there are two reasons to have a constructor.

* The first is so that when you try this code out, you will see that when state changes, the constructor does not fire after the two initial times, proving that state changes don’t result in component recreation. Note that in a constructor for a React.Component instance, you must call the superclass’s constructor and pass it the props that will be passed into the constructor.
* The second reason is the line where title is set as a member variable, and its initial value is taken from the props that are passed in. This variable becomes the state of this Bookmark component.

In Line 26, note the change there: rather than getting the value from ‘this.props’ like before, it now comes from ‘this.title’.

A new child element has been added at the end, this one a <button> HTML element. The second argument to React.createElement(), which you’ll recall is the props to pass to the component, can includes functions, which is an onClick event handler function. React will create a <button> HTML element, and attach that function. The button changes the title property of the class when it is clicked.

By making changes in variables, React would not necessary reflect those changes on the screen. Once React initially renders all components, React will not recognize changes in those components. That’s when state comes in handy. State informs React that changes had occur in a component and React needs to scan the component again and reflect the changes. That’s precisely what the setState() method is for. This method is provided by the base React.Component class that our Bookmark custom component class extends. It informs React that this component, and its children, may need to be re-rendered (React will make the final determination).

The argument passed to setState() is one of two things: either a function or an object. If it’s a function, then it’s what is called an *updater function* . This function receives two arguments: the current state of the object and its props. This function must then return an object that will be the new state of the component. By calling the function with a new value as an argument, the function does not only return the new value, but also let React to recognize the change and re-render the component with the updated value.

Alternatively, in the screenshot, an empty object was passed in setState(). What this does causes React to perform a shallow merge of the object with the component’s current state. In this case, since I’ve already altered the state variable, that means that the resulting object has the new value, so what’s returned is a valid new state object.

It is asynchronous in other words, and as a result, you can also pass a second argument to setState(), a callback function. This function will be called after the update has occurred.

Let’s run the code to see what we have.

> If you have the same webpage open, you should see that there are buttons now.

If you have closed the webpage, you can click the *Go Live* again

If you click those buttons, you will see that “-CHANGED” is added.

That is, because React re-renders that component after recognizing the change via state.

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application

Description automatically generated

***Please take a screenshot of your output.***

***Open a new terminal, enter following command to commit the section 2 work to git.***

$ git add --all

$ git commit -m "Section 2"

**3. Making Them Look Good: Style**

When building an app with React, we’re still using HTML, JavaScript, and CSS.

Components will always render down into some combination of those.

For example, if we want to make the color of our Bookmark titles red, we could add this to the page:

In your index.html file, add the line 14 – 17 to match the screenshot below.

A picture containing text

Description automatically generated

Check your output to see that h2 writings have changed their colors.

Another alternative is to explicitly name a CSS class to use in the component’s code. So, let’s alter that style definition a little bit:

Modify your index.html file to match the screenshot below.

Graphical user interface, text, chat or text message

Description automatically generated

Now, in our Bookmark code, the call to React.createElement() that creates the <h2> for the title specifically, let’s use that style:

Graphical user interface, text, application, chat or text message

Description automatically generated

Since the attribute name ‘class’ is a reserved word in JavaScript, React makes us use ‘className’ instead. But that will result in the style being applied all the same.

Check your output to see that the output is the same.

Now, the Bookmark component’s style is defined within the Bookmark class, achieving encapsulation, but then within the class the style information is abstracted from the code that produces the layout in the render() method. This is arguably a cleaner way to write component code.

Modify your index.html file <script> to match the screenshot below.

Text

Description automatically generated

Check your output to see that the output is the same.

Graphical user interface, text, application

Description automatically generated

***Please take a screenshot of your output.***

***Open a new terminal, enter following command to commit the section 3 work to git.***

$ git add --all

$ git commit -m "Section 3"

**Pushing your work to GitHub**

Run the following commands to push your work to the GitHub repository:

Open the terminal from the VSCode by hitting the control + ~ key and type the following command:

>>> git add .

>>> git commit -m “Submission for Module 1--yourname”

>>> git push