**CS 628 Modern Full-Stack Development**

**HOS04: 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 JSX
* Understand how Babel works

**Resources**

* Zammetti, F. (2022). *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, Python, Django, and Docker* (2nd ed.). Apress.
  + Ch. 4: A Few More Words: Advanced React
* React - <https://reactjs.org/>
* Babel - <https://babeljs.io/>

**Introduction**

In the last chapter, we began looking at React, the popular JavaScript application framework. There, you were introduced to the basic concepts that every React developer deals with. In this HIOS, we’ll look at a few more important concepts for writing React apps, JSX, which provides us a better way to write React apps.

**A Better Way to Write React Code: JSX**

JSX, which stands for JavaScript XML, is an extension to the JavaScript language that adds XML syntax to the language. It allows us to embed XML (or HTML) inside JavaScript. JSX allows us to define tree structures with attributes in more simple way compared to only pure JavaScript, which we had in the last HOS.

Let’s take a look at JSX code.

Graphical user interface, text

Description automatically generated

As you can see, this code has both JavaScript, CSS, and HTML characters.

MaterialButton component is assigned to button variable.

Text

Description automatically generated

This is what the code would look like without JSX.

As you can see, the code above is more simple, and JSX becomes powerful when the tree structure is more complex.

But, if we load that in a browser, the browser would not understand the code because it’s not valid JavaScript. How do we get valid JavaScript from JSX?

**Babel**

Babel is a JavaScript compiler, meaning it transforms and compiles at the same time.

JavaScript has been evolving quickly and new JavaScript syntax gets introduced in new versions, and sometimes, a particular browser or engine might never understand modern features you want to use. For example, if we run JavaScript code, it will work in the latest version of Chrome, Firefox, and Opera, but it might not work in Internet Explorer 11.

Solutions for that is an approach is often called a *polyfill*.

Polyfill can refactor every new feature in the language and write using the earlier language features, often with some compromises, but essentially functioning as the new features do.

Babel allows writing code using the new language features.

Once the code is run through Babel, Babel takes care of generating the appropriate polyfill code.

Let’s see how Babel works.

Open VS code and then open the **CS628-HOS4/FirstnameLastname** folder

A screenshot of a cell phone

Description automatically generated

> Create “test.js” file

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

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

***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 4--yourname”

>>> git push