CS628 Full-Stack Development – Web App

**HOS03A: React - User Interface**

Created by Veerendra Jagatha on July 6, 2023

Reviewed by Anahita Raeiszadeh July 11, 2023

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* Version numbers may not match the most current version at the time of writing. If given the option to choose between the stable release (long-term support) or the most recent, please select the **stable release** rather than the beta-testing version.
* There might be subtle discrepancies along with the steps. Please **use your best judgment** while going through this cookbook-style tutorial to complete each step.
* If you are not familiar with a terminal, command line, and bash scripts, check out this video: <https://youtu.be/Dp7uw9c6QH8>
* All the steps and concepts in this tutorial are from references, so if you encounter problems, please **try to read and compare the references to solve the problem**. If you still can't solve the problem, please contact your course TA.
* **Avoid copy-pasting code from the book or the GitHub repository**. Instead, type out the code yourself. Resort to copy-pasting only when you are stuck and find things not working as expected.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

#### **Readings and Examples:**

* Visit [CS 628 Repository for Examples](https://github.com/samchung0117/cs628-examples).
  + Select the related module.
  + Visit the README.md file.
  + Find examples for your practices.

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces
* Section 2: Module import/export for code organization and reusability
* Section 3: Understanding JSX syntax for creating React elements
* Section 4: Dynamic rendering with JSX expressions and JavaScript objects
* Section 5: Passing data and configuration through props in React
* Section 6: Implementing conditional rendering in React
* Section 7: Rendering data lists using JSX and mapping
* Section 8: Keeping Components Pure

**Section 1: Accessing GitHub Codespaces**

Refer the steps from [TA Center](https://cityuseattle.github.io/docs/git/github_codepsace/) to get started with this week’s module GitHub Codespace.

Once the codespace is ready, in the terminal type the commands below to start a new react project and start it.

**>>npx create-react-app hos03**

**>> npm start**

**Section 2: Module import/export for code organization and reusability**

Components in React are highly reusable, and as they become more complex or nested, it is beneficial to split them into separate files for better organization and reusability. This involves creating a new file for the component, exporting it using either default or named exports, and then importing it in the file where it will be used. This allows for modularity and reusability of components across different files.

**Default export example:**

Create a file with name Gallery.js under src folder and type the following code in it.

A screen shot of a computer code

Description automatically generated

Now update the App.js to match the following to import the Gallery.js,

A screenshot of a computer

Description automatically generated

Refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Named export example:**

Now update the Gallery.js to match the following,

A screenshot of a computer program

Description automatically generated

Update the code in App.js to match the following and import the named export Profile,

A screenshot of a computer code

Description automatically generated

Refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Section 3: Understanding JSX syntax for creating React elements**

JSX syntax is a fundamental aspect of creating React elements. JSX is a syntax extension for JavaScript that allows developers to write HTML-like markup within JavaScript code. It provides a more intuitive and concise way to define the structure and appearance of React components.

JSX resembles HTML syntax, but it is not actually HTML. It is a syntactic sugar that gets transformed into regular JavaScript function calls during the build process. JSX elements look like HTML tags, but they represent React components or HTML elements.

Update your App.js code to match the following,

A screenshot of a computer program

Description automatically generated

You will receive runtime error as shown below,

A black and red background with white text

Description automatically generated

This is because, Adjacent JSX elements must be wrapped in an enclosing tag.

Now update the code in App.js to match the following,

A screenshot of a computer program

Description automatically generated

Refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Section 4: Dynamic rendering with JSX expressions and JavaScript objects**

JSX also allows embedding JavaScript expressions within **curly braces { }.** This enables dynamic content and computation within the JSX elements. You can use JavaScript variables, functions, or any valid JavaScript expression inside the curly braces to dynamically generate content or compute values.

Under src folder create a file AvatarCSS.css and type the code below in it,

A computer screen shot of a computer code

Description automatically generated

Now, update your App.js content to match the following,

A screenshot of a computer program

Description automatically generated

In the given example, notice that the curly braces { } are used in JSX to embed JavaScript expressions or variables.

Now, refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Section 5: Passing data and configuration through props in React**

Props allow communication between parent and child components by passing data and configuration from the parent to the child. Props can be any JavaScript value and are similar to HTML attributes. The parent component passes props to its child component by including them in the JSX tags. The child component can then access and use the passed props using the destructuring syntax. Default values can be specified for props, providing fallback values if a prop is not explicitly provided. Props can also be passed as JSX children, allowing for nested components. Props are immutable, meaning they cannot be changed within a component, and when updates are needed, the parent component must provide new props.

Create a file with name utils.js under the src folder with the following content,

A computer screen shot of a computer code

Description automatically generated

Now, update your App.js to match the following,

A screen shot of a computer code

Description automatically generated

In the given code, props are used to customize and pass data to the Avatar component and to render nested content in the Card component. In the Avatar component, the person and size props are destructured from the passed props object. The person prop is used to determine the image URL using the getImageUrl function, and the size prop is used to set the width and height of the image. In the Card component, the children prop is used to render the nested content passed to the Card component, which in this case is the Avatar component with specific props. The App component then uses the Card component to wrap and display the Avatar component with the specified props.

Refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Section 6: Implementing conditional rendering in React**

Implementing conditional rendering in React allows developers to selectively render content based on specific conditions or states. By using conditional statements or logical operators within the component's JSX, different parts of the UI can be displayed or hidden dynamically. This enables the creation of dynamic and responsive user interfaces. Conditional rendering can be achieved by using if-else statements, ternary operators, or logical operators like && and ||. The following example used && operator to conditionally render the content.

Update your App.js to match the following code and observe that the checkmark is not rendered for some components,

A screenshot of a computer program

Description automatically generated

Refresh the browser window running the development server and capture the screenshot and save it in the module folder.

**Section 7: Rendering data lists using JSX and mapping**

Rendering lists in React involves mapping over an array of data and generating a list of components dynamically. In the below example, the App component renders a list of scientists by mapping over the people array. For each person in the array, a list item (<li>) is created with a unique key assigned using the person.id. Within each list item, an image element (<img>) is rendered with the image URL obtained from the getImageUrl function and the person's name as the alt text. Additionally, a paragraph (<p>) is included to display the person's name, profession, and accomplishment. The list items are stored in the listItems variable using the map method, and then the listItems variable is rendered within an unordered list (<ul>) element inside the App component. This allows the list of scientists to be dynamically rendered based on the data in the people array.

Add a file with name data.js under src folder with the following content,

A screenshot of a computer program

Description automatically generated

Update you App.js to match the following,

A screenshot of a computer program

Description automatically generated

Refresh the browser window running the dev server to observe how the list is rendered. Capture the screenshot and store it in the module folder.

**Section 8: Keeping Components Pure**

Components in React must be pure, meaning they should not modify pre-existing objects or variables, and always produce the same output for the same inputs. They should not depend on the rendering order of other components and should avoid mutating props, state, and context. Striving to express logic within the JSX and using event handlers or useEffect for updates helps maintain purity. By adhering to these principles, React's paradigm can be fully harnessed.

Consider the following example, which is breaking the rule,

Update the App.js to match the following and observe the output which is not as expected,

A screenshot of a computer program

Description automatically generated

This component uses a variable called "guest" from outside its scope. As a result, calling this component multiple times can lead to different JSX output. Additionally, other components that also use the "guest" variable may generate different JSX depending on when they are rendered. This lack of predictability should be avoided.

To avoid this, we can pass guest as a prop. Update your App.js to match the following and observe that this time the component is rendered properly. Capture the screenshot and save it in the module folder.

A screen shot of a computer program

Description automatically generated

**Pushing your work to GitHub**

* 1. Go to Source Control on your GitHub codespace and observe the pending changes.

Graphical user interface, text, application

Description automatically generated

* 1. Type the Message for your changes in the Message box on the top. For example,” **Submission for Module02 – Your Name**”
  2. Click on the dropdown beside the commit button and select **Commit & Push** to update the changes to your repository main branch.
  3. Select **Yes** when prompted.

Graphical user interface, application

Description automatically generated