CS628 Full-Stack Development – Web App

**HOS04A: React - Interaction and State**

Created by Veerendra Jagatha on July 17, 2023

Reviewed by Sam Chung on July 19, 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: Event Handling in React
* Section 3: Understanding State in React Components
* Section 4: Snapshot of State
* Section 5: Queuing and Managing State Updates
* Section 6: Updating Objects in State
* Section 7: Updating Arrays in State
* Section 8: Reacting to User Input with State
* Section 9: Choosing the Right State Structure
* Section 10: Sharing State Between Components
* Section 11: Pushing your work to GitHub

**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 hos04**

**>> cd hos04**

**>> npm start**

**Section 2: Event Handling in React**

In React, you can attach event handlers to your JSX. These event handlers are custom functions that are invoked when users interact with your application, such as clicking, hovering, or focusing on form inputs. You also have the flexibility to create your own components and assign event handler props with custom names tailored to your specific application requirements.

In this React example, the “<App>” component renders a “<Toolbar>” component, which contains two “<Button>” components. Each “<Button>” component receives an event handler prop (“onPlayMovie” and “onUploadImage”) that triggers an alert when clicked. This demonstrates how event handlers can be passed as props between components in React to enable custom behavior in response to user interactions.

Update your “App.js” to match the following and refresh the browser to notice the changes. Click on each button and take the screenshot and save it in the module folder with this section number.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 3: Understanding State in React Components**

In React, components use state to remember and update information based on user interactions. The “useState” Hook provides a way to declare and manage state variables within components, enabling easy updates and tracking of current state values.

**Example**: The code snippet showcases a React component called “Gallery” that utilizes state. It uses the “useState” Hook to manage two state variables: “index” and “showMore.” The index variable keeps track of the current sculpture being displayed, while “showMore” determines whether additional details about the sculpture should be shown or hidden. The component also includes event handlers (“handleNextClick” and “handleMoreClick”) that update the state based on user interactions.

Create a new file named “data.js” under “src” folder and update the code in it from this module examples (“Section03-SculpturesData.js”).

Now update your “App.js” file code with the code from the module examples file “Section03-UnderstandingState.js”.

Refresh the browser to observe the changes. Click on next and show details buttons and save a screenshot with the section number.

A statue of a hand holding a brain

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 4: Snapshot of State**

State as a snapshot refers to capturing and preserving the current data within a React component, enabling it to be referenced and utilized at a later point in time.

The provided example illustrates the concept of state as a snapshot in React. The Form component manages two state variables: “to” and “message.” The “to” state represents the recipient of the message, while the “message” state holds the content of the message itself. As the user selects a recipient or types a message, the state is updated accordingly using the “setTo” and “setMessage” functions. When the form is submitted, the current snapshot of the state is captured and used to display an alert message after a delay of 5 seconds, confirming the user's chosen recipient and message.

Update your “App.js” to match the following code,  
  
A screen shot of a computer program

Description automatically generated

Update the styling in your “index.css” under “src” folder from the module examples file “section4-style.css.”

Now refresh the browser and observe the changes. Provide your custom message and select “Bob” from the dropdown and save the screenshot to the module folder with section number.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 5: Queuing and Managing State Updates**

Update your “App.js” to match the following code. Refresh the browser and observe that clicking in “+3” increments only once.

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

When setting state in React, it triggers a re-rendering of the component but does not immediately change the value in the existing running code. Therefore, if you call “setScore(score + 1),” the score will still remain as 0 in the subsequent code right after the call.

To address this issue, you can resolve it by utilizing an updater function when setting the state. By updating your “App.js” file with the provided code snippet, you will notice that the increment buttons function correctly. Remember to save a screenshot including the section number for reference.

A screenshot of a computer program

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Section 6: Updating Objects in State**

When working with state in React, it is important to avoid directly modifying objects and arrays stored within the state. Instead, it is recommended to create new objects or make copies of existing ones when updating the state. The spread syntax (...) is commonly used to create copies of objects and arrays that need to be modified. By following this approach, you can ensure that state updates are properly handled and do not result in unintended side effects.

Update your “App.js” code using the code from the examples file “Section06-UpdatingObjects.js.”

Update the “index.css” under the “src” folder to match the styling code from module example file “Section06-style.css.”

Refresh the browser and you will observe the changes as shown below. Try to update the fields and observe that the state is getting updated without any issue. Capture the screenshot and save it under the module folder.

A person in a garment

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 7: Updating Arrays in State**

In React, arrays are mutable JavaScript objects that can be stored in state, but it is recommended to treat them as read-only. Like objects, when updating an array stored in state, it is necessary to create a new array or make a copy of the existing one, and then update the state to utilize the new array.

Update your “App.js” to match the code from the module example file “Section07-UpdatingArrays.js.” Refresh the browser and notice the changes. Click on the list items to notice the update in the array. Save the screenshot under the module folder.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 8: Reacting to User Input with State**

In React, you focus on describing the desired UI for different component states and triggering state changes in response to user input, rather than directly manipulating the UI through code, resembling the approach designers take when thinking about UI.

Observe the implementation of a React quiz form, where the utilization of the "status" state variable is determining the enabling or disabling of the submit button, as well as displaying the success message accordingly.

Update your “App.js” to match the code from the module example file “Section08-ReactingToUserInput.js”. Also update the styling in “index.css” under the “src” folder to match the module example file “Section08-style.css”.

Refresh the browser and notice the changes. Type some city name in the textbox and submit. Notice how the submit button gets enabled and disabled. Save the screenshot of the browser window in the module folder.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 9: Choosing the Right State Structure**

Properly structuring state is crucial for a well-maintained and bug-free component, as it ensures the elimination of redundant or duplicated information, preventing update-related bugs and facilitating easy modification and debugging.

Update the “App.js” code to match the code in module example file “Section09-Redundant.js”. Observe the changes by refreshing the browser and providing first and last name.

A screenshot of a computer

Description automatically generated

We can have the simplified version of the same code by removing fullName from state. Replace App.js code with module example file “Section9-Simplified.js” and observe that we have the same functionality after removing fullName from the state.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Section 10: Sharing State Between Components**

In certain cases, when you need two components to consistently update their state together, you can achieve this by removing the state from both components, placing it in their closest shared parent component, and passing it down as props. This approach, known as "**lifting state up**," is a frequently used practice in React development.

Update your App.js code to match the code from the module example file “Section10-SharingState.js.” Also, update the index.css to match the code from “Section10-style.css.”

Refresh the browser and click on the show button to see the changes. Rather than maintaining the active state within each individual panel, the state is managed by the parent component, which then provides the necessary props to its children components.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Section 11: 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 Module04 – 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