React Hooks

We will create a React application that uses React Hooks to manage state and handle activities.

Part one: Display a list of articles

1. Add this to App.js:

<h2>React Hooks App</h2>

<li>one</li>

<li>two</li>

<li>three</li>

</div>

);

}

1. Get these from react-hooks-starter: index.css, App.css
2. We’ve just added a hard coded list. We want to display the data programmatically. Open the articles.json file and take a look. It contains an array of articles.
3. First, add the following array into App.js after the imports and before the App function:

const initialArticles = [{title:'ONE'},{title:'TWO'},{title:'THREE'} ];

1. We want the App to hold its own state. We can do that with the help of the 'useState' hook. To use this hook we first need to import it:

import React, {useState} from 'react';

1. Insert the following as the first line of the App() function (before the return statement):

const [articles, setArticles] = useState(initialArticles);

1. Delete the existing <li> elements being returned from the App() function and replace them with code that uses the JavaScript array map function to output the contents of the array as <li> elements. Your App() function should now look like this:

export default function App() {

const [articles, setArticles] = useState(initialArticles); return (<div className={'app'}>

<h2>React Hooks App</h2>

<ul>

{articles.map( (article, index) => {

return <li key={index} >

{article.title}</li>

}

)}

</ul>

</div>

);

}

1. Next we will retrieve the articles from a “network.” The current version of the app displays the contents of the initialArticles array. What we really want though is to retrieve the data from a network. By default the development server that’s used to serve the app can also serve static files that are placed in the project's /public directory.
2. Make sure articles.json is in the public directory
3. We can use the 'useEffect' method to insert code that will retrieve data from the article file after the first render. To do that we first need to update the import statement at the top of the file to import the 'useEffect' method:

import React, {useState, useEffect } from 'react';

1. We will need to add a function to retrieve the articles.json file contents over the network. We can do this inside of the App() function by creating a 'const' type with the name 'getArticles' and assigning an anonymous function to it like this:

const getArticles = function(){};

1. Make sure to place the above code inside the App() function, after the 'useState' line and before the return statement.
2. Update the anonymous function you just created with a JavaScript fetch statement that retrieves the url 'articles.json':

const getArticles = function(){ fetch('articles.json')

.then(response => response.json())

.then(data => { setArticles(data)

}

);

};

1. The code uses the 'setArticles' function that was returned from our 'useState' call.
2. Insert a 'useEffect' method call inside the App() method so that it appears on the line before the return statement:

useEffect(() => {getArticles()}, [] );

1. Notice the use of the empty array for the second parameter of useEffect. This makes sure the code is only called once – after the first render. Save the App.js file.
2. We will add a 'Select Item from List.’ We will add the ability to click an item to select it and show its article text.
3. We'll need to save the index of the selected item in a state variable. To do that add another call to 'useState'. Name the variable 'selectedArticleId' and its modifier method 'setSelectedArticleId'. Add the statement as the first line of the App() method:

const [selectedArticleId, setSelectedArticleId] = useState(-1);

1. Add an onClick handler to the <li> elements that sets the selectedArticleId using the associated modifier method:

onClick={(event) => setSelectedArticleId(index)}

1. When an item is selected we want to apply the 'selected' CSS class to the <li>element which will render the element in bold text:

className={(selectedArticleId === index) ? 'selected' : ''}

1. The App() method's return statement should like this:

return (<div className={'app'}>

<h2>React Hooks App</h2>

<ul>

{articles.map( (article, index) => {

return <li key={index}

className={(selectedArticleId === index) ? 'selected' : ''} onClick={(event) => setSelectedArticleId(index)} >

{article.title}</li>

}

)}

</ul>

</div>

);

1. Save the App.js file. The app should refresh in the browser. Try clicking on an item to select it. The selected item should appear in bold lettering. Once an article is selected, we'd like to see its text displayed below the list. When the app comes up though, before an item is selected we'd like to see the word 'none' instead. Lets add a line of code into the App() function that implements that logic and assigns a value to a 'const' variable named 'selectedArticle'. The code can be added just before the return statement:

const selectedArticle = (articles[selectedArticleId]) ? articles[selectedArticleId].content : 'none';

1. Now we will add some HTML that displays the heading “Selected Article” and a paragraph right after that with the article text. These elements should be inserted near the end of the return statement after </ul> and before </div>:

<br /><span className={'bold'}>Selected Article:</span>

<p>{selectedArticle}</p><br />

1. Save the App.js file.
2. Next, implement “Add an Article” capability. We want to be able to add and delete articles from the list. We'll start by adding a section called 'Controls' after the selected article. The section will hold two input fields (for title and content) and two buttons – 'Add Article' and 'Delete Selected'. Before we add the controls section we need to create a state variable to hold the input field contents. The shape of the data will be an object with two properties - title and content. You can add this statement as the first line in the App() function:

const [formObject, setFormObject] = useState({ title: 'title1', content: 'content1' });

1. The input fields we display will get their values from formObject.title and formObject.content.
2. Add the following HTML after the paragraph with the selected article and before the closing </div> element:

<div className={'controls'}>

<span className={'bold'}>Controls:</span><br/>

<button onClick={null}>Add Article</button>&nbsp;

<button onClick={null}>Delete Selected</button>

<br />

<input type={'text'} name={'title'} placeholder={'title'} value={formObject.title}

/> <br />

<input type={'text'} name={'content'} placeholder={'content'} value={formObject.content}

/> <br />

</div>

1. Save the App.js file. The buttons don't do anything yet, and the input fields don't update when you type into them. We will change that.
2. Create an anonymous function and assign it to a 'const' type named 'changeHandler'. The function should get a new value for the field being changed and assign it to the property of the same name in the formObject. The function should accept an 'event' parameter. (Hint: 'event.target' represents the input field being changed) . Add this code right before the return statement:

const changeHandler = function (event) { const name = event.target.name;

const value = event.target.value; formObject[name] = value; setFormObject({ ...formObject })

}

1. Call the 'changeHandler' function from the 'onChange' event of both <input> elements:

onChange= {(e)=>changeHandler(e)}

1. Save the App.js file. Now keystrokes you enter into the input fields will be added to the formObject and displayed when the component is re-rendered.
2. Update the onClick handler for the 'Add' button to create a new array based on the existing articles array and add the formObject to it. This can be done using the spread operator. Use the 'setArticles' modifier function to update the articles variable with the new array:

onClick={() => setArticles([...articles, formObject])}

1. Save the App.js file.
2. Type some text into the title and contents input fields. Click on the 'Add Article' button. You should see an article added to the list. Click on the new article in the list and you will see its contents.
3. Add “delete selected” capability” once an article has been selected we want to have the ability to delete it from the list.
4. We have a button named 'Delete Selected' and we want that button to be disabled if no article is selected in the list. Lets create a 'const' type named 'validSelectedArticleId' that we can use to disable/enable the button. This code can be added right before the return statement:

const validSelectedArticleId = function () {

return( selectedArticleId >= 0 && selectedArticleId <articles.length);

}

1. Set the disabled property of the delete button from the value that was just created:

disabled={!validSelectedArticleId()}

1. Save the file and let the browser refresh. The 'Delete Selected' button should be grayed out until an article is selected.
2. Create a 'const' type named 'deleteSelected' and assign an anonymous function to it that deletes the selected article from the articles array and updates the articles array using its setArticles function. You can use the JavaScript array splice function to implement the delete. Add this code right before the return statement:

const deleteSelected = function () { if (validSelectedArticleId()) {

articles.splice(selectedArticleId, 1); setArticles([...articles]);

}

}

1. Update the onClick handler of the 'Delete Selected' button to call the 'deleteSelected' function:

onClick={() => deleteSelected()}

1. Save the App.js file. Select one of the articles by clicking on it in the list. The 'Delete Selected' button should become active. Click on 'Delete Selected'. The selected item should be removed from the list. Refreshing the page will restore the original set of articles.