React Custom Hooks

Let’s create a React application that uses a custom hook to encapsulate an article repository

1. We will create a React component that displays a list of articles. Replace the contents of App.js with the following and then save the file:

import React from 'react'; import './App.css';

export default function App() { return (<div className={'app'}>

<h2>React Custom Hooks</h2>

<li>one</li>

<li>two</li>

<li>three</li>

</div>

);

}

1. Save the file. Get the new index.css, App.css and articles.json.
2. Creating a custom hook: we will create a custom hook to encapsulate the list of articles. Create a new file in the src directory called repository.js
3. Add the following import at the top:

import { useEffect, useState } from 'react';

1. Let’s create the hook. Add the following function below the import statement:

export default function useArticles() {

}

1. Hooks can call other hooks. In the next set of steps, we will use the useEffect and useState hook to setup the repository.
2. In the function, let’s use a state to manage the list of articles. Add the line in bold:

export default function useArticles() {

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

}

1. To populate the list of articles, we will use an effect hook. Below the last line we’ve added, enter the following functions:

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

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

.then(data => { setArticles(data)

}

);

}

1. And load the list of articles by adding the following code below the function:

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

1. The useArticles hook is now able to load and manage a list of article using React hooks. Next, let’s provide that list of articles to the application via a repository.
2. Now let’s create a repository interface to interact with the list of articles.
3. At the end of the useArticles function, add the following constant:

const repository = {

};

1. Next, inside the curly brackets, add the list() function which will return the list of article:

list() {

return articles;

},

1. Let’s provide a count() function to retrieve the number of articles in the repository:

count() {

return articles.length;

},

1. Following that, we will create a helper to determine if an article id (index) is valid. Enter the following code:

isValidId(id) {

return id >= 0 && id < articles.length;

},

1. We will need to be able to retrieve articles by id. Let’s create the byId() function to achieve this:

byId(id) {

if (this.isValidId(id)) { return articles[id];

} else {

return null;

}

},

1. We will want to add an article. Let’s provide a function to achieve this:

add(title, content) {

setArticles([...articles, { title, content }]);

},

1. And finally, here’s a function to remove an article from the repository:

delete(id) {

articles.splice(id, 1);

setArticles([...articles]);

},

1. We are done the repository. At the end of the useArticles function (before the closing curly bracket), add the following line:

return repository;

1. Save the file.
   1. For the sake of reference, here’s what the completed repository.js should look like:

import { useEffect, useState } from 'react'; export default function useArticles() {

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

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

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

.then(data => { setArticles(data)

}

);

}

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

const repository = { list() {

return articles;

},

count() {

return articles.length;

},

isValidId(id) {

return id >= 0 && id < articles.length;

},

byId(id) {

if (this.isValidId(id)) { return articles[id];

} else {

return null;

}

},

add(title, content) {

setArticles([...articles, { title, content }]);

},

delete(id) {

articles.splice(id, 1);

setArticles([...articles]);

},

};

return repository;

}

1. Let’s integrate the repository. At the top of App.js, add useState to the list of imports:

import React, { useState } from 'react';

1. Let’s import our repository. Add the following line at the top:

import useArticles from './repository';

1. At the beginning of the App() function, add the following code:

const articles = useArticles();

1. The role of the repository is to maintain a list of article. To track the user interaction, we will rely on additional useStates. Enter the following code inside the App() function following the line we previously added:

const [formObject, setFormObject] = useState(

{ title: 'title1', content: 'content1' }

);

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

const selectedArticle = articles.byId(selectedArticleId)?.content || 'none';

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

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

}

1. Let’s now build the user interface. Amend the return statement as follows:

return (

<div className={'app'}>

<h2>React Custom Hooks App</h2>

<ul>

</ul>

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

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

</div >

);

1. Let’s display the list of articles by inserting the following code between the <ul>:

<ul>

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

return <li key={index} className={

(selectedArticleId === index) ? 'selected' : ''

}

)}

</ul>

}

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

{article.title}</li>

1. Save the file. At this point, the React application should display a default list of articles.
2. Modifying articles: In this last part, let’s add controls to allows the user to modify the list of articles. Before the closing </div> of the return statement, add the following code:

<br />

<div className={'controls'}>

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

<button onClick={false}>Add</button>

<button onClick={false} disabled={false} >Delete</button>

<br />

<input type={'text'} name={'title'} placeholder={'title'} value={formObject.title} onChange={(e) => changeHandler(e)}

/><br />

<input type={'text'} name={'content'} placeholder={'content'} value={formObject.content} onChange={(e) => changeHandler(e)}

/>

<br />

</div>

1. Save the file.
2. Let’s ensure that the Delete button is only enabled when an article is selected. Add the code in bold to the disabled attribute:

<button onClick={false} disabled={! articles.isValidId(selectedArticleId)}>Delete</button>

1. Save the file. Test the changes. Ensure that the Delete button is initially disabled, but is enabled once an article is selected.
2. Let’s now add the code to delete an article. In the onClick handler for the Delete button, add the following code:

onClick={() => articles.delete(selectedArticleId)}

1. Save the file and test the delete behavior.
2. Now let’s add the functionality behind the Add button.
3. Add the following code to the onClick handler of the Add button:

onClick={() => articles.add(formObject.title, formObject.content)}

1. Save the file and test.