**Objectives**

* Explain the need and Benefits of component life cycle
* Identify various life cycle hook methods
* List the sequence of steps in rendering a component

In this hands-on lab, you will learn how to:

* Implement componentDidMount() hook
* Implementing componentDidCatch() life cycle hook.

## **Prerequisites**

The following is required to complete this hands-on lab:

* Node.js
* NPM
* Visual Studio Code

## **Notes**

Estimated time to complete this lab: **60 minutes.**

1. Create a new react application using *create-react-app* tool with the name as “blogapp”
2. Open the application using VS Code
3. Create a new file named as **Post.js** in **src folder** with following properties



Figure 2: Post class

1. Create a new class based component named as **Posts** inside **Posts.js** file



Figure 3: Posts Component

1. Initialize the component with a list of Post in state of the component using the constructor
2. Create a new method in component with the name as **loadPosts()** which will be responsible for using Fetch API and assign it to the component state created earlier. To get the posts use the url (<https://jsonplaceholder.typicode.com/posts>)



Figure 4: loadPosts() method

1. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts



Figure 5: componentDidMount() hook

1. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.



Figure 6: render() method

1. Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.



Figure 7: componentDidCatch() hook

1. Add the Posts component to App component.
2. Build and Run the application using *npm start* command.

**Program:**

**Index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta

name="description"

content="Web site created using create-react-app"

/>

<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />

<!--

manifest.json provides metadata used when your web app is installed on a

user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/

-->

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

<!--

Notice the use of %PUBLIC\_URL% in the tags above.

It will be replaced with the URL of the `public` folder during the build.

Only files inside the `public` folder can be referenced from the HTML.

Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running `npm run build`.

-->

<title>React App</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

<!--

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.

To create a production bundle, use `npm run build` or `yarn build`.

-->

</body>

</html>

**Manifest.json:**

{

"short\_name": "React App",

"name": "Create React App Sample",

"icons": [

{

"src": "favicon.ico",

"sizes": "64x64 32x32 24x24 16x16",

"type": "image/x-icon"

},

{

"src": "logo192.png",

"type": "image/png",

"sizes": "192x192"

},

{

"src": "logo512.png",

"type": "image/png",

"sizes": "512x512"

}

],

"start\_url": ".",

"display": "standalone",

"theme\_color": "#000000",

"background\_color": "#ffffff"

}

**Robots.txt:**

# https://www.robotstxt.org/robotstxt.html

User-agent: \*

Disallow:

**App.css:**

.App {

text-align: center;

}

.App-logo {

height: 40vmin;

pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

.App-logo {

animation: App-logo-spin infinite 20s linear;

}

}

.App-header {

background-color: #282c34;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;

}

.App-link {

color: #61dafb;

}

@keyframes App-logo-spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

**App.js:**

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**App.test.js:**

import { render, screen } from '@testing-library/react';

import App from './App';

test('renders learn react link', () => {

render(<App />);

const linkElement = screen.getByText(/learn react/i);

expect(linkElement).toBeInTheDocument();

});

**Post.js:**

class Post {

constructor( id, title, body ) {

this.id = id;

this.title = title;

this.body = body;

}

}

export default Post;

**Posts.js:**

import React from "react";

import Post from "./Post";

class Posts extends React.Component {

state = {

posts : [],

error : null,

success : false

}

loadPosts() {

fetch("https://jsonplaceholder.typicode.com/posts")

.then(posts => posts.json())

.then(posts => {

const postObjects = posts.map(postData => new Post(postData.id, postData.title, postData.body))

this.setState({

posts : postObjects,

success : true

})

})

.catch(error => {

this.setState({

error : error.message

})

})

}

componentDidMount() {

this.loadPosts();

}

render() {

return (

<div>

<h1>Posts</h1>

<div>

{

this.state.success && this.state.posts.map((post, index) => {

return (

<div key={index} style={{border: "2px solid black", padding: "10px", margin: "10px", display: "flex"}}>

<div style={{paddingRight: "50px", alignSelf: "center"}}>

<p>{post.id}</p>

</div>

<div style={{display: "flex", flexDirection: "column", textAlign: "left"}}>

<p style={{fontWeight: 800, fontSize: "1.5rem"}}>{post.title}</p>

<p>{post.body}</p>

</div>

</div>

)

})

}

</div>

</div>

);

}

componentDidCatch(error, info) {

console.log(error);

console.log(info);

}

}

export default Posts;

**index.css:**

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

**Index.js:**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<App />

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

**reportWebVitals.js:**

const reportWebVitals = onPerfEntry => {

if (onPerfEntry && onPerfEntry instanceof Function) {

import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {

getCLS(onPerfEntry);

getFID(onPerfEntry);

getFCP(onPerfEntry);

getLCP(onPerfEntry);

getTTFB(onPerfEntry);

});

}

};

export default reportWebVitals;

**setupTests.js:**

// jest-dom adds custom jest matchers for asserting on DOM nodes.

// allows you to do things like:

// expect(element).toHaveTextContent(/react/i)

// learn more: https://github.com/testing-library/jest-dom

import '@testing-library/jest-dom';

**package.json:**

{

"name": "blogapp",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/dom": "^10.4.0",

"@testing-library/jest-dom": "^6.6.3",

"@testing-library/react": "^16.3.0",

"@testing-library/user-event": "^13.5.0",

"react": "^19.1.0",

"react-dom": "^19.1.0",

"react-scripts": "5.0.1",

"web-vitals": "^2.1.4"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

**Output:**

