**React JS-NOTES**

**===========================**

**JSON SERVER**

**1** npm install -g json-server@0.17.4(install in a folder)

2db.json

3npx json-server --watch db.json --port 3004

npm install @reduxjs/toolkit react-redux

**creating a react app**

**1 )npx create-react-app contacts**

**2)npm start**

**3)** import "./login.css";

**4)** import { Login } from './components/login/login';

**5)** install bootstrap bootstrap-icons axios formik react-router-dom mongodb express cors --save

**6)in index.js (**import '../node\_modules/bootstrap-icons/font/bootstrap-icons.css';

import '../node\_modules/bootstrap/dist/css/bootstrap.css';

import bootstrap from 'bootstrap';**)**

**7)** const [productName, setProductName] = useState('');

8) useEffect(()=>{ setProductName("Samsung TV"); }, [ ]);

9) { categories.map((category, index) => <li key={index}>{category}</li> ) }

10)

**React Features**

1. It is component based. 2. It uses virtual DOM. 3. It is modular. 4. It is loosely coupled and extensible

**FAQ: What is DOM?**

- Browser Architecture

User Interface

UI Backend

Browser Engine

Rendering Engine

Networking

JavaScript Interpreter

Data Persistence

**- HTML Parsing**

**Markup => Bytes => Chars => Tokens [tags] => Nodes [Elements] => DOM => Layout => Render => Paint**

**FAQ: What is shadow DOM?**

Ans : It is a hierarchy of elements in a component. HTML provides various pre-defined

components like date, email, file, color etc..

HTML also allows to create custom components using JavaScript.

**FAQ: What is virtual DOM?**

Ans:

It is a copy of actual DOM in browser memory.

The changes in page a updated in virtual DOM and later committed on actual DOM.

It make React application faster in interactions.

**FAQ: What is Modular?**

Ans: Modular refers to "part-by-part".

Software programming uses modular approach in building any application.

It allows to uses application

It makes application specific library.

It is light weight and faster.

It is easy to extend.

It enables reusability, maintainability, extensibility and testability.

**FAQ: What is Legacy?**

Ans: It is an approach where entire library is imported into application in order to use only few functionalities. It is heavy on application and uses more memory.

**Concerns of React:**

- It is not designed for what you are using.

- Hence lot of GAP's.

- You have to depend on lot of 3rd party libraries

- Poor documentation

- Pace of development in versions

**17-06-2024**

**Setup Environment for React**

1. Download and Install Node JS on your PC

https://nodejs.org/en

- Node JS is used to build web applications and create server.

- It provides an environment for managing web applications.

- It provides a package manager, which is used by developers to install, update or

uninstall any library required for project.

**Yarn Bower NuGet Composer RubyGems npm**

2. Open your command prompt and test

C:\> node -v

C:\> npm -v

3. Download and Install Visual Studio Code Editor

https://code.visualstudio.com/

4. Install following extensions for VS Code

- Live Server

- vscode-icons

- IntelliSense for CSS class names in HTML

**Create a new traditional web application:**

1. Create a new folder for your project

D:\web-app

2. Open project folder in VS code

- Open Terminal in VS Code and run the command

>npm init -y => generates package.json file

- Create a new file "README.md"

- Add following folders into project

a) public : It is used to keep all static resources, like

html pages, images, documents, media content etc..

b) src : It is used to keep all dynamic resources like

.js, .ts, .jsx, .tsx, .css, .scss, .less, etc..

3. Add following files into public folder

- index.html

- home.html

4. Add react environment to "home.html"

- You can add Legacy environment which is suitable up to 17x version.

- You can add modern environment for 18x versions.

- React 18x is a complete re-write of React library.

https://legacy.reactjs.org/ => for react up to 17x

https://react.dev/ => for react 18x

Adding React 17x environment to Home Page:

- Every react environment requires 3 basic libraries

a) react

b) react-dom

c) babel

- "react" is the core library for React JS.

- "react-dom" is virtual DOM library, it creates and manipulates virtual DOM.

- "babel" is JavaScript compiler.

a) JIT [ Just-in-Time ]

b) AOT [ Ahead-of-Time]

- You can get the libraries from CDN or you can download using package manager.

React CDN:

https://legacy.reactjs.org/docs/cdn-links.html

Babel CDN: [Standalone]

https://babeljs.io/docs/babel-standalone

- React 17 uses "ReactDOM.render()" method to create virtual DOM and render elements into virtual DOM.

Syntax:

<script type="text/babel">

ReactDOM.render("message", document.getElementById("root"));

</script>

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Index</title>

</head>

<body>

<h2>Welcome</h2>

<p>It is a traditional Web application.</p>

<p>React is not active in this page.</p>

<p>We are using react in <a href="home.html">Home</a> page.</p>

</body>

</html>

**home.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

<script type="text/babel">

ReactDOM.render("Welcome to React JS", document.getElementById("root"));

</script>

</head>

<body>

<noscript>Please enable JavaScript on your browser</noscript>

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

</body>

</html>

**18-06-2024**

- Setup environment

- React in existing web application with CDN links

react

react-dom

babel

Installing libraries for using react in web application:

1. Open project terminal

2. Run the following commands

>npm install react --save

>npm install react-dom --save

>npm install @babel/standalone --save

3. Library files are copied into a folder "node\_modules"

4. Link the "umd" module system files to your HTML page.

react.development.js

react-dom.development.js

babel.js

**Ex: home.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

<script src="../node\_modules/react/umd/react.development.js"></script>

<script src="../node\_modules/react-dom/umd/react-dom.development.js"></script>

<script src="../node\_modules/@babel/standalone/babel.js"></script>

<script type="text/babel">

ReactDOM.render("Welcome to React JS", document.getElementById("root"));

</script>

</head>

<body>

<noscript>Please enable JavaScript on your browser</noscript>

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

</body>

</html>

**React Components**

- React is component based library.

- Components are building blocks for react application.

- Every component comprises of

a) Presentation : HTML

b) Styles : CSS

c) Logic : JavaScript & TypeScript

- You can create a component in React by using 2 techniques

a) JavaScript Function b) JavaScript Class

**JavaScript Functions:**

- Function Declaration

- Function Expression

- Anonymous Function

- Function Recursion

- Function Callback

- Function Promises

- Function Closure

- Function Generator

- Function Return

- Function Rest Parameters

- Function Spread Operator

- Async Functions

Creating a component in React using JavaScript Function:

1. You can configure function using declaration or expression.

Syntax:

function Login() => Declaration

{

}

const Login = function() { => Function Expression

}

2. A component function can't be void type. It must return JSX element.

3. JSX refers to JavaScript Extension lib. It is JavaScript extension lang used to design components in React.

JSX Rules:

- JSX returns one fragment as a function.

- You can't return multiple fragments of markup.

return(

<h1> </h1> => invalid

<p> </p>

)

return(

<div>

<h1> </h1> => valid

<p> </p>

</div>

)

- JSX will not allow void elements syntax.

- Every element in JSX must have end token.

<img> => invalid

<img> </img> => valid

<img /> => valid

<input> </input>

<input />

- JSX element will not allow attributes, you have to use only properties.

<img src="" class="img-style">

document.querySelector("img").src = " ";

document.querySelector("img").className = "";

<img class="img-style"> // invalid

<img className="img-style"> // valid

home.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Home</title>

<style>

.container {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.frm-login {

padding: 20px;

border:1px solid gray;

border-radius: 10px;

}

.nav-bar {

display: flex;

justify-content: space-between;

padding: 5px;

border:1px solid black;

}

.nav-bar span {

width: 100px;

border:1px solid black;

padding: 5px;

text-align: center;

}

</style>

<script src="../node\_modules/react/umd/react.development.js"></script>

<script src="../node\_modules/react-dom/umd/react-dom.development.js"></script>

<script src="../node\_modules/@babel/standalone/babel.js"></script>

<script type="text/babel">

function Navbar(){

return(

<nav className="nav-bar">

<span>Home </span>

<span>Shop </span>

<span>Blog </span>

<span>Contact </span>

</nav>

)

}

function Login()

{

return (

<div className="container">

<form className="frm-login">

<h2>Login</h2>

<dl>

<dt>User Name</dt>

<dd><input type="text" /> </dd>

<dt>Password</dt>

<dd><input type="password"/> </dd>

</dl>

<button>Login</button>

</form>

</div>

)

}

ReactDOM.render(

<section>

<Navbar />

<Login />

</section> ,

document.getElementById("root"));

</script>

</head>

<body>

<noscript>Please enable JavaScript on your browser</noscript>

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

</body>

</html>

19-06-2024

React Components

- Function Components

- Class Components

- JSX Rules

-Component Name must start with uppercase letter.

Setup Bootstrap & Icons for React Project:

1. Install bootstrap & icons in your project

>npm install bootstrap bootstrap-icons --save

2. To use bootstrap & icons in any page link the following files

<link rel="stylesheet" href="../node\_modules/bootstrap-icons/font/bootstrap-icons.css">

<link rel="stylesheet" href="../node\_modules/bootstrap/dist/css/bootstrap.css">

<script src="../node\_modules/bootstrap/dist/js/bootstrap.bundle.js"> </script>

Ex:

Netflix.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Netflix - Watch Movies</title>

<style>

.bg-style {

background-image: url("images/netflix-banner.jpg");

background-size: cover;

height: 100vh;

}

.bg-shade {

background-color: rgba(0,0,0,0.5);

height: 100vh;

}

header {

padding: 20px;

display: flex;

justify-content: space-between;

}

.brand-title {

font-size: 40px;

color:red;

font-weight: bold;

text-shadow: 1px 1px 2px white;

}

main {

margin-top: 150px;

}

</style>

<link rel="stylesheet" href="../node\_modules/bootstrap-icons/font/bootstrap-icons.css">

<link rel="stylesheet" href="../node\_modules/bootstrap/dist/css/bootstrap.css">

<script src="../node\_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>

<script src="../node\_modules/react/umd/react.development.js"></script>

<script src="../node\_modules/react-dom/umd/react-dom.development.js"></script>

<script src="../node\_modules/@babel/standalone/babel.js"></script>

<script type="text/babel">

function NetflixIndex(){

return(

<div className="bg-style">

<div className="bg-shade">

<NetflixHeader />

<NetflixMain />

</div>

</div>

)

}

function NetflixHeader(){

return(

<header>

<div>

<span className="brand-title">NETFLIX</span>

</div>

<div>

<div className="input-group">

<span className="bi bi-translate input-group-text"></span>

<select className="form-select">

<option>Language</option>

<option>English</option>

<option>हिंदी</option>

</select>

<button className="btn btn-danger ms-2">Signin</button>

</div>

</div>

</header>

)

}

function NetflixMain(){

return(

<main>

<div className="text-white text-center">

<div className="h1">

Unlimited movies, TV shows and more

</div>

<div className="h3">

Watch anywhere. Cancel anytime.

</div>

<NetflixRegister />

</div>

</main>

)

}

function NetflixRegister(){

return(

<div>

<p>Ready to watch? Enter your email to create or restart your membership.</p>

<div className="d-flex justify-content-center">

<form className="input-group input-group-lg w-50">

<input type="email" placeholder="Your email address" className="form-control" />

<button className="btn ms-1 btn-danger"> Get Started <span className="bi bi-chevron-right"></span> </button>

</form>

</div>

</div>

)

}

ReactDOM.render(<NetflixIndex />, document.getElementById("root"));

</script>

</head>

<body>

<noscript>Please enable JavaScript on your browser</noscript>

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

</body>

</html>

Ex: With Bootstrap Model

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Netflix - Watch Movies</title>

<style>

.bg-style {

background-image: url("images/netflix-banner.jpg");

background-size: cover;

height: 100vh;

}

.bg-shade {

background-color: rgba(0,0,0,0.5);

height: 100vh;

}

header {

padding: 20px;

display: flex;

justify-content: space-between;

}

.brand-title {

font-size: 40px;

color:red;

font-weight: bold;

text-shadow: 1px 1px 2px white;

}

main {

margin-top: 150px;

}

</style>

<link rel="stylesheet" href="../node\_modules/bootstrap-icons/font/bootstrap-icons.css">

<link rel="stylesheet" href="../node\_modules/bootstrap/dist/css/bootstrap.css">

<script src="../node\_modules/bootstrap/dist/js/bootstrap.bundle.js"></script>

<script src="../node\_modules/react/umd/react.development.js"></script>

<script src="../node\_modules/react-dom/umd/react-dom.development.js"></script>

<script src="../node\_modules/@babel/standalone/babel.js"></script>

<script type="text/babel">

function NetflixIndex(){

return(

<div className="bg-style">

<div className="bg-shade">

<NetflixHeader />

<NetflixMain />

</div>

</div>

)

}

function NetflixHeader(){

return(

<header>

<div>

<span className="brand-title">NETFLIX</span>

</div>

<div>

<div className="input-group">

<span className="bi bi-translate input-group-text"></span>

<select className="form-select">

<option>Language</option>

<option>English</option>

<option>हिंदी</option>

</select>

<button data-bs-target="#login" data-bs-toggle="modal" className="btn btn-danger ms-2">Sign In</button>

<div className="modal fade" id="login">

<div className="modal-dialog modal-dialog-centered">

<div className="modal-content">

<div className="modal-header">

<h3 className="bi bi-person-fill"> User Login </h3>

<button data-bs-dismiss="modal" className="btn btn-close"></button>

</div>

<div className="modal-body">

<dl>

<dt>User Id</dt>

<dd><input type="text" className="form-control" /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control" /></dd>

</dl>

</div>

<div className="modal-footer">

<button className="btn btn-warning"> Login </button>

<button data-bs-dismiss="modal" className="btn btn-danger"> Cancel </button>

</div>

</div>

</div>

</div>

</div>

</div>

</header>

)

}

function NetflixMain(){

return(

<main>

<div className="text-white text-center">

<div className="h1">

Unlimited movies, TV shows and more

</div>

<div className="h3">

Watch anywhere. Cancel anytime.

</div>

<NetflixRegister />

</div>

</main>

)

}

function NetflixRegister(){

return(

<div>

<p>Ready to watch? Enter your email to create or restart your membership.</p>

<div className="d-flex justify-content-center">

<form className="input-group input-group-lg w-50">

<input type="email" placeholder="Your email address" className="form-control" />

<button className="btn ms-1 btn-danger"> Get Started <span className="bi bi-chevron-right"></span> </button>

</form>

</div>

</div>

)

}

ReactDOM.render(<NetflixIndex />, document.getElementById("root"));

</script>

</head>

<body>

<noscript>Please enable JavaScript on your browser</noscript>

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

</body>

</html>

Creating a new React Application:

- You can create a react application using various bundling tools like

a) Webpack

b) Parcel

c) Vite

- Bundling tools allow to configure a project with development, quality and production environment.

- Node JS have in-built support for "Webpack" bundler.

20-06-2024

**React 18 Application**

- React 18 is a complete re-write of React.

- It provides a new approach for building interfaces.

- You can build interactive interface for web and native apps.

- NPM provides support for "Web Pack" bundling tool, which is used to configure and create a React application.

1. Open your PC location in command prompt to create react project

2. Run the following command

F:\> npx create-react-app react-project

3. Open project folder "f:\react-project" in visual studio code

4. Project Infrastructure comprises of various files & folder

node\_modules It contains the library files installed on your project.

public It comprises of static resources like .html, images,

docs, media clips etc.

src It comprises of dynamic resources like .js, .jsx, .ts,

.tsx, .css, .scss, .less etc.

.gitignore It configure the repository to ignore while publishing

on GIT.

package.json It contains project meta data.

package.lock.json It is used for migrations.

README.md It is a help document for developers.

**React Application Flow: [High Level]**

1. React Webpack hosts your application on local server. It creates a server locally that runs on port number "3000".

2. Open your project terminal in VS code and run the command

> npm start

You can server starts listening on "http://127.0.0.1:3000" (or) http://localhost:3000"

3. Application starts with "index.html"

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

4. Logic for index.html is defined in "index.js"

React up to 17x versions:

ReactDOM.render(<component/>, document.getElementById("root"));

**React 18x versions**

const root = createRoot(document.getElementById("root"));

root.render(

<React.StrictMode>

<component />

</React.StrictMode>

)

Note:

- createRoot() is used to create virtual DOM from 18x versions.

- <React.StrictMode> is used to turn strict mode ON for developers.

- Strict mode will prevent code inconsistency.

- It is used only in "development" not for "production".

Add a new component into React application:

- Traditionally every component comprises of following files

.jsx : It comprises of markup to return & the logic for component

.css : It comprises of styles

.test.js : It comprises for test functions used to test your component.

- Go to "src" folder and add a sub folder for every component

src/components/login

- login.jsx

- login.css [optional]

- login.test.js [optional]

**login.css**

.form-login {

border: 1px solid gray;

padding: 10px;

border-radius: 10px;

}

.container {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

**login.jsx**

import "./login.css";

export function Login(){

return(

<div className="container">

<form className="form-login">

<h3>User Login</h3>

<dl>

<dt>User Name</dt>

<dd><input type="text" /></dd>

<dt>Password</dt>

<dd><input type="password"/></dd>

</dl>

<button>Login</button>

</form>

</div>

)

}

index.js

import { Login } from './components/login/login';

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

root.render(

<React.StrictMode>

<Login />

</React.StrictMode>

);

**21-06-2024**

Creating React Application using NPX [Web Pack Bundler]

> npx create-react-app app-name

> npm start

FAQ: What is "npm start" ?

Ans: It is a batch program to compile and run react application.

FAQ: What is a batch program?

Ans: A batch program comprises of series of commands defined under specified name.

So that you can execute the commands in sequential order by running the command name.

FAQ: Where to add batch command?

Ans : In package.json => "scripts"

"scripts": {

"shorthand" : "your commands … "

}

**Setup Bootstrap for React application:**

1. Install bootstrap and icons using npm package manager

> npm install bootstrap bootstrap-icons --save

2. Go to "index.js" and import the following

import '../node\_modules/bootstrap-icons/font/bootstrap-icons.css';

import '../node\_modules/bootstrap/dist/css/bootstrap.css';

import bootstrap from 'bootstrap';

**Ex:login.jsx**

import "./login.css";

export function Login(){

return(

<div className="container">

<form className="form-login alert alert-warning alert-dismissible">

<h3 className="bi bi-person-fill">User Login</h3>

<button data-bs-dismiss="alert" className="btn btn-close"></button>

<dl>

<dt>User Name</dt>

<dd><input type="text" className="form-control" /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control"/></dd>

</dl>

<button className="btn btn-warning w-100">Login</button>

</form>

</div>

)

}

**React Data Binding**

- Databinding is a technique of web application.

- Data binding allows to access data from memory source and bind to UI elements.

- It also allows to identify the change in data from UI and update back into memory source.

- JavaScript requires lot of DOM methods and Events to handle data binding.

- React library can manage one-way-binding by using a binding expression defined in embedded block "{ }".

Syntax: var title = "User Login";

<h3> { title } </h3>

var uname = "David";

<input type="text" value={uname} />

**Note: Never use variables to store and handle data for a component.**

**Variables are immutable types, and a component requires mutable memory.**

**State in React**

- Web applications use "http" as protocol.

- "Http" is a state less protocol.

- It can't remember information between requests.

- It requires various state management techniques.

- State management is classified into

a) Client Side State b) Server Side State

- Client Side state management includes

**a) Query String b) Local Storage c) Session Storage d) Cookies**

- React can use all client side state management techniques.

- React also provides a local state for components, so that they can manage data in virtual DOM.

- React provides "useState()" hook [function] to maintain a local state for component.

- React have various state related hooks

useState()

useContext()

useReducer()

useMemo()

useCallBack()

etc...

Syntax:

import {useState} from 'react';

const [getter, setter] = useState(anyValue);

<p> {getter} </p>

Ex:

import "./login.css";

import { useState } from "react";

export function Login(){

const [title, setTitle] = useState('User Login');

const [userName, setUserName] = useState('John');

return(

<div className="container">

<form className="form-login alert alert-warning alert-dismissible">

<h3 className="bi bi-person-fill">{title}</h3>

<button data-bs-dismiss="alert" className="btn btn-close"></button>

<dl>

<dt>User Name</dt>

<dd><input type="text" value={userName} className="form-control" /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control"/></dd>

</dl>

<button className="btn btn-warning w-100">Login</button>

</form>

</div>

)

}

22-06-2024

**React Data Binding**

**- Databinding Expression { }**

**- useState()**

**Syntax:**

const [getter, setter] = useState(anyValue);

**FAQ: Can we use "var" or "let" to declare state in React?**

Ans: Yes. But not recommended.

**FAQ: Why state is defined with "const"? Why not "let or var"?**

Ans: Initialization is mandatory for state, You have to initialize state at the time of

creating component hence "const" is recommended.

"var or let" may skip initialization, which is not recommended for state.

**FAQ: How you can assign a value into state if you declared using "const"?**

Ans: State will not assign values, It initializes memory every time using a "setter",

that allows to store data.

**Syntax:**

const [name, setName] = useState('John');

setName = newValue; // invalid

setName(newValue); // valid

**FAQ: What type of memory is allocated for state? [useState()]**

Ans: Component allocates "Static" memory.

Memory allocated for first request and the same is used across requests.

**Note: Component renders into virtual DOM when requested by client.**

**The assignment and initialization of values in component will repeat the rendering.**

**React will not allow re-rendering infinite times.**

**Component Mount Phase**

- Component is created when requested by client.

- Component will mount after creation.

- You can define actions such as initialization of values into state and other functions in mount phase.

- The component mount phase is configured by using "useEffect()" hook.

**Syntax:**

useEffect(function(){ }, [dependencies])

useEffect(()=>{ },[ dependencies ])

- Component mount occurs only once.

- If you want component to mount across various changes, then it is mandatory to define the dependencies.

- Dependency configuration is required even when there are no dependencies.

**Ex:data-binding.jsx**

import { useEffect, useState } from "react";

export function DataBinding(){

const [productName, setProductName] = useState('');

useEffect(()=>{

setProductName("Samsung TV");

}, []);

return(

<div className="container-fluid">

<h2>Data Binding</h2>

Product Name : {productName}

</div>

)

}

**Handling Various Data Types**

**1. Number Type**

Signed Integer -24

Unsigned Integer 24

Float 34.56

Double 234.56

Decimal 35600.45003 [21 to 29]

Hexadecimal 0x4673

Octa 0o745

Binary 0b1010

Exponent 2e3

Bigint 9994888272n

- Parsing is same as in JS.

parseInt()

parseFloat()

- Verifying number is same as in JS

isNaN()

- Converting to locale string is also same as in JS

toLocaleString()

Syntax: const [price, setPrice] = useState(560000.55);

{ price.toLocaleString('en-in', { style:'currency', currency:'INR' }) }

- You can crop values using precision & fixed methods

toPrecision() : crop the value up to specified chars length

toFixed() : to set fractions.

Syntax: price.toPrecision(8); 56000.456

price.toFixed(2); 56000.46

- JavaScript Math library is same in React to handle various Math operations on a number.

Math.PI Math.sqrt() Math.pow() Math.random() Math.round() Math.sin() etc...

Syntax: const [code, setCode] = useState();

setCode(Math.random());

{ code }

**2. Boolean Type**

- JS Boolean is same in React.

- It handles "true or false".

- React JSX can't display a Boolean value. It can only use a Boolean value.

- Hence you have to use various operators to present relative content based on Boolean value.

Syntax:

const [stock, setStock] = useState();

setStock(true);

Stock : {stock} // returns empty

Stock : { (stock)?"true":"false" } // returns "true"

**Note: JavaScript boolean can use "1" for true and "0" for false, which is not recommended in react. Always use only "true or false" keywords for boolean.**

**3. String4. Undefined5. Null6. Symbol7. Date8. Array9. Object10. Map / Set**

**24-06-2024**

**3. String Type**

- String is configured using

a) Double Quote " "

b) Single Quote ' '

c) Backtick ` `

- Backtick allows a data binding expression "${ }" to bind dynamic value.

**Syntax:**

const [val, setVal] = useState('bg-dark');

<h2 className={ `text-center text-warning ${val}` }> Your Text </h2>

<h2 className={ 'text-center text-warning' + val' }> Your Text </h2>

- All string formatting and manipulation methods are same as in JavaScript.

bold() charAt()

italics() charCodeAt()

sup() indexOf()

sub() lastIndexOf()

fontcolor() match()

fontsize() startsWith()

toUpperCase() trim() etc...

toLowerCase()

Ex:

import { D, useState } from "react";

export function DataBinding(){

const [val, setVal] = useState();

const [title] = useState('Data Binding');

useEffect(()=>{

setVal('bg-dark p-2');

}, []);

return(

<div className="container-fluid">

<h2 className={`text-center text-warning ${val}`}> {title.toUpperCase()} </h2>

</div>

)

}

**4. Undefined**

- It is same as in JS.

- You can use "undefined" keyword to verify value at compile time.

**5. Null**

- It is same as in JS.

- You can use "null" keyword to verify value at runtime.

**Syntax:**

const [uname, setUname] = useState(prompt('Enter Name'));

{ (uname==null) ? 'please provide a name' : uname }

{ (uname==undefined) ? 'please provide a name' : uname }

**6. Symbol**

- It is same as in JS.

- It is used to configure a unique hidden field in object.

**Syntax:**

var Id = Symbol();

const [product] = useState({ [Id]:1, Name:"TV", Price: 45000.55 });

**Ex:**

<script>

var Id = Symbol();

var product = {

[Id]: 101,

Name: "TV",

Price: 45000.44

}

for(var property in product)

{

document.write(`${property} : ${product[property]}<br>`);

}

document.write(product[Id]);

</script>

**7. Date Type**

- JavaScript date() constructor is same in React.

**Syntax:**

const [mfd, setMfd] = useState();

setMfd(Date()); // stores current date and time

setMfd(Date('yy-mm-dd hrs:min:sec:milliSec')); // stores specfic date & time

- React can use all Date methods but can't handle Locale formats.

getHours() setHours()

getMinutes() setMinutes()

getSeconds() setSeconds()

getMilliSeconds() setMilliSeconds()

getDate() setDate()

getMonth() setMonth()

getFullYear() setYear() etc..

getDay()

toLocaleDateString() \*

toDateString()

toLocaleTimeString() \*

toTimeString()

toString()

etc..

- Date() is a getter, which gets the current date.

- You can configure "new Date()" which allocates dynamic memory to store value.

Syntax:

const [mfd] = useState(Date());

const [mfd] = useState(new Date('2024-06-29'));

{ mdf.toLocaleDateString() } ] => not recommended as it belongs to DOM.

{ mdf.toDateString() } ]

- Always use a "Date Adapter" for Virtual DOM.

- React don't have any in-build date adapter, you have to depend on 3rd party like

a) dayjs

b) moment

**npmjs.com**

1. Install moment library

> npm install moment --save

2. configure date in react component

const [mfd ] = useState(new Date('2024-06-29'));

3. Import moment function

import moment from "moment";

4. configure the format for date value

{ moment(mfd).format('DD dddd, MM yyyy') }

DD - numeric

dd - short string

dddd - long string

MM - month number

MMM - short month string

MMMM - long month string

yy - 2 digits year

yyyy - 4 digits year

Ex:

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [mfd, setMfd] = useState(new Date('2024-06-29'));

useEffect(()=>{

}, []);

return(

<div className="container-fluid">

<p>

Manufactured : {moment(mfd).format('DD dddd, MMMM yyyy')}

</p>

</div>

)

}

**8. Regular Expressions**

- A regular expression comprises of meta characters and quantifiers.

- Expression is enclosed in "/ /".

**Syntax:**const [regExp] = useState(/(?=.\*[A-Z])\w{4,15}/);

- Expression is verified using "match()" method.

{ ('david123'.match(regExp)) ? "Verified" : "Invalid" }

**9. Array**

- All array concepts are same as in JS.

- All array methods are same.

- You have to use various array iterators to read all elements and present in UI.

a) map() b) forEach() c) find() d) filter() e) join() f) toString()

Ex:

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [categories] = useState(['All', 'Electronics', 'Footwear', 'Fashion']);

useEffect(()=>{

}, []);

return(

<div className="container-fluid">

<ol>

{

categories.map(category=> <li>{category}</li>)

}

</ol>

<select>

{

categories.map(category=><option>{category}</option>)

}

</select>

</div>

)

}

**25-06-2024**

**Array Type**

- Array configuration is same as in JS

a) [ ] b) Array()

- Array methods as same as in JS

- Array is presented using iterators

map() forEach()

**Syntax:**

const [categories] = useState( ['All', 'Electronics' ])

categories.map(category=> <container> { category } </container>)

- Container you defined in iteration will repeat for every element in array.

- Every repeating element in React must have a unique key, which you can define by using "key" property.

**Syntax:**

categories.map(category => <li key={category}> {category} </li>

Ex:**data-binding.jsx**

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [categories] = useState(['All', 'Electronics', 'Footwear', 'Fashion']);

useEffect(()=>{

}, []);

return(

<div className="container-fluid">

<ol>

{

categories.map((category, index) => <li key={index}>{category}</li> )

}

</ol>

<select>

{

categories.map((category, index)=> <option key={index}> {category} </option>)

}

</select>

{

categories.map((category, index)=> <h2 key={index}>{category}</h2>)

}

</div>

)

}

**Object Type**

- JavaScript object and its manipulations are same in React.

- Object keeps all related data and logic together.

- Object is a key-value collection.

- Key is string type and value can be any type.

{

"Key" : value,

"Key" : value

}

**Syntax:**

const [obj] = useState({key:value});

{ obj.key }

**Ex:**

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [product] = useState({Name:"Samsung TV", Price:45000.33,cost:35000, Cities:["Delhi","Hyd"], Rating:{Rate:4.3, Count:200}});

useEffect(()=>{

}, []);

return(

<div className="container-fluid">

<h2>Product Details</h2>

<dl>

<dt>Name</dt>

<dd>{product.Name}</dd>

<dt>Price</dt>

<dd>{product.Price.toLocaleString('en-in', {style:'currency', currency:'INR'})}</dd>

<dt>Cities</dt>

<dd>

<ol>

{

product.Cities.map(city=> <li key={city}>{city}</li>)

}

</ol>

</dd>

<dt>Rating</dt>

<dd>

{product.Rating.Rate} <span className="bi bi-star-fill text-success"></span>

[{product.Rating.Count}]

</dd>

</dl>

</div>

)

}

- Object manipulation techniques are same as in JS.

**1. How to delete any key from object?**

A. by using JS "delete" operator.

**2. How to read all keys from object?**

A. by using JS "Object.keys()" method

**3. How to verify the existence of a Key in object?**

A. by using JS "in" operator

**4. How to know the data type of value in key?**

A. by using JS "typeof" operator

**FAQ: What are the issues with object type?**

Ans:

- Keys are string type.

- You can't have number or Boolean type keys.

- It requires lot of explicit operators and methods for manipulation.

- It is slow in interactions.

- However is structured data. [schema based]

**FAQ: What is alternative for object type?**

Ans: "map" is alternative.

- Key can be any type in map.

- Map provides implicit methods for manipulation.

- Map is faster than object.

- It is schema less. [structure less]

**Syntax:**

const [data] = useState(new Map(1, "TV"));

**data.get() data.set() data.delete() data.size data.keys() data.values() data.entries()**

**data.has() data.clear() etc..**

**JavaScript Ajax Methods to Access Data**

- JavaScript provides 2 techniques for Ajax request

a) XMLHttpRequest

b) fetch()

26-06-2024

**Various Ajax Request Techniques**

**1. JavaScript XMLHttpRequest object**

- It is the native browser method to communicate with remote API's.

- It is defined by JavaScript "window".

- It can create async requests and handle interactions with remote file or URL.

**Step-1:** Create a new object for XMLHttpRequest

var http = new XMLHttpRequest();

**Step-2:** Configure the request by using "open()"

http.open("method", "file / url", async:boolean);

method => GET, POST, PUT, PATCH, DELETE

**Step-3:** Send request to file or URL through browser

http.send();

**Step-4:** Check the response of Http Request and configure a function to process

the actions after response.

http.onreadystatechange = function() {

}

**Step-5:** Check the status of response which can be a numeric value from 1 to 4.

http.readyState => 1=Initial, 2=Success, 3=Complete, 4=Ready

**Step-6:** On ready get the response text by using "responseText" property

var ref = http.responseText;

**Issues with XMLHttpRequest:**

- It is not implicitly async.

- You have to explicitly configure async.

- It returns response in various formats. [XML, Text, HTML]

- You need various parsing techniques.

- It is not good in error handling.

- Browser may block the request when data is received in various formats.

Ex:

1. Go to public folder and add "product.json"

{

"title": "Apple iPhone 15 (Green, 128 GB)",

"price": 79999,

"rating": {"rate":4.8, "ratings":43000, "reviews":12000},

"offers": [

"Bank OfferGet ₹50 Instant Discount on first Flipkart UPI transaction on order of ₹200 and aboveT&C",

"Bank Offer5% Cashback on Flipkart Axis Bank CardT&C",

"Bank Offer₹2000 Off On ICICI Bank Credit Non EMI, Credit and Debit Card EMI TransactionsT&C",

"Special PriceGet extra ₹7901 off (price inclusive of cashback/coupon)T&C"

],

"image": "iphone-green.jpg"

}

**2. data-binding.jsx**

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [product, setProduct] = useState({title:'', price:0, rating:{rate:0, ratings:0, reviews:0}, offers:[], image:''});

function LoadData(){

var http = new XMLHttpRequest();

http.open("get", "product.json", true);

http.send();

http.onreadystatechange = function(){

if(http.readyState==4)

{

setProduct(JSON.parse(http.responseText));

}

}

}

useEffect(()=>{

LoadData();

}, []);

return(

<div className="container-fluid">

<div className="row mt-4">

<div className="col-3">

<img src={product.image} width="100%" />

</div>

<div className="col-9">

<div className="fs-4 fw-bold">{product.title}</div>

<div className="mt-2">

<span className="bg-success rounded text-white p-1"> <span>{product.rating.rate}</span> <span className="bi bi-star-fill"></span> </span>

<span className="ms-3 fw-bold text-secondary"> {product.rating.ratings} ratings & {product.rating.reviews} reviews </span>

</div>

<div className="mt-3">

<span className="fw-bold fs-3"> {product.price.toLocaleString('en-in',{style:'currency', currency:'INR'})} </span>

</div>

<div className="mt-3">

<h5>Available Offers</h5>

<ul className="list-unstyled">

{

product.offers.map(offer=><li className="bi bi-tag-fill text-success my-2" key={offer}> <span className="text-secondary">{offer}</span> </li>)

}

</ul>

</div>

</div>

</div>

</div>

)

}

**2. JavaScript "fetch()" promise**

- JavaScript function promise is async by default.

- It provides a better error handling technique when compared to call backs.

- "fetch()" is a JavaScript promise provided by "window" object.

Syntax:

fetch("URL / File").then().catch().finally()

then() executes on success and returns a response object, that contains

the response data.

catch() executes on error and returns an error object, that contains error

details.

finally() executes always.

Issues with fetch() promise:

- Data is returned in binary format.

- Explicit conversion is required.

- Binary conversion may be blocked by firewall.

[Firewall blocks COM level marshalling].

- It uses catch() for handling errors, but can't track the status of request.

**Ex: data-binding.jsx**

import { useEffect, useState } from "react";

import moment from "moment";

export function DataBinding(){

const [product, setProduct] = useState({title:'', price:0, rating:{rate:0, ratings:0, reviews:0}, offers:[], image:''});

function LoadData(){

fetch("product.json")

.then(response=> {

return response.json();

})

.then(product => {

setProduct(product);

})

}

useEffect(()=>{

LoadData();

}, []);

return(

<div className="container-fluid">

<div className="row mt-4">

<div className="col-3">

<img src={product.image} width="100%" />

</div>

<div className="col-9">

<div className="fs-4 fw-bold">{product.title}</div>

<div className="mt-2">

<span className="bg-success rounded text-white p-1"> <span>{product.rating.rate}</span> <span className="bi bi-star-fill"></span> </span>

<span className="ms-3 fw-bold text-secondary"> {product.rating.ratings} ratings & {product.rating.reviews} reviews </span>

</div>

<div className="mt-3">

<span className="fw-bold fs-3"> {product.price.toLocaleString('en-in',{style:'currency', currency:'INR'})} </span>

</div>

<div className="mt-3">

<h5>Available Offers</h5>

<ul className="list-unstyled">

{

product.offers.map(offer=><li className="bi bi-tag-fill text-success my-2" key={offer}> <span className="text-secondary">{offer}</span> </li>)

}

</ul>

</div>

</div>

</div>

</div>

)

}

**3. jQuery Ajax methods**

- jQuery introduces the approach of "write less, do more".

- It provides various DOM interaction methods and Ajax methods.

- Ajax methods of jQuery provides various life cycle events.

- It is good in tracking the errors.

- It doesn't require parsing methods.

$.ajax()

$.getJSON()

.ajaxStart()

.ajaxStop()

.ajaxSuccess()

.ajaxComplete()

.ajaxError()

Setup jQuery for React:

1. Install jQuery library for react project

> npm install jquery --save

2. Import jQuery core into your "index.js"

import $ from "jquery";

$ => is jQuery core that comprises of all jQuery methods.

3. jQuery Ajax call is configured by using Ajax method

$.ajax({

method: "get | post | put | patch | delete",

url : "file / URL",

success: function(){ }, => returns response object

error: function(){ } => returns error object

})

Note: Import jQuery library into your component if you are not using any routing technique for navigation in application.

Issues with jQuery Ajax:

- It is not good for virtual DOM.

- It can't configure life cycle in virtual DOM.

- CORS [Cross Origin Resource Sharing]

- Request Forgery

- XSS [Cross Site Scripting] Attacks

**Ex: data-binding.jsx**

import { useEffect, useState } from "react";

import moment from "moment";

import $ from "jquery";

export function DataBinding(){

const [product, setProduct] = useState({title:'', price:0, rating:{rate:0, ratings:0, reviews:0}, offers:[], image:''});

function LoadData(){

$.ajax({

method: "get",

url: "product.json",

success: (response)=> {

setProduct(response);

},

error:(xhre) =>{

alert(xhre);

}

})

}

useEffect(()=>{

LoadData();

}, []);

return(

<div className="container-fluid">

<div className="row mt-4">

<div className="col-3">

<img src={product.image} width="100%" />

</div>

<div className="col-9">

<div className="fs-4 fw-bold">{product.title}</div>

<div className="mt-2">

<span className="bg-success rounded text-white p-1"> <span>{product.rating.rate}</span> <span className="bi bi-star-fill"></span> </span>

<span className="ms-3 fw-bold text-secondary"> {product.rating.ratings} ratings & {product.rating.reviews} reviews </span>

</div>

<div className="mt-3">

<span className="fw-bold fs-3"> {product.price.toLocaleString('en-in',{style:'currency', currency:'INR'})} </span>

</div>

<div className="mt-3">

<h5>Available Offers</h5>

<ul className="list-unstyled">

{

product.offers.map(offer=><li className="bi bi-tag-fill text-success my-2" key={offer}> <span className="text-secondary">{offer}</span> </li>)

}

</ul>

</div>

</div>

</div>

</div>

)

}

**4. React 3rd Party Ajax Libraries**

a) Telerik

b) Axios

c) WhatwgFetch

etc..

27-06-2024

**Axios**

- It is a JavaScript library used to communicate with API.

- You can configure and handle Ajax calls using axios.

- It provides virtual DOM interactions. [implicitly uses XMLHttpRequest]

- It can handle CORS. [Cross Origin Resource Sharing]

- It can prevent XSS. [Cross Site Scripting Attack]

- It can prevent Cross Page Posting.

- It can prevent Request Forgery.

- It is good in error handling.

- It can handle multiple requests simultaneously at the same time.

**Note: You can find various 3rd party libraries @ https://www.npmjs.com/**

1. Install axios for your project

> npm install axios --save

2. Import axios into component

import axios from "axios";

3. You can make a request using various short hand methods

axios.get() to fetch data

axios.post() to submit data

axios.put() to allow modify

axios.patch() to allow partial modify

axios.delete() to remove

Syntax:

axios.get("url").then(()=>{}).catch(()=>{}).finally(()=>{})

then() It is a callback on success, which returns response object with

various response details.

a) status [code] 200, 404

b) statusText OK, Not Found

c) headers GET, POST, MIME TYPE

d) data Data returned by URL

Ex:

**data-binding.jsx**

import { useEffect, useState } from "react";

import axios from "axios";

export function DataBinding(){

const [product, setProduct] = useState({title:'', price:0, rating:{rate:0, ratings:0, reviews:0}, offers:[], image:''});

function LoadData(){

axios.get("product.json")

.then(response=>{

setProduct(response.data);

})

.catch(err=> {

console.log(err);

})

}

useEffect(()=>{

LoadData();

}, []);

return(

<div className="container-fluid">

<div className="row mt-4">

<div className="col-3">

<img src={product.image} width="100%" />

</div>

<div className="col-9">

<div className="fs-4 fw-bold">{product.title}</div>

<div className="mt-2">

<span className="bg-success rounded text-white p-1"> <span>{product.rating.rate}</span> <span className="bi bi-star-fill"></span> </span>

<span className="ms-3 fw-bold text-secondary"> {product.rating.ratings} ratings & {product.rating.reviews} reviews </span>

</div>

<div className="mt-3">

<span className="fw-bold fs-3"> {product.price.toLocaleString('en-in',{style:'currency', currency:'INR'})} </span>

</div>

<div className="mt-3">

<h5>Available Offers</h5>

<ul className="list-unstyled">

{

product.offers.map(offer=><li className="bi bi-tag-fill text-success my-2" key={offer}> <span className="text-secondary">{offer}</span> </li>)

}

</ul>

</div>

</div>

</div>

</div>

)

}

- React supports "One Way Binding" implicitly.

**FAQ: What is One Way Binding?**

Ans: It is a data binding technique, which allows to access data from source and bind to UI elements. It will not allow changes to data in UI.

It can secure the data from auto updates.

It prevents accidental data modifications.

**FAQ: What is Two Way Binding?**

Ans: It is a data binding technique, which allows to access data from source and bind to

UI elements, identify the change in value and update back into data source.

Several technologies support two way data binding with implicit techniques.

React requires explicit methods to handle two way binding.

**React Two way binding**

- React will not support two way binding implicitly.

- It requires event binding techniques to handle two way binding.

- There are various events supported by React but two way binding is managed only with "onChange" event.

Syntax:

function handleChange(e)

{

e.target.value; // to access element value

}

<input type="text" onChange={handleChange}>

**Ex: one-way.jsx**

import { useState } from "react"

export function OneWay(){

const [userName, setUserName] = useState('John');

function handleNameChange(event){

setUserName(event.target.value);

}

return(

<div className="container-fluid">

<h2>One Way Binding</h2>

<label>User Name </label>

<input type="text" onChange={handleNameChange} value={userName} />

<h3>Hello ! {userName}</h3>

</div>

)

}

**React Event Binding**

**1. What is Event?**

A. Event is a message sent by sender to its subscriber in order to notify the change.

It follows a "delegate" mechanism, which is a function pointer mechanism.

Event follow a software design pattern called "Observer".

**2. What is a sender in event?**

A. It refers to a trigger that points towards specified function.

onclick="InsertClick()" => onclick trigger pointing towards InsertClick

**3. What is subscriber in event?**

A. It defines the actions to perform when element triggers event.

function InsertClick()

{

// actions

}

**4. What is Event handler?**

A. Event handler refers to a trigger and function pointer configured for elements in design. It comprises of event name and function name.

<button onclick="InsertClick()">

----------------------------

Event Handler

onclick => event

InsertClick() => function

onclick="InsertClick()" => event handler

28-06-2024

**5. What is Event Listener?**

A. Event Listener configures event for element dynamically.

JavaScript uses "addEventListener()" method for configuring event listener.

**Syntax:**

var btn = document.createElement("button");

btn.innerHTML = "Insert";

btn.addEventListener("event", ()=>{ // event => click, change, blur …

})

Ex:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function bodyload(){

var btn = document.createElement("button");

btn.innerHTML = "Update";

btn.addEventListener("click", ()=>{

alert("Updated");

})

document.getElementById("container").appendChild(btn);

document.getElementById("btnInsert").addEventListener("click", ()=>{

alert("Inserted");

})

}

</script>

</head>

<body onload="bodyload()">

<div id="container">

<button id="btnInsert">Insert</button>

</div>

</body>

</html>

**6. What are event arguments?**

A. Event argument comprises of "Payload", which is data about element and event sent to subscriber.

JavaScript events support

a) Default Arguments

b) Custom Arguments

- Default event arguments are

**a) this**  : It sends information about the current element

**b) event** : It sends information about the current event

- "this" payload contains details like: id, name, className, value, src, href …

- "event" payload contains details like : clientX, clientY, ctrlKey, altKey, shiftKey …

**Syntax:** function InsertClick(obj, e)

{

obj.name,

obj.value,

obj.id,

e.clientX,

e.clientY,

e.ctrlKey

}

<button onclick="InsertClick(this, event)" id="btn" name="Insert">

- Event can configure custom arguments, it supports any type of args.

a) Primitive

b) Non Primitive

- It also supports spread and rest approach.

**Syntax:**

function Details(...product)

{

}

<button onclick="Details(1, 'TV', [ ], { })">

- You can pass default arguments along with custom args.

**Ex:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function Details(obj, ...product){

var [id, name, stock, cities, rating] = product;

document.write(`ButtonId:${obj.id}<br> Rating=${rating.rate}<br>Cities:${cities}<br>Name=${name}`);

}

</script>

</head>

<body>

<button id="btnDetails" onclick="Details(this, 1, 'TV', true, ['Delhi','Hyd'], {rate:4.5})">Get Details</button>

</body>

</html>

**Note: Event Listener will not support different types of args. It implicitly uses only a**

**reference for "event". Hence only one parameter is allowed in listener, which**

**can access both event and element details.**

**Syntax:**

document.querySelector("button").addEventListener("click", (e)=>{

e.clientX

e.clientY

e.ctrlKey

e.target.id

e.target.name

})

**Ex:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function bodyload(){

document.querySelector("button").addEventListener("click",(e)=>{

document.write(`X Positon: ${e.clientX}<br>Button Id=${e.target.id}`);

})

}

</script>

</head>

<body onload="bodyload()">

<button id="btnDetails" name="Details">Get Details</button>

</body>

</html>

**7. What is event propagation? How to prevent propagation?**

A. It is a mechanism where the child event triggers its parent element events.

Child element simulates the event that triggers parent events.

You can prevent propagation of child event by using the event argument method.

"stopPropagation()"

**Syntax:** function ChildEvent(e)

{

e.stopPropagation();

}

Event propagation is a mechanism, which is often referred as "Event Bubbling".

**Ex:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

nav {

background-color : black;

color : white;

padding : 20px;

}

</style>

<script>

function NavClick(){

alert("Navbar Clicked");

}

function SearchClick(e){

e.stopPropagation();

alert("Search Button Click");

}

</script>

</head>

<body>

<nav onclick="NavClick()">

<h2>Amazon</h2>

<button onclick="SearchClick(event)">Search</button>

</nav>

</body>

</html>

**6. How to disable the default generic functionality of elements?**

A. By using event argument method "preventDefault()".

**Syntax:**

function FormSubmit(e)

{

e.preventDefault();

}

<form onsubmit="FormSubmit(event)">

<button> Submit </button>

</form>

**Synthetic Events**

- "Events" in JavaScript belong to browser "window" object.

- React can't use the browser events directly.

- React have "Synthetic Events" library, which provides events for "Virtual DOM" that map to actual DOM events.

Actual DOM Virtual DOM

onclick onClick

onchange onChange

onblur onBlur

etc...

01-07-2024

**Event Binding in React**

- React uses synthetic events that map to browser events

onClick [Synthetic Event] => onclick [Browser Event]

- React uses event handler that points towards a subscriber function.

function InsertClick()

{

}

<button onClick={InsertClick}> Insert </button>

- React event handler have default payload sent to subscriber as Event Argument.

- React Subscriber function can access and use the event argument.

- React can't send data using "this" keyword. It uses event to send both element and event details.

- React passes the default event argument implicitly.

**Syntax:**

function InsertClick(e)

{

e.clientX;

e.clientY;

e.shiftKey;

e.target.value;

e.target.id;

}

<button onClick={InsertClick}> Insert </button>

**Ex:event-demo.jsx**

export function EventDemo(){

function InsertClick(e){

console.log(`X=${e.clientX}\nCtrl Key=${e.ctrlKey}\nButtonId=${e.target.id}`);

}

return(

<div className="container-fluid">

<h2>Event Demo</h2>

<button id="btnInsert" onClick={InsertClick}>Insert</button>

</div>

)

}

- React will not allow to pass custom arguments directly using "delegate" signature.

[function pointer signature]

onClick={InsertClick} => Delegate

- Custom arguments can be passed by using a function return in Event handler.

onClick={()=> InsertClick(args) }

args => can be any type and any count

- IF you configure a function return then it discards the default event argument.

- You have to configure default event argument in function return.

onClick={ (e) => InsertClick(e, args…) }

function InsertClick(e, ...args)

{

}

**Ex:event-demo.jsx**

export function EventDemo(){

function InsertClick(e,...msg){

console.log(`X=${e.clientX}\n${msg}`);

}

return(

<div className="container-fluid">

<h2>Event Demo</h2>

<button id="btnInsert" onClick={(e)=> InsertClick( e,1, 'TV', true)}>Insert</button>

</div>

)

}

**Ex:**

export function EventDemo(){

function InsertClick(e, ...args){

console.log(`Button Id=${e.target.id}\n${args}`)

}

return(

<div className="container-fluid">

<h2>Event Demo</h2>

<button id="btnInsert" onClick={(e)=> InsertClick(e, 'TV', {Rate:4.2}, ['Delhi', 'Hyd']) }>Insert</button>

</div>

)

}

- Propagation of event can be stopped just like in JavaScript.

e.stopPropagation()

- You can prevent default generic events using the function

e.preventDefault()

**Ex:event-demo.jsx**

export function EventDemo(){

function NavBarClick(){

alert('Navbar Clicked');

}

function SearchClick(e){

e.stopPropagation();

alert('Search Clicked');

}

function handleSubmit(e){

e.preventDefault();

alert('Submitted..');

}

return(

<div className="container-fluid">

<h2>Event Demo</h2>

<nav onClick={NavBarClick} className="p-3 bg-dark text-white">

<h2>Navbar</h2>

<button onClick={SearchClick} className="bi bi-search btn btn-primary"></button>

</nav>

<section className="mt-4">

<form onSubmit={handleSubmit}>

User Name : <input type="text" name="UserName" /> <button type="submit">Submit</button>

</form>

</section>

</div>

)

}

**- React Synthetic Event Types**

**1. Mouse Events 2. Keyboard Events 3. Button Events 4. Element State Events**

**5. Clipboard Events 6. Timer Events 7. Touch Events etc...**

**Mouse Events**

- onMouseover - onMouseout

- onMouseup - onMousedown

- onMousemove

**Ex: mouse-demo.jsx**

import axios from "axios";

import { useEffect, useState } from "react";

import './mouse-demo.css';

export function MouseDemo(){

const [images, setImages] = useState([{img\_src:''}]);

const [previewSource, setPreviewSource] = useState('m1.jpg');

function LoadImages(){

axios.get('mobiles.json') .then(response=> { setImages(response.data); } )

}

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

function GeneratePreview(e){

setPreviewSource(e.target.src);

}

return(

<div className="container-fluid">

<div className="mt-4 row">

<div className="col-2">

{

images.map(image=>

<img onMouseOver={GeneratePreview} key={image.img\_src} className=

"d-block my-3" width="50%" src={image.img\_src} />

)

}

</div>

<div className="col-10">

<img width="350" src={previewSource} className="mt-4" height="450"/>

</div>

</div>

</div>

)

}

**mouse-demo.css**

.col-2 img:hover { border:2px solid blue; }

**public/mobiles.json**

[

{ "img\_src": "m1.jpg" },

{ "img\_src": "m2.jpg" },

{ "img\_src": "m3.jpg" },

{ "img\_src": "m4.jpg" }

]

02-07-2024

**Style Binding in React**

- Style binding is a technique where the styles are dynamically configured for element.

- You can apply CSS styles to any element dynamically.

Syntax: JavaScript

<h1> Welcome </h1>

document.querySelector("h1").style.textAlign = "center";

text-align textAlign

background-color backgroundColor

font-size fontSize

color color

- JSX element can bind styles dynamically using binding expression.

<h1 style={ { textAlign:'center', fontSize:'40px' } }>

<h1 style="text-align:center; font-size:40px"> // invalid in react

- Style object contains a key and value pair.

- Key is style property defined in camel case.

- Value is defined in string format.

{ backgroundColor: 'red', fontSize: '20px' }

- Style Binding requires a style object dynamically.

Syntax:

const [obj, setObj] = useState({color:'red', textAlign:'center'});

<h1 style={obj}> Welcome </h1>

setObj( {color:'green', textAlign:'right'} );

**Ex:style-binding.jsx**

import { useState } from "react"

export function StyleBinding(){

const [textColor, setTextColor] = useState({color:'black'});

function GetColor(color){

setTextColor({color:color})

}

function ResetColor(){

setTextColor({color:'black'});

}

return(

<div className="container-fluid">

<h2>Style Binding</h2>

<div className="d-flex">

<div onMouseOver={()=> GetColor('red') } onMouseOut={ResetColor} style={{backgroundColor:'red', height:'30px', width:'50px'}}></div>

<div onMouseOver={()=> GetColor('green')} onMouseOut={ResetColor} style={{backgroundColor:'green', height:'30px', width:'50px'}}></div>

<div onMouseOver={()=> GetColor('blue') } onMouseOut={ResetColor} style={{backgroundColor:'blue', height:'30px', width:'50px'}}></div>

</div>

<div>

<h1 style={textColor}>Sample Text</h1>

</div>

</div>

)

}

**Ex: Mouse Move style-binding.jsx**

import { useState } from "react"

export function StyleBinding(){

const [styleObj, setStyleObj] = useState({position:'', top:'', left:''});

function GetPosition(e){

setStyleObj({position:'fixed', top:e.clientY + 'px', left:e.clientX + 'px'});

}

return(

<div onMouseMove={GetPosition} className="container-fluid">

<div style={{height:'1000px'}}>

<p>move mouse pointer to test</p>

</div>

<img style={styleObj} src="flag.gif" width="50" height="50"/>

</div>

)

}

**React Class Binding**

- Class Binding is a technique used to configure CSS classes to elements dynamically.

- JSX can't use "class" attribute to bind CSS class.

- JSX requires "className" property to binding a CSS class.

- Class Name is defined in a string format.

**Syntax:** <h1 className="text-primary bg-danger">

- Multiple classes a separated with blank space.

- You can bind a CSS class dynamically using data binding expression.

**Syntax:**const [style, setStyle] = useState('text-danger bg-warning');

<h1 className={style}> Welcome </h1>

**Ex:class-binding.jsx**

import { useState } from "react"

export function ClassBinding(){

const [theme, setTheme] = useState('border border-1 p-4');

const [btnTheme, setBtnTheme] = useState('btn btn-dark w-100');

function ToggleTheme(e){

if(e.target.checked)

{

setTheme('border border-1 p-4 bg-dark text-white');

setBtnTheme('btn btn-warning w-100');

} else {

setTheme('border border-1 p-4');

setBtnTheme('btn btn-dark w-100');

}

}

return(

<div className="container-fluid">

<div className="d-flex justify-content-center align-items-center" style={{height:'100vh'}}>

<form className={theme}>

<div className="form-switch">

<input onChange={ToggleTheme} className="form-check-input" type="checkbox" /> <label> Dark Theme</label>

</div>

<h3 className="bi bi-person-fill">User Login</h3>

<dl>

<dt>User Name</dt>

<dd><input type="text" className="form-control"/></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control" /></dd>

</dl>

<button className={btnTheme}>Login</button>

</form>

</div>

</div>

)

}

**Ex: animation**

**animation-demo.css**

@keyframes Spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

@keyframes Zoom {

from { transform: scale(0.5); }

to { transform: scale(3); }

}

.image-spin {

animation-name: Spin;

animation-duration: 5s;

animation-iteration-count: infinite;

animation-timing-function: linear;

}

.image-zoom {

animation-name: Zoom;

animation-duration: 5s;

animation-iteration-count: infinite;

animation-timing-function: linear;

}

**animation-demo.jsx**

import { useState } from 'react';

import './animation-demo.css';

export function AnimationDemo()

{

const [aniStyle, setAniStyle] = useState('');

function SpinClick(){

setAniStyle('image-spin');

}

function ZoomClick(){

setAniStyle('image-zoom');

}

return(

<div className="container-fluid">

<div className="d-flex justify-content-center bg-dark align-items-center" style={{height:'100vh'}}>

<div className='text-center'>

<img src="logo192.png" className={aniStyle} />

<div className='mt-3'>

<button onClick={SpinClick} className='btn me-2 btn-primary'>Spin</button>

<button onClick={ZoomClick} className='btn btn-warning'>Zoom</button>

</div>

</div>

</div>

</div>

)

}

03-07-2024

Style Binding

Class Binding

Mouse Events

**Keyboard Events**

**onKeyUp onKeyDown onKeyPress**

- KeyUp & KeyDown are the events used with elements when you are verifying input chars.

- KeyPress is used with elements when you are verifying the ASCII code of input chars.

- KeyPress fire up when you finish current key and start keying in another key.

- The commonly used keyboard event arguments

keyCode It returns actual ASCII code of key

charCode It returns code of char according to UTF.

which It is similar to keycode but compatible with various layouts.

shiftKey ]

ctrlKey ] return true when used

altKey ]

**Ex:key-demo.jsx**

import axios from "axios";

import { useEffect, useState } from "react"

export function KeyDemo(){

const [users, setUsers] = useState([{UserId:''}]);

const [userError, setUserError] = useState('');

const [pwdError, setPwdError] = useState('');

const [errorClass, setErrorClass] = useState('');

const [pwdStrength, setPwdStrength] = useState({width:''});

const [bgClass, setBgClass] = useState('');

const [codeError, setCodeError] = useState('d-none');

useEffect( ()=>{axios.get('users.json').then(response=>{ setUsers(response.data);} ) }, [])

function VerifyUserId(e){

var userid = e.target.value;

for(var user of users)

{

if(user.UserId===userid){

setUserError('UserId Taken - Try Another');

setErrorClass('text-danger');

break;

} else {

setUserError('UserId Available');

setErrorClass('text-success');

}

}

}

function VerifyPassword(e)

{

if(e.target.value.match(/(?=.\*[A-Z])\w{4,15}/))

{

setPwdError('Strong Password');

setErrorClass('text-success');

setPwdStrength({width:'100%'});

setBgClass('bg-success');

} else {

if(e.target.value.length<4){

setPwdError('Poor Password');

setErrorClass('text-danger');

setPwdStrength({width:'30%'});

setBgClass('bg-danger');

} else {

setPwdError('Weak Password');

setErrorClass('text-warning');

setPwdStrength({width:'60%'});

setBgClass('bg-warning');

}

}

}

function VerifyCode(e){

if(e.which>=65 && e.which<=90)

{ setCodeError('d-block'); }

else { setCodeError('d-none'); }

}

return(

<div className="container-fluid">

<h2>Register User</h2>

<dl className="w-25">

<dt>User Id</dt>

<dd><input type="text" className="form-control" onKeyUp={VerifyUserId} /></dd>

<dd className={errorClass}>{userError}</dd>

<dt>Password</dt>

<dd><input type="password" className="form-control" onKeyUp={VerifyPassword}/></dd>

<dd>

<div className="progress">

<div style={pwdStrength} className={`progress-bar progress-bar-animated progress-bar-striped ${bgClass}`}>

{pwdError}

</div>

</div>

</dd>

<dd className={errorClass}>{pwdError}</dd>

<dt>Verify Code</dt>

<dd><input type="text" onKeyPress={VerifyCode} className="form-control" placeholder="Lowercase Code Only" /></dd>

<dd className={codeError}>

<div className="text-warning">

<span className="bi bi-exclamation-triangle-fill"></span> Warning Caps ON

</div>

</dd>

</dl>

</div>

)

}

04-07-2024

**Element State Events**

onChange Actions when value changes.

onBlur Actions when element lost focus.

onFocus Actions when element gets focus.

**Ex:state-event-demo.jsx**

import { useState } from "react"

export function StateEventDemo()

{

const [msg, setMsg] = useState('');

const [code, setCode] = useState('');

function handleFocus(){ setMsg('IFSC CODE : SBIN0000HY, HDF0000TN'); }

function handleBlur(){ setMsg(''); }

function handleKeyUp(e){ setCode(e.target.value.toUpperCase()); }

return(

<div className="container-fluid">

<h2>State Events</h2>

<dl>

<dt>Bank IFSC Code</dt>

<dd><input type="text" onChange={handleKeyUp} value={code} onBlur={handleBlur} onFocus={handleFocus} /></dd>

<dd className="text-danger">{msg}</dd>

</dl>

</div>

)

}

**Button Events**

onClick Actions on single click

onDoubleClick Actions on double click

onContextMenu Actions on right click

onSelectStart Actions when you hold down button and drag. [N/A]

**Note: To disable any element you have to map to actual DOM event that returns false.**

**Syntax:**

function handleContext()

{

document.oncontextmenu = function() { return false; }

}

**Ex:state-event-demo.jsx**

import { useState } from "react"

export function StateEventDemo()

{

function handleDblClick(){ window.open('iphone-green.jpg','Iphone','width=300 height=400'); }

function handleContextMenu(){

document.oncontextmenu = function(){

alert('Right Click Not Allowed');

return false;

}

}

return(

<div className="container-fluid" onContextMenu={handleContextMenu}>

<h2>Button Events</h2>

<img src="iphone-green.jpg" onDoubleClick={handleDblClick} width="100" height="100" />

<p>Double Click to View Large - Right Click is disabled on this page</p>

</div>

)

}

**Clipboard Events**

onCut Actions on cut

onCopy Actions on copy

onPaste Actions on paste

**Syntax:**

<input type="text" onPaste={handlePaste} />

function handlePaste()

{

document.paste = function(){ return false; }

}

**Form Events**

onSubmit Actions on form submit

onReset Actions on form reset

**Syntax:**

<form onSubmit={handleSubmit} onReset={handleReset}>

<button type="submit"> Submit </button>

<button type="reset"> Reset </button>

</form>

**Timer Events**

setInterval()

clearInterval()

setTimeout()

clearTimeout()

**FAQ: What is debounce in JavaScript?**

Ans: Debounce is a electronic term, which is used to control the actions in memory.

It is a technique implemented to keep the task inactive in memory for specific duration. [thread sleep]

**FAQ: How to implement debounce?**

Ans: by using "setTimeout()"

**Syntax:**

setTimeout(function(){ }, interval);

**Ex:**

import { useEffect, useState } from "react"

export function StateEventDemo()

{

const [msg, setMsg] = useState('');

function Msg1(){ setMsg('Hello !'); }

function Msg2(){ setMsg('How are you?'); }

function Msg3(){ setMsg('Welcome to React !'); }

function handleShowClick()

{

setTimeout(Msg1, 3000);

setTimeout(Msg2,6000);

setTimeout(Msg3,10000);

}

return(

<div className="container-fluid">

<button onClick={handleShowClick}>Show Msgs</button>

<p className="mt-4 h1">{msg}</p>

</div>

)

}

05-07-2024

**Timer Events**

setTimeout() : It is used to control the task bounce mechanism in process.

It is to configure debounce for tasks in application.

**FAQ: What is debounce?**

Ans: It is a technique where task can be kept inactive for a specific duration of time, and

invoke when ever required.

**Syntax:** setTimeout(function(){ }, interval);

**clearTimeout()** : It is used to clear the task from memory before it is released into process.

It requires a memory reference for task in memory.

**Syntax:** refName = setTimeout(function(){ }, interval);

clearTimeout(refName);

**React useRef() hook**

- React 18 provides useRef() hook, which is used to create a memory reference.

- The memory reference created by useRef() is designed for the tasks performed by internal process.

- It can handle data or logic good for process and not for rendering output.

**Syntax:** let thread = useRef(null);

thread.current = value/function;

**Ex:** let thread = useRef(null);

thread.current = setTimeout(function(){ }, interval);

clearTimeout(thread.current);

**Ex:state-event-demo.jsx**

import { useEffect, useRef, useState } from "react"

export function StateEventDemo()

{

const [msg, setMsg] = useState('');

let m1 = useRef(null);

function Msg1(){ setMsg('Hello !'); }

function Msg2(){ setMsg('How are you?'); }

function Msg3(){ setMsg('Welcome to React !'); }

function handleShowClick(){

setTimeout(Msg1, 3000);

m1.current = setTimeout(Msg2,6000);

setTimeout(Msg3,10000);

}

function handleCancelClick(){ clearTimeout(m1.current); }

return(

<div className="container-fluid">

<button onClick={handleShowClick}>Show Msgs</button>

<button onClick={handleCancelClick}>Cancel 2nd Message</button>

<p className="mt-4 h1">{msg}</p>

</div>

)

}

**setInterval()** : It can load task into memory and release a copy into process, so that

the task executed repeatedly until removed from memory.

**Syntax:** setInterval(function(){ }, interval);

**clearInterval():**It can remove the task from memory using a reference name.

**Syntax:** clearInterval(refName);

**Ex: interval-demo.jsx**

import { useRef, useState } from "react"

export function IntervalDemo()

{

const [toggleProgress, setToggleProgress] = useState('d-none');

const [toggleImage, setToggleImage] = useState('d-none');

const [toggleBtn, setToggleBtn] = useState('d-block');

const [progressValue, setProgressValue] = useState(1);

let count = useRef(1);

let suspendThread = useRef(null);

function StartProgress(){

count.current = count.current + 1;

setProgressValue(count.current);

if(count.current==100)

{

setToggleProgress('d-none');

setToggleImage('d-block');

}

}

function handleLoadClick(){

setToggleBtn('d-none');

setToggleProgress('d-block');

suspendThread.current = setInterval(StartProgress,100);

}

function handlePauseClick(){ clearInterval(suspendThread.current); }

function handlePlayClick(){ suspendThread.current = setInterval(StartProgress,100); }

return(

<div className="container-fluid">

<div className="d-flex justify-content-center align-items-center" style={{height:'100vh'}}>

<div>

<div className={toggleBtn}>

<button onClick={handleLoadClick} className="btn btn-primary">Load Image</button>

</div>

<div className={toggleProgress}>

<progress value={progressValue} min="1" max="100" style={{width:'350px', height:'30px'}}></progress>

<p className="text-center">

<button onClick={handlePlayClick} className="btn btn-success bi bi-play"></button>

<button onClick={handlePauseClick} className="btn btn-danger bi bi-pause ms-1"></button>

</p>

<p className="text-center">{progressValue}% completed</p>

</div>

<div className={toggleImage}>

<img src="iphone-green.jpg" width="300" height="300" />

</div>

</div>

</div>

</div>

)

}

08-07-2024

**Touch Events**

onTouchStart

onTouchEnd

onTouchMove

**Note:Touch events map to browser touch events that require a console enabled with touch.**

**Touch event args for position are defined in "touches" property.**

**Syntax:** event.touches[0].clientX

event.touches[0].clientY

**Ex:**

import { useState } from "react"

export function TimerDemo(){

const [msg, setMsg] = useState('');

const [styleObj, setStyleObj] = useState({position:'', top:'', left:'', width:'250px'})

function GetDetails(message, e)

{

console.log(e);

setMsg(message);

}

function MoveAd(e){

setStyleObj({

position: 'fixed',

left: e.touches[0].clientX + 'px',

top: e.touches[0].clientY + 'px',

width: '200px'

})

}

return(

<div className="container-fluid">

<aside onTouchMove={MoveAd} style={styleObj} className="border border-2 p-3 border-primary">

<h4>iPhone</h4>

<img src="iphone-green.jpg" width="200" height="200" />

</aside>

<img onTouchStart={(e)=> GetDetails('iPhone Green - 128 GB, 65,000', e)} src="iphone-green.jpg" width="200" height="300" />

<img onTouchStart={(e)=> GetDetails('iPhone Black - 256 GB 79,000', e)} src="iphone-black.jpg" width="200" height="300" />

<div>

<h1>{msg}</h1>

</div>

</div>

)

}

**Summary: Event Binding**

1. Mouse Events 2. Keyboard Events 3. Button Events

4. Clipboard Events 5. Form Events 6. Timer Events

7. Element State Events 8. Touch Events

**Handling API's**

**1. What is Distributed Computing?**

- Distributed computing allows communication between two applications running on two different machines.

- It also allows communication between two different objects running in two different process of same machine.

**2. What are the various distributed computing technologies?**

CORBA : Common Object Request Broken Architecture [14 languages]

RMI : Remote Method Invocation [J2EE]

EJB : Enterprise Java Beans [Java]

DCOM : Distribute Component Object Model [Visual Basic]

Web Service : All technologies C#, Java, PHP, Python etc..

Remoting : .NET

**3. What is the popular distributed technology for web applications?**

A. "Web Service"

**4. What are the issues with Web Service?**

A. - It can run only on Http protocol

- It can be hosted only on web server.

- It provides data in XML format.

**5. What is the solution? What is alternative for Web Service?**

A. API [Application Programming Interface]

**6. What is API?**

A - It is a distributed computing architecture.

- It configures business logic which provides data across various devices and OS services.

- It can run on any protocol.

- It can be hosted on any local server or file system.

- It provides REST with JSON.

**7. What are the communication specifications?**

a) SOAP

b) REST

c) JSON

**SOAP :**

- Service Oriented Architecture Protocol

- Consumer will make an XML request

<Product>

<Category> Electronics </Category>

</Product>

- Provider will give an XML response

<Products>

<Product>

<Name> TV </Name>

<Price> 45000 </Price>

<Category> Electronics </Category>

</Product>

….

….

</Products>

REST:

- Representational State Transfer

- Consumer sends a simple query request.

http://server.com/shop?category=electronics

- Provider sends XML and JSON as response

[

{

Name:TV,

Price: 40000,

Category: Electronics

},

….

]

**JSON**

- JavaScript Object Notation

- Consumer sends a JSON request

- Provider sends a JSON response

09-07-2024

**api.nasa.gov - NASA API Portal**

**Ex:nasa-api.jsx**

import { useEffect, useState } from "react";

import axios from "axios";

export function NasaAPI()

{

const [marsObj, setMarsObj] = useState({});

useEffect(()=>{

axios.get('https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos?sol=1000&api\_key=DEMO\_KEY')

.then(response=>{

setMarsObj(response.data);

})

},[])

return(

<div className="container-fluid">

<h2>Mars Rover Photos</h2>

<section className="d-flex flex-wrap">

{

marsObj.photos.map(item=>

<div key={item.id} className="card p-2 m-2" style={{width:'200px'}}>

<a href={item.img\_src} target="\_blank"><img className="card-img-top" height="150" src={item.img\_src} /></a>

<div className="card-header">

<div className="fs-4">{item.id}</div>

</div>

<div className="card-body">

<dl>

<dt>Camera</dt>

<dd className="bi bi-camera"> {item.camera.full\_name} </dd>

<dt>Rover</dt>

<dd> {item.rover.name} </dd>

</dl>

</div>

</div>

)

}

</section>

</div>

)

}

https://fakestoreapi.com

- It provides REST full services for ERP. [Online Shopping]

Request Route Description

------------------------------------------------------------------------------------------------------------------

GET /products It returns array of all products [ { }, { } ]

GET /products/1 It returns specific product by id { }

GET /products/categories It returns a string array of all categories

GET /products/category/electronics It returns all products of specific

category. [ { }, { } ]

**Ex:fakestore.jsx**

import { useEffect, useState } from "react";

import axios from "axios";

export function Fakestore(){

const [categories, setCategories] = useState([]);

const [products, setProducts] = useState([{id:0, title:'', category:'', description:'', image:'', price:0, rating:{rate:0, count:0}}]);

function LoadCategories(){

axios.get('https://fakestoreapi.com/products/categories')

.then(response=> {

response.data.unshift('all');

setCategories(response.data);

})

}

function LoadProducts(url){

axios.get(url)

.then(response=>{

setProducts(response.data);

})

}

function handleCategoryChange(e){

if(e.target.value==="all"){

LoadProducts('https://fakestoreapi.com/products');

} else {

LoadProducts(`https://fakestoreapi.com/products/category/${e.target.value}`);

}

}

function NavbarButtonClick(categoryName)

{

if(categoryName==="all"){

LoadProducts('https://fakestoreapi.com/products');

} else {

LoadProducts(`https://fakestoreapi.com/products/category/${categoryName}`);

}

}

useEffect(()=>{

LoadCategories();

LoadProducts('https://fakestoreapi.com/products');

},[])

return(

<div className="container-fluid">

<header className="d-flex text-white justify-content-between p-2 bg-dark">

<div>

<span className="h4">Fakestore.</span>

</div>

<div className="fs-5">

<span className="px-2"> <button onClick={()=>NavbarButtonClick('all')} className="btn btn-light">Home</button> </span>

<span className="px-2">Electronics</span>

<span className="px-2">Jewelery</span>

<span className="px-2">Men's Fashion</span>

<span className="px-2">Women's Fashion</span>

</div>

<div>

<button className="bi bi-cart4 btn btn-warning position-relative me-4 "> <span className="badge rounded position-absolute rounded-circle bg-danger">0</span> </button>

</div>

</header>

<section className="mt-4 row">

<nav className="col-2">

<label className="form-label fw-bold">Select Category</label>

<div>

<select className="form-select" onChange={handleCategoryChange}>

{

categories.map(category=> <option key={category} value={category}> {category.toUpperCase()} </option>)

}

</select>

</div>

</nav>

<main className="col-10 d-flex flex-wrap overflow-auto" style={{height:'500px'}}>

{

products.map(product=>

<div key={product.id} className="card p-2 m-2" style={{width:'200px'}}>

<img src={product.image} className="card-img-to" height="120px" />

<div className="card-header" style={{height:'130px'}}>

<div>

{product.title}

</div>

</div>

<div className="card-body">

<dl>

<dt>Price</dt>

<dd>{product.price.toLocaleString('en-us', {style:'currency', currency:'USD'})}</dd>

<dt>Rating</dt>

<dd>

<span className="badge bg-success p-1 rounded"> {product.rating.rate} <span className="bi bi-star-fill"></span> </span> <span className="text-secondary fw-bold">{product.rating.count} ratings</span>

</dd>

</dl>

</div>

<div className="card-footer">

<button className="btn btn-warning bi bi-cart4 w-100"> Add to Cart </button>

</div>

</div>

)

}

</main>

</section>

</div>

)

}

10-07-2024

Ex: fakestore.jsx

import { useEffect, useState } from "react";

import axios from "axios";

export function Fakestore(){

const [categories, setCategories] = useState([]);

const [products, setProducts] = useState([{id:0, title:'', category:'', description:'', image:'', price:0, rating:{rate:0, count:0}}]);

const [cartCount, setCartCount] = useState(0);

const [cartItems, setCartItems] = useState([]);

function LoadCategories(){

axios.get('https://fakestoreapi.com/products/categories')

.then(response=> {

response.data.unshift('all');

setCategories(response.data);

})

}

function LoadProducts(url){

axios.get(url)

.then(response=>{

setProducts(response.data);

})

}

function handleCategoryChange(e){

if(e.target.value==="all"){

LoadProducts('https://fakestoreapi.com/products');

} else {

LoadProducts(`https://fakestoreapi.com/products/category/${e.target.value}`);

}

}

function NavbarButtonClick(categoryName)

{

if(categoryName==="all"){ LoadProducts('https://fakestoreapi.com/products');

} else {

LoadProducts(`https://fakestoreapi.com/products/category/${categoryName}`);

}

}

function handleAddClick(product){

cartItems.push(product);

alert(`${product.title}\nAdded to Cart`);

setCartCount(cartItems.length);

}

useEffect(()=>{

LoadCategories();

LoadProducts('https://fakestoreapi.com/products');

},[])

return(

<div className="container-fluid">

<header className="d-flex text-white justify-content-between p-2 bg-dark">

<div>

<span className="h4">Fakestore.</span>

</div>

<div className="fs-5">

<span className="px-2"> <button onClick={()=>NavbarButtonClick('all')} className="btn btn-light">Home</button> </span>

<span className="px-2">Electronics</span>

<span className="px-2">Jewelery</span>

<span className="px-2">Men's Fashion</span>

<span className="px-2">Women's Fashion</span>

</div>

<div>

<button data-bs-target="#cart" data-bs-toggle="modal" className="bi bi-cart4 btn btn-warning position-relative me-4 "> <span className="badge rounded position-absolute rounded-circle bg-danger"> {cartCount} </span> </button>

<div className="modal fade" id="cart">

<div className="modal-dialog">

<div className="modal-content">

<div className="modal-header">

<h2 className="text-primary">Your Cart Items</h2>

</div>

<div className="modal-body">

<table className="table table-hover">

<thead>

<tr>

<th>Title</th>

<th>Preview</th>

<th>Price</th>

</tr>

</thead>

<tbody>

{

cartItems.map(item=>

<tr key={item.id}>

<td>{item.title}</td>

<td>

<img width="50" height="50" src={item.image}/>

</td>

<td>

{item.price}

</td>

</tr>

)

}

</tbody>

</table>

</div>

<div className="modal-footer">

<button className="btn btn-primary" data-bs-dismiss="modal">OK</button>

</div>

</div>

</div>

</div>

</div>

</header>

<section className="mt-4 row">

<nav className="col-2">

<label className="form-label fw-bold">Select Category</label>

<div>

<select className="form-select" onChange={handleCategoryChange}>

{

categories.map(category=> <option key={category} value={category}> {category.toUpperCase()} </option>)

}

</select>

</div>

</nav>

<main className="col-10 d-flex flex-wrap overflow-auto" style={{height:'500px'}}>

{

products.map(product=>

<div key={product.id} className="card p-2 m-2" style={{width:'200px'}}>

<img src={product.image} className="card-img-to" height="120px" />

<div className="card-header" style={{height:'130px'}}>

<div>

{product.title}

</div>

</div>

<div className="card-body">

<dl>

<dt>Price</dt>

<dd>{product.price.toLocaleString('en-us', {style:'currency', currency:'USD'})}</dd>

<dt>Rating</dt>

<dd>

<span className="badge bg-success p-1 rounded"> {product.rating.rate} <span className="bi bi-star-fill"></span> </span> <span className="text-secondary fw-bold">{product.rating.count} ratings</span>

</dd>

</dl>

</div>

<div className="card-footer">

<button onClick={()=> handleAddClick(product)} className="btn btn-warning bi bi-cart4 w-100"> Add to Cart </button>

</div>

</div>

)

}

</main>

</section>

</div>

)

}

**React Forms**

- Form provides an UI from where user can interact with the data.

- It enables CRUD operations.

C - Create

R - Read

U - Update

D - Delete

- React have "One Way Binding". Hence you have to configure "onChange" for every element in form to collect the data and submit to API.

- Every element change requires a function that collects the data and store in a state.

**Ex:form-demo.jsx**

import { useState } from "react"

export function FormDemo()

{

const [product, setProduct] = useState({Name:'', Price:0, City:'', Stock:false});

const [newProduct, setNewProduct] = useState({Name:'', Price:0, City:'', Stock:false});

function handleNameChange(e){

setProduct({

Name: e.target.value,

Price: product.Price,

City: product.City,

Stock: product.Stock

})

}

function handlePriceChange(e){

setProduct({

Name: product.Name,

Price: e.target.value,

City: product.City,

Stock: product.Stock

})

}

function handleCityChange(e){

setProduct({

Name: product.Name,

Price: product.Price,

City: e.target.value,

Stock: product.Stock

})

}

function handleStockChange(e){

setProduct({

Name: product.Name,

Price: product.Price,

City: product.City,

Stock: e.target.checked

})

}

function handleSubmitClick(e){

setNewProduct(product);

}

return(

<div className="container-fluid">

<div className="row">

<div className="col-3">

<h3>Register Product</h3>

<form>

<dl>

<dt>Name</dt>

<dd><input type="text" onChange={handleNameChange} /></dd>

<dt>Price</dt>

<dd><input type="number" onChange={handlePriceChange} /></dd>

<dt>Shipped To</dt>

<dd>

<select onChange={handleCityChange}>

<option>Select City</option>

<option>Delhi</option>

<option>Hyd</option>

</select>

</dd>

<dt>Stock</dt>

<dd className="form-switch">

<input type="checkbox" onChange={handleStockChange} /> <label>Available</label>

</dd>

</dl>

<button onClick={handleSubmitClick} type="button">Submit</button>

</form>

</div>

<div className="col-9">

<h3>Product Details</h3>

<dl>

<dt>Name</dt>

<dd>{newProduct.Name}</dd>

<dt>Price</dt>

<dd>{newProduct.Price}</dd>

<dt>Stock</dt>

<dd>{(newProduct.Stock==true)?'Available':'Out of Stock'}</dd>

<dt>City</dt>

<dd>{newProduct.City}</dd>

</dl>

</div>

</div>

</div>

)

}

**- React provides support for various 3rd party form libraries that simplify the form interactions.**

a) Formik

b) React Hook Form [npmjs.com]

c) Telerik Form

etc...

**React Formik Library**

- It provides built-in functions and components to handle forms in React.

- It simplifies the form interactions.

- It simplifies the form design.

- It can improve the performance of form.

- It implicitly implements AJAX.

Step-1: Install Formik library for project

> npm install formik --save

Step-2: Import "useFormik()" from "formik" module

import { useFormik } from "formik"

Step-3: Configure form by using "useFormik()" hook

const formik = useFormik({

initialValues: { }, // values that a form have to handle

onSubmit: function(){ }, // defines the actions to perform on submit

validation: function(){ }, // uses function to validate the form values

validationSchema: { }, // uses a schema to validation values

....

})

Step-4: Bind formik with your form elements

<input type="text" onChange={formik.handleChange}>

<form onSubmit={formik.handleSubmit}>

**Ex:formik-demo.jsx**

import { useFormik } from "formik";

export function FormikDemo(){

const formik = useFormik({

initialValues: { Name: '', Price: 0, City: '', Stock: false },

onSubmit: (values) => { console.log(values); }

})

return(

<div className="container-fluid">

<h3>Register Product</h3>

<form onSubmit={formik.handleSubmit}>

<dl>

<dt>Name</dt>

<dd><input type="text" onChange={formik.handleChange} name="Name" /></dd>

<dt>Price</dt>

<dd><input type="number" onChange={formik.handleChange} name="Price" /></dd>

<dt>City</dt>

<dd>

<select name="City" onChange={formik.handleChange}>

<option>Select City</option>

<option>Delhi</option>

<option>Hyd</option>

</select>

</dd>

<dt>Stock</dt>

<dd>

<input type="checkbox" onChange={formik.handleChange} name="Stock" /> <label>Available</label>

</dd>

</dl>

<button type="submit">Submit</button>

</form> </div> )

}

11-07-2024

React Form

Formik Library

Rules:

- Every form element must have a "name" attribute.

- Formik initialValues must match with the name defined for element.

initialValues : { Price:0 }

<input type="text" name="Price">

- Name is binded with initial values using a technique known as "Model Binding".

- Every element must have onChange event mapping to "formik.handleChange".

- onChange is the only event that handle two way binding.

- Formik can use the following events

a) handleChange

b) handleBlur

c) handleSubmit

- Form can be submitted only with "onSubmit" configured for <form> element.

- Submit action triggers only with button type submit.

Form Validation

- Validation is the process of verifying user input.

- Validation is required to ensure that contradictory and unauthorized data is not get stored into the database.

- User interface in the front-end technologies can be configured with various types of validation techniques.

HTML & CSS Validation:

- HTML 5 provides various validation attributes for elements

a) minlength

b) required

c) min

d) max

e) email | url | number [types]

f) pattern

Syntax:

<input type="text" name="Mobile" required pattern="\+91\d{10}">

- CSS provides various validation classes to configure styles for different validations

:required

:optional

:in-range

:out-of-range

:valid

:invalid

:checked ]

:disabled ] state

:read-only ]

:focus ]

:hover ]

:active ]

Ex:

1. formik-demo.css

input:optional+span {

display: none;

}

input:required+span {

display: inline;

color:red;

}

input:required {

border:1px solid red;

}

input:valid+span {

display: none;

}

input:invalid~div {

display: block;

color:red;

}

input:valid~div {

display: none;

}

input:valid {

border:1px solid black;

}

.terms:checked~button {

display: block;

}

.terms~button {

display: none;

}

.terms+label {

color:red;

}

.terms:checked+label {

color:green;

}

#Price:focus~div {

display: block;

}

2. formik-demo.jsx

import { useFormik } from "formik";

import './formik-demo.css';

export function FormikDemo(){

const formik = useFormik({

initialValues: {

Name: '',

Price: 0,

City: '',

Stock: false,

Vendor: ''

},

onSubmit: (values) => {

console.log(values);

}

})

return(

<div className="container-fluid">

<h3>Register Product</h3>

<form onSubmit={formik.handleSubmit}>

<dl>

<dt>Name</dt>

<dd><input type="text" required onChange={formik.handleChange} name="Name" />

<span>\*</span> <div>Name Required</div> </dd>

<dt>Price</dt>

<dd><input type="number" id="Price" onChange={formik.handleChange} name="Price" />

<span>\*</span> <div className="text-warning bi bi-exclamation-triangle"> Price 5000 to 10000 only </div> </dd>

<dt>City</dt>

<dd>

<select name="City" onChange={formik.handleChange}>

<option>Select City</option>

<option>Delhi</option>

<option>Hyd</option>

</select>

</dd>

<dt>Stock</dt>

<dd>

<input type="checkbox" onChange={formik.handleChange} name="Stock" /> <label>Available</label>

</dd>

<dt>Vendor Contact</dt>

<dd><input type="text" required pattern="\+91\d{10}" placeholder="+91 and 10 digits"

onChange={formik.handleChange}name="Vendor"/>

<span>\*</span> <div>Invalid Mobile +91 10 digits</div> </dd>

<dt>Terms of Service</dt>

<dd>

<textarea disabled rows="4" cols="40" value="Lorem ipsum dolor sit amet consectetur adipisicing elit. Magnam alias,

recusandae facere error exercitationem incidunt placeat? Nostrum similique voluptas dolor officia

deserunt nobis ad corporis. Ut veniam perspiciatis vitae dicta."> </textarea>

<br/>

<input type="checkbox" className="terms"/> <label>I Accept</label>

<br /><br/>

<button type="submit">Submit</button>

</dd>

</dl>

</form>

</div>

)

}

Formik Validation Technique:

- Formik provides two types of validation techniques

a) validate

b) validateSchema

- "Validate" uses a function that validates the input values and returns an error object.

- "validateSchema" is uses a built-in library that provides schema [structure] for validation.

Syntax:

function ValidateForm(formData)

{

var errors = { Name: ' ' , Price: ' ' }

// logic for validating fields

return errors;

}

const formik = useFormik({

initialValues: { },

validate: ValidateForm,

onSubmit: (values)=>{ }

})

<p> { formik.errors.Name } </p>

Ex: formik-demo.jsx

import { useFormik } from "formik";

import './formik-demo.css';

export function FormikDemo(){

function ValidateForm(FormData){

var errors = { Name:'', Price:'', City:'', Stock:'', Vendor:'' }

if(FormData.Name.length==0){

errors.Name = "Name Required";

} else {

if(FormData.Name.length<4){

errors.Name = "Name min 4 chars";

} else {

errors.Name = "";

}

}

if(FormData.City=="-1"){

errors.City = "Please select a city";

} else {

errors.City = "";

}

return errors;

}

const formik = useFormik({

initialValues: {

Name: '',

Price: 0,

City: '',

Stock: false,

Vendor: ''

},

validate: ValidateForm,

onSubmit: (values) => {

console.log(values);

}

})

return(

<div className="container-fluid">

<h3>Register Product</h3>

<form onSubmit={formik.handleSubmit}>

<dl>

<dt>Name</dt>

<dd><input type="text" onChange={formik.handleChange} name="Name" /><span>\*</span> </dd>

<dd className="text-danger">{formik.errors.Name}</dd>

<dt>Price</dt>

<dd><input type="number" id="Price" onChange={formik.handleChange} name="Price" /><span>\*</span> <div className="text-warning bi bi-exclamation-triangle"> Price 5000 to 10000 only </div> </dd>

<dt>City</dt>

<dd>

<select name="City" onChange={formik.handleChange}>

<option value="-1">Select City</option>

<option value="Delhi">Delhi</option>

<option value="Hyd">Hyd</option>

</select>

</dd>

<dd className="text-danger">{formik.errors.City}</dd>

<dt>Stock</dt>

<dd>

<input type="checkbox" onChange={formik.handleChange} name="Stock" /> <label>Available</label>

</dd>

<dt>Vendor Contact</dt>

<dd><input type="text" required pattern="\+91\d{10}" placeholder="+91 and 10 digits" onChange={formik.handleChange} name="Vendor"/><span>\*</span> <div>Invalid Mobile +91 10 digits</div> </dd>

<dt>Terms of Service</dt>

<dd>

<textarea disabled rows="4" cols="40" value="Lorem ipsum dolor sit amet consectetur adipisicing elit. Magnam alias, recusandae facere error exercitationem incidunt placeat? Nostrum similique voluptas dolor officia deserunt nobis ad corporis. Ut veniam perspiciatis vitae dicta."> </textarea>

<br/>

<input type="checkbox" className="terms"/> <label>I Accept</label>

<br /><br/>

<button type="submit">Submit</button>

</dd>

</dl>

</form>

</div>

)

}

12-07-2024

- HTML Validation Attributes & CSS Validation State Classes

- Formik Validate Function

FAQ: How to disable the default HTML validations in a Form container?

Ans: By using "novalidate" attribute for "form".

Syntax:

<form novalidate>

</form>

Formik Validate Function

1. Create a function that returns error object

Syntax:

function ValidateForm(formCollection)

{

var errors = { Field:" " };

return errors;

}

2. Write custom functions to validate field value from form collection.

Syntax:

if(formCollection.UserName.length===0)

{

errors.UserName = "Required";

}

3. Configure validation function for formik form.

Syntax:

const formik = useFormik({

initialValues: { },

validate: ValidateForm => error function

})

4. Display error message using "formik.errors"

{ formik.errors.FieldName }

Ex: validation-demo.jsx

import { useFormik } from "formik"

export function ValidationDemo(){

function ValidateForm(formCollection)

{

var errors = {UserName:'', Age:'', Mobile:''};

if(formCollection.UserName.length===0){

errors.UserName = "User Name Required";

} else {

if(formCollection.UserName.length<4){

errors.UserName = "Name too short";

} else {

errors.UserName = "";

}

}

if(isNaN(formCollection.Age)){

errors.Age = "Age must be a number";

} else {

errors.Age = "";

}

if(formCollection.Mobile.match(/\+91\d{10}/))

{

errors.Mobile = "";

} else {

errors.Mobile = "Invalid Mobile";

}

return errors;

}

const formik = useFormik({

initialValues: {

UserName:'',

Age:0,

Mobile: ''

},

validate: ValidateForm,

onSubmit: (values)=>{

console.log(values);

}

});

return(

<div className="container-fluid">

<form noValidate onSubmit={formik.handleSubmit}>

<dl>

<dt>User Name</dt>

<dd><input type="text" onChange={formik.handleChange} name="UserName" /></dd>

<dd className="text-danger">{formik.errors.UserName}</dd>

<dt>Age</dt>

<dd><input type="text" onChange={formik.handleChange} name="Age" /></dd>

<dd className="text-danger">{formik.errors.Age}</dd>

<dt>Mobile</dt>

<dd><input type="text" onChange={formik.handleChange} name="Mobile" /></dd>

<dd className="text-danger">{formik.errors.Mobile}</dd>

</dl>

<button>Submit</button>

</form>

</div>

)

}

Yup Library for Validation

- It is a library for handling form validations.

- It provides validation schema. [built-in structure]

- It provides various validation services, which you can inject and implement in a form.

- Services use "Single Ton" design pattern.

[object is created for first request and the same is used across requests].

1. Install Yup library for project

>npm install yup --save

2. Import yup services into your component

import \* as yup from "yup"; // imports all services

import {requried, minLength, pattern} from "yup"; // import only specified

3. Yup provides a validation schema, which you have to configure for formik.

const formik = useFormik({

initialValues: { },

validationSchema: yup.object({

FieldName: yup.service1().service2()...

})

})

- Yup service includes

number()

string()

required()

minLength()

maxLength()

pattern() etc..

4. Error messages are displayed by using "formik.errors" object.

{ formik.errors.FieldName }

Ex:

validation-demo.jsx

import { useFormik } from "formik";

import \* as yup from "yup";

export function ValidationDemo(){

const formik = useFormik({

initialValues: {

UserName:'',

Age:0,

Mobile: ''

},

validationSchema: yup.object({

UserName: yup.string().required("Name Required").min(4, "Name too short"),

Age: yup.number("Age must be a number").required("Age Required").min(15, "Age min 15 required").max(30, "Age max 30 only"),

Mobile: yup.string().matches(/\+91\d{10}/, "Invalid Mobile")

}),

onSubmit: (values)=>{

console.log(values);

}

});

return(

<div className="container-fluid">

<form noValidate onSubmit={formik.handleSubmit}>

<dl>

<dt>User Name</dt>

<dd><input type="text" onChange={formik.handleChange} name="UserName" /></dd>

<dd className="text-danger">{formik.errors.UserName}</dd>

<dt>Age</dt>

<dd><input type="text" onChange={formik.handleChange} name="Age" /></dd>

<dd className="text-danger">{formik.errors.Age}</dd>

<dt>Mobile</dt>

<dd><input type="text" onChange={formik.handleChange} name="Mobile" /></dd>

<dd className="text-danger">{formik.errors.Mobile}</dd>

</dl>

<button>Submit</button>

</form>

</div>

)

}

Note: Formik Validations fire up on various events, which you can configure for every

element.

formik.handleSubmit

formik.handleChange

formik.handleBlur

Syntax:

<input type="text" onChange={formik.handleChange} onBlur="formik.handleBlur" />

<input type="text" name="UserName" {...formik.getFieldProps("UserName") } />

15-07-2024

Formik Validation

a) Validate

b) ValidationSchema

Yup Library for validation

Formik Components

- Formik provides pre-defined components for handling elements and validation.

- Components are built with design, styles and functionality. You can customize according to the requirements.

- Formik Components include

<Formik> It configures a container to handle form

<Form> It configures a form element

<Field> It configures form elements like <input> <select> ..

<ErrorMessage> It configures a container to map with errors and display error messages.

Syntax:

<Formik initialValues={ } validate={ } validationSchema={ } onSubmit={ }>

<Form>

<Field type="text"> </Field>

<ErrorMessage> </ErrorMessage>

</Form>

</Formik>

- It uses model binding technique hence the name defined for field and error message must map with initial values.

Syntax:

<Field type="text" name="Age" />

<ErrorMessage name="Age" />

Ex:

formik-component-demo.jsx

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as yup from "yup";

export function FormikComponentDemo()

{

return(

<div className="container-fluid">

<h3>Register User</h3>

<Formik initialValues={{UserName:'', Age:0, Mobile:''}}

validationSchema={yup.object({UserName:yup.string().required('User Name Required').min(4, 'Name too short'), Age:yup.number().required('Age Required').min(15, "Age min 15").max(30, 'Age max 30'), Mobile:yup.string().matches(/\+91\d{10}/,'Invalid Mobile')})} onSubmit={(values)=> {console.log(values)}} >

<Form>

<dl>

<dt>User Name</dt>

<dd>

<Field type="text" name="UserName" />

</dd>

<dd className="text-danger">

<ErrorMessage name="UserName" />

</dd>

<dt>Age</dt>

<dd>

<Field type="text" name="Age" />

</dd>

<dd className="text-danger">

<ErrorMessage name="Age" />

</dd>

<dt>Mobile</dt>

<dd>

<Field type="text" name="Mobile" />

</dd>

<dd className="text-danger">

<ErrorMessage name="Mobile" />

</dd>

</dl>

<button type="submit">Submit</button>

</Form>

</Formik>

</div>

)

}

- <Form> component can return Form state validation services.

a) isValid

b) dirty

c) untouched

d) touched

e) pristine

Syntax:

<Formik>

{

form => <Form> </Form>

}

</Formik>

{form.dirty}

{form.touched}

{form.isValid}

Note: Form validation usually comprises of 2 states

a) Input State

b) Form State

Input state verifies a field individually.

Form state verifies all fields simultaneous at the same time.

The common form state validation services are:

a) Pristine [ not touched ]

b) Touched [ any one field touched ]

c) Dirty [ any one field modified ]

d) IsValid [ if all fields are valid ]

Form state services return Boolean "true / false".

Syntax:

<button type="submit" disabled={ !form.isValid }> Submit </button>

Ex:

formik-component-demo.jsx

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as yup from "yup";

export function FormikComponentDemo()

{

return(

<div className="container-fluid">

<h3>Register User</h3>

<Formik initialValues={{UserName:'', Age:0, Mobile:''}} validationSchema={yup.object({UserName:yup.string().required('User Name Required').min(4, 'Name too short'), Age:yup.number().required('Age Required').min(15, "Age min 15").max(30, 'Age max 30'), Mobile:yup.string().matches(/\+91\d{10}/,'Invalid Mobile')})} onSubmit={(values)=> {console.log(values)}} >

{

form =>

<Form className={(form.isValid)?'bg-success':'bg-danger'} style={{padding:'10px'}}>

<dl>

<dt>User Name</dt>

<dd>

<Field type="text" name="UserName" />

</dd>

<dd className="text-danger">

<ErrorMessage name="UserName" />

</dd>

<dt>Age</dt>

<dd>

<Field type="text" name="Age" />

</dd>

<dd className="text-danger">

<ErrorMessage name="Age" />

</dd>

<dt>Mobile</dt>

<dd>

<Field type="text" name="Mobile" />

</dd>

<dd className="text-danger">

<ErrorMessage name="Mobile" />

</dd>

</dl>

<button className={(form.isValid)?'d-inline':'d-none'} disabled={!form.isValid} type="submit">Submit</button>

<button type="submit" disabled={!form.dirty} >Save</button>

</Form>

}

</Formik>

</div>

)

}

React Hook Form Library

- It is a 3rd party form library for react.

- It provides various "hooks" to handle forms and validations.

useForm()

useController()

etc..

- It allows to use HTML validations.

https://react-hook-form.com/

16-07-2024

React Hook Form

- PreFormant

- Easy to extend and reuse

- Simple HTML validations

- Various hooks to handle different states

1. Install React Hook Form library for project

>npm install react-hook-form --save

2. To use hook form in any component you have to import "useForm()" hook.

import { useForm } from "react-hook-form";

3. Configure the form using "useForm()" hook

const { register, handleSubmit, formState: {errors} } = useForm();

4. Write a function to handle submit functionality.

const submit = (values) => {

console.log(values);

}

5. You have to bind hook form with form elements using spread operator.

<input type="text" name="UserName" { ...register("UserName", { required:true, minLength:4, maxLength:10, pattern:/regExp/ }) } />

6. Display Errors by verifying the error type.

{ (errors.fieldName?.type==="required")? <span> true </span> : <span> false </span> }

You can use logical operators to handle multiple validation expressions.

7. Bind handleSubmit to <form> container

<form onSubmit={handleSubmit(submit)}>

</form>

Ex:

hook-form-demo.jsx

import { useForm } from "react-hook-form";

export function HookFormDemo(){

const {register, handleSubmit, formState:{errors}} = useForm();

const submit = (values) => {

console.log(values);

}

return(

<div className="container-fluid">

<h2>Register</h2>

<form onSubmit={handleSubmit(submit)}>

<dl>

<dt>UserName</dt>

<dd><input type="text" name="UserName" {...register("UserName", { required:true, minLength:4 })} /></dd>

<dd className="text-danger">

{

((errors.UserName?.type==="required")?<span>Name Required</span>:<span></span> )

&&

((errors.UserName?.type==="minLength")?<span>Name too short..</span>:<span></span>)

}

</dd>

<dt>Mobile</dt>

<dd><input type="text" name="Mobile" {...register("Mobile", { required:true, pattern:/\+91\d{10}/ })} /></dd>

<dd className="text-danger">

{

((errors.Mobile?.type==="required")?<span>Mobile Required</span>:<span></span> )

&&

((errors.Mobile?.type==="pattern")?<span>Invalid Mobile</span>:<span></span>)

}

</dd>

</dl>

<button type="submit">Submit</button>

</form>

</div>

)

}

Summary:

- Data Binding

a) One Way

b) Two Way

- Style Binding

- Class Binding

- Event Binding

- API Ajax Calls

- Forms

Component Properties

- Component properties allow to modify the component according to the requirements.

- It allows to reuse and customize a component.

- Properties control the behaviour and allow to modify the functionality.

Syntax:

export function ComponentName(props)

{

}

"props" is object type.

props.key = value;

- Component with properties is known as "Controlled Component".

- Parent component controls by passing the properties.

- It is controlled by the data passed from parent component.

- Component without properties is known as "Uncontrolled Component".

Syntax:

export function Component(props)

{

return(

<div {prop.key1}> </div>

)

}

<Component Key1="value" />

- A component property can handle any type of data

a) Primitive

b) Non Primitive

Ex:

1. Add a new folder

"Custom-Components"

2. Add a new file

"navbar.jsx"

export function NavBar(props)

{

return(

<div className="container-fluid">

<nav className={`d-flex justify-content-between p-2 border border-1 ${props.Theme}`}>

<div>

<span className="fw-bold fs-5"> {props.BrandTitle} </span>

</div>

<div>

{

props.MenuItems.map(item=> <span key={item} className="px-3"> {item} </span>)

}

</div>

<div>

<div className="input-group">

<input type="text" className="form-control" placeholder="Search" />

<button className="btn btn-warning bi bi-search"></button>

</div>

</div>

</nav>

</div>

)

}

3. Add home.jsx component

import { NavBar } from "../../custom-components/navbar";

export function Home(){

return(

<div className="container-fluid">

<h3>Amazon</h3>

<NavBar Theme="bg-dark text-white" BrandTitle="Amazon" MenuItems={["Home", "Shop", "Contact", "Offers", "Sale"]} />

<h3>Flipkart</h3>

<NavBar Theme="bg-success text-white" BrandTitle="Flipkart" MenuItems={["Home", "Deals", "Electronics", "Fashion"]} />

</div>

)

}

17-07-2024

FAQ: Which component(s) is most commonly used in real-world project development?

Ans:

Data Grid => Handle CRUD Operations on data.

Date Picker => It allows to pick a date or time from calendar.

Navbar => Dynamic navigation

Charts => Graphs [for designing dashboard]

Ex:

1. Go to "Custom-Components"

2. Add "data-grid.jsx"

export function DataGrid(props){

return(

<div className="container-fluid">

<table className="table table-hover caption-top">

<caption>{props.caption}</caption>

<thead>

<div>

<button data-bs-target="#AddNew" data-bs-toggle="modal" className="btn btn-primary"> + New</button>

<div className="modal fade" id="AddNew">

<div className="modal-dialog modal-dialog-centered">

<div className="modal-content">

<div className="modal-header">

<h3>Add New {props.caption} </h3>

</div>

<div className="modal-body">

<dl>

{

props.fields.map(field=>

<div key={field}>

<dt>{field}</dt>

<dd><input type="text" name={field} /></dd>

</div>

)

}

</dl>

</div>

<div className="modal-footer">

<button className="btn btn-primary">Register</button>

</div>

</div>

</div>

</div>

</div>

<tr>

{

props.fields.map(field=><th key={field}> {field} <div className="dropdown d-inline"> <button data-bs-toggle="dropdown" data-bs-target="#menu" className="btn dropdown-toggle" ></button> <ul className="dropdown-menu" id="menu"> <li className="dropdown-item"> <span className="bi bi-sort-alpha-down"></span> Sort Ascending </li> <li className="dropdown-item"> <span className="bi bi-sort-alpha-up"></span> Sort Descending </li> </ul> </div> </th>)

}

<th>Actions</th>

</tr>

</thead>

<tbody>

{

props.data.map(item=>

<tr key={item}>

{

Object.keys(item).map(key => <td key={key}> {item[key]} </td> )

}

<td> <button className="btn btn-warning bi bi-pen-fill"></button> <button className="btn btn-danger bi bi-trash-fill"></button> </td>

</tr>

)

}

</tbody>

</table>

</div>

)

}

3. Go to "home.jsx"

import { DataGrid } from "../../custom-components/data-grid";

export function Home(){

return(

<div className="container-fluid">

<DataGrid caption="Employee Details" fields={['EmployeeId', 'Name', 'Salary']} data={[{EmployeeId:101, Name:'Samson', Salary: 45000}]} />

<DataGrid caption="Products Details" fields={['Id', 'Name', 'Price', 'Stock']} data={[{Id:1, Name:'Samsung TV', Price:24000, Stock:'Availabe'}, {Id:2, Name:'Mobile', Price:12000, Stock:'Out of Stock'}]} />

</div>

)

}

Conditional Rendering

- Conditional rendering is the process of returning various types of content according to the state and situation.

- A component can render different content dynamically.

Syntax:

function Name()

{

if(condition)

{

return (<> </>)

}

else

{

return (<> </> )

}

}

Ex:

data-grid.jsx

export function DataGrid(props){

if(props.layout=='grid') {

return(

<div className="container-fluid">

<table className="table table-hover caption-top">

<caption>{props.caption}</caption>

<thead>

<div>

<button data-bs-target="#AddNew" data-bs-toggle="modal" className="btn btn-primary"> + New</button>

<div className="modal fade" id="AddNew">

<div className="modal-dialog modal-dialog-centered">

<div className="modal-content">

<div className="modal-header">

<h3>Add New {props.caption} </h3>

</div>

<div className="modal-body">

<dl>

{

props.fields.map(field=>

<div key={field}>

<dt>{field}</dt>

<dd><input type="text" name={field} /></dd>

</div>

)

}

</dl>

</div>

<div className="modal-footer">

<button className="btn btn-primary">Register</button>

</div>

</div>

</div>

</div>

</div>

<tr>

{

props.fields.map(field=><th key={field}> {field} <div className="dropdown d-inline"> <button data-bs-toggle="dropdown" data-bs-target="#menu" className="btn dropdown-toggle" ></button> <ul className="dropdown-menu" id="menu"> <li className="dropdown-item"> <span className="bi bi-sort-alpha-down"></span> Sort Ascending </li> <li className="dropdown-item"> <span className="bi bi-sort-alpha-up"></span> Sort Descending </li> </ul> </div> </th>)

}

<th>Actions</th>

</tr>

</thead>

<tbody>

{

props.data.map(item=>

<tr key={item}>

{

Object.keys(item).map(key => <td key={key}> {item[key]} </td> )

}

<td> <button className="btn btn-warning bi bi-pen-fill"></button> <button className="btn btn-danger bi bi-trash-fill"></button> </td>

</tr>

)

}

</tbody>

</table>

</div>

)

} else {

return(

<div className="container-fluid d-flex">

{

props.data.map(item=>

<div key={item} className="card p-2 m-2" style={{width:'200px'}}>

<div className="card-header">

{

item[props.fields[0]]

}

</div>

<div className="card-body">

<dl>

<dt> {props.fields[1]} </dt>

<dd>

{

item[props.fields[1]]

}

</dd>

<dt> {props.fields[2]} </dt>

<dd>

{

item[props.fields[2]]

}

</dd>

</dl>

</div>

<div className="card-footer">

<button className="btn btn-warning bi bi-pen-fill"></button>

<button className="btn btn-danger bi bi-trash-fill"></button>

</div>

</div>

)

}

</div>

)

}

}

home.jsx

import { useState } from "react";

import { DataGrid } from "../../custom-components/data-grid";

export function Home(){

const [layout, setLayout] = useState('grid');

function handleChange(e){

setLayout(e.target.value)

}

return(

<div className="container-fluid">

<div>

<h3>Layout</h3>

<select onChange={handleChange}>

<option>Choose Layout</option>

<option value='grid'>Grid</option>

<option value='card'>Card</option>

</select>

</div>

<DataGrid layout={layout} caption="Products Details" fields={['Id', 'Name', 'Price', 'Stock']} data={[{Id:1, Name:'Samsung TV', Price:24000, Stock:'Availabe'}, {Id:2, Name:'Mobile', Price:12000, Stock:'Out of Stock'}]} />

</div>

)

}

18-07-2024

React Hooks

- Hook is a service.

- Service is a pre-defined business logic, which allows to implement and customize according to the requirements.

- Service comprises various components

a) Provider

b) Consumer

c) Injector

- Provider is responsible for locating values and services in memory.

- Injector is responsible for injecting the values and functions into application.

- Consumer implements the service.

- Service uses "Dependency Injection" mechanism.

- Dependency Injection enables features like

a) Reusability

b) Extensibility

c) Testability

d) Maintainability

- React Hooks are functions that return pre-defined functionality, which a component can implement and customize according to the requirements.

- React introduced hooks from version 17x.

- React provides pre-defined hooks from Meta community and also allows to create custom hooks.

Hook Rules:

- It must be a function.

- It can't be void type.

- It must return.

- It must be configured at higher level in a component.

- You can't configure a hook in any another function or block.

function handleChange() {

const [uname, setUname] = useState(' '); // invalid

}

- It must be defined in camelCase.

- Hooks are not allowed in class components.

- Hook can be parameter less or parameterized.

- If hook is parameterized then it is mandatory to define the parameters.

FAQ: Why to create a hook, if you can do the same with a general function of JavaScript?

Ans: Hook uses "Single Ton" pattern.

Function uses a "Single Call" pattern.

FAQ: What is Single call?

Ans : It is an approach where a object is created every time when ever a resource is requested.

FAQ: What is Single ton?

Ans: It is an approach where an object is created for first request and the same is used across multiple requests.

Ex: hooks folder

usecaptcha.jsx

export function useCaptcha()

{

var code = '';

code = `${Math.round(Math.random()\*10)} ${Math.round(Math.random()\*10)} ${Math.round(Math.random()\*10)} ${Math.round(Math.random()\*10)} ${Math.round(Math.random()\*10)} ${Math.round(Math.random()\*10)}`

return code;

}

getapidata.jsx

import { useEffect, useState } from "react";

export function useGetAPI(url)

{

const [data, setData] = useState([]);

useEffect(()=>{

fetch(url)

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

.then(collection=> {

setData(collection);

})

},[url]);

return data;

}

login.jsx

import "./login.css";

import { useState } from "react";

import { useCaptcha } from "../../hooks/usecaptcha";

import { useDateFormat } from "../../hooks/usedateformat";

export function Login(){

const [title, setTitle] = useState('User Login');

const [userName, setUserName] = useState('John');

const code = useCaptcha();

return(

<div className="container">

<div>

<div> {mdf} </div>

<form className="form-login alert alert-warning alert-dismissible">

<h3 className="bi bi-person-fill">{title}</h3>

<button data-bs-dismiss="alert" className="btn btn-close"></button>

<dl>

<dt>User Name</dt>

<dd><input type="text" value={userName} className="form-control" /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control"/></dd>

<dt>Verify Code</dt>

<dd>{code}</dd>

</dl>

<button className="btn btn-warning w-100">Login</button>

</form>

</div>

</div>

)

}

home.jsx

import { useState } from "react";

import { DataGrid } from "../../custom-components/data-grid";

import { useGetAPI } from "../../hooks/getapidata";

export function Home(){

const [layout, setLayout] = useState('grid');

let products = useGetAPI(`https://fakestoreapi.com/products`);

function handleChange(e){

setLayout(e.target.value)

}

return(

<div className="container-fluid">

<div>

<ol>

{

products.map(product=><li key={product.id}>{product.title}</li>)

}

</ol>

</div>

<div>

<h3>Layout</h3>

<select onChange={handleChange}>

<option>Choose Layout</option>

<option value='grid'>Grid</option>

<option value='card'>Card</option>

</select>

</div>

<DataGrid layout={layout} caption="Products Details" fields={['Id', 'Name', 'Price', 'Stock']} data={[{Id:1, Name:'Samsung TV', Price:24000, Stock:'Availabe'}, {Id:2, Name:'Mobile', Price:12000, Stock:'Out of Stock'}]} />

</div>

)

}

19-07-2024

React Hooks

- Function Components

-Service

-Single Ton

-DI

-Provider

-Consumer

-Injector

-Custom Hooks

-Hook Rules

React Built-in Hooks

1. useState()

- It configures a local state.

- Local state is defined for current component.

- It is accessible only to the current component.

- It is created when component is requested.

- It is destroyed when component changes.

[You navigate from one component to another]

Syntax:

const [getter, setter] = useState(any);

FAQ: How useState can handle any type data?

Ans: It is a Generic type function.

Generic types are type safe and dynamically typed.

Syntax:

function useState<T>(param:T)

{

}

2. useContext()

- Context refers to memory.

- Context memory is created for current component and it is made available to other components that run within the context of current component.

- Context memory is created by using the method

"createContext()"

let contextName = createContext(null);

- You need to configure the context scope.

<contextName.Provider>

// components...

</contextName.Provider>

- Components that are defined inside context scope can use context.

- Components outside context scope can't use the context.

- Provider locates value in memory and injects into components by using the injector

"useContext()"

<contextName.Provider value={ }>

<Component />

</contextName.Provider>

function Component()

{

let context = useContext(contextName);

}

Note: Context memory can use any type of data. It is dynamically typed.

Ex:

Context-Demo.jsx

import { createContext, useContext, useState } from "react";

let UserContext = createContext(null);

export function Level1Component()

{

let context = useContext(UserContext);

return(

<div className="bg-warning text-danger p-4 m-3">

<h2>Level-1 Component - {context}</h2>

<Level2Component />

</div>

)

}

export function Level2Component()

{

let context = useContext(UserContext);

return(

<div className="bg-danger text-white p-4 m-3">

<h2>Level-2 Component - {context}</h2>

</div>

)

}

export function ContextDemo(){

const [username, setUserName] = useState();

function handleNameChange(e){

setUserName(e.target.value);

}

return(

<div className="container-fluid p-4 bg-dark text-white">

<div className="my-3">

<input type="text" placeholder="User Name" onChange={handleNameChange} />

</div>

<UserContext.Provider value={username}>

<h2>Context Demo - {username}</h2>

<Level1Component />

</UserContext.Provider>

</div>

)

}

3. useReducer() | Redux

20-07-2024

React Hooks

- Custom Hooks

- Built-in Hooks

a) useState

b) useContext [createContext]

useReducer

- It is used to configure application memory.

- Every application have various phases to maintain state.

a) Application\_Start

b) Session\_Start

c) Session\_End

d) Application\_End

- Reducer maintains application state, which provides data to all sessions in application.

- It is available from application start to application end.

- It configures predictable, maintainable and debuggable global state for application.

- At larger scale you can use JS library like "Redux", but at smaller scale you can use the hook "userReducer".

- Reducer provides various components

a) Store

b) State

c) UI

d) Reducer

- Store is a global location where the data is kept.

- State is responsible for accessing the data from store and provides to UI component.

- UI is the location where we present the data and handle data manipulations.

- Reducer configures the actions and manages the state. It triggers the actions and updates the state into store.

Configuring and using Application State:

1. Configure "Store" with initial values, i.e define initial state.

let initialState = { likesCount: 0 };

- Make sure that the store is an object with key and value references.

2. Configure a reducer function that handle the actions and data.

function reducer(state, action)

{

}

3. Define the action and state behaviour in reducer function

function reducer(state, action)

{

switch(action.type)

{

case "addLike":

return { likesCount: state.LikesCount + 1 };

case "addUnlike":

return { likesCount: state.LikesCount - 1 };

}

}

4. Every component can access and use the reducer and state defined globally. It requires "useReducer()" hook.

const [state, dispatch] = useReducer(reducer, initialState);

dispatch => defines the actions performed in component. It is a trigger that

dispatches the actions.

state => It comprises of the "Payload", which is the new data to update

into global store.

5. Component uses dispatch to trigger the action

function handleLikeClick()

{

dispatch({type: "addLike" })

}

function handleDislike()

{

dispatch({type: "addUnlike"})

}

Ex:

reducer-demo.jsx

import { useReducer } from "react";

let initialState = { likesCount: 0 };

function reducer(state, action){

switch(action.type){

case 'addLike':

return { likesCount: state.likesCount + 1 }

case 'addDislike':

return { likesCount: state.likesCount - 1 }

}

}

export function ReducerDemo()

{

const [state, dispatch] = useReducer(reducer, initialState);

function handleLikeClick(){

dispatch({type:'addLike'});

}

return(

<div className="container-fluid">

<iframe width="400" height="300" src="https://youtube.com/embed/\_mhtfJQ5lxU"></iframe>

<div className="my-2">

<button onClick={handleLikeClick} className="btn btn-primary bi bi-hand-thumbs-up"> </button> <span> {state.likesCount} Likes </span>

</div>

</div>

)

}

22-07-2024

useState

useContext

useReducer

React Cookies

- Cookie is a simple text document where client details are stored and used across requests.

- Cookie have 2 behaviours

a) In memory

b) Persistent

- In memory is temporary cookie, which is deleted automatically when browser is closed.

- Persistent is permanent cookie, saved in client hard drive. It is set with expiry date so that it is automatically deleted after specified duration.

- React requires "react-cookie" library to create and manage cookies for application

1. Install Cookie Library for project

> npm install react-cookie --save

2. Configure cookies provider at application level, provider is responsible for locating the cookies in memory and injecting them into components.

index.js

import { CookiesProvider } from 'react-cookie';

<React.StrictMode>

<CookiesProvider>

<App />

</CookiesProvider>

</React.StrictMode>

3. To create an manage cookies in a component you need "useCookies()" hook.

import { useCookies } from "react-cookie";

const [cookies, setCookie, removeCookie] = useCookies(['cookieName']);

cookies : to access a cookie and its value

setCookie : to create a cookie and append data into cookie

removeCookie: to remove cookie from memory

Syntax:

setCookie("cookieName", "cookieValue", { expires : noOfDays });

IF expires not defined then it is a temporary cookie.

removeCookie("CookieName"); // to delete cookie

cookies["CookieName"] // to access cookie value

Note: "react-cookie" returns "undefined" as value if cookie is not defined.

Syntax:

useEffect(()=>{

if(cookies["username"] === undefined) {

// navigate to login

}

},[])

(or)

if(cookies["username"])

{

}

else

{

// navigate to login

}

Ex:

1. Login.jsx

import "./login.css";

import { useState } from "react";

import { useCaptcha } from "../../hooks/usecaptcha";

import { useCookies } from "react-cookie";

export function Login(){

const [title, setTitle] = useState('User Login');

const [userName, setUserName] = useState('');

const code = useCaptcha();

const [cookies, setCookie, removeCookie] = useCookies(['username']);

function handleNameChange(e){

setUserName(e.target.value);

}

function handleLoginClick(){

setCookie('username', userName);

alert('Cookie Created');

}

return(

<div className="container">

<div>

<form className="form-login alert alert-warning alert-dismissible">

<h3 className="bi bi-person-fill">{title}</h3>

<button data-bs-dismiss="alert" className="btn btn-close"></button>

<dl>

<dt>User Name</dt>

<dd><input type="text" onChange={handleNameChange} className="form-control" /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control"/></dd>

<dt>Verify Code</dt>

<dd>{code}</dd>

</dl>

<button type="button" onClick={handleLoginClick} className="btn btn-warning w-100">Login</button>

</form>

</div>

</div>

)

}

2. reducer-demo.jsx

import { useEffect, useReducer } from "react";

import { useCookies } from "react-cookie";

let initialState = { likesCount: 0 };

function reducer(state, action){

switch(action.type){

case 'addLike':

return { likesCount: state.likesCount + 1 }

case 'addDislike':

return { likesCount: state.likesCount - 1 }

}

}

export function ReducerDemo()

{

const [state, dispatch] = useReducer(reducer, initialState);

const [cookies, setCookie, removeCookie] = useCookies(['username']);

useEffect(()=>{

console.log(cookies['username']);

},[])

function handleLikeClick(){

dispatch({type:'addLike'});

}

function handleSignout(){

removeCookie('username');

alert('Signed Out Successfully..');

}

return(

<div className="container-fluid">

<h2> { cookies['username'] } - SignedIn <button onClick={handleSignout} className="btn btn-link">Signout</button> </h2>

<iframe width="400" height="300" src="https://youtube.com/embed/\_mhtfJQ5lxU"></iframe>

<div className="my-2">

<button onClick={handleLikeClick} className="btn btn-primary bi bi-hand-thumbs-up"> </button> <span> {state.likesCount} Likes </span>

</div>

</div>

)

}

FAQ: What is difference between sessionStorage and localStorage?

Ans: Session Store is temporary.

Session is accessible only from current tab.

Session is removed when browser closed.

Local Storage is permanent.

It is accessible to components in another tab.

It is kept in memory even after browser closed.

You have to remove explicitly.

23-07-2024

React Cookie

Session Storage

Local Storage

useRef

- It creates a reference memory to store the data.

- Reference memory is intended to use internally for a process.

- It is not for rendering output.

- You can keep reference values, thread details using useRef() and use internally for component.

Syntax:

let thread = useRef(null);

thread.current = setTimeout(()=>{}, interval);

clearTimeout(thread.current);

useMemo

- It is used to save round trip for values that are used frequently in a component.

- Round trip is a mechanism where the resource is fetched from server every time when requested by client.

- You can cache the values in memory and save round trips.

- useMemo() can store the values returned by a function and improve the performance.

Syntax:

useMemo(()=>{ }, [ dependency ])

Ex:

function GetData(data){

// fetch data from server

return data;

}

let userData = useMemo(()=> GetData() ,[]);

let sampleData = GetData();

useCallback

- It is similar to useMemo but it can cache the function into memory.

- If a function is frequently requests across multiple requests then you can cache the function and save round trips.

Syntax:

useCallback(()=>{

function name() {

}

})

Ex:

let userData = useCallback(()=> {

function GetData(data){

// fetch data from server

return data;

}

} ,[]);

useEffect

- It defines the functionality to perform at the time of mount and unmount.

- useEffect defines mount actions by default.

- You can configure unmount by using a return function.

- useEffect mounts the component only once. It requires a dependency to mount again.

Syntax:

useEffect(()=>{

// actions on mount

return ()=> {

// actions on unmount

}

}, [ dependencies ])

Note: Remove <React.StrictMode> from index.js to test only production.

React Class Components

JavaScript Class:

- Class is a program template.

- A program template comprises of data and logic which you can customize and implement according to the requirements.

- If a class is mapping to the business requirements then it is referred as "Entity".

- If a class is mapping to the data requirements then it is referred as "Model".

- It have a behaviour of "blue print" as it can create multiple instances to reuse the logic.

Configuring Class:

a) Declaration

b) Expression

class Name

{

=> class declaration

}

const Name = class {

=> class expression

}

Class Members:

a) Property

b) Accessor

c) Constructor

d) Method

24-07-2024

useEffect :

Ex:

import { useEffect, useState } from "react"

export function Login(){

useEffect(()=>{

console.log('Login Component Mounted');

return ()=>{

console.log('Login Component Unmounted');

}

},[])

return(

<div>

<h2>Login</h2>

<dl>

<dt>User Name</dt>

<dd><input type="text" /></dd>

</dl>

</div>

)

}

export function Register(){

useEffect(()=>{

console.log('Register Component Mounted');

return ()=>{

console.log('Register Component Unmounted');

}

},[])

return(

<div>

<h2>Register</h2>

</div>

)

}

export function EffectDemo()

{

const [component, setComponent] = useState('');

function handleLoginClick(){

setComponent(<Login />);

}

function handleRegisterClick(){

setComponent(<Register />);

}

return(

<div className="container-fluid">

<h3>Home</h3>

<button onClick={handleLoginClick}>Login</button>

<button onClick={handleRegisterClick}>Register</button>

<hr />

{component}

</div>

)

}

Classes in JavaScript

- What is Class?

- Why it is an entity or model?

- Why it is a blue print?

- How to configure class?

- What are class members?

a) Property

b) Accessor

c) Method

d) Constructor

FAQ's:

1. Can we have a variable as class member?

A. No.

2. Why variable can't be class member?

A. Variable is immutable and class member can't be immutable.

3. Can we have a variable in class?

A. Yes.

Property

- Class uses a property to store data.

- Property can handle any type of data

a) Primitive

b) Non-Primitive

- Property is mutable.

Syntax:

class Name

{

PropertyName = value;

}

- You can access class member within class by using "this" keyword.

- You can access outside class by using instance of class.

Syntax:

let obj = new className();

obj.PropertyName;

Accessors

- Accessor will provide a fine grained control over the property.

- Accessors will allow to handle read and write operations on a property.

- Class provides 2 accessors

a) get() getter

b) set() setter

- get() is used to read and return a value of property.

- set() is used to write and assign a new value into property.

Syntax: without accessor

obj.Property // returns value stored in property

obj.Property=value; // assigns a new value into property

Syntax: with accessors

get aliasName()

{

return Property;

}

set aliasName(newValue)

{

Property = newValue;

}

Ex:

<script>

var userName = prompt("Enter User Name");

var role = prompt("Enter Your Role", "admin | user");

class Product

{

\_productName;

get ProductName()

{

return this.\_productName;

}

set ProductName(newName){

if(role==="admin") {

this.\_productName = newName;

} else {

document.write(`Hello ! ${userName} you are not authorized to set product name`);

}

}

}

let obj = new Product();

obj.ProductName = prompt("Enter Name for Product");

if(obj.ProductName)

{

document.write(`Product Name=${obj.ProductName}`);

}

</script>

Ex:

<script>

class Product

{

Rating = {

Hyd : {

Vendor: {

Ratings: {Rate:4.5, Reviews:5600}

},

customer: {

Ratings: {Rate:4.1, Reviews:100}

}

},

Delhi: {

Vendor: {

Ratings: {Rate:4.8, Reviews:8900}

},

customer: {

Ratings: {Rate:3.1, Reviews:600}

}

}

}

get HydCustRating(){

return this.Rating.Hyd.customer.Ratings.Rate;

}

set HydCustRating(newRating){

this.Rating.Hyd.customer.Ratings.Rate = newRating;

}

}

let obj = new Product();

obj.HydCustRating = prompt("Enter New Customer Rating for Hyd");

document.write(`Hyd Customer Rating : ${obj.HydCustRating}`)

</script>

Method

- Method defines the actions to be performed by class instance.

- Methods are mutable.

- JavaScript methods can be void or with return.

- Method can be parameterized or parameter less.

Syntax:

class Name

{

Method(params?)

{

}

}

FAQ's:

1. Can we define function as class member?

A. No.

2. Why function can't be a class member?

A. function is immutable.

3. Can we have a function in class?

A. Yes

What is JavaScript?

- JavaScript is light weight interpreted and JIT compiled programming language.

- It supports various programming approaches functional, structural, imperative, object oriented etc.

- JavaScript is a language used client side, server side, database, animation tools.

- Brendan Eich, ECMA, ES22, ESNext [MDN]

- JavaScript is used client side

a) Browser Interactions

b) DOM manipulations

c) Validations

- JavaScript is used server side

a) Creating server

b) Handling Request and Response

c) Routing

d) File Handling

25-07-2024

JavaScript Class

- Class Configuration

- Class Members

- Property

- Accessor

- Method

- Constructor

Constructor

- A constructor is responsible for instantiation.

- Instantiation is the process of creating an object for class.

- Every class have a constructor implicitly, which is known as "Default" Constructor.

- JavaScript constructor is anonymous type defined using constructor keyword.

Syntax:

class Name

{

constructor() {

}

}

- You can configure actions to perform at the time of creating object for class by using explicit constructor.

- Constructor is special type of sub-routine that executes automatically for an object.

- Constructor will execute only once for an object.

- JavaScript constructor can't overload.

Syntax: anonymous function

<script>

(function(){

document.write("i am anonymous");

})(); // IIFE

</script>

- JavaScript constructor invoking is optional

var obj = new Class; // valid

var obj = new Class(); // valid IIFE

Immediately Invoking Function Expression.

- IIFE is required for class if constructor is parameterized.

Ex:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

class Database

{

constructor(dbName){

document.write(`Connected with ${dbName} Database..<br>`);

}

Insert() {

document.write("Record Inserted..");

}

Delete(){

document.write("Record Deleted");

}

}

function InsertClick(){

let obj = new Database(document.querySelector("select").value);

obj.Insert();

}

function DeleteClick(){

let obj = new Database(document.querySelector("select").value);

obj.Delete();

}

</script>

</head>

<body>

<select>

<option>Select Database</option>

<option>Oracle</option>

<option>MySQL</option>

<option>MongoDB</option>

</select>

<button onclick="InsertClick()">Insert</button>

<button onclick="DeleteClick()">Delete</button>

</body>

</html>

Code Reusability & Extensibility

- Every application have to follow "DRY" principle. [ Don't Repeat Yourself ]

- Reusability is about reusing the code without rewriting the code.

- Extensibility is about extending the functionality without disturbing the existing.

- You can achieve reusability and extensibility using 2 techniques

a) Aggregation

b) Inheritance

Aggregation:

- It is the process of accessing the members of one class in another without configuring any relation between classes.

- It uses Object-to-Object communication.

- It is often referred as "Has-A-Relation".

Ex:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

class HDFC\_Bank\_Version1

{

Personal = "Personal Banking Services-Savings, Current";

Print(){

document.write(`${this.Personal}<br>`);

}

}

class HDFC\_Bank\_Version2

{

NRI = "NRI Banking Services";

Print(){

let obj = new HDFC\_Bank\_Version1();

obj.Print();

document.write(`${this.NRI}<br>`);

}

}

class HDFC\_Bank\_Version3

{

Loans = "Housing and Personal Loans";

Print(){

let obj = new HDFC\_Bank\_Version2();

obj.Print();

document.write(`${this.Loans}<br>`);

}

}

function InstallClick(){

var ver = document.querySelector("select").value;

switch(ver){

case "ver1":

let obj1 = new HDFC\_Bank\_Version1();

document.write("<h2> HDFC Bank APP - Ver-1 Installed </h2>");

obj1.Print();

break;

case "ver2":

let obj2 = new HDFC\_Bank\_Version2();

document.write("<h2> HDFC Bank APP - Ver-2 Installed </h2>");

obj2.Print();

break;

case "ver3":

let obj3 = new HDFC\_Bank\_Version3();

document.write("<h2> HDFC Bank APP - Ver-3 Installed </h2>");

obj3.Print();

break;

default:

document.write("Please Select a Version");

break;

}

}

</script>

</head>

<body>

<h3>Install Bank App</h3>

<select>

<option>Select Version</option>

<option value="ver1">Version 2022</option>

<option value="ver2">Version 2023</option>

<option value="ver3">Version 2024 [Latest]</option>

</select>

<button onclick="InstallClick()">Install</button>

</body>

</html>

Inheritance:

- It is the mechanism of configuring relationship between classes in order to access the members of one class in another.

- A new class can inherit existing class.

- New class is known as "Derived" class.

- Existing class is known as "Super" class.

- Derived class "extends" super class.

Syntax:

class Super

{

}

class Derived extends Super

{

}

- This mechanism is often known as "Is-A-Relation".

- The members of super class are accessible to derived by using "super" keyword.

Ex:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

class HDFC\_Bank\_Version1

{

Personal = "Personal Banking Services-Savings, Current";

Print(){

document.write(`${this.Personal}<br>`);

}

}

class HDFC\_Bank\_Version2 extends HDFC\_Bank\_Version1

{

NRI = "NRI Banking Services";

Print(){

super.Print();

document.write(`${this.NRI}<br>`);

}

}

class HDFC\_Bank\_Version3 extends HDFC\_Bank\_Version2

{

Loans = "Housing and Personal Loans";

Print(){

super.Print();

document.write(`${this.Loans}<br>`);

}

}

function InstallClick(){

var ver = document.querySelector("select").value;

switch(ver){

case "ver1":

let obj1 = new HDFC\_Bank\_Version1();

document.write("<h2> HDFC Bank APP - Ver-1 Installed </h2>");

obj1.Print();

break;

case "ver2":

let obj2 = new HDFC\_Bank\_Version2();

document.write("<h2> HDFC Bank APP - Ver-2 Installed </h2>");

obj2.Print();

break;

case "ver3":

let obj3 = new HDFC\_Bank\_Version3();

document.write("<h2> HDFC Bank APP - Ver-3 Installed </h2>");

obj3.Print();

break;

default:

document.write("Please Select a Version");

break;

}

}

</script>

</head>

<body>

<h3>Install Bank App</h3>

<select>

<option>Select Version</option>

<option value="ver1">Version 2022</option>

<option value="ver2">Version 2023</option>

<option value="ver3">Version 2024 [Latest]</option>

</select>

<button onclick="InstallClick()">Install</button>

</body>

</html>

Inheritance Rule:

- A derived class constructor must call super class constructor.

- Super class constructor executes first then followed by derived class constructor.

- JavaScript calls super constructor in derived class using "super()".

Ex:

<script>

class Super

{

constructor(){

document.write("Super Class Constructor<br>");

}

}

class Derived extends Super

{

constructor(){

super();

document.write("Derived class constructor");

}

}

let obj = new Derived();

</script>

Polymorphism

27-07-2024

JavaScript Class

Class Declaration & Expression

Class Members

- Properties

- Methods

- Accessors

- Constructor

Reusability & Extensibility

- Aggregation

- Inheritance

Polymorphism

- Poly means "Many".

- Morphos means "Forms".

- Polymorphism is behaviour where one component can handle various types of functionalities.

- Technically polymorphism allows single base class object to use the memory of multiple derived classes.

Syntax:

let baseClassObj = new Array(new Class1(), new Class2());

for(var obj of baseClassObj)

{

}

obj[0] => new Class1();

obj[1] => new Class2();

Ex:

<script>

class Employee

{

FirstName;

LastName;

Designaiton;

Print(){

document.write(`${this.FirstName} ${this.LastName} - ${this.Designaiton}<br>`);

}

}

class Developer extends Employee

{

FirstName = "Raj";

LastName = "Kumar";

Designaiton = "Developer";

Role = "Developer Role : Build, Debug, Test, Deploy";

Print(){

super.Print();

document.write(`${this.Role}`);

}

}

class Admin extends Employee

{

FirstName = "Kiran";

LastName = "Kumar";

Designaiton = "Admin";

Role = "Admin Role : Authorizations";

Print(){

super.Print();

document.write(`${this.Role}`);

}

}

class Manager extends Employee

{

FirstName = "Tom";

LastName = "Hanks";

Designaiton = "Manager";

Role = "Manager Role : Approvals";

Print(){

super.Print();

document.write(`${this.Role}`);

}

}

let employees = new Array(new Developer(), new Admin(), new Manager());

var designation = prompt("Enter Designation");

for(var employee of employees)

{

if(employee.Designaiton===designation){

employee.Print();

}

}

</script>

React Class Components

- Class components are still supported in React but not recommended in modern code.

FAQ: What are issues with OOP?

Ans:

- It will not support low level features.

- It can't directly interact with hardware services.

- It uses more memory.

- It is heavy and complex.

- It is tedious.

- React class component is just a JavaScript class that implements component behaviour from the base classes

a) React.Component

b) React.PureComponent

Syntax:

export class Login extends React.PureComponent | React.Component

{

}

- "React.Component" loads the component every time when ever changes are tracked.

It reloads entire component to update any specific change.

- "React.PureComponent" loads and updates only the changes without reloading the complete component.

- Every class component must configure a constructor in order to use state.

- React component constructor must call the super constructor.

Syntax:

export class Product extends React.Component

{

constructor(){

super();

}

}

- Class component must return "JSX" using "render()" method.

- "render()" is a base class method used to render JSX. [React.Component]

Syntax:

export class Product extends React.Component

{

constructor() {

super();

}

render() {

return(

<div>

.. JSX ..

</div>

);

}

}

State in class Component

- Class component can use a property to store data.

- Properties in component class are immutable.

- Hence it is recommended to use "State".

- Class component can't use React Hooks.

- Class component is configured with implicit State, which is derived from "Component" base.

Syntax:

constructor(){

super();

this.state = { key: value }

}

<p> { this.state.key } </p>

Ex:

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

ProductName: "Samsung TV",

Cities : ["Delhi", "Hyd"],

Rating: {Rate:4.2}

}

}

render() {

return(

<div className="container-fluid">

<h3>Product Class Component</h3>

<dl>

<dt>Product Name</dt>

<dd>{this.state.ProductName}</dd>

<dt>Cities</dt>

<dd>

{

this.state.Cities.map(city=> <li key={city}>{city}</li>)

}

</dd>

<dt>Rating</dt>

<dd>

{this.state.Rating.Rate}

</dd>

</dl>

</div>

)

}

}

29-07-2024

State in Class Component

- State is defined for a component implicitly.

- You can define state in component constructor.

- It must be initialized before the component mounts.

Syntax:

constructor()

{

super();

this.state = {

key : value

}

}

- State is an object type with key and value reference.

- Configure state using "this.state".

- Access state using "this.state.key".

- Assign values into state by using "this.setState()".

Note: Class components are often known as state full components, as they have

implicit state.

Ex:

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

ProductName: "Samsung TV",

Cities : ["Delhi", "Hyd"],

Rating: {Rate:4.2}

}

}

render() {

return(

<div className="container-fluid">

<h3>Product Class Component</h3>

<dl>

<dt>Product Name</dt>

<dd>{this.state.ProductName}</dd>

<dt>Cities</dt>

<dd>

{

this.state.Cities.map(city=> <li key={city}>{city}</li>)

}

</dd>

<dt>Rating</dt>

<dd>

{this.state.Rating.Rate}

</dd>

</dl>

</div>

)

}

}

Mount and Unmount in Class Component

- Class component uses the following life cycle hooks [methods] to handle mount and unmount phase.

a) componentDidMount()

b) componentWillMount()

c) componentWillUnmount()

Syntax:

componentDidMount()

{

this.setState({

key : value

})

}

Note: You can set state only after component mount or during mount. You can't

assign a value into state for a component which is not yet mounted.

Ex:

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

ProductName: "Samsung TV",

Cities : ["Delhi", "Hyd"],

Rating: {Rate:4.2}

}

}

componentDidMount(){

this.setState({

ProductName: 'Samsung LED TV',

Rating: {Rate:4.9}

})

}

render() {

return(

<div className="container-fluid">

<h3>Product Class Component</h3>

<dl>

<dt>Product Name</dt>

<dd>{this.state.ProductName}</dd>

<dt>Cities</dt>

<dd>

{

this.state.Cities.map(city=> <li key={city}>{city}</li>)

}

</dd>

<dt>Rating</dt>

<dd>

{this.state.Rating.Rate}

</dd>

</dl>

</div>

)

}

}

Style & Class Binding

- The technique of binding styles and classes dynamically similar to function component.

- However the values are dynamically accessed using state of class component.

Syntax:

this.state = {

styleObj : {backgroundColor: 'yellow', color: 'red' },

align: 'text-center p-2'

}

<h2 style={this.state.styleObj} className={this.style.align}>

Your Text

</h2>

Event Binding

- Event handling in class component is similar to function component.

- All Synthetic Events are similar to function components.

- Event in class component point towards a method in class.

Syntax:

handleInsertClick()

{

}

<button onClick={this.handleInsertClick}> Insert </button>

Ex:

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

}

handleInsertClick(){

alert('Record Inserted');

}

render() {

return(

<div className="container-fluid">

<button onClick={this.handleInsertClick} className="btn btn-primary mt-3">Insert</button>

</div>

)

}

}

Note: Event method can't use state of class directly.

You have to register the event method while creating state.

This requires "bind()" method that binds event to current component class memory, so that it can use the state defined in memory.

Syntax:

constructor()

{

super();

this.state= { };

this.handleInsertClick = this.handleInsertClick.bind(this);

}

Ex:

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

msg : ""

}

this.handleInsertClick = this.handleInsertClick.bind(this);

this.handleDeleteClick = this.handleDeleteClick.bind(this);

}

handleInsertClick(){

this.setState({

msg: "Record Inserted"

})

}

handleDeleteClick(){

this.setState({

msg: "Record Deleted Successfully.."

})

}

render() {

return(

<div className="container-fluid">

<button onClick={this.handleInsertClick} className="btn btn-primary mt-3">Insert</button>

<button onClick={this.handleDeleteClick} className="btn btn-danger mt-3">Delete</button>

<p>{this.state.msg}</p>

</div>

)

}

}

- You can register the event method with current class memory directly from event handler. This allows to register the method only when it is used.

a) Registering all methods in constructor => Eager Loading

b) Register only when it is used => Lazy Loading

Syntax:

<button onClick={this.handleInsertClick.bind(this)}> Insert </button>

FAQ: How to use state without using the bind() method in event handler?

Ans: You can configure event handler that returns method memory, so that the

memory is alive and available across multiple requests.

Syntax:

<button onClick={ ()=> this.handleInsertClick() }> Insert </button>

API Request , Forms, Two Way Binding

Props

TypeScript

30-07-2024

Class Components

- State

- Style Binding

- Class Binding

- Event Binding

Ex: Data Binding in Class Component [Two Way Binding]

product.jsx

import React from "react";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

Name: '',

Price: 0,

Stock: false

}

this.handleNameChange = this.handleNameChange.bind(this);

this.handlePriceChange = this.handlePriceChange.bind(this);

this.handleStockChange = this.handleStockChange.bind(this);

}

handleNameChange(e){

this.setState({

Name: e.target.value,

Price: this.state.Price,

Stock: this.state.Stock

})

}

handlePriceChange(e){

this.setState({

Name: this.state.Name,

Price: e.target.value,

Stock: this.state.Stock

})

}

handleStockChange(e){

this.setState({

Name: this.state.Name,

Price: this.state.Price,

Stock: e.target.checked

})

}

render() {

return(

<div className="container-fluid">

<h2>Register Product</h2>

<dl>

<dt>Name</dt>

<dd><input type="text" onChange={this.handleNameChange} /></dd>

<dt>Price</dt>

<dd><input type="number" onChange={this.handlePriceChange} /></dd>

<dt>Stock</dt>

<dd><input type="checkbox" onChange={this.handleStockChange} /> <label>Available</label> </dd>

</dl>

<h3>Details</h3>

<dl>

<dt>Name</dt>

<dd>{this.state.Name}</dd>

<dt>Price</dt>

<dd>{this.state.Price}</dd>

<dt>Stock</dt>

<dd>{(this.state.Stock===true)?"Available":"Out of Stock"}</dd>

</dl>

</div>

)

}

}

Formik in class Component

- Formik is a 3rd party form library for React.

- It provides "useFormik()" hook to configure and handle form interactions.

- Class component can't use the hooks.

- It can use the built-in components of formik library

<Formik>

<Form>

<Field>

<ErrorMessage>

Ex:

product.jsx

import React from "react";

import { Formik, Form, Field, ErrorMessage } from "formik";

import \* as yup from "yup";

export class Product extends React.Component

{

constructor(){

super();

}

render() {

return(

<div className="container-fluid">

<Formik initialValues={{Name:'', Price:0, Stock:false}} validationSchema={yup.object({Name: yup.string().required('Name Required')})} onSubmit={(values)=>{console.log(values)}} >

<Form>

<h2>Register Product</h2>

<dl>

<dt>Name</dt>

<dd> <Field type="text" name="Name" /> </dd>

<dd className="text-danger"> <ErrorMessage name="Name" /> </dd>

<dt>Price</dt>

<dd> <Field type="number" name="Price" /> </dd>

<dt>Stock</dt>

<dd> <Field type="checkbox" name="Stock" /> <label>Available</label> </dd>

</dl>

<button type="submit">Submit</button>

</Form>

</Formik>

</div>

)

}

}

API Requests in Class Component

- You can use all Ajax techniques in class components

a) XMLHttpRequest

b) fetch()

c) jQuery Ajax

d) axios

Ex:

product.jsx

import React from "react";

import axios from "axios";

export class Product extends React.Component

{

constructor(){

super();

this.state = {

categories: [],

products: []

}

}

LoadCategories(){

axios.get('https://fakestoreapi.com/products/categories')

.then(response => {

this.setState({

categories: response.data

})

})

}

componentDidMount(){

this.LoadCategories();

}

render() {

return(

<div className="container-fluid">

<h2>Select Category</h2>

<select>

{

this.state.categories.map(category=><option key={category}> {category}</option>)

}

</select>

</div>

)

}

}

FAQ: Is it possible to call any method in constructor?

Ans: Yes. It is not recommended if you are handling state in the defined method.

It is not good to assign state while creating state.

However constructor can execute only once, hence method can't run for

multiple requests.

Properties in Class Component

- Properties are used to create controlled components.

- "Component" base class provides "props" for configuring properties.

- Custom components that implement "Component" base can use "props".

- "props" is object type with key and value collection.

Syntax:

class ComponentName extends React.Component

{

constructor(){

super();

}

render() {

return(

<div>

{this.props.key}

</div>

)

}

}

Ex:

side-bar.jsx

import React from "react";

export class SideBar extends React.Component

{

constructor(){

super();

}

render(){

return(

<div>

<nav className="w-25 p-2 bg-light">

<ul className="list-unstyled">

<h3>{this.props.title}</h3>

{

this.props.navitems.map(item=> <li key={item} className="p-2 my-3 bg-secondary text-white rounded rounded-2" >{item}</li>)

}

</ul>

</nav>

</div>

)

}

}

product.jsx

import React from "react";

import axios from "axios";

import { SideBar } from "./side-bar";

export class Product extends React.Component

{

constructor(){

super();

}

render() {

return(

<div className="container-fluid">

<header className="mt-4">

<SideBar title="Shopper." navitems={['Home', 'Shop', 'Blog', 'Page', 'Contact']} />

</header>

</div>

)

}

}

React Life Cycle Hooks

- Life Cycle represents various phases of a component from start to end.

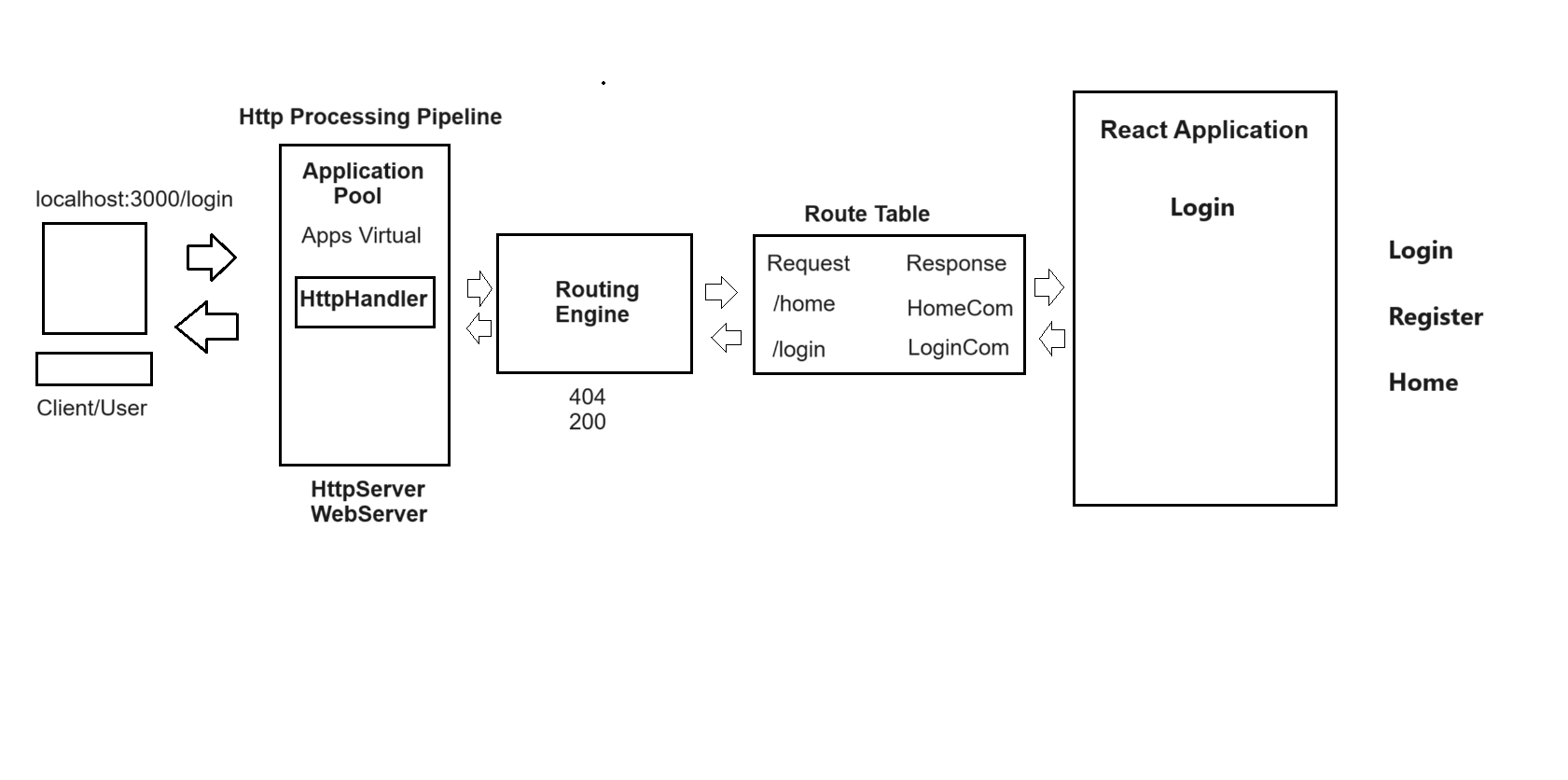
- Life cycle for class component comprises of 3 major phases

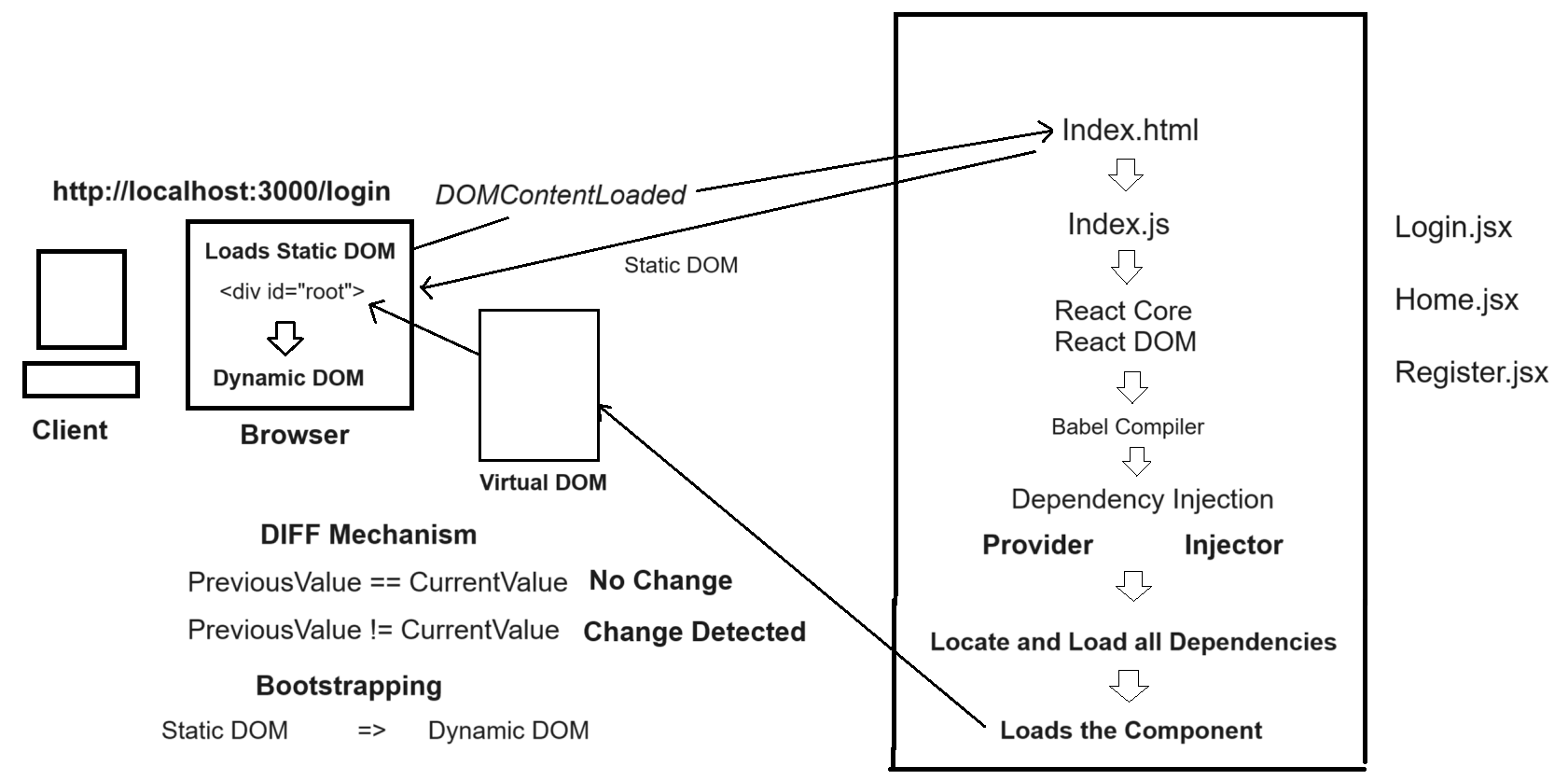
a) Mount

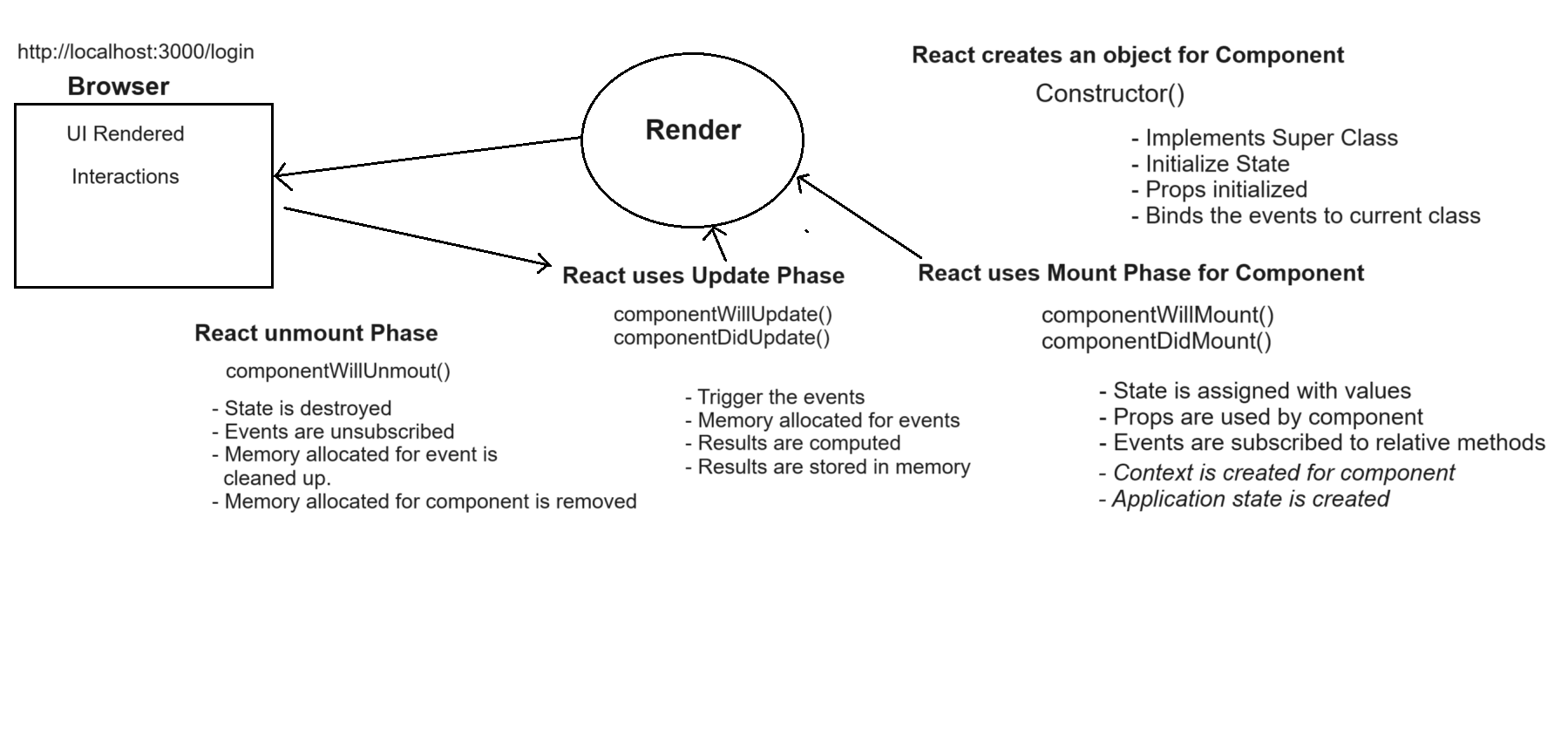
b) Update

c) Unmount

31-07-2024







Routing in React

- Routing is technique used in web application to configure user and SEO friendly URL's.

- User friendly URL allows to query any content directly from address bar. It can fetch data from a page and return as result.

Without Route:

https://www.amazon.in/shop.jsp?category=electronics&product=mobiles&model=apple

With Route:

https://www.amazon.in/shop/electronics/mobiles/apple

- SEO friendly URL's allow to track the exact location in a page and navigate user to specified location.

- Routing can be configured both

a) Server Side

b) Client Side

- Server side routing is used by server side applications for handling interaction with users and API's.

- Client Side routing allows to design SPA [Single Page Application] environment.

- User stays on one page and gets access to everything on to the page.

- Routing uses implicit AJAX, it loads new content into page without reloading the page.

- React provides 2 major versions of routing

a) react-router-dom V5

b) react-router-dom V6

- V6 is a complete re-build of routing for React 18x version.

https://reactrouter.com/en/main

Setup Routing:

1. Install Router DOM library for project

> npm install react-router-dom --save

2. Router DOM library provides various components to configure routes

<BrowserRouter>

<Routes>

<Route>

<Outlet>

<Link>

3. BrowserRouter creates a virtual DOM route that maps to actual DOM "location" object. All routes must be configured inside "BrowserRouter".

4. "Routes" is a collection of routes, it configures a Routing Table.

5. Every individual route in routing table is defined by using "Route".

6. Each "Route" comprises of 2 basic properties

a) path

b) element

Syntax:

<BrowserRouter>

<Routes>

<Route path="name" element={ <markup> | <component> } />

<Route path="name" element={ <markup> | <component> } />

</Routes>

</BrowserRouter>

Ex:

portfolio.css

.profile-pic {

border-radius: 50%;

width: 100px;

height: 100px;

}

portfolio.jsx

import { BrowserRouter, Link, Route, Routes } from "react-router-dom";

import "./portfolio.css";

import { Login } from "../login/login";

export function Portfolio(){

return(

<div className="container-fluid">

<BrowserRouter>

<header style={{height:'150px'}} className="d-flex justify-content-around p-3 bg-dark text-white">

<div className="profile-pic">

<img className="profile-pic" src="developer.jpg" width="100" height="100" />

<h4>JOHN</h4>

</div>

<div>

<span className="h1">React Developer</span>

</div>

<div className="fs-5 d-flex justify-content-between">

<span className="mx-2"> <Link className="btn btn-warning" to="about"> About Me </Link> </span>

<span className="mx-2"> <Link className="btn btn-danger" to="projects"> My Projects </Link> </span>

<span className="mx-2"> <Link className="btn btn-light" to="resume" > My Resume</Link> </span>

<span className="mx-2"> <Link className="btn btn-light" to="login" > <span className="bi bi-person-fill"></span> </Link> </span>

</div>

</header>

<section className="bg-secondary text-white p-3" style={{height:'100vh'}}>

<Routes>

<Route path="/" element={<div><h2>UI / UX Developer</h2><p>HTML | CSS | JavaScript | React</p></div>} />

<Route path="about" element={<div><h2>About Me</h2><i>i am a React developer</i><br /> Lorem ipsum dolor sit amet consectetur adipisicing elit. Aliquam, assumenda mollitia. Asperiores magni enim incidunt deserunt alias ipsam dignissimos harum ex aperiam sed ratione quos vel rem, recusandae consectetur labore?</div>} />

<Route path="projects" element={<div><h2>My Projects</h2><ul><li>To DO App</li><li>Shopping App</li><li>Video Library</li></ul></div>} />

<Route path="resume" element={<div><h2>Download My Resume</h2> <a className="text-white" download href="resume.pdf"> Download </a> </div>} />

<Route path="login" element={<Login />} />

<Route path="\*" element={<div><h1>OOPs !</h1> Unable to find your requested content</div>} />

</Routes>

</section>

</BrowserRouter>

</div>

)

}

02-08-2024

react-router-dom v6

<BrowserRouter>

<Routes>

<Route>

<Link>

<Outlet>

FAQ: What are wild card routes?

Ans: "/" & "\*"

<Route path="/" /> refers to content that display when user is not

requesting any specific.

<Route path="\*" /> refers to content that display when requested path

not found.

FAQ: What is difference between absolute path & relative path?

Ans: "Relative Path" adds the specified to existing path. It is mostly used for child route.

"Absolute Path" adds the specified by removing existing. [New Path]

<Route path="categories"> Relative

<Route path="/categories"> Absolute

Route Parameters

- A route parameter is defined in the URL.

- It is used to query any content in a component.

- You can transport data from one component to another by using "Route Parameter".

- The parameters are configured in Route path

<Route path="name/:param1/:param2" element={ <component /> } />

- The values are passed from URL by using Link or Address bar.

<Link to={`name/${value1}/${value2}`}> Click </Link>

http://localhost:3000/name/value1/value2

- React Router DOM provides "useParams()" hook to access the route parameters.

- It returns an object with Key/value collection.

Synax:

let params = useParams();

params.key

Child Routes

03-08-2024

Routes

Route Parameters

useParams()

Nested Routes / Child Routes

- A module in application can have routes within the context.

- Navigation to module is the parent route.

- Navigation within the module is defined using child routes.

Syntax:

<Route path="parent" element={ <ParentComponent /> }>

<Route path="child1" element={ <Child1Component />} />

<Route path="child2" element={ <Child2Component />} />

</Route>

- Child route requires <Outlet> to render its content into existing route.

- Outlet defines the target location where resulting markup is rendered.

FAQ: Can we define multiple outlets for a component in parent?

Ans: Yes.

Ex:

1. fakestore-index.jsx

import { BrowserRouter, Route, Routes } from "react-router-dom";

import { FakestoreHome } from "./fakestore-home";

import { FakestoreCategories } from "./fakestore-categories";

import { FakestoreProducts } from "./fakestore-products";

import { FakestoreDetails } from "./fakestore-details";

export function FakestoreIndex(){

return(

<div>

<BrowserRouter>

<header className="d-flex justify-content-between p-4 bg-dark text-white">

<div>

<span className="fs-4 fw-bold">Fakestore.</span>

</div>

<div className="fs-5">

<span className="mx-2">Home</span>

<span className="mx-2">Categories</span>

<span className="mx-2">Blog</span>

<span className="mx-2">Pages</span>

</div>

<div className="fs-5">

<span className="bi bi-search mx-2"></span>

<span className="bi bi-person mx-2"></span>

<span className="bi bi-cart3 mx-2"></span>

</div>

</header>

<section className="mt-4 p-4">

<Routes>

<Route path="/" element={<FakestoreHome />} />

<Route path="home" element={<FakestoreHome />} />

<Route path="categories" element={<FakestoreCategories />} />

<Route path="products/:category" element={<FakestoreProducts />}>

<Route path="details/:id" element={<FakestoreDetails />} />

</Route>

<Route path="\*" element={<h2 className="text-danger">Not Found</h2>} />

</Routes>

</section>

</BrowserRouter>

</div>

)

}

2. fakestore-details.jsx

import axios from "axios";

import { useEffect, useState } from "react";

import { Link, useParams } from "react-router-dom"

export function FakestoreDetails(){

let params = useParams();

const [product, setProduct] = useState({id:0, title:'', category:'', price:0, description:'', rating:{rate:0, count:0}, image:''});

useEffect(()=>{

axios.get(`https://fakestoreapi.com/products/${params.id}`)

.then(response=>{

setProduct(response.data);

})

},[params.id])

return(

<div>

<h3>Details</h3>

<div className="card p-2 m-2" style={{width:'200px'}}>

<img className="card-img-top" src={product.image} height="120"/>

<div className="card-header">

{product.title}

</div>

<div className="card-body">

<dl>

<dt>Price</dt>

<dd>{product.price}</dd>

<dt>Rating</dt>

<dd>

{product.rating.rate} <span className="bi bi-star-fill text-success"></span> [{product.rating.count}]

</dd>

</dl>

</div>

<div className="card-footer">

<textarea rows="4" disabled value={product.description}></textarea>

</div>

</div>

</div>

)

}

Note: Child routes mount a component only once, you have configure the dependency

to mount again. Usually the route parameter will be the dependency for child

route.

Syntax:

useEffect(()=>{

},[params.id])

Dynamic Navigation

- Application uses a static navigation technique, which is implemented using <Link> in routing.

- <Link> is configured with a path, and always uses the same path.

- Dynamic navigation allows to change path according state and situation.

- React Router DOM V6 recommends "useNavigate()" hook to configure dynamic navigation.

Syntax:

let navigate = useNavigate();

navigate("route\_path");

navigate("route\_path/params");

React MUI

05-08-2024

FAQ: What are search parameters?

Ans: Search Parameter refers to query string.

Route path can use query string to configure complex queries in a component.

Syntax:

path/param?key=value&key=value

You can access the search parameters by using "useSearchParams()" hook.

Syntax:

const [searchParam] = useSearchParams();

Search Parameter returns a "map", which is a key and value collection.

Syntax:

searchParam.get("key");

.set()

.entries() returns both keys and values

.keys() returns all keys

.values() returns all values

.delete() remove any specific key

.clear() remove all keys

Ex:

fakestore-index.jsx

import { BrowserRouter, Link, Route, Routes } from "react-router-dom";

import { FakestoreHome } from "./fakestore-home";

import { FakestoreCategories } from "./fakestore-categories";

import { FakestoreProducts } from "./fakestore-products";

import { FakestoreDetails } from "./fakestore-details";

import { CustomerLogin } from "./customer-login";

import { FakestoreError } from "./fakestore-error";

import { FakestoreResults } from "./fakestore-results";

import { FakestoreSearch } from "./fakestore-search";

export function FakestoreIndex(){

return(

<div>

<BrowserRouter>

<header className="d-flex justify-content-between p-4 bg-dark text-white">

<div>

<span className="fs-4 fw-bold">Fakestore.</span>

</div>

<div className="fs-5">

<Link to="/home" className="link link-light"><span className="mx-2">Home</span></Link>

<Link to="/categories" className="link-light"><span className="mx-2">Categories</span></Link>

<span className="mx-2">Blog</span>

<span className="mx-2">Pages</span>

</div>

<div className="fs-5">

<Link to="/search" className="link-light"><span className="bi bi-search mx-2"></span></Link>

<Link to="/login" className="link-light"><span className="bi bi-person mx-2"></span></Link>

<span className="bi bi-cart3 mx-2"></span>

</div>

</header>

<section className="mt-4 p-4">

<Routes>

<Route path="/" element={<FakestoreHome />} />

<Route path="home" element={<FakestoreHome />} />

<Route path="categories" element={<FakestoreCategories />} />

<Route path="products/:category" element={<FakestoreProducts />}>

<Route path="details/:id" element={<FakestoreDetails />} />

</Route>

<Route path="login" element={<CustomerLogin />} />

<Route path="error" element={<FakestoreError />} />

<Route path="search" element={<FakestoreSearch />} />

<Route path="results" element={<FakestoreResults />} />

<Route path="\*" element={<h2 className="text-danger">Not Found</h2>} />

</Routes>

</section>

</BrowserRouter>

</div>

)

}

fakestore-search.jsx

export function FakestoreSearch(){

return(

<div>

<form method="get" action="/results">

<h2>Search</h2>

<div className="input-group w-25">

<input type="text" placeholder="Search Fakestore" name="search" className="form-control"/>

<button type="submit" className="btn btn-warning bi bi-search"></button>

</div>

</form>

</div>

)

}

fakestore-results.jsx

import axios from "axios";

import { useEffect, useState } from "react";

import { useSearchParams } from "react-router-dom"

export function FakestoreResults(){

const [products, setProducts] = useState([{id:0, title:'', category:'', price:0, description:'', rating:{rate:0, count:0}, image:''}]);

let [searchParams] = useSearchParams();

useEffect(()=>{

axios.get(`https://fakestoreapi.com/products/category/${searchParams.get('search')}`)

.then(response=>{

setProducts(response.data);

})

},[])

return(

<div>

<h2>Searching for {searchParams.get('search')}</h2>

<div className="d-flex">

{

products.map(product=>

<div key={product.id} className="card p-2 m-2">

<img src={product.image} width="100" height="100" />

</div>

)

}

</div>

</div>

)

}

FAQ: Can we have multiple submit buttons in a form?

Ans: Yes.

FAQ: What is the role of multiple submit buttons?

Ans: You can submit to various locations using single form.

FAQ: How form submits to different locations?

Ans: Form submit buttons can use "formAction" attribute, which specifies the target location. Every button can have a different formAction.

Syntax:

<form method="get">

<button type="submit" formAction="/result1"> Search </button>

<button type="submit" formAction="/result2"> Top Search </button>

</form>

Ex:

fakestore-search.jsx

export function FakestoreSearch(){

return(

<div>

<form method="get">

<h2>Search</h2>

<div className="input-group w-25">

<input type="text" placeholder="Search Fakestore" name="search" className="form-control"/>

<button formAction="/results" type="submit" className="btn btn-warning bi bi-search"></button>

<button formAction="/lucky" type="submit" className="btn btn-danger bi bi-search-heart"></button>

</div>

</form>

</div>

)

}

fakestore-lucky.jsx

import axios from "axios";

import { useEffect, useState } from "react";

import { useSearchParams } from "react-router-dom";

export function FakestoreLucky(){

const [products, setProducts] = useState([{id:0, title:'', category:'', price:0, description:'', rating:{rate:0, count:0}, image:''}]);

let [searchParams] = useSearchParams();

useEffect(()=>{

axios.get(`https://fakestoreapi.com/products/category/${searchParams.get('search')}`)

.then(response=>{

setProducts(response.data);

})

},[])

return(

<div>

<h3>Top Search Results </h3>

<div className="d-flex">

{

products.filter(item=> item.rating.rate>4.5).map(product=>

<div key={product.id} className="card p-2 m-2">

<img src={product.image} width="100" height="100" />

<div className="card-body">{product.rating.rate}</div>

</div>

)

}

</div>

</div>

)

}

fakestore-index.jsx

import { BrowserRouter, Link, Route, Routes } from "react-router-dom";

import { FakestoreHome } from "./fakestore-home";

import { FakestoreCategories } from "./fakestore-categories";

import { FakestoreProducts } from "./fakestore-products";

import { FakestoreDetails } from "./fakestore-details";

import { CustomerLogin } from "./customer-login";

import { FakestoreError } from "./fakestore-error";

import { FakestoreResults } from "./fakestore-results";

import { FakestoreSearch } from "./fakestore-search";

import { FakestoreLucky } from "./fakestore-lucky";

export function FakestoreIndex(){

return(

<div>

<BrowserRouter>

<header className="d-flex justify-content-between p-4 bg-dark text-white">

<div>

<span className="fs-4 fw-bold">Fakestore.</span>

</div>

<div className="fs-5">

<Link to="/home" className="link link-light"><span className="mx-2">Home</span></Link>

<Link to="/categories" className="link-light"><span className="mx-2">Categories</span></Link>

<span className="mx-2">Blog</span>

<span className="mx-2">Pages</span>

</div>

<div className="fs-5">

<Link to="/search" className="link-light"><span className="bi bi-search mx-2"></span></Link>

<Link to="/login" className="link-light"><span className="bi bi-person mx-2"></span></Link>

<span className="bi bi-cart3 mx-2"></span>

</div>

</header>

<section className="mt-4 p-4">

<Routes>

<Route path="/" element={<FakestoreHome />} />

<Route path="home" element={<FakestoreHome />} />

<Route path="categories" element={<FakestoreCategories />} />

<Route path="products/:category" element={<FakestoreProducts />}>

<Route path="details/:id" element={<FakestoreDetails />} />

</Route>

<Route path="login" element={<CustomerLogin />} />

<Route path="error" element={<FakestoreError />} />

<Route path="search" element={<FakestoreSearch />} />

<Route path="results" element={<FakestoreResults />} />

<Route path="lucky" element={<FakestoreLucky />} />

<Route path="\*" element={<h2 className="text-danger">Not Found</h2>} />

</Routes>

</section>

</BrowserRouter>

</div>

)

}

React Material UI [MUI]

06-08-2024

React Material UI

- It is a library provided by react community to build interactive UI for React apps.

- It a library with built-in components for UI & UX.

- MUI provides various products

a) MUI Core

b) MUI X

c) Tool Pad

d) Templates

e) Design Kits

MUI Core:

- It is a component library for React.

- It provides free functional components for React.

- It is an open source and cross platform library.

- MUI re-written the library for React 18x version.

- MUI provides "@emotion" library as new integration to its functional component.

- MUI provides various products

a) MUI

b) Joy UI

c) Base UI

d) System UI

Setup MUI for Project:

1. Install MUI Core library with @emotion support

> npm install @mui/material @emotion/react @emotion/styled

2. Import the built-in components into your component. All MUI components are

controlled components.

[ You have to customize by using various properties ]

Customer-Login.jsx

import { useState } from "react"

import { useCookies } from "react-cookie";

import { useNavigate } from "react-router-dom";

import { TextField, Button } from "@mui/material";

export function CustomerLogin(){

const [userId, setUserId] = useState('');

const [password, setPassword] = useState('');

const [cookies, setCookie, removeCookie] = useCookies(['userid']);

let navigate = useNavigate();

function handleIdChange(e){

setUserId(e.target.value);

}

function handlePassword(e){

setPassword(e.target.value);

}

function handlLoginClick(){

if(userId==="john\_nit") {

setCookie("userid", userId);

navigate("/categories");

} else {

navigate("/error");

}

}

return(

<div className="row">

<div className="col">

<dl className="w-50">

<h4>Bootstrap Design</h4>

<dt>User Id</dt>

<dd><input type="text" className="form-control" onChange={handleIdChange} /></dd>

<dt>Password</dt>

<dd><input type="password" className="form-control" onChange={handlePassword} /></dd>

<button onClick={handlLoginClick} className="btn w-100 btn-primary">Login</button>

</dl>

</div>

<div className="col">

<h4>MUI Design</h4>

<div>

<TextField onChange={handleIdChange} className="w-50" label="User Id" variant="standard" ></TextField>

</div>

<div >

<TextField onChange={handlePassword} className="w-50" label="Password" variant="standard" ></TextField>

</div>

<div className="my-4">

<Button onClick={handlLoginClick} color="error" className="w-50" variant="contained" > Login </Button>

</div>

</div>

</div>

)

}

08-08-2024

MUI

- Input

- Button

- Card

- Dialog

MUIX Date Picker

- MUI X requires individual library to setup for UI and UX.

- Every UX component have its own dependencies, which you have to install and configure in project.

1. Install the following library for Date Picker

> npm install @mui/x-date-pickers --save

2. Install any one of the date adapters like : dayjs, luxon, moment, date-fns

> npm install dayjs --save

Date adapters are required to display date and time values in custom formats.

3. To use date picker in a component, you have to configure <LocalizationProvider>

with date adapter [ dayjs ]. [index.js]

import { LocalizationProvider } from '@mui/x-date-pickers';

import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs'

<LocalizationProvider dateAdapter={AdapterDayjs} >

<FakestoreIndex />

</LocalizationProvider>

4. In your component import and use date picker

import { DatePicker } from "@mui/x-date-pickers";

<DatePicker />

Issues with JavaScript:

- It is not strongly typed.

- It is not implicitly strictly typed.

- It is not an OOP language.

- Extensibility issues.

- Code Level Security Issues.

TypeScript

- TypeScript is strictly typed mode of JavaScript.

- Designed by "Anders Hejlsberg" the prime architect of "C#" language at Microsoft.

- It is strongly typed.

- It is implicitly strictly typed.

- It is an OOP language.

- TypeScript is built with TypeScript.

- It supports low level features.

- It can directly interact with hardware services.

- It is faster in interactions.

- It can build application at large scale.

TypeScript Architecture:

1. Core Compiler [core.ts]

core.ts : It checks the language keywords

program.ts : It checks the code structure and syntax.

scanner.ts : It handles input.

emitter.ts : It handles output.

parser.ts : It handles data conversion

checker.ts : It verifies the reference and value types.

2. Standalone Compiler [tsc.ts]

It trans compiles the Typescript code into JavaScript.

3. Language Service [services.ts]

It provides all functions and values required for Typescript

4. TS Server [server.ts]

It hosts your typescript language, and sets environment for

build, debug, test and deploy.

5. VS Shim [shims.ts]

It is a library used to make TypeScript cross platform.

6. Managed Language Service

It comprises of cross platform functions and values.

09-08-2024

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Get-ExecutionPolicy

Get-ExecutionPolicy -List

======================================================================

Setup Environment for TypeScript:

1. Open your command prompt on PC

2. Type the command

C:\>npm install -g typescript

3. Check the version

C:\> tsc -v

4. Create a new folder for typescript project

D:\typescript-project

5. Open folder in VS code

6. Run the command "tsc init" in terminal

> tsc -init

It generates typescript configuration file "tsconfig.json"

7. Add a new typescript file "index.ts"

var x = 10;

console.log("x=" + x);

8. Trans compile Typescript into JavaScript

> tsc index.ts

It generates "index.js"

9. You can link index.js to HTML page or directly run in terminal

> node index.js

10. Link to HTML page if you are using DOM methods

<script src="./index.js"> </script>

TypeScript Language

1. Variables

- Variable configuration is same as in JS with "var, let & const".

- TypeScript is strongly typed hence data type specification is required.

Syntax:

var name:dataType;

let name:dataType;

const name:dataType = value;

- TypeScript supports "Type Inference", which is a technique of determining the data type of variable according to the value initialized.

Syntax:

var price:number; // without type inference

var price = 45000; // number

price = "A"; // invalid

- If value is not initialized and the data type is not defined then the default data type is "any", which is root for all types.

- TypeScript types are the same as JS types

Primitive Types:

:number

:string

:boolean

:null

:undefined

Syntax:

var Name:string = "TV";

var Price:number = 45000.44;

var Stock:boolean = true;

- TypeScript supports "Union" of types, which allow to handle multiple types.

Syntax:

var name:type1 | type2 | type3 | … ;

var name:string | undefined;

Ex:

var userName:string|null;

userName = prompt("Enter name");

- Null and Undefined are verified in the same way how you do in JS.

if(userName==null) => returns true if value is null

{

}

if(price) => returns true if defined with value

{

}

Non Primitive Types

1. Array

- Arrays are used in computer programming to reduce overhead and complexity.

- Array reduces overhead by storing values in sequential order.

- Array reduces complexity by storing multiple values under the reference of one name.

- Array can handle various types of data in sequential order.

- Array size can change dynamically.

10-08-2024

Arrays

- Arrays are used to reduce overhead and complexity.

- Array reduce overhead by storing values in sequential order.

- Array reduce complexity by storing multiple values under one reference name.

- TypeScript & JavaScript array can handle various types of elements.

- TypeScript additionally can restrict array to similar type of elements.

- Array size in TS & JS can change dynamically.

TypeScript Array:

- TypeScript allows array declaration using var, let and const.

- If you want array to handle various types then define type as "any".

Syntax:

var values:any[] ;

- If you want to restrict specific type then define the data type.

Syntax:

var values:string[];

var values:number[];

- You have to allocate memory for array, it can be initialized or assigned.

Syntax:

var values:any[] = [ ]; // initialization

var values:any[];

values = [ ]; // assignment

var values:string[] = new Array(); // initialization with constructor

var values:string[];

values = new Array(); // assignment with constructor

- Array() constructor will not allow to initialize different types of values even when the data type is configured as "any". You can assign any type of data.

Syntax:

var values:any[] = new Array(10,...numbers[]);

var values:any[] = new Array("A", ...string[]);

Syntax:

var values:any[] = new Array();

values[0] = 10;

values[1] = "A";

values[2] = true;

FAQ: What is a Tuple?

Ans: It is a collection that allows initialization and assignment of various types of values.

Syntax:

var values:any[] = [ ]; // Tuple

- Array configuration in TS allows union of type. But you can initialize only one type, however assignment can be specified types.

Syntax:

var values:string[] | number[] = [10, "A"] ; // invalid ...numbers[], ...string[]

values[0] = 10; // valid

values[1] = "A"; // valid

Note: Type Inference of TypeScript allows a union of initialization.

Syntax:

var values = [10, "A", 20]; // string[] | number[]

FAQ: Is it possible to create a union for array to initialize multiple types?

Ans: Yes.

Syntax:

var values:(string | number)[] = [10, "A", 30];

- Array manipulation methods all are same as in JS.

Object Type

- JS object uses dynamic keys and values.

- It can assign a different type of value, that mismatch with initialized value.

- TS can restrict the values for every key.

- TS requires a schema to define as data type.

Syntax:

var obj:{schema} = { data\_according\_to\_schema };

Ex:

var product:{Id:number, Name:string} = { }

- Every key defined in schema is mandatory to configure in definition.

- TS supports nullable types, which allow to configure an optional key.

- Nullable type is defined with "?".

Syntax:

var product:{Id:number, Name:string, Price?:number} = { }; // price is optional

Ex:

index.ts

var product:{Name:string, Price:number, Stock:boolean, Rating?:number} = {

Name: "Samsung TV",

Price: 34000.44,

Stock: true,

Rating: 3.5

};

if(product.Rating) {

console.log(`Name=${product.Name}\nPrice=${product.Price}\nStock=${product.Stock}\nRating=${product.Rating}`);

} else {

console.log(`Name=${product.Name}\nPrice=${product.Price}\nStock=${product.Stock}`);

}

- Object schema can contain definition for both properties and methods.

- A method schema defines the signature and return type.

Syntax:

var obj = { property:string, method():void | type } = { }

- A method can be optional in object configuration.

Syntax:

var obj = {property:string, method?():void|type} = { }

Ex:

var product:{Name:string, Price:number, Qty:number, Total():number, Print?():void} = {

Name: "TV",

Price: 40000.33,

Qty: 2,

Total: function(){

return this.Qty \* this.Price

},

Print: function(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

product.Print();

- All object manipulations are same as in JS.

Ex:

var product:{Name:string, Cities:string[], Rating:{rate:number, count:number}, Price:number, Qty:number, Total():number, Print?():void} = {

Name: "TV",

Price: 40000.33,

Qty: 2,

Cities:["Delhi", "Hyd"],

Rating: {rate: 4.2, count: 3400},

Total: function(){

return this.Qty \* this.Price

},

Print: function(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

product.Print();

Array of Object

- JavaScript array with objects is schema less.

- TS configures structured data for a collection of objects.

Syntax:

var values: { key:type }[ ] = [ ];

Ex:

var values:{Name:string, Price?:number}[ ] = [ ];

var products:{Name:string, Price:number}[] = [

{Name: "TV", Price: 4500},

{Name: "Mobile", Price: 12000}

];

12-08-2024

Primitive

Non Primitive

- Array

- Object

- Array of Objects

Map Type

- JS map type is a key and value collection

- Key can be any type and value also any type.

- It provides implicit methods and properties to handle interactions.

- It is faster than object.

- TypeScript define Map as "Generic" type. You can restrict the key and value types.

Syntax:

var data:Map<keyType, valueType> = new Map();

var data:Map<number, any> = new Map();

var data:Map<string, string> = new Map();

var data:Map<any, any> = new Map();

- Map manipulations are handled by using

set(key, value)

get(key)

delete(key)

clear()

has(key)

keys()

values()

entries()

Regular Expression

- Typescript provides "RegExp" as type to handle regular expression.

- It is enclosed in "/ /" with meta characters and quantifiers.

Syntax:

var pattern:RegExp = /meta\_char\_quantifiers/;

- RegExp is verified by using "match()" method.

Ex:

index.ts

var password:string = "aDmin123";

var pattern:RegExp = /(?=.\*[A-Z])\w{4,15}/;

if(password.match(pattern)){

console.log(`Verified..`);

} else {

console.log(`Password 4 to 15 with at least 1 uppercase letter`);

}

Date Type

- Typescript "Date" type is used to configure date and time values.

- All date & time functions are same as in JS.

Syntax:

var mfd : Date = new Date(); // current date and time

var mfd : Date = new Date('yy-mm-dd hr:min:sec.milliSec'); // specific date & time

mfd.toLocaleDateString()

mfd.getMonth()

mfd.setMonth()

etc..

- Operators are same

- Statements are same

TypeScript Functions

- TS functions requires type specification for parameters and function.

Syntax:

function Name(param:type):type|void

{

}

- Every parameter is a required parameter.

- You can configure optional parameter by using null reference character "?".

- A required parameter can't follow an optional parameter. Hence make sure that

all optional parameters are last in formal list.

Syntax:

function Name(param1:type, param2?:type): void

{

}

Ex:

function Details(id:number, name:string, price?:number):void

{

if(price) {

console.log(`Id=${id}\nName=${name}\nPrice=${price}`);

} else {

console.log(`Id=${id}\nName=${name}`);

}

}

function Addition(a:number, b:number):number{

return a + b;

}

Details(1, "TV");

console.log("Addition=" + Addition(40,50));

- Rest parameters in a function requires type to be configured as "any[]".

Syntax:

function Name(...params:any[]) : void

{

}

- Function can also use the "spread" operator to spread argument into multiple

parameters.

Syntax:

Name(...[argument]);

Ex:

index.ts

function Details(...product:any[]):void

{

var [id, name, price] = product;

if(price) {

console.log(`Id=${id}\nName=${name}\nPrice=${price}`);

} else {

console.log(`Id=${id}\nName=${name}`);

}

}

function Addition(a:number, b:number):number{

return a + b;

}

Details(1, "TV");

console.log("Addition=" + Addition(40,50));

React:

function Login(): JSXElement

{

return (<div> </div>);

}

Types OOP Features

Contracts in OOP:

- A contract defines set of rules for designing a component.

- Rules are defined in OOP languages by using "interface".

- You can reuse and extend the rules.

- It can contain rules for both properties and methods.

- It can contain only rules not their definition.

Ex:

interface IProduct

{

Id:number;

Name:string;

Price:number;

Qty:number;

Total():number;

Print():void;

}

var product:IProduct = {

Id: 4,

Name: "TV",

Price: 45000.44,

Qty: 2,

Total: function(){

return this.Qty \* this.Price

},

Print: function(){

console.log(`Id=${this.Id}\nName=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

product.Print();

13-08-2024

- A contract can have optional rules.

- Optional rules are required to define goals for a component.

FAQ: What is difference between Objective and Goal?

Ans: Objective is time bound and must be achieved.

Goal is not have any time bound and may or many not be achieved always.

- Optional rules are defined by using null reference character.

Syntax:

interface IProduct

{

Name:string;

Price:number;

Rating?:number;

Print?():void;

}

- Contract can have readonly rules.

- You can't assign values into rules that are configured as readonly.

- You can initialize values but can't assign.

Syntax:

interface IProduct

{

Name:string;

readonly Price:number;

Qty:number;

}

var product : IProduct = {

Name : "TV",

Price: 35000.44,

Qty:2

}

product.Price = 50000.33; // Invalid

product.Qty = 3; // valid

- A contract can be extended by using another contract.

- Contract supports multiple and multi level inheritance.

Syntax:

interface A

{

}

interface B

{

}

interface Derived extends A, B

{

}

FAQ: Why multiple inheritance is not supported for class and it is supported for interface?

Ans: classes encounter "constructor dead lock", which will not allow to complete

multiple inheritance.

Interface will not have a constructor, hence it supports multiple inheritance.

Ex:

index.ts

interface IVendor

{

VendorId:number;

VendorName:string;

}

interface ICategory

{

CategoryName:string;

}

interface IProduct extends ICategory, IVendor

{

Id:number;

Name:string;

readonly Price:number;

Qty:number;

Total():number;

Print():void;

}

var product:IProduct = {

Id: 4,

Name: "TV",

Price: 45000.44,

Qty: 2,

CategoryName: "Electronics",

VendorId: 101,

VendorName : "Reliance Digital",

Total: function(){

return this.Qty \* this.Price

},

Print: function(){

console.log(`Id=${this.Id}\nName=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}\nCategory:${this.CategoryName}\nVendor:${this.VendorName}`);

}

}

product.Print();

2. Components in TypeScript [Classes]

- TypeScript classes can have 2 types of members

a) Static

b) Non Static

Static:

- It refers to continuous memory.

- The memory allocated for first object with continue for other objects.

- Static uses a connected architecture.

- Static members are accessible within and outside class by using class name.

Non-Static: [Dynamic]

- It refers to discreet memory.

- Memory is newly allocated for every object.

- It uses a disconnected architecture.

- Non static members are accessible within class by using "this" keyword and

outside class by using instance of class.

Ex:

index.ts

class Demo

{

static s = 0;

n = 0;

constructor(){

Demo.s = Demo.s + 1;

this.n = this.n + 1;

}

Print(){

console.log(`s=${Demo.s} n=${this.n}`);

}

}

let obj1 = new Demo();

obj1.Print();

let obj2 = new Demo();

obj2.Print();

let obj3 = new Demo();

obj3.Print();

- TypeScript class member can have access modifiers

a) public

b) private

c) protected

- public is accessible from any location and by using any instance i.e super class or

derived class.

- private is accessible only within the class.

- protected is accessible within the class and outside class, but only within derived class and also only by using derived class reference.

Ex:

class Product

{

public Name:string = "TV";

private Price:number = 45000.44;

protected Stock:boolean = true;

public Print():void {

}

}

class Derived extends Product

{

public Print(): void {

let obj = new Derived();

obj.Name;

obj.Stock;

obj.Print();

}

}

// only public are allowed

Abstraction, Generics, Enum, Modules, Namespace

14-08-2024

TypeScript Classes

Static and Non Static

Access Modifiers

public

private

protected

- Public is the default access modifiers for members in class.

- Other class members are same as in JS

a) Constructor

b) Property

c) Accessor

d) Method

- A class can implement contract. [implements]

- A class can extend another class. [extends]

- A class have to implement every required feature of a contract, however as it is a template it can allow customization by adding new features.

- You can't customize the structure of any member. [Data Type]

Syntax:

class className implements interfaceName

{

}

class derived extends super

{

}

- A class can implement multiple interfaces but can't extend multiple classes.

Ex:

interface IProduct

{

Name:string;

Price:number;

Qty:number;

Total():number;

Print():void;

}

interface ICategory

{

CategoryName:string;

}

class Product implements IProduct, ICategory

{

public Name: string = "Samsung TV";

public Price: number = 40000.44;

public Qty:number = 2;

public CategoryName: string = "Electronics";

public Stock:boolean = true;

public Total(): number {

return this.Qty \* this.Price

}

public Print(): void {

console.log(`Name=${this.Name}\nPrice=${this.Price}\nStock=${this.Stock}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

let tv = new Product();

tv.Print();

Templates in OOP

- A template provides pre-defined data and logic which you can implement and customize according to requirements.

- A template hides the structure and provides only functionality.

- The process of hiding structure and providing functionality is known as "Abstraction".

- It is mostly used "Rollouts" & "Modules" that have sensitive functionality to implement.

- Templates are designed by using "abstract" classes.

- Abstract class comprises of members

a) Which have structure and data implemented.

b) Which have structure but functionality or data not implemented.

- Any incomplete member is known as "abstract" member.

Syntax:

abstract class Template

{

public abstract property:type;

public property:type;

}

class Component extends Template

{

}

- You can't create a instance for abstract class, as it is incomplete in structure.

Ex:

interface IProduct

{

Name:string;

Price:number;

Qty:number;

Total():number;

Print():void;

}

abstract class ProductTemplate implements IProduct

{

public Name:string = "";

public Price:number = 0;

public Qty: number = 0;

public abstract Total():number;

public abstract Print(): void;

}

class Product extends ProductTemplate

{

Name = "Samsung TV";

Price = 45000.44;

Qty = 2;

Total(){

return this.Qty \* this.Price;

}

Print(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

let tv = new Product();

tv.Print();

Generics in TypeScript

16-08-2024

Contracts - Interface

Component - Class

Templates - Abstract Class

Generics

- Generic refers to "Type Safe".

- Type Safe is initially ready to accept any type, and can make it as strongly typed for the type defined into reference.

- TypeScript generic types can be defined for

a) Variables

b) Functions

c) Methods

d) Parameters

e) Class

- Generic type requires marking with "<type>".

Syntax:

function Name<T>(param:T) : T

{

}

- Generic types will not allow any operator to handle operands.

- You have handle operations using methods and functions.

Syntax:

function Name<T>(param:T): T

{

return function() {

}

}

Ex:

function Sum(a:any, b:any)

{

return a + b;

}

function Print<T>(a:T, b:T):T

{

return Sum(a, b);

}

console.log(Print<number>(10, 30));

console.log(Print<string>("A", "B"));

Ex: Generic Functions

interface IProduct

{

Name:string;

Price:number;

}

interface IEmployee

{

FirstName:string;

LastName:string;

Salary:number;

}

function FetchData<T>(response:T):void

{

for(var property in response) {

console.log(`${property} : ${response[property]}`);

}

}

FetchData<IEmployee>({FirstName:"Raj", LastName:"Kumar", Salary: 45000});

FetchData<IProduct>({Name: "TV", Price: 5000});

Ex: Generic Class

interface IOracle

{

UserId:string;

Password:string;

Database:string;

}

interface IMongoDB{ Url:string;}

class Database<T>

{

constructor(connectionString:T) {

for(var property in connectionString) {

console.log(`${property} : ${connectionString[property]}`);

}

}

}

let oracle = new Database<IOracle>({UserId:'scott', Password:'tiger', Database:'EmpDb'});

let mongo = new Database<IMongoDB>({Url: 'mongodb://127.0.0.1:27017'});

Ex: Generic class and Property

interface IOracle

{

UserId:string;

Password:string;

Database:string;

}

interface IMongoDB

{

Url:string;

}

class Database<T>

{

public ConnectionString:T|null = null;

public Print():void {

for(var property in this.ConnectionString) {

console.log(`${property} : ${this.ConnectionString[property]}`);

}

}

}

let oracle = new Database<IOracle>();

oracle.ConnectionString = {UserId:"scott", Password:"tiger", Database:"EmpDB"};

oracle.Print();

let mongo = new Database<IMongoDB>();

mongo.ConnectionString = {Url:"mongodb://127.0.0.1:27017"};

mongo.Print();

Enum [Enumeration]

- Enum is a collection of constants.

- Constants must be initialized with a value.

- Constants will not allow to change the value.

- Enum in typescript can handle

a) number

b) string

c) expression

- Numeric constants in Enum have auto implementation.

- Enum can't have a Boolean value or Boolean expression.

Ex:

enum StatusCodes

{

Found,

OK = 200,

Method,

NotFound = 404,

Timeout,

}

console.log(StatusCodes.NotFound);

Ex:

enum StatusCodes

{

A = 10,

B = 20,

C = A + B

}

console.log(`Addition=${StatusCodes.C}`);

Ex: Reverse Mapping

enum StatusCodes

{

NotFound = 404

}

console.log(`${StatusCodes.NotFound} : ${StatusCodes[404]}`);

Module & Namespace

17-08-2024

TypeScript Modules

- A modules is a set of values, functions, contracts, templates, components etc.

- Modular approach allows to build light weight application.

- You can build & import application specific library.

- It enables reusability, maintainability and testability.

- Every Typescript file is a module.

- Members of module are private in access, you have to mark them as "export" in order to access from other modules.

export function Name(){ }

export class Name() { }

- Module allows any one member as "default export".

- You can import and use the functions, classes, variables from another module.

import { Name } from "moduleName";

- To use module system, you have to setup a module system for project

a) CommonJS

b) UMD

c) AMD

d) RequireJS

e) ESModule

- Module system uses DI [Dependency Injection] Mechanism which enable easy reusability, extensibility, maintainability and testability.

Ex:

1. Add the folders

a) library

b) app

2. library/contracts/ProductContract.ts

export interface ProductContract

{

Name:string;

Price:number;

Qty:number;

Total():number;

Print():void;

}

3. library/templates/ProductTemplate.ts

import { ProductContract } from "../contracts/ProductContract";

export abstract class ProductTemplate implements ProductContract

{

public Name:string = "";

public Price:number = 0;

public Qty:number = 0;

abstract Total():number;

abstract Print():void;

}

4. library/components/ProductComponent.ts

import { ProductTemplate } from "../templates/ProductTemplate";

export class ProductComponent extends ProductTemplate

{

Name = "Samsung TV";

Price = 40000.44;

Qty = 2;

Total(){

return this.Qty \* this.Price;

}

Print() {

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

5. app/index.ts

import { ProductComponent } from "../library/components/ProductComponent";

let obj = new ProductComponent();

obj.Print();

6. Compile

> tsc indext.ts

> node index.js

Namespace

- A namespace is a collection sub-namespace and modules.

- It enables to configure a library for module system.

- Large scale application use namespace for building library.

- It allows to reduce ambiguity issues in library.

Syntax:

namespace Parent

{

namespace Child

{

// members : contracts, templates, components..

}

}

- You have to import namespace using "///<reference />" directive.

- You can directly uses the members of namespace or define as alias name.

import aliasName = Parent.Child.Member;

- To compile the library with namespace you have to use

> tsc -outFile file.js file.ts

19-08-2024

FAQ: What is fully qualified and aliasing namespace?

Ans: "Fully Qualified" refers to complete hierarchy of library.

Project.Contracts.ContractName

"Aliasing" is the process of importing fully qualified and defining a shortcut.

import ContractName = Project.Contracts.ContractName;

Ex:

1. library/contracts/ProductContract.ts

namespace Project

{

export namespace Contracts

{

export interface ProductContract

{

Name:string;

Price:number;

Qty:number;

Total():number;

Print():void;

}

}

}

2. library/templates/ProductTemplate.ts

///<reference path="../contracts/ProductContract.ts" />

import ProductContract = Project.Contracts.ProductContract;

namespace Project

{

export namespace Templates

{

export abstract class ProductTemplate implements ProductContract

{

public Name:string = "";

public Price:number = 0;

public Qty:number = 0;

public abstract Total():number;

public abstract Print():void;

}

}

}

3. library/components/ProductComponent.ts

///<reference path="../templates/ProductTemplate.ts" />

import ProductTemplate = Project.Templates.ProductTemplate

namespace Project

{

export namespace Components

{

export class ProductComponent extends ProductTemplate

{

Name = "TV";

Price = 40000.44;

Qty = 2;

Total(){

return this.Qty \* this.Price;

}

Print(){

console.log(`Name=${this.Name}\nPrice=${this.Price}\nQty=${this.Qty}\nTotal=${this.Total()}`);

}

}

}

}

4. app/index.ts

///<reference path="../library/components/ProductComponent.ts" />

import ProductComponent = Project.Components.ProductComponent;

let obj = new ProductComponent();

obj.Print();

5. Generate output file

>tsc --outFile index.js index.ts

Create React TypeScript Application:

1. Run the command

D:\>npx create-react-app app\_name --template typescript

2. NPX uses "Webpack" as bundler for creating react typescript application.

3. It requires typescript to install on the device.

D:\>npm install -g typescript

4. React TypeScript application file system [similar to JS application]

a) tsconfig.json Typescript configuration file.

b) .tsx For components & hooks

c) .ts Contracts, Templates

FAQ: How to setup the target JavaScript version for TypeScript?

Ans: Go to "tsconfig.json"

{

"compilerOptions": {

"target": "es5", // es6, es7

….

}

FAQ: What is the default module system of React typescript application?

Ans: EsNext [tsconfig.json]

Components in TS:

- Component file extension must be "tsx".

- A function component must have type defined as "JSX.Element"

[If type inference is configured then it is implicitly defined]

export function Name(): JSX.Element

{

return (

<div>

</div>

}

Configuring State in TS:

- State is defined with "useState()" hook.

- useState is generic type function.

Syntax:

useState<T>()

- You can configure the type while defining the state.

Syntax:

useState<string>();

useState<{key:value}>();

useState<{key:value}[]>();

- You can also configure a contract for state.

Syntax:

useState<Contract>();

useState<Contract[]>();

DataBinding, StyleBinding, ClassBinding, EventBinding:

- TS application uses the same techniques as JS.

- Binding expression "{ }"

- DataBinding requires null reference specification for any state value that you are using in JSX. [ ? ]

- Ajax requests are same.

a) XMLHttpRequest

b) Fetch

c) jQuery Ajax

d) Axios etc..

Syntax:

const [categories, setCategories] = useState<string[]>();

categories?.map() => If not initialized then it returns null

const [categories, setCategories] = useState<string[]>( [ ' ' ] );

categories.map() => null reference not required

Forms in TS application:

- You can use all the form libraries same as in JS.

- Formik for forms

- Yup for validations.

Routing & Hooks:

- Hooks functions are same as in JS.

- Routing library is same.

- react-router-dom

a) BrowserRouter

b) Routes

c) Route

d) Outlet

Ex:

src/contracts/FakestoreContract.ts

export interface FakestoreContract

{

id:number;

title:string;

price:number;

description:string;

image:string;

category:string;

rating:{rate:number, count:number};

}

data-binding.tsx

import { useEffect, useState } from "react";

import { FakestoreContract } from "../../contracts/FakestoreContract";

import axios from "axios";

export function DataBinding()

{

const [categories, setCategories] = useState<string[]>(['']);

const [product, setProduct] = useState<FakestoreContract>();

const [products, setProducts] = useState<FakestoreContract[]>();

function LoadCategories(){

axios.get(`https://fakestoreapi.com/products/categories`)

.then(response => {

setCategories(response.data);

})

}

function LoadProducts(){

axios.get(`https://fakestoreapi.com/products`)

.then(response=>{

setProducts(response.data);

})

}

useEffect(()=>{

LoadCategories();

LoadProducts();

},[])

return(

<div>

<ol>

{

categories.map(category=> <li key={category}>{category}</li>)

}

</ol>

<main style={{display:'flex', flexWrap:'wrap'}}>

{

products?.map(product=> <div key={product.id}><img src={product.image} width="100" height="100" /></div> )

}

</main>

</div>

)

}

Project with TypeScript

- MERN Stack

MongoDB Database

Express JS Middleware

React UI

Node JS Server Side

- To Do Application

- CRUD Operations

Create

Read

Update

Delete

20-08-2024

MERN Stack Application

MongoDB Database

- It is non-SQL or no-SQL database.

- It is document based database.

- It is schema less.

- It supports ORM. [Object Relational Mapping]

- ORM introduces features like

a) Identity Resolution

b) Lazy Loading

c) Change Tracking etc..

- It enables various approaches

a) Code First

b) Database First

- Developers can write for business requirements instead of data requirements.

- It supports indexing.

- It supports data replications.

- It supports Ad-hoc queries.

Setup MongoDB on your PC:

1. Visit

https://www.mongodb.com/try/download/community

2. Download Community Server

3. Install by selecting "MongoDB Compass" in options.

4. MongoDB Compass provides both GUI and CLI for database.

5. Start MongoDB server on your PC

Services.msc => MongoDB Server => Start

6. Open MongoDB compass tool and connect to MongoDB Database server

Connection String : mongodb://127.0.0.1:27017

MongoDB Terminology:

Oracle, MySQL, SQL Server... MongoDB

-----------------------------------------------------------------------------------------------

Database Database

Table Collection

Record/Row Document [JSON]

Field Field / Key

Joins Embedded Documents

MongoDB Shell Commands:

1. To view the existing databases

> show dbs

2. To view the active database

> db

3. To create a new database or to start using existing database

> use databaseName

Database name can be a new name or existing. If it is a new name, then

mongodb will create database.

Database is displayed in list only when there are collections in database.

Ex:

> use todo-react

4. To add collection [table] into database

> db.createCollection("name", { options })

> db.createCollection("name")

Option Description

------------------------------------------------------------------------------

autoIndexId Boolean It sets an auto generated primary key of document.

max number It defines the maximum number of documents to allow

size number It defines memory allocated for collection. [bytes]

capped Boolean It depends on "max", if it is set to false then old

documents are replaced with new documents.

Note: Default autoIndexId and capped are "true".

5. To view collections

> show collections

21-08-2024

MongoDB

Features

Commands

- Creating Database

use dbName

- Creating Collection

db.createCollection("name", { options })

6. Adding data into collection

insertOne()

insertMany()

Syntax:

> db.collectionName.insertOne({ key:value })

> db.collectionName.insertMany([ { }, { } ])

- MongoDB uses JavaScript types.

Ex:

> db.users.insertOne({UserId:'john\_nit', Age:22, Email:'john@gmail.com'})

> db.users.insertMany([ { }, { }, { } ])

Note: Date type data is stored in collection by using "$date" operator, which uses an object type.

{

"DoB": {

"$date": "1998-08-22T09:02:43.000Z"

}

}

Ex:

>db.users.insertOne({UserId:"john", DoB: { "$date": "yy-mm-ddThr:min:sec" } })

7. Querying Data from collection

find({query})

Syntax:

>db.collection.find({ }) // return all

>db.collection.find({key:value}) // return records with exact match

Ex:

>db.products.find({id:4})

>db.products.find({category:"jewelery"})

- MongoDB Query supports various operators

$gt greater than

$lt less than

$gte greater than or equal

$lte less than or equal

$ne not equal

$eq equal

$and AND

$or OR

Query: Records with price above 600

> db.products.find({price:{$gt:600}})

Query: Records with price between 200 to 600

> db.products.find({$and:[ {price:{$gt:200}}, {price:{$lt:600} } ]})

{ $and: [ { query1 }, { query2 } ] }

{ $or: [ { query1 }, { query2 }, { query3 } ] }

Query: Records with price above 400 and category Jewelery

Query: Records with price above 400 and all jewellery.

8. Updating data into document [Record]

updateOne()

updateMany()

> db.collection.updateOne({find}, {update})

- Update requires operators

$set to update value

$unset to remove field

$rename to rename field

Ex:

>db.products.updateOne({id:3}, {$set:{price:100.55}})

Note: Querying and updating object requires the key specification in string format.

'object.key':value

>db.products.find({'rating.rate':{$gt:4.5}})

9. Delete documents

deleteOne()

deleteMany()

Syntax:

> db.collection.deleteOne({findQuery})

Ex:

> db.products.deleteOne({id:3})

> db.products.deleteMany({'rating.rate':{$lte:3.0}})

10. Delete collection & Database

> db.collection.drop()

> db.dropDatabase() // current database

Summary:

CRUD

- insertOne() => C

- insertMany()

- find() => R

- updateOne() => U

- updateMany()

- deleteOne() => D

- deleteMany()

Distributed Computing

- Distributed computing allows communication between 2 applications running in 2 different machines.

- It also allows communication between 2 objects running in 2 different process of same machine.

- The popular distributed computing technologies

CORBA

DCOM

RMI

EJB

Web Service

Remoting

22-08-2024

CRUD Operation commands in MongoDB

Distributed Computing

Web Service

Web Service Specifications:

1. SOAP

2. REST

3. JSON

SOAP

- Service Oriented Architecture Protocol

- Consumer makes an XML request

- Provider sends an XML response

Request:

<Products>

<Product>

<Category> Electronics </Category>

</Product>

</Products>

Response:

<Products>

<Product>

<Id>1</Id>

<Name>TV</Name>

<Category>Electronics</Category>

</Product>

….

</Products>

REST

- Representational State Transfer

- Consumer sends a simple query request.

- Provider sends XML response, optionally JSON.

Request:

http://server.com/products?category=electronics

Response:

[

{

"Id": 1,

"Name": "TV",

"Category": "Electronics"

}

…

]

JSON

- JavaScript Object Notation

- Consumer sends JSON request.

- Provider sends JSON response.

FAQ: Why XML & JSON?

Ans:

- Light weight

- No conversions require [COM to Marshal]

- Can flow through firewall.

- Cross platform

- Can reach any device from PC to Watch.

FAQ: What are the issues with Web Service?

Ans:

- It uses only SOAP.

- It can run only on HttpServer

- It can use only Http as protocol.

- Its primary specification is XML.

API

- Application Programming Interface

- Uses REST & JSON

- Run anywhere. [any server, any protocol]

- Run on any device or OS.

- API provides a business logic that handle communication between database and client application.

- It can process the request and send response in XML or JSON.

Creating API with Node & Express JS:

- Node is required to create server side web application.

- Express JS is a middleware that handle communication in network application.

1. Open your React project in VS code

2. Create a new folder "Server" for business logic

3. Install the following libraries

> npm install express --save [ middleware library]

> npm install mongodb --save [ mongodb drivers library]

> npm install cors --save [ Cross Origin Resource Sharing ]

4. Server side business logic is written in a JavaScript file

"server.js"

5. Import "express" library and create a server app

var express = require("express");

var app = express();

6. Configure the routes for server app

app.get("/", (request, response)=> { })

app.post("/", (request, response)=>{ } )

app.put()

app.delete()

etc...

7. Configure port number to run server app

app.listen(5050);

8. Start your business logic

>node server.js

Ex:

var express = require("express");

var app = express();

app.get("/", (request, response)=>{

response.send("<h1>Welcome to API</h1>");

response.end();

});

app.get("/product", (req, res)=>{

var product = {

Name: "TV",

Price: 35000.44,

Stock: true

}

res.send(product);

res.end();

});

app.get("/details/:id/:name/:price", (req, res)=>{

res.send(`Id=${req.params.id}<br>Name=${req.params.name}<br>Price=${req.params.price}`);

res.end();

})

app.get("\*", (req, res)=> {

res.send("<code>Page you requested : Not found</code>");

res.end();

});

app.listen(5050);

console.log('Server Started : http://127.0.0.1:5050');

23-08-2024

MongoDB Connection with API

1. Install MongoDB database drivers library.

>npm install mongodb --save

2. Import "MongoClient" class from MongoDB drivers

var mongoClient = require("mongodb").MongoClient;

3. Connect to MongoDB server using connection string

mongoClient.connect("connectionString").then().catch().finally()

4. then() uses a function that returns client object, which is responsible for communicating with database on server.

mongoClient.connect("connectionString")

.then(function(clientObject){

var database = clientObject.db("databaseName");

})

5. "database" reference will allow to communicate with collections in database.

database.collection("name").find()

.insertOne()

.updateOne()

.deleteOne()

etc..

Ex: api.js

var mongoClient = require("mongodb").MongoClient;

mongoClient.connect("mongodb://127.0.0.1:27017").then(clientObject=>{

var database = clientObject.db("northwind");

database.collection("categories").find({}).toArray().then(documents=>{

console.log(documents);

})

});

Note: To test API requests you have to download web debuggers like

a) Postman b) Fiddler c) Swagger etc..

Ex: CRUD API Methods

api.js

var express = require("express");

var mongoClient = require("mongodb").MongoClient;

var cors = require("cors");

var app = express();

app.use(cors());

app.use(express.urlencoded({extended:true}));

app.use(express.json());

var conString = "mongodb://127.0.0.1:27017";

//API Routes

app.get("/users", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").find({}).toArray().then(documents=>{

res.send(documents);

res.end();

});

});

});

app.post("/register-user",(req, res)=>{

var user = {

UserId: req.body.UserId,

UserName: req.body.UserName,

Password: req.body.Password,

Email: req.body.Email,

Mobile: req.body.Mobile

}

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").insertOne(user).then(()=>{

console.log('User Registered..');

res.end();

});

});

});

app.put("/edit-user/:userid", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").updateOne({UserId:req.params.userid},{$set:{UserId:req.body.UserId, UserName:req.body.UserName, Password:req.body.Password, Email:req.body.Email, Mobile:req.body.Mobile}}).then(()=>{

console.log('User Details Updated..');

res.end();

});

});

});

app.delete("/delete-user/:userid", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").deleteOne({UserId:req.params.userid}).then(()=>{

console.log('User Deleted..');

res.end();

});

});

});

app.listen(3200);

console.log(`Server Started : http://127.0.0.1:3200`);

24-08-2024

React TO-DO Application

User Module

- User can register

- User can login

- User can add appointment

- User can view, edit and delete appointments

Data Models

- users collection [tblusers]

UserId [PK]

UserName

Password

Email

Mobile

- appointments collection [tblappointments]

Appointment\_Id [PK]

Title

Description

Date

UserId [FK]

API Requests [Routes]:

Method Route Description

----------------------------------------------------------------------------------------------------------

GET /users Get all users data

GET /users/userid Get specific user details

POST /register-user Add [Insert] user data into collection

PUT /edit-user/userid Edit specific user details

DELETE /delete-user/userid Delete specific user

GET /appointments/userid Get all appointments of specific user

POST /add-appointment Insert new appointment for user

PUT /edit-appointment/id Edit appointment details by id

DELETE /delete-appointment/id Delete specific appointment by id

React TypeScript Application:

1. Create React TypeScript app

>npx create-react-app react-todo - -template typescript

2. Install the libraries

Server Side:

> npm install mongodb express cors --save

Client Side:

> npm install bootstrap bootstrap-icons axios formik yup react-router-dom --save

3. Go to project and add a new folder by name "server"

4. Add API logic into server folder

api.js

var express = require("express");

var mongoClient = require("mongodb").MongoClient;

var cors = require("cors");

var app = express();

app.use(cors());

app.use(express.urlencoded({extended:true}));

app.use(express.json());

var conString = "mongodb://127.0.0.1:27017";

//API Routes

app.get("/users", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").find({}).toArray().then(documents=>{

res.send(documents);

res.end();

});

});

});

app.post("/register-user",(req, res)=>{

var user = {

UserId: req.body.UserId,

UserName: req.body.UserName,

Password: req.body.Password,

Email: req.body.Email,

Mobile: req.body.Mobile

}

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").insertOne(user).then(()=>{

console.log('User Registered..');

res.end();

});

});

});

app.put("/edit-user/:userid", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").updateOne({UserId:req.params.userid},{$set:{UserId:req.body.UserId, UserName:req.body.UserName, Password:req.body.Password, Email:req.body.Email, Mobile:req.body.Mobile}}).then(()=>{

console.log('User Details Updated..');

res.end();

});

});

});

app.delete("/delete-user/:userid", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblusers").deleteOne({UserId:req.params.userid}).then(()=>{

console.log('User Deleted..');

res.end();

});

});

});

// Route for Appointments

app.get("/appointments/:userid", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblappointments").find({UserId:req.params.userid}).toArray().then(documents=>{

res.send(documents);

res.end();

});

});

});

app.post("/add-appointment",(req, res)=>{

var appointment = {

Appointment\_Id: parseInt(req.body.Appointment\_Id),

Title: req.body.Title,

Description: req.body.Description,

Date: new Date(req.body.Date),

UserId: req.body.UserId

}

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblappointments").insertOne(appointment).then(()=>{

console.log('Appointment added');

res.end();

});

});

});

app.put("/edit-appointment/:id", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblappointments").updateOne({Appointment\_Id:req.params.id},{$set:{Appointment\_Id:parseInt(req.body.Appointment\_Id), Title: req.body.Title, Description: req.body.Description, Date:new Date(req.body.Date), UserId:req.body.UserId}}).then(()=>{

console.log('Appointment Details Updated..');

res.end();

});

});

});

app.delete("/delete-appointment/:id", (req, res)=>{

mongoClient.connect(conString).then(clientObj=>{

var database = clientObj.db("todo-react");

database.collection("tblappointments").deleteOne({Appointment\_Id:parseInt(req.params.id)}).then(()=>{

console.log('User Deleted..');

res.end();

});

});

});

app.listen(3200);

console.log(`Server Started : http://127.0.0.1:3200`);

5. Start API

D:\...\server > node api.js

6. Add "contracts" folder into project

UserContract.ts

export interface UserContract

{

UserId:string;

UserName:string;

Password:string;

Email:string;

Mobile:string;

}

AppointmentContract.ts

export interface AppointmentContract

{

Appointment\_Id:number;

Title:string;

Description:string;

Date:Date;

UserId:string;

}

26-08-2024

Redux in Video Library Project

1. Install Redux tool kit into project

>npm install @reduxjs/toolkit react-redux --save

2. Create a new slicer in project with initial state

- Slicer is a redux component used to configure initial state

- Initial state defines the data to store.

- In video library project initial state contains

videosCount : number

videos[ ] : array

- Add a new folder into project by name "slicers" [in src]

- Add a new file into slicers folder

video-slicer.jsx

import { createSlice } from "@reduxjs/toolkit";

- createSlice is used to configure initial state and actions to perform

const initialState = {

videosCount : 0,

videos: [ ]

}

const videoSlice = createSlice({

name: "video",

initialState,

reducers: {

addToSaveList: (state, action) => {

state.videos.push(action.payload);

state.videosCount = state.videos.length;

}

}

})

export const { addToSaveList } = videoSlice.actions;

export default videoSlice.reducer;

3. Create a global store into project to save the data and use across components.

- Add a new file by name "store.jsx"

- Import "configureStore" from tool kit.

- It is a component of redux tool kit used to create global store.

- Global store keep the data globally so that is predictable and debuggable.

store.jsx

import { configureStore } from "@reduxjs/toolkit";

import videoSlicer from "../slicers/video-slicer";

export default configureStore({

reducer : {

store : videoSlicer

}

})

4. Go to index.js and import store and define for application

- Store is a service, it requires a provider.

import { Provider } from "react-redux";

import store from "../store/store";

<Provider store={store}>

<App />

</Provider>

5. Go to User Dashboard in Video Library project and configure "Save" action

- It requires dispatch from redux tool kit

- dispatch can dispatch any action with payload

- action carries the payload and saves in store

user-dashboard.jsx

import { useDispatch } from "react-redux";

import { addToSaveList} from "slicers/video-slicer";

let dispatch = useDispatch();

function handleSaveClick(video)

{

dispatch(addToSaveList(video));

}

6. You can display the store data in UI

import store from "store/store";

<p> { store.getState().store.videosCount } </p>

Testing React Components

- Testing is the process of verifying "AS-IS" with "TO-BE" and asserts the result.

- "AS-IS" => the developers design.

- "TO-BE" => the client requirement.

AS-IS === TO-BE => Test Pass

AS-IS !== TO-BE => Test Fail

- Testing JavaScript libraries and frameworks requires various testing frameworks like

JEST, Jasmine Karma, Protractor etc..

- React Project template is enabled with JEST framework.

- Testing every component comprises of 3 phases

a) Arrange

b) Act

c) Assert

Ex:

- Login component must have title "User Login"

- Login component must have a link with text "Forgot Password"

- Link href must have a route path "/ForgotPassword".

1. Add new test file into project

"login.test.js" (or) "login.spec.js"

2. Import "render" & "screen" from JEST library

render : It renders the component to test.

screen : It can access component elements and test.

Syntax:

test("Test-Case-Name", ()=>{

render(<Component />);

let ref = screen.getByTestId("your\_test\_id");

expect(ref).toBe()

.toBeEqual()

.toHaveAttribute()

.toBeNumber() etc..

})

- Run test

>npm run test

Ex:

login.jsx

import axios from "axios"

import { useFormik } from "formik"

import { useCookies } from "react-cookie";

import { useNavigate } from "react-router-dom"

export function UserLogin(){

let navigate = useNavigate();

const [cookies, setCookies, removeCookie] = useCookies('user-id');

const formik = useFormik({

initialValues: {

UserId: '',

Password: ''

},

onSubmit: (user)=> {

axios.get('http://127.0.0.1:3030/get-users')

.then(response=>{

var data = response.data.find(item => item.UserId===user.UserId);

if(data){

if(data.Password===user.Password){

setCookies('user-id', user.UserId);

navigate('/user-dashboard');

} else {

navigate('/user-error');

}

} else {

navigate('/user-error');

}

})

}

})

return(

<div>

<h2>User Login</h2>

<form onSubmit={formik.handleSubmit}>

<dl>

<dt>User Id</dt>

<dd><input type="text" onChange={formik.handleChange} name="UserId" /></dd>

<dt>Password</dt>

<dd><input type="password" onChange={formik.handleChange} name="Password" /></dd>

</dl>

<button type="submit" className="btn btn-warning">Login</button>

</form>

</div>

)

}

login.test.js

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

import { Login } from "./login";

//Test Case 1

test('Title Test',()=>{

render(<Login />);

let title = screen.getByTestId("title");

expect(title).toHaveTextContent("User Login");

})

//Test Case 2

test('Forget Link Test',()=>{

render(<Login />)

let link = screen.getByText(/Forgot Password/);

expect(link).toBeInTheDocument();

expect(link).toHaveAttribute("href", "/ForgotPassword");

})

- Deployment

- Webpack

- React Native

31-08-024