Client-side web application frameworks and libraries are essential tools for building modern, interactive web applications. Here are some popular options:

**1. React:** A JavaScript library for building user interfaces. It allows you to create reusable UI components and efficiently update the UI when data changes.

**2. Angular:** A full-featured framework for building web applications. It provides a comprehensive set of tools and features for building complex applications.

**3. Vue.js:** A progressive JavaScript framework for building user interfaces. It is known for its simplicity and ease of integration with existing projects.

**4. Ember.js:** A framework for creating ambitious web applications. It follows the convention over configuration principle and provides a rich set of features.

**5. Backbone.js:** A lightweight JavaScript library that provides structure to web applications. It helps in organizing code and simplifies data synchronization.

**6. jQuery:** A fast, small, and feature-rich JavaScript library. It simplifies HTML document traversal, event handling, and animation.

**7. Bootstrap:** A popular CSS framework that provides a responsive grid system and pre-styled components. It helps in building mobile-first, responsive web applications.

**8. Material-UI:** A set of React components that implement Google's Material Design. It provides a rich set of UI components for building modern web applications.

These are just a few examples, and there are many more frameworks and libraries available. The choice depends on your specific requirements, project complexity, and personal preferences.

A library is a collection of pre-written code that provides specific functionality that can be used by other programs. It typically consists of a set of functions, classes, or modules that can be imported and used in a program.

A framework, on the other hand, is a more comprehensive and structured collection of libraries, tools, and guidelines that provide a foundation for building applications. It often includes a set of predefined rules, conventions, and patterns that developers can follow to create applications more easily and efficiently.

In summary, a library provides specific functionality that can be used in a program, while a framework provides a more comprehensive structure and guidelines for building applications.

A framework is a software development environment that provides developers with a foundation for building applications. It's a pre-written code base that provides a structured approach to application development.

Here are some key points about frameworks:

**Inversion of Control**: Unlike libraries, where the flow of control is determined by the developer, in a framework, the control is inverted - the framework calls the developer's code. This is often referred to as the "Hollywood Principle" - "Don't call us, we'll call you".

**Default Behavior**: Frameworks often provide a default behavior, which developers can then customize or extend to suit their specific needs. This can significantly speed up the development process.

**Convention over Configuration**: Many frameworks follow the principle of "convention over configuration", meaning they provide a set of sensible defaults and conventions that reduce the amount of boilerplate code a developer needs to write.

**Components and Libraries**: Frameworks often come with a set of pre-built components or libraries for common tasks, such as database access, templating, routing, and more.

**Lifecycle Management**: Frameworks often manage the lifecycle of your application, handling things like initialization, routing, and cleanup.

Examples of popular frameworks include Django and Flask for Python, Angular and React for JavaScript, and Rails for Ruby.

1. Component based Architecture
2. It is used to create SPA

**1.React is declarative**

**You can tell How the Dom is rest is taken care by the react.**

**Goto shop-> Javascript example**

*import* React *from* 'react';

class App *extends* React*.*Component {

  constructor(props) {

    super(props);

    this*.*state = {

      count: 0

    };

  }

*handleClick*() {

    this*.setState*(prevState => ({

      count: *prevState.*count + 1

    }));

  }

*render*() {

*return* (

      <div>

        <h1>Counter: {this*.state.*count}</h1>

        <button *onClick*={() => this*.handleClick*()}>Increment</button>

      </div>

    );

  }

}

*export* *default* App;

**1. React Declarative Example**

In React, you manage the state and UI declaratively. Here’s a basic example of a React component with a text input and a display of the entered text.

**App.js (React Component)**

import React, { useState } from 'react';

function App() {

const [text, setText] = useState('');

const handleChange = (event) => {

setText(event.target.value);

};

return (

<div>

<h1>React Declarative Example</h1>

<input

type="text"

value={text}

onChange={handleChange}

placeholder="Type something..."

/>

<p>Entered Text: {text}</p>

</div>

);

}

export default App;

**Explanation:**

* **State Management:** useState is used to manage the text state.
* **Declarative Rendering:** The text input and paragraph elements automatically update based on the state.
* **Event Handling:** handleChange updates the state whenever the input value changes.

**2. JavaScript Imperative Example**

In a vanilla JavaScript approach, you handle the DOM directly and update it imperatively. Here’s a simple example with an HTML file and JavaScript.

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>JavaScript Imperative Example</title>

</head>

<body>

<h1>JavaScript Imperative Example</h1>

<input type="text" id="textInput" placeholder="Type something..." />

<p id="displayText">Entered Text: </p>

<script src="script.js"></script>

</body>

</html>

**script.js**

// Select DOM elements

const textInput = document.getElementById('textInput');

const displayText = document.getElementById('displayText');

// Function to update the display text

const updateDisplay = () => {

displayText.textContent = 'Entered Text: ' + textInput.value;

};

// Add event listener to input

textInput.addEventListener('input', updateDisplay);

**Explanation:**

* **DOM Manipulation:** Use getElementById to get references to DOM elements.
* **Event Handling:** addEventListener is used to listen for input events and call updateDisplay to update the text content.
* **Imperative Updates:** Directly update the DOM based on user input.

**Summary**

* **Declarative (React):** Focuses on describing what the UI should look like based on the state. The framework takes care of updating the DOM.
* **Imperative (Vanilla JavaScript):** Focuses on explicitly updating the DOM based on user actions and events.

Both approaches achieve the same result but use different methodologies. React handles UI updates declaratively, making it more suitable for complex state management, while vanilla JavaScript uses imperative code to manage and update the DOM directly.

**2.React is Component based Architecture?**

<div>

<h2>Name: Name1</h2>

<h2>Age : 34 </h2>

1.</div>

<div>

<h2>Name: Name2</h2>

<h2>Age : 45 </h2>

</div>

.pers{

width:300px;

border:1px solid black;

padding : 10px;

margin : 10px;

display:inline-block;

}

So this name one, this name two, this 34 or maybe 45, all these informations can be passed to one component and you can keep on repeating this component.

That's the charm of component based architecture.

Now we can try the above code to component based architecture.

in code pen ->settings-> babbel-> react and react DOM

header,footer everthing is component.

How to convert above html code to component

function PersonInfo{

<html> inside javascript using Babel compiler is called JSX where Babel acts as a compiler helping to write the html inside javascript.

}

Every function is going to return JSX

function personInfo{

return (

<div class=”pers”>

<h2>Name:name1</h2>

<h2>Age:65</h2>

</div>

);

}

**<div id="p1">**

**</div>**

**.personInfo{**

**width:300px;**

**border:1px solid black;**

**margin: 10px;**

**padding: 10px;**

**display:block;**

**}**

**import reactIs from "https://esm.sh/react-is";**

**import React from "https://esm.sh/react";**

**import ReactDOM from "https://esm.sh/react-dom";**

**function PersonInfo(){**

**return (**

**<div className="personInfo">**

**<h2>Name:abc</h2>**

**</div>**

**);**

**}**

**ReactDOM.render(<PersonInfo />,document.getElementById("p1"))**

import reactIs from "https://esm.sh/react-is";

import React from "https://esm.sh/react";

import ReactDOM from "https://esm.sh/react-dom";

function PersonInfo(props){

return (

<div className="personInfo">

<h2>Name:{props.name}</h2>

</div>

);

}

ReactDOM.render(<PersonInfo name="n1"/>,document.getElementById("p1"))

ReactDOM.render(<PersonInfo name="n2" />,document.getElementById("p2"))

**3.Single Page Application**

**Create React App**

1.npx create-react-app <projname>

2.src->index.js->

3. npm start

**Difference between npm and npx(node package execute)**

**npm install create-react-app p1**

**or**

**npx create-react-app p1**

To understand this, I'm going to use one very simple package called Cowsay.

**npm install cowsay -g**

**cowsay hello**

Then there is a cow getting displayed and it says hello.Whatever message you have passed, in short, is displayed along with this cow image.This is how the cow say package works. Now, if I want to check whether this package is installed or not, I can use NPM list cow say if I run this command.

**npm list cowsay**

it shows empty because in the current folder I don’t have cowsay

**npm list cowsay -g**

[**--cowsay@1.5.0**](mailto:--cowsay@1.5.0)

So in my system cowsay installed and it occupy space also

Le me remove this

**npm uninstall cowsay -g**

**npx cowsay hello**

**npm list cowsay -g**

**empty**

So now cowsay is not installed it is just executed.

Props destructuring

Const {username,id}=props;

**State and useState**

*import* React, { useState } *from* "react";

*import* "./App.css";

function *App*() {

  const[*i*, *setCount*]= *useState*(0);

  function *increment*() {

*setCount*(i + 1);

*console.log*(i);

  }

*return* (

    <div *className*="App">

      <h2>{i} </h2>

      <button *onClick*={increment}>increment</button>

    </div>

  );

}

*export* *default* App;

**lazy initialization of a state or lazy initial state.**

function biggerProcess(){

console.log(“welcome”);

return 0;

}

Const [i,setCount]=useState(()=>biggerProcess()];

**Event Handling**

1.create decrement also

2.but it should be same functon

3.passing argument

4. return the function

JSX

let a=10;

let b=20

msg=<h1 style={{color:’blue’}}>Hi{a+b}</h1>

return (

<div>

{msg}

</div>

)

**htmlVs Jsx**

class in html className in Javascript

**Object in useState**

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

**function MyComponent() {**

**// Define state for an object**

**const [userInfo, setUserInfo] = useState({**

**name: 'John Doe',**

**age: 30,**

**email: 'john@example.com'**

**});**

**// Function to update name**

**const updateName = () => {**

**setUserInfo(prevState => ({**

**...prevState,**

**name: 'Jane Doe'**

**}));**

**};**

**return (**

**<div>**

**<p>Name: {userInfo.name}</p>**

**<p>Age: {userInfo.age}</p>**

**<p>Email: {userInfo.email}</p>**

**<button onClick={updateName}>Update Name</button>**

**</div>**

**);**

**}**

**export default MyComponent;**

**Class Component**

A class component in versions before React 16.8 class was the only component which could use state in it. But after React 16.8, even the functional component can have states. So earlier there were terms like stateful component which used to refer to class component and stateless component, which used to refer to functional component. But now that is just class and functional component, as both have states.\

Components defined as class still provide more features than a functional component.The React component needs to be extended to create a class component.

Stateful component refered as class component . Stateless omponenet refered as Functional component.But now both have states.

**How to upfdate the state**

**The set state method is used to send a request for the updation of the state.**

There are two ways or syntaxes of set state. The first method is you use set state. That is an updater function and there is an optional callback function.

The second method is that you directly specify the change object that is the new value. You pass and again it has an optional callback function. Now, when I say that it is a request for the abdication of the state, **it means that the set state method does not immediately update the state. Rather, it makes a queue of changes.**

*import* { Component } *from* "react";

*export* *default* class Person *extends* Component {

  state = {

    counter: 0,

  };

*increment* = () => {

    this*.setState*({ counter: this*.state.*counter + 1 });

  };

*render*() {

*return* (

      <div>

        <h2>{this*.state.*counter}</h2>

        <button *onClick*={this*.*increment}>Increment</button>

      </div>

    );

  }

}

this.setState({ counter: this.state.counter + 1 });

In this approach, you're directly updating the state based on the current state. You're not relying on the previous state value passed as an argument. This approach can be problematic when multiple state updates are batched together or when state updates are asynchronous.

The reason is that this.state may not always reflect the latest state value immediately after calling setState. React batches multiple setState calls for performance reasons, and this.state may not be updated immediately after calling setState. Therefore, relying on this.state for updating state values can lead to unexpected behavior.

this.setState(prevState => ({ count: prevState.count - 1 }));

In this approach, you're using the functional form of setState, where you pass a function that receives the previous state as an argument and returns an object representing the updated state. This ensures that you're always working with the latest state, even if multiple setState calls are batched together or executed asynchronously.

By using the functional form of setState, you avoid potential issues related to accessing this.state directly. React guarantees that the function passed to setState will be called with the latest state and will apply updates correctly, ensuring the integrity of your state changes.

In summary, while both approaches may work in simple cases, it's generally recommended to use the functional form of setState (the second approach) to ensure correctness and avoid potential issues, especially in more complex scenarios.

class Person extends React.Component {

constructor(props) {

super(props);

this.state = {

counter: 0

};

}

increment = () => {

this.setState((prevState, props) => {

return { counter: prevState.counter + props.step };

});

}

render() {

return (

<div>

<p>Counter: {this.state.counter}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

**chageObject**

**class Counter extends React.Component {**

**constructor(props) {**

**super(props);**

**this.state = {**

**count: 0**

**};**

**}**

**incrementCount = () => {**

**this.setState({ count: this.state.count + 1 }, () => {**

**console.log('Count updated:', this.state.count);**

**});**

**}**

**render() {**

**return (**

**<div>**

**<p>Count: {this.state.count}</p>**

**<button onClick={this.incrementCount}>Increment</button>**

**</div>**

**);**

**}**

**}**

**Component Lifecycle**

*import* React *from* "react";

*export* *default* class Person *extends* React*.*Component {

  constructor(props) {

    super(props);

    this*.*state = { counter: 0 };

  }

*componentDidMount*() {

*console.log*("Component did mount");

    // *You can perform any data fetching here.*

  }

*componentDidUpdate*(prevProps, prevState) {

*console.log*("Component did update");

    // *This method is invoked immediately after updating occurs.*

  }

*componentWillUnmount*() {

*console.log*("Component will unmount");

    // *This method is invoked immediately before a component is unmounted and destroyed.*

    // *Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().*

  }

*increment* = () => {

    this*.setState*({ counter: this*.state.*counter + 1 });

  };

*render*() {

*return* (

      <div>

        <h2>{this*.state.*counter}</h2>

        <button *onClick*={this*.*increment}>Increment</button>

      </div>

    );

  }

}

**useEffect Hook**

is one of the hook to handle life cycle method in functional component

useEffect hook is called everytime when the componenet is renderes or rerendered.

*import* React, { useState, useEffect } *from* "react";

*import* "./App.css";

function *App*() {

  //*Object state variable*

  const[*cnt*, *setCnt*]= *useState*(0);

*useEffect*(() => {

*console.log*("rendered");

  });

*return* (

    <div>

      {cnt}

      <button *onClick*={() => *setCnt*(cnt + 1)}>OK</button>

    </div>

  );

  //*return <Person />;*

}

*export* *default* App;

For example, assume that you are putting the setinterval here.

Of course, right now I'm not keeping curly brackets. Let me do that also.

So this is the JavaScript thing I'm putting inside the JSX in case if you call the Setinterval inside the component that is here in this JSX,

every time when interval changes something, it has to render. Now, in the rendering section, if you try another thing which is already getting rendered, then it is kind of a mess happening inside the component.

*import* React, { useState, useEffect } *from* "react";

*import* "./App.css";

function *App*() {

  //*Object state variable*

  const[*cTime*, *setTime*]= *useState*(new *Date*()*.toString*());

*useEffect*(() => {

*setInterval*(() => {

*setTime*(new *Date*()*.toString*());

    }, 1000);

  });

*return* (

    <div>

      {cTime}

      <button *onClick*={() => *setTime*(new *Date*()*.toString*())}>OK</button>

    </div>

  );

  //*return <Person />;*

}

*export* *default* App;

*import* React *from* "react";

*export* *default* class Person *extends* React*.*Component {

  state = { name: "" };

  constructor(props) {

    super(props);

*console.log*("In con");

  }

*componentDidMount*() {

    this*.setState*({ name: this*.props.*name });

  }

*componentDidUpdate*() {

*console.log*("in component");

  }

*componentWillUnmount*(){

}

*render*() {

*return* <div></div>;

  }

}

Handling Methods with events

*import* React, { Component } *from* "react";

class Person *extends* Component {

*handleClick*() {

    // *Handle the click event here*

*console.log*("Button clicked!");

  }

*render*() {

*return* (

      <div>

        <h1>Person Component</h1>

        <button *onClick*={this*.*handleClick}>Click me</button>

      </div>

    );

  }

}

*export* *default* Person;

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

const *Person* =()=>{

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

const *handleClick* =()=>{

*console.log*('Button clicked!');

    // *Perform any actions you want here*

};

*return* (

<div>

<input

*type*="text"

*value*={name}

*onChange*={(e) => *setName*(*e.target.*value)}

      />

<button *onClick*={handleClick}>Click me</button>

</div>

);

};

*export* *default* Person;

*import* React, { Component } *from* 'react';

class Person *extends* Component {

  constructor(props) {

    super(props);

    this*.*state = {

      name: ''

    };

  }

*handleClick* = () => {

*console.log*('Button clicked!');

    // *Perform any actions you want here*

  };

*handleChange* = (e) => {

    this*.setState*({ name: *e.target.*value });

  };

*render*() {

*const* { *name* }= *this.state*;

*return* (

      <div>

        <input

*type*="text"

*value*={name}

*onChange*={this*.*handleChange}

        />

        <button *onClick*={this*.*handleClick}>Click me</button>

      </div>

    );

  }

}

**How to skip for property**

*export* *default* Person;

*import* React, { useState, useEffect } *from* "react";

const *MyComponent* =()=>{

const[*data*, *setData*]= *useState*(*null*);

const[*isLoading*, *setIsLoading*]= *useState*(*false*);

const[*error*, *setError*]= *useState*(*null*);

*useEffect*(()=>{

    // *Fetch data only when isLoading is true*

if(isLoading){

*fetchData*()

*.then*((*result*)=>{

*setData*(result);

*setError*(*null*);

*setIsLoading*(*false*);

})

*.catch*((*error*)=>{

*setError*(error);

*setIsLoading*(*false*);

});

}

},[isLoading]);// *Only re-run the effect if isLoading changes*

const *fetchData* = *async* ()=>{

    // *Simulating an API call*

    // *const response = await fetch("https://api.example.com/data");*

const *response* ={

*ok*: *true*,

*json*:()=>{

*return* { *name*:"test"};

},

};

if(!*response.ok*){

*throw* new *Error*("Failed to fetch data");

}

*return response.json*();

};

const *handleClick* =()=>{

    // *Trigger data fetching when the button is clicked*

*setIsLoading*(*true*);

};

*return* (

<div>

{isLoading?(

<div>*Loading...*</div>

):error?(

<div>*Error:* {*error.message*}</div>

):data?(

<div>*Data:* {*JSON.stringify*(data)}</div>

):(

<button *onClick*={handleClick}>Fetch Data</button>

)}

</div>

);

};

**Component lifecycle**

*export* *default* MyComponent;

*import* React, { useEffect } *from* "react";

function *Person*() {

*useEffect*(() => {

    // *componentDidMount*

*console.log*("Component mounted");

*return* () => {

      // *componentWillUnmount*

*console.log*("Component unmounted");

    };

  }, []);

*useEffect*(() => {

    // *componentDidUpdate*

*console.log*("Component updated");

  });

*return* (

    <div>

      <h1>Person Component</h1>

    </div>

  );

}

*export* *default* Person;

**List and keys**

**import React from 'react';**

**import MyListComponent from './MyListComponent';**

**const ParentComponent = () => {**

**const data = ['Item 1', 'Item 2', 'Item 3'];**

**return (**

**<div>**

**<h1>Parent Component</h1>**

**<MyListComponent items={data} />**

**</div>**

**);**

**};**

**export default ParentComponent;**

**import React from 'react';**

**const MyListComponent = ({ items }) => {**

**return (**

**<ul>**

**{items.map((item, index) => (**

**<li key={index}>{item}</li>**

**))}**

**</ul>**

**);**

**};**

**const App = () => {**

**const data = ['Item 1', 'Item 2', 'Item 3'];**

**return (**

**<div>**

**<h1>List Example</h1>**

**<MyListComponent items={data} />**

**</div>**

**);**

**};**

**export default App;**

**inline style**

*import* React *from* 'react';

const *Person* =()=>{

const *personStyle* ={

*backgroundColor*:'lightblue',

*padding*:'10px',

*borderRadius*:'5px',

*margin*:'10px',

*textAlign*:'center',

};

*return* (

<div *style*={personStyle}>

      <h2>Person Component</h2>

      <p>This is a sample component with inline styles.</p>

    </div>

);

};

*export* *default* Person;

**How to debugging**

**Event Handling**

**Textbox onChange() event**

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

function *Person*() {

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

    const *handleInputChange* =(*event*)=>{

*setName*(*event.target.value*);

};

*return* (

        <div>

            <input *type*="text" *value*={name} *onChange*={handleInputChange} />

            <p>Hello, {name}!</p>

        </div>

    );

}

*export* *default* Person;

*import* React, { useState } *from* "react";

function *Person*() {

  const[*gender*, *setGender*]= *useState*("");

  const *handleGenderChange* =(*event*)=>{

*setGender*(*event.target.value*);

};

*return* (

    <div>

      <label>

        <input

*type*="radio"

*value*="male"

*checked*={gender === "male"}

*onChange*={handleGenderChange}

        />

        Male

      </label>

      <label>

        <input

*type*="radio"

*value*="female"

*checked*={gender === "female"}

*onChange*={handleGenderChange}

        />

        Female

      </label>

      <label>

        <input

*type*="radio"

*value*="other"

*checked*={gender === "other"}

*onChange*={handleGenderChange}

        />

        Other

      </label>

    </div>

  );

}

*export* *default* Person;

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

const *Person* =()=>{

const[*selectedOption*, *setSelectedOption*]= *useState*('');

const *handleDropdownChange* =(*event*)=>{

*setSelectedOption*(*event.target.value*);

};

*return* (

<div>

<select *value*={selectedOption} *onChange*={handleDropdownChange}>

                <option *value*="">Select an option</option>

                <option *value*="option1">Option 1</option>

                <option *value*="option2">Option 2</option>

                <option *value*="option3">Option 3</option>

            </select>

<p>*Selected option:* {selectedOption}</p>

</div>

);

};

*export* *default* Person;

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

const *Person* =()=>{

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

const[*age*, *setAge*]= *useState*('');

const *handleSubmit* =(*e*)=>{

*e.preventDefault*();

        // *Handle form submission logic here*

*console.log*('Name:',name);

*console.log*('Age:',age);

};

*return* (

<form *onSubmit*={handleSubmit}>

            <label>

                Name:

                <input *type*="text" *value*={name} *onChange*={(e) => *setName*(*e.target.*value)} />

            </label>

            <br />

            <label>

                Age:

                <input *type*="number" *value*={age} *onChange*={(e) => *setAge*(*e.target.*value)} />

            </label>

            <br />

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

        </form>

);

};

*export* *default* Person;

*import* React *from* "react";

function *Person*(props) {

  const{ *name*, *age* }=props;

*return* (

    <div>

      <h2>{name}</h2>

      {age >= 18 ? <p>Adult</p> : <p>Minor</p>}

    </div>

  );

}

*export* *default* Person;

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

function *App*() {

    const[*isAdmin*, *setIsAdmin*]= *useState*(*false*);

    const *handleCheckboxChange* =()=>{

*setIsAdmin*(!isAdmin);

};

*return* (

        <div>

            <label>

                <input

*type*="checkbox"

*checked*={isAdmin}

*onChange*={handleCheckboxChange}

                />

                Is Admin

            </label>

            {/\* *Display the list of persons if isAdmin is checked* \*/}

                    </div>

    );

}

*export* *default* App;

const *persons* =[

{ *name*:'John', *age*:25},

{ *name*:'Jane', *age*:30},

{ *name*:'Bob', *age*:35},

];

if (isAdmin) {

*return* (

        <ul>

            {*persons.map*((person, index) => (

                <li *key*={index}>

                    Name: {*person.*name}, Age: {*person.*age}

                </li>

            ))}

        </ul>

    );

}

Validation i

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

**function TextBoxValidation() {**

**const [value, setValue] = useState('');**

**const [isValid, setIsValid] = useState(true);**

**const handleChange = (event) => {**

**const inputValue = event.target.value;**

**setValue(inputValue);**

**// Perform validation here**

**if (inputValue.length >= 5) {**

**setIsValid(true);**

**} else {**

**setIsValid(false);**

**}**

**};**

**return (**

**<div>**

**<label>**

**Enter text:**

**<input**

**type="text"**

**value={value}**

**onChange={handleChange}**

**style={{ borderColor: isValid ? 'initial' : 'red' }}**

**/>**

**</label>**

**{!isValid && <p style={{ color: 'red' }}>Text must be at least 5 characters long</p>}**

**</div>**

**);**

**}**

**export default TextBoxValidation;**

**Forms Error Handling**

*import* React, { useState } *from* "react";

const *Person* =()=>{

const[*name*, *setName*]= *useState*("");

const[*email*, *setEmail*]= *useState*("");

const[*error*, *setError*]= *useState*("");

const *handleSubmit* =(*e*)=>{

*e.preventDefault*();

    // *Validate form inputs*

if(!name||!email){

*setError*("Please fill in all fields");

*return*;

}

    // *Submit form data*

    // *...*

    // *Clear form inputs and error message*

*setName*("");

*setEmail*("");

*setError*("");

};

*return* (

<div>

<h2>*Person Form*</h2>

{error&&<p>{error}</p>}

<form *onSubmit*={handleSubmit}>

        <div>

          <label *htmlFor*="name">Name:</label>

          <input

*type*="text"

*id*="name"

*value*={name}

*onChange*={(e) => *setName*(*e.target.*value)}

          />

        </div>

        <div>

          <label *htmlFor*="email">Email:</label>

          <input

*type*="email"

*id*="email"

*value*={email}

*onChange*={(e) => *setEmail*(*e.target.*value)}

          />

        </div>

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

      </form>

</div>

);

};

*export* *default* Person;

*import* React, { useState } *from* "react";

const *Person* =()=>{

const[*formData*, *setFormData*]= *useState*({

*name*:"",

*email*:"",

*password*:"",

*confirmPassword*:"",

});

const[*errors*, *setErrors*]= *useState*({});

const *handleChange* =(*e*)=>{

*setFormData*({...formData,[*e.target.name*]: *e.target.value* });

};

const *handleSubmit* =(*e*)=>{

*e.preventDefault*();

    // *Perform validation*

const *validationErrors* ={};

if(!*formData.name*){

*validationErrors.name* ="Name is required";

}

if(!*formData.email*){

*validationErrors.email* ="Email is required";

}elseif(!/*\S*+*@\S*+*\.\S*+/*.test*(*formData.email*)){

*validationErrors.email* ="Invalid email format";

}

if(!*formData.password*){

*validationErrors.password* ="Password is required";

}elseif(*formData.password.length* <6){

*validationErrors.password* ="Password must be at least 6 characters long";

}

if(*formData.password* !== *formData.confirmPassword*){

*validationErrors.confirmPassword* ="Passwords do not match";

}

if(*Object.keys*(validationErrors)*.length* >0){

*setErrors*(validationErrors);

}else{

      // *Submit the form*

      // *You can make an API call here to send the form data to the server*

*console.log*("Form submitted:",formData);

}

};

*return* (

<div>

<h1>*Registration Form*</h1>

<form *onSubmit*={handleSubmit}>

        <div>

          <label>Name:</label>

          <input

*type*="text"

*name*="name"

*value*={*formData.*name}

*onChange*={handleChange}

          />

          {*errors.*name && <span>{*errors.*name}</span>}

        </div>

        <div>

          <label>Email:</label>

          <input

*type*="email"

*name*="email"

*value*={*formData.*email}

*onChange*={handleChange}

          />

          {*errors.*email && <span>{*errors.*email}</span>}

        </div>

        <div>

          <label>Password:</label>

          <input

*type*="password"

*name*="password"

*value*={*formData.*password}

*onChange*={handleChange}

          />

          {*errors.*password && <span>{*errors.*password}</span>}

        </div>

        <div>

          <label>Confirm Password:</label>

          <input

*type*="password"

*name*="confirmPassword"

*value*={*formData.*confirmPassword}

*onChange*={handleChange}

          />

          {*errors.*confirmPassword && <span>{*errors.*confirmPassword}</span>}

        </div>

        <button *type*="submit">Register</button>

      </form>

</div>

);

};

*export* *default* Person;

import React from 'react';

import PersonForm from './PersonForm';

const App = () => {

return (

<div>

<h1>Person Form</h1>

<PersonForm />

</div>

);

};

export default App;

import React from 'react';

import useFormValidation from './useFormValidation';

import Input from './Input';

const PersonForm = () => {

const { handleSubmit, handleChange, values, errors } = useFormValidation();

return (

<form onSubmit={handleSubmit}>

<Input

name="name"

type="text"

placeholder="Name"

value={values.name}

onChange={handleChange}

error={errors.name}

/>

<Input

name="email"

type="email"

placeholder="Email"

value={values.email}

onChange={handleChange}

error={errors.email}

/>

<Input

name="age"

type="text"

placeholder="Age"

value={values.age}

onChange={handleChange}

error={errors.age}

/>

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

</form>

);

};

export default PersonForm;

import { useState } from 'react';

const useFormValidation = () => {

const [values, setValues] = useState({ name: '', email: '', age: '' });

const [errors, setErrors] = useState({});

const handleChange = (e) => {

const { name, value } = e.target;

setValues({

...values,

[name]: value

});

};

const handleSubmit = (e) => {

e.preventDefault();

const isValid = validateForm();

if (isValid) {

console.log('Form submitted successfully');

}

};

const validateForm = () => {

let errors = {};

if (!values.name) {

errors.name = 'Name is required';

}

if (!values.email) {

errors.email = 'Email is required';

} else if (!/\S+@\S+\.\S+/.test(values.email)) {

errors.email = 'Email is invalid';

}

if (!values.age) {

errors.age = 'Age is required';

} else if (isNaN(values.age)) {

errors.age = 'Age must be a number';

}

setErrors(errors);

return Object.keys(errors).length === 0;

};

return { handleSubmit, handleChange, values, errors };

};

export default useFormValidation;

import { useState } from 'react';

const useFormValidation = () => {

const [values, setValues] = useState({ name: '', email: '', age: '' });

const [errors, setErrors] = useState({});

const handleChange = (e) => {

const { name, value } = e.target;

setValues({

...values,

[name]: value

});

};

const handleSubmit = (e) => {

e.preventDefault();

const isValid = validateForm();

if (isValid) {

console.log('Form submitted successfully');

}

};

const validateForm = () => {

let errors = {};

if (!values.name) {

errors.name = 'Name is required';

}

if (!values.email) {

errors.email = 'Email is required';

} else if (!/\S+@\S+\.\S+/.test(values.email)) {

errors.email = 'Email is invalid';

}

if (!values.age) {

errors.age = 'Age is required';

} else if (isNaN(values.age)) {

errors.age = 'Age must be a number';

}

setErrors(errors);

return Object.keys(errors).length === 0;

};

return { handleSubmit, handleChange, values, errors };

};

export default useFormValidation;

import React from 'react';

const Input = ({ name, type, placeholder, value, onChange, error }) => {

return (

<div>

<label>{name.charAt(0).toUpperCase() + name.slice(1)}:</label>

<input

type={type}

name={name}

placeholder={placeholder}

value={value}

onChange={onChange}

/>

{error && <span>{error}</span>}

</div>

);

};

export default Input;

*import* React, { useState } *from* "react";

const *Person* =()=>{

const[*name*, *setName*]= *useState*("");

const[*error*, *setError*]= *useState*("");

const *handleNameChange* =(*e*)=>{

*setName*(*e.target.value*);

*setError*("");

};

const *handleSubmit* =(*e*)=>{

*e.preventDefault*();

if(*name.trim*()===""){

*setError*("Name is required");

*return*;

}

if(!/*^*[*a-zA-Z*]+*$*/*.test*(name)){

*setError*("Name must contain only alphabetic characters");

*return*;

}

if(*name.length* <3){

*setError*("Name must be at least 3 characters long");

*return*;

}

if(*name.length* >20){

*setError*("Name must be at most 20 characters long");

*return*;

}

    // *Add more validation conditions here*

if(*name.includes*("admin")){

*setError*("Name cannot contain 'admin'");

*return*;

}

if(*name.includes*("123")){

*setError*("Name cannot contain '123'");

*return*;

}

if(*name.includes*("password")){

*setError*("Name cannot contain 'password'");

*return*;

}

if(*name.includes*("pass")){

*setError*("Name cannot contain 'pass'");

*return*;

}

if(*name.includes*("word")){

*setError*("Name cannot contain 'word'");

*return*;

}

if(*name.includes*("123")){

*setError*("Name cannot contain '123'");

*return*;

}

if(*name.includes*("secret")){

*setError*("Name cannot contain 'secret'");

*return*;

}

    // *Perform other actions with the valid name*

    // *Reset the name input*

*setName*("");

};

*return* (

<div>

<form *onSubmit*={handleSubmit}>

        <label>

          Name:

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

        </label>

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

      </form>

{error&&<p>{error}</p>}

</div>

);

};

*export* *default* Person;

*import* React, { useState } *from* "react";

*import* "./Person.css";

const *Person* =()=>{

const[*name*, *setName*]= *useState*("");

const[*error*, *setError*]= *useState*({});

const *handleSubmit* =(*e*)=>{

*e.preventDefault*();

if(*name.length* <3)

*setError*({

...error,

*invalidNameLength*:"Length should be greater than 3",

});

else{

const{ *invalidNameLength*,...*rest* }=error;

*setError*(rest);

// *alert("Form submitted");*

}

    //*setError({});*

*return Object.keys*(error)*.length* ===0;

};

const *changeValue* =(*e*)=>{

*setName*(*e.target.value*);

};

*return* (

<div>

<form *onSubmit*={handleSubmit}>

        <label>

          Name

          <input *type*="text" *name*="name" *onChange*={changeValue} />

          {*error.*invalidNameLength && <p>{*error.*invalidNameLength}</p>}

        </label>s

        <input *type*="submit" />

      </form>

</div>

);

};

*export* *default* Person;

**Creating a Custom Dynamic Input Component**

1. **Create a Dynamic componenet**

import React from 'react';

const CustomInput = ({ type, name, value, onChange, placeholder, error }) => {

return (

<div className="input-style">

<input

type={type}

name={name}

value={value}

onChange={onChange}

placeholder={placeholder}

className={error ? 'error' : ''}

/>

{error && <span className="error-message">{error}</span>}

</div>

);

};

export default customInput;

import React, { useState } from 'react';

import CustomInput from './CustomInput';

const MyForm = () => {

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

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [errors, setErrors] = useState({});

const handleInputChange = (e) => {

const { name, value } = e.target;

if (name === 'name') setName(value);

else if (name === 'email') setEmail(value);

else if (name === 'password') setPassword(value);

};

const handleSubmit = (e) => {

e.preventDefault();

if (!name.trim()) {

errors.name = 'Name is required';

}

if (!email.trim()) {

errors.email = 'Email is required';

} else if (!/\S+@\S+\.\S+/.test(email)) {

errors.email = 'Email is invalid';

}

if (!password.trim()) {

errors.password = 'Password is required';

}

if (Object.keys(errors).length === 0) {

// Form submission logic

} else {

setErrors(errors);

}

};

return (

<form onSubmit={handleSubmit}>

<CustomInput

type="text"

name="name"

value={name}

onChange={handleInputChange}

placeholder="Enter your name"

error={errors.name}

/>

<CustomInput

type="email"

name="email"

value={email}

onChange={handleInputChange}

placeholder="Enter your email"

error={errors.email}

/>

<CustomInput

type="password"

name="password"

value={password}

onChange={handleInputChange}

placeholder="Enter your password"

error={errors.password}

/>

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

</form>

);

};

export default MyForm;

.input-wrapper {

margin-bottom: 10px;

}

.input-wrapper input {

width: 100%;

padding: 8px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

transition: border-color 0.3s ease;

}

.input-wrapper input.error {

border-color: red;

}

.error-message {

color: red;

font-size: 14px;

margin-top: 5px;

}

// *Define the custom input component*

const *CustomComponent* =({ *label*, *value*, *onChange* })=>{

*return* (

<div>

<label>{label}</label>

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

</div>

);

};

*import* React *from* "react";

*import* "./CustomComponent";

const *Person* =()=>{

*return* (

<div>

<h1>*Person*</h1>

<CustomComponent *label*="Name" *value*="John" *onChange*={() => {}} />

</div>/

);

};

const formConfig = {

name: {

type: 'text',

name: 'name',

label: 'Name',

placeholder: 'Enter your name',

required: true,

minLength: 3,

maxLength: 50,

validationMessage: 'Name is required and must be between 3 and 50 characters',

},

email: {

type: 'email',

name: 'email',

label: 'Email',

placeholder: 'Enter your email',

required: true,

validationRegex: /\S+@\S+\.\S+/,

validationMessage: 'Email is required and must be valid',

},

password: {

type: 'password',

name: 'password',

label: 'Password',

placeholder: 'Enter your password',

required: true,

minLength: 6,

validationMessage: 'Password is required and must be at least 6 characters',

},

role: {

type: 'select',

name: 'role',

label: 'Role',

options: ['Admin', 'User', 'Guest'],

placeholder: 'Select your role',

required: true,

validationMessage: 'Role is required',

},

};

import React, { useState } from 'react';

import DynamicInput from './DynamicInput';

const MyForm = () => {

const [formData, setFormData] = useState({

name: '',

email: '',

password: '',

role: '',

});

const [errors, setErrors] = useState({});

const handleInputChange = (e) => {

const { name, value } = e.target;

setFormData({ ...formData, [name]: value });

};

const handleSubmit = (e) => {

e.preventDefault();

const newErrors = validateForm(formData);

if (Object.keys(newErrors).length === 0) {

// Form submission logic

console.log('Form submitted successfully');

} else {

setErrors(newErrors);

}

};

const validateForm = (data) => {

const newErrors = {};

for (const fieldName in formConfig) {

const fieldConfig = formConfig[fieldName];

const value = data[fieldName].trim();

if (fieldConfig.required && !value) {

newErrors[fieldName] = `${fieldConfig.label} is required`;

} else if (fieldConfig.minLength && value.length < fieldConfig.minLength) {

newErrors[fieldName] = `${fieldConfig.label} must be at least ${fieldConfig.minLength} characters`;

} else if (fieldConfig.maxLength && value.length > fieldConfig.maxLength) {

newErrors[fieldName] = `${fieldConfig.label} must be at most ${fieldConfig.maxLength} characters`;

} else if (fieldConfig.validationRegex && !fieldConfig.validationRegex.test(value)) {

newErrors[fieldName] = fieldConfig.validationMessage;

}

}

return newErrors;

};

return (

<form onSubmit={handleSubmit}>

{Object.keys(formConfig).map(fieldName => {

const fieldConfig = formConfig[fieldName];

if (fieldConfig.type === 'select') {

return (

<div key={fieldName} className="input-wrapper">

<label>{fieldConfig.label}</label>

<select

name={fieldConfig.name}

value={formData[fieldName]}

onChange={handleInputChange}

className={errors[fieldName] ? 'error' : ''}

>

<option value="">{fieldConfig.placeholder}</option>

{fieldConfig.options.map(option => (

<option key={option} value={option}>{option}</option>

))}

</select>

{errors[fieldName] && <span className="error-message">{errors[fieldName]}</span>}

</div>

);

} else {

return (

<DynamicInput

key={fieldName}

type={fieldConfig.type}

name={fieldConfig.name}

value={formData[fieldName]}

onChange={handleInputChange}

placeholder={fieldConfig.placeholder}

error={errors[fieldName]}

/>

);

}

})}

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

</form>

);

};

export default MyForm;

import React from 'react';

import './App.css';

import MyForm from './MyForm';

function App() {

return (

<div className="App">

<h1>Form Demo</h1>

<MyForm />

</div>

);

}

export default App;

.input-wrapper {

margin-bottom: 10px;

}

.input-wrapper input,

.input-wrapper select {

width: 100%;

padding: 8px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

transition: border-color 0.3s ease;

}

.input-wrapper input.error,

.input-wrapper select.error {

border-color: red;

}

.error-message {

color: red;

font-size: 14px;

margin-top: 5px;

}

**Http request**

function *Person*() {

*async* function *fetchData*() {

    const *response* = *await fetch*("https://jsonplaceholder.typicode.com/users");

*console.log*(response);

*console.log*(*response.*ok);

*console.log*(*response.*status);

    const *data* = *await response.text*();

*console.log*(data);

  }

*fetchData*();

}

*export* *default* Person;

function *Person*() {

  const *fetchData* =()=>{

*fetch*("https://jsonplaceholder.typicode.com/users")

*.then*((*response*)=>{

*return response.json*();

})

*.then*((*data*)=>{

*console.log*(data);

})

*.catch*((*e*)=>{

*console.log*(e);

});

};

*fetchData*();

}

*export* *default* Person;

import axios from 'axios';

function Person() {

const fetchData = () => {

axios.get("https://jsonplaceholder.typicode.com/users")

.then((response) => {

console.log(response.data);

})

.catch((error) => {

console.error(error);

});

};

fetchData();

}

export default Person;

**fetch with local**

*import* React, { useEffect, useState } *from* "react";

function *Person*() {

  const[*data*, *setData*]= *useState*([]);

*useEffect*(() => {

    const *fetchData* =()=>{

*return fetch*("http://localhost:3000/api/users")

*.then*((*response*)=>{

if(!*response.ok*){

*throw* new *Error*("Network response was not ok.");

}

*return response.json*();

})

*.then*((*jsonData*)=>{

*setData*(jsonData);

})

*.catch*((*error*)=>{

*console.error*("Error fetching data:",error);

});

};

*fetchData*();

  }, []); // *Empty dependency array ensures this effect runs only once after the component mounts*

*return* (

    <div *key*="personData">

      <ul>

        {*data.map*((d) => (

          <li *key*={*d.*id}>{*d.*name}</li>

        ))}

      </ul>

    </div>

  );

}

*export* *default* Person;

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

import axios from 'axios';

function Person() {

const [data, setData] = useState([]);

useEffect(() => {

const fetchData = async () => {

try {

const response = await axios.get("http://localhost:3000/api/users");

setData(response.data);

} catch (error) {

console.error("Error fetching data:", error);

}

};

fetchData();

}, []); // Empty dependency array ensures this effect runs only once after the component mounts

return (

<div key="personData">

<ul>

{data.map((d) => (

<li key={d.id}>{d.name}</li>

))}

</ul>

</div>

);

}

export default Person;

*import* axios *from* "axios";

*import* React, { useEffect, useState } *from* "react";

const *Person* =()=>{

const[*persons*, *setPersons*]= *useState*([]);

const[*name*, *setName*]= *useState*("");

const[*age*, *setAge*]= *useState*("");

*useEffect*(()=>{

*fetchPersons*();

},[]);

const *fetchPersons* = *async* ()=>{

*try* {

const *response* = *await axios.get*("http://localhost:3000/api/users");

*setPersons*(*response.data*);

} *catch* (error){

*console.error*(error);

}

};

const *addPerson* = *async* ()=>{

*try* {

const *response* = *await axios.post*("http://localhost:3000/api/users",{

name,

age,

});

*setPersons*([...persons, *response.data*]);

*setName*("");

*setAge*("");

} *catch* (error){

*console.error*(error);

}

};

const *deletePerson* = *async* (*id*)=>{

*try* {

*await axios.delete*(`*/api/users/*${id}`);

*setPersons*(*persons.filter*((*person*)=> *person.id* !==id));

} *catch* (error){

*console.error*(error);

}

};

*return* (

<div>

<h1>*Persons*</h1>

<ul>

{*persons.map*((*person*)=>(

<li *key*={*person.*id}>

            {*person.*name} - {*person.*age}

            <button *onClick*={() => *deletePerson*(*person.*id)}>Delete</button>

          </li>

))}

</ul>

<input

*type*="text"

*placeholder*="Name"

*value*={name}

*onChange*={(e) => *setName*(*e.target.*value)}

      />

<input

*type*="number"

*placeholder*="Age"

*value*={age}

*onChange*={(e) => *setAge*(*e.target.*value)}

      />

<button *onClick*={addPerson}>Add Person</button>

</div>

);

};

*export* *default* Person;

*import* axios *from* "axios";

*import* React, { useState } *from* "react";

const *Person* =()=>{

const[*data*, *setData*]= *useState*([]);

const *fetchData* =()=>{

axios

*.get*("http://localhost:3000/api/users")

*.then*((*response*)=> *setData*(*response.data*))

*.catch*((*error*)=> *console.log*(error));

axios

*.post*("http://localhost:3000/api/users",{ *id*:2, *name*:"John Doe"})

*.then*((*response*)=> *console.log*(*response.data*))

*.catch*((*error*)=> *console.log*(error));

axios

*.put*("http://localhost:3000/api/users/2",{ *name*:"Jane Doe"})

*.then*((*response*)=> *console.log*(*response.data*))

*.catch*((*error*)=> *console.log*(error));

axios

*.delete*("http://localhost:3000/api/users/2")

*.then*((*response*)=> *console.log*(*response.data*))

*.catch*((*error*)=> *console.log*(error));

    //*console.log(data);*

};

*fetchData*();

*return* (

<div>

{*data.map*((*d*, *i*)=>(

<li *key*={i}>{*d.*name}</li>

))}

</div>

);

};

*export* *default* Person;

**Working with http**

All client side application will not directly call the database instead it will make the request to the server side programs it can be webservices, rest API, or we may have graphql whatever technology that might be on the server. But the point is that communicate with database and it fetches the data and sends back to the server..

Now to make the http request in the react either we can use core javascript or we can have axios.or we can use jquery also

The fetch API core java Api is beteer,faster and more efficient way to deal wlkjlkjlkith the http or network requests.Before Es6 we used XMLHTTP request from ES6 onwards we use fetch API.

**SYntax**

**const responsePromise=fetch(url,[optional object]);**

**where url->the resource where we want to make the request.**

**Optional object->for post request**

**If you don’t mention optional fetch API will get get request.**

**const response=fetch(‘http://url’,{**

**method:’post’,**

**headers:{‘contentType’:’appliction/json;charset=utf-8’},**

**body:JSON.stringyfy(data)**

**})**

async function fetchData(){

const response=await fetch(‘https://typicode.com/usere’,);

console.log(response);

let data=await reposne.json();

console.log(data);

}

fetchData();

we can write response.text() if it is image then response.blob().

Instead of this await we can use promise syntax also as follows.

fetch(‘https://typicode.com/usere’,)

.then(res=>{rs.json()})

.then(data=>console.log(dta));

For post

const todo={};

fetch(‘https:url’,{

method:’POST’.

headers:{

“content-type”:”application/json:charset=utf-8”,

},

body:JSON.stringyfy(data)

}).then(r=>r).then(d=>c.l(d))

Fetch API helps in msaking AJAx call

Crud op

Downloading and uploading files

**Creating the new App**

Steps:

1.Using npx create-react-app cruddemo

2.in app.js

Add 2 buttons

3.const function fetchUser(){

const data=fetch(‘url).then(response->response.json())

.then(data=>console.log(data));

}

4. create state variable

const [user,setUser]=useState([]);

import React, { useState } from "react";

import "./App.css";

**function** App() {

**const** [users, setUsers] = useState([]);

**function** fetchData() {

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

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

      .then((json) **=>** {

        setUsers(json);

      });

  }

  return (

    <div>

      <button onClick={fetchData}>User</button>

      {users.map((user) **=>** {

        return <div>{user.name}</div>;

      })}

    </div>

  );

}

export default App;

import logo from './logo.svg';

import './App.css';

import { useState } from 'react';

import User from './components/User';

import Todo from './components/Todo';

import Error from './components/Error';

**function** App() {

**const** [users,setUsers] = useState([]);

**const** [todos,setTodos] = useState([]);

**const** [userData,setUserData] = useState(true);

**const** [errorFlag,setErrorFlag] = useState(false);

**const** fetchUsers=()=>{

    fetch('https://jsonplaceholder.typicode.com/user')

      .then(response **=>** {

        if(response.ok){

          return response.json();

        }else{

          throw new Error("Error!");

        }

      })

      .then(json **=>**{

        setUsers(json);

      })

      .catch((error)**=>**{

        setErrorFlag(true);

      });

      setUserData(true);

  }

**const fetchTodos=()=>{**

    fetch('https://jsonplaceholder.typicode.com/todos')

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

      .then(json **=>** {

          setTodos(json);

        })

        .catch((error)**=>**{

          setErrorFlag(true);

        });

        setUserData(false);

  }

  if(errorFlag){

    return(

      <Error/>

    )

  }

  return (

    <div className="App">

      <div className="topbar"></div>

      <button onClick={fetchUsers}>Users</button>

      <button onClick={fetchTodos}>Todos</button>

      <br/>

      {userData ? users.map((user,index)**=>**{

          return(

            <User user={user}/>

          )

        })

      :

        todos.map((todo,index)**=>**{

          return(

            <Todo todo={todo}/>

          )

        })

      }

    </div>

  );

}

export default App;