**Index**

1. **create a new React application**
2. **React features**
3. **React Prerequisite -JavaScript Fundamental**
4. **create a new React application**

To create a new React application, you can use the create-react-app command-line tool. Here's the step-by-step process:

First, make sure you have Node.js and npm installed on your machine. You can check this by running node -v and npm -v in your terminal. If these commands return a version number, you're good to go. If not, you'll need to install Node.js and npm first.

Install create-react-app globally on your machine. This is a package provided by Facebook to help bootstrap new React applications.

Use create-react-app to create a new React application.

# Step 1: Check Node.js and npm versions

node -v

npm -v

# Step 2: Install create-react-app

npm install -g create-react-app

# Step 3: Create a new React application

npx create-react-app my-app

Replace my-app with the name you want for your new application. After running these commands, you'll have a new directory with the same name as your application, filled with a basic React application structure. You can start the application by navigating into the directory and running npm start.

**2- React features**

1. Hot Reload

2.

1. **React Prerequisite -JavaScript Fundamental**

* Function Declarations and Arrow Functions
* Template Literals
* Short Conditionals: &&, ||, Ternary Operator
* Array Methods: .map(), .filter(), .reduce()
* Object Tricks: Property Shorthand, Destructuring, Spread Operator
* Promises + Async/Await Syntax
* ES Modules + Import / Export syntax
* Spread Operator + Rest Parameters
* Optional Chaining + Nullish Coalescing

**1. Function Declarations and Arrow Functions**

The basis of any React application is the component. In React, components are defined with both JavaScript functions. In React components return JSX elements .

// JavaScript function: returns any valid JavaScript type

function javascriptFunction() {

return "Hello world";

}

// React function component: returns JSX

function ReactComponent(props) {

return <h1>{props.content}</h1>;

}

==> Arrow function

// Arrow function syntax

const MyComponent = (props) => {

return <div>{props.content}</div>;

};

// Arrow function syntax (shorthand)

const MyComponent = (props) => <div>{props.content}</div>;

/\*

In the last example we are using several shorthands that arrow functions allow:

1. No parentheses around a single parameter

2. Implicit return (as compared to using the "return" keyword)

3. No curly braces for function body

\*/

**2. 2. Template Literals**

With the addition of ES6, we were given a newer form of string called a template literal,

which consists of two back ticks `` instead of single or double quotes.

Instead of having to use the + operator,

we can connect strings by putting a JavaScript expression within a special ${} syntax:

xpression (such as a variable) within a special ${} syntax:

/\*

Concatenating strings prior to ES6.Notice the awkward space after the word Hello?

\*/

function sayHello(text) {

return "Hello " + text + "!";

}

sayHello("React"); // Hello React!

/\*

Concatenating strings using template literals.

See how much more readable and predictable this code is?

\*/

function sayHelloAgain(text) {

return `Hello again, ${text}!`;

}

sayHelloAgain("React"); // Hello again, React!

**2.3. Short Conditionals: &&, ||, Ternary Operator**

conditionally show (or hide) JSX elements using simple if statements .

const isLoggedIn = true;

if (isLoggedIn) {

return <div>Welcome back!</div>;

}

return <div>Who are you?</div>;

using Ternary operator

return isLoggedIn ? <div>Welcome back!</div> : <div>Who are you?</div>;

**2.4 Three Array Methods: .map(), .filter(), .reduce()**

map

const programmers = ["Reed", "John", "Jane"];

return (

<ul>

{programmers.map((programmer) => (

<li>{programmer}</li>

))}

</ul>

);

**JSX ( Js + html ) in depth**

**alternative of React.CreateElement you don’t need to import React for jsx**

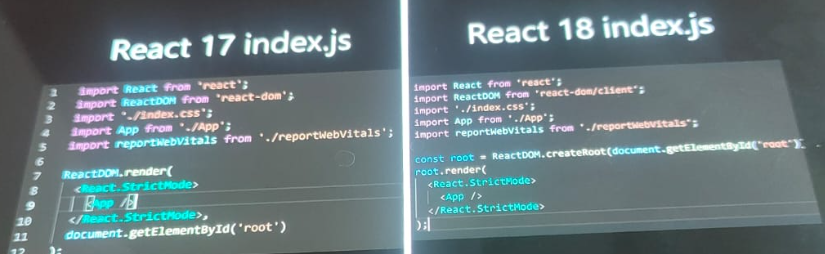
**HTML =>** <h1 class =”abc”>

**JSX =>** <h1 className =”abc”>

React 18

1. Dom Rendering

ReactDOM.render() is no longer supported . However it work better to switch CreateRoot() API .



**Props destructing**

Destructuring was introduced in ES6. It’s a JavaScript feature that allows us to extract multiple pieces of data from an array or object and assign them to their own variables.

const person = {

  firstName: "Lindsay",

  lastName: "Criswell",

  city: "NYC"

}

Before ES6, you had to access each property individually:

console.log(person.firstName) // Lindsay

console.log(person.lastName) // Criswell

console.log(person.city) // NYC

const { firstName, lastName, city } = person;

is equivalent to

const firstName = person.firstName

const lastName = person.lastName

const city = person.city

<div className = "App">

                <Greet active="KAPIL GARG" activeStatus = "CSE"/>

    </div>

const Greet = props =>{

    // Destructuring

    const {active, activeStatus} = props;

    return (

        <div>

              <h3> {active} </h3>

              <h1>{activeStatus}</h1>

        </div>

        )

  }

**Concept of state and useState() hook**

Without useState() react will not track the variable state and updated value will not reflect in dom .

export default function Counter() {

  const [count, setCount] = useState(0);

  function handleClick() {

    setCount(count + 1);

  }

  return (

    <button onClick={handleClick}>

      You pressed me {count} times

    </button>

  );

Update for obj literal

import React, { useState } from "react";

const CounterwithObjLitral = () => {

  const [state, setState] = useState({ cnt: 0, flag: false });

  const incrementCount = () => {

    setState((prevState) => ({

      ...prevState,

      cnt: prevState.cnt + 1,

    }));

  };

  const decrementCount = () => {

    setState((prevState) => ({

      ...prevState,

      cnt: prevState.cnt - 1,

    }));

  };

  const toggleFlag = () => {

    setState((prevState) => ({

      ...prevState,

      flag: !prevState.flag,

    }));

  };

  return (

    <div>

      <p>Count: {state.cnt}</p>

      <p>Flag: {state.flag ? "true" : "false"}</p>

      <button onClick={incrementCount}>Increment Count</button>

      <button onClick={decrementCount}>decrement Count</button>

      <button onClick={toggleFlag}>Toggle Flag</button>

    </div>

  );

};

export default CounterwithObjLitral;

**Lazy initial state/ load**

Normal

export default function Counter() {

  function initialState() {

    console.log('Initial state called -Expensive process' + Date.now());

    return 0;

  }

  const [count, setCount] = useState(initialState());

  function handleClick() {

    setCount(count + 1);

  }

  return (

    <button onClick={handleClick}>

      You pressed me {count} times

    </button>

  );

}

With Lazy initialization / load

export default function Counter() {

  function initialState() {

    console.log('Initial state called -Expensive process' + Date.now());

    return 0;

  }

  const [count, setCount] = useState(()=>initialState());

  function handleClick() {

    setCount(count + 1);

  }

  return (

    <button onClick={handleClick}>

      You pressed me {count} times

    </button>

  );

**Basic eventHandling & Parameter passing**

Use fat arrow or return function rather calling

     <button onClick={() => setCount(count + 1)}>Increment</button>

      <button onClick={() => setCount(count - 1)}>Decrement</button>

**Apply internal and external css**

**External Stylesheet**

import "./styles.css";

use directly using className

**Inline CSS**

To implement inline CSS, you can create an object containing style references, which can be then called using the style attribute. For example:

const styles = {

  section: {

    fontSize: "18px",

    color: "#292b2c",

    backgroundColor: "#fff",

    padding: "0 20px"

  },

  wrapper: {

    textAlign: "center",

    margin: "0 auto",

    marginTop: "50px"

  }

}

It is then added to an element like this:

<section style={styles.section}>

  <div style={styles.wrapper}>

  </div>

</section>

**Primitive Types & Object Literal with useState()**

**Primitive Types with useState()**

import React, { useState } from "react";

const Counter = () => {

  const [count, setCount] = useState(0);

  const styles = {

    section: {

      fontSize: "18px",

      color: "#292b2c",

      backgroundColor: "#fff",

      padding: "0 20px",

    },

    wrapper: {

      margin: "0 auto",

      marginTop: "50px",

    },

  };

  // const increment = () => {

  //     setCount(count + 1);

  // };

  // const decrement = () => {

  //     setCount(count - 1);

  // };

  return (

    <>

      <section style={styles.section}>

        <div style={styles.wrapper}>

          <h1>Counter: {count}</h1>

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

            <button onClick={decrement}>Decrement</button> \*/}

          <button onClick={() => setCount(count + 1)}>Increment</button>

          <button onClick={() => setCount(count - 1)}>Decrement</button>

        </div>

      </section>

    </>

  );

};

export default Counter;

**Object Literal with useState()**

import React, { useState } from "react";

const CounterwithObjLitral = () => {

  const [state, setState] = useState({ cnt: 0, flag: false });

  const incrementCount = () => {

    setState((prevState) => ({

      ...prevState,

      cnt: prevState.cnt + 1,

    }));

  };

  const decrementCount = () => {

    setState((prevState) => ({

      ...prevState,

      cnt: prevState.cnt - 1,

    }));

  };

  const toggleFlag = () => {

    setState((prevState) => ({

      ...prevState,

      flag: !prevState.flag,

    }));

  };

  return (

    <div>

      <p>Count: {state.cnt}</p>

      <p>Flag: {state.flag ? "true" : "false"}</p>

      <button onClick={incrementCount}>Increment Count</button>

      <button onClick={decrementCount}>decrement Count</button>

      <button onClick={toggleFlag}>Toggle Flag</button>

    </div>

  );

};

export default CounterwithObjLitral;

**Class component**

The React.Component needs to extended to create a class component

Instead of return a render() method .

import React, { Component } from "react";

class CounterClass extends Component {

  state: { counter: number } = {

    counter: 0,

  };

  increment = () => {

    let c = this.state.counter;

    c++;

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

      return { counter: c };

    });

  };

  decrement = () => {

    let c = this.state.counter;

    c--;

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

      return { counter: c };

    });

  };

  render() {

    return (

      <div>

        <h2>Count: {this.state.counter}</h2>

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

        <button onClick={this.decrement}>Decrement</button>

      </div>

    );

  }

}

export default CounterClass;

**Component life cycle methods**