WEEK 5

Nischal k
497CS23033

1: Introduction to React

• Introduction to React

React is a popular JavaScript library used for building user interfaces, particularly for single-page applications where you want a fast and interactive user experience. React allows you to create reusable UI components, which are the building blocks of your application.

• Why Use React?

• Component-Based: You can build encapsulated components that manage their own state, then compose them to make complex UIs.

• Declarative: React makes it easy to create interactive UIs by updating and rendering the right components when your data changes.

• Learn Once, Write Anywhere: You can use React to create applications for the web, mobile, and even desktop.

• Steps to Install React

1. Install Node.js and npm:

◦ First, you need to install Node.js, which includes npm (Node Package Manager). You can download it from Node.js official website.

◦ After installing, check if Node.js and npm are installed by running the following commands in your terminal:

◦ node -v

◦ npm -v

2. Create a React App:

React provides a tool called create-react-app to set up a new React project easily.

Open your terminal or command prompt and run the following command:

```
npx create-react-app my-app
```

WEEK 5

Nischal k
497CS23033

This command creates a new directory called my-app with a basic React setup.

3. Navigate to Your Project Folder:
Go into your newly created project

directory:bash cd my-app

4. Start the Development Server: Start your React app by running:sql npm start

This will open your React app in your web browser at http://localhost:3000.

• Understanding React Files and Components

When you create a React app using create-react-app, you'll notice several files and folders.

Here's a simple explanation of what they are:

• node_modules/: Contains all the npm packages your project depends on.
• public/: This folder contains static files, like index.html, where your React app is injected.
• src/: This is where you'll write all your React code. It includes:
◦ index.js: The entry point of your React app. It renders the root component (App.js) into the DOM.
◦ App.js: A sample component that serves as the root component for your app.
◦ App.css: CSS file for styling the App.js component.
◦ index.css: Global styles for your app.
◦ logo.svg: A React logo that you can see in your default app.

• React Components
In React, everything is a component. A component is a reusable piece of code that represents part of the user interface. Components can be either functional or class-based:
Functional Components: Simple functions that return JSX (HTML-like syntax). They are easier to write and understand.

```
javascript
function Welcome(props) {
return <h1>Hello, {props.name}</h1>;
}
```

Class Components: More powerful and can manage their own state, but are more complex.

javascript

WEEK 5

Nischal k
497CS23033

```
class Welcome extends React.Component {
render() {
return <h1>Hello, {this.props.name}</h1>;
}}
```

• Simple React program with explanation: import React from 'react';
import ReactDOM from 'react-dom/client'; import './index.css';

```
// Creating a simple React component
function Greeting() {
return <h1>Hello, Welcome to React!</h1>;
}
// Getting the root element from the HTML file
const rootElement = document.getElementById('root');
// Creating a root using React DOM
const root = ReactDOM.createRoot(rootElement);
// Rendering the Greeting component inside the root element
root.render(<Greeting />);
```

Description of Elements

1. import React from 'react';:
   ◦ This line imports the React library, which is necessary to create React components.
2. import ReactDOM from 'react-dom/client';:
   ◦ This line imports the ReactDOM library, which is used for rendering React components to the DOM (the Document Object Model, which represents the page content in the browser).
3. import './index.css';:
   ◦ This imports a CSS file that can be used to style your React components. In this example, it's used to style the application globally.

4. function Greeting() { return <h1>Hello, Welcome to React!</h1>; }:
   ◦ Here, we define a simple functional component called Greeting. It returns some JSX, which is essentially HTML written inside JavaScript. This component displays a heading with the text "Hello, Welcome to React!”.
5. const rootElement = document.getElementById('root');:
   ◦ This line selects the HTML element with the ID root. This is where our React app will be displayed. Usually, this root element is defined in the index.html file within the public folder.

6. const root = ReactDOM.createRoot(rootElement);:

Nischal k
497CS23033

◦ This creates a React root, which is a starting point for your React application. The root is attached to the rootElement we selected earlier.

7. root.render(<Greeting />);:
◦ This line renders the Greeting component inside the root element. The Greeting component is what will be shown on the webpage.

Explanation

- React Component: The Greeting function is a React component, a small, reusable piece of the user interface.
- JSX: The HTML-like syntax inside the return statement is called JSX, which stands for JavaScript XML. It allows you to write HTML elements in JavaScript.
- Rendering: The root.render() function takes a React component (like Greeting) and displays it inside the specified DOM element (rootElement).

2: Write a program to demonstrate props in react.

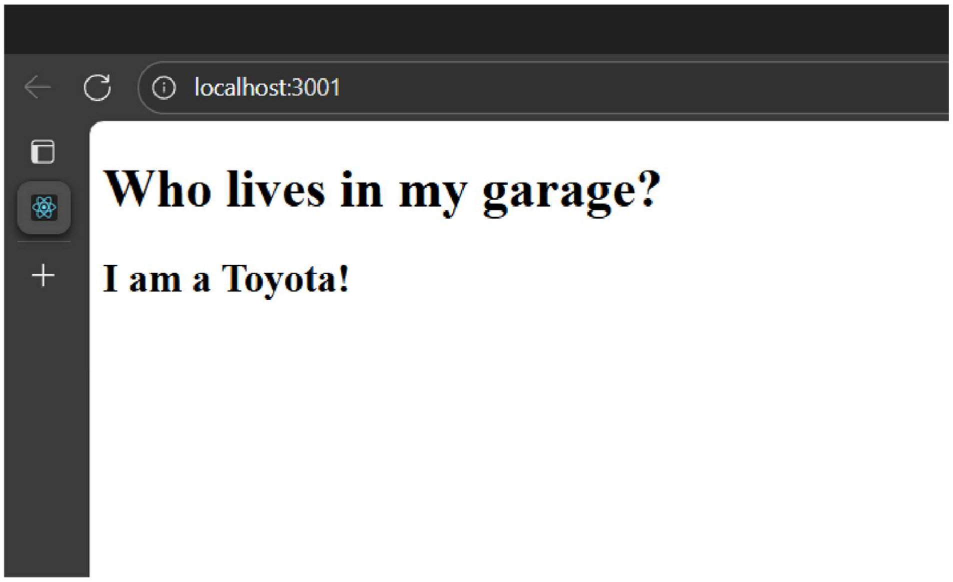
- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand="Toyota" />
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Garage />);
```

WEEK 5

Nischal k
497CS23033

//OUTPUT:



localhost:3001

Who lives in my garage?

I am a Toyota!

3: Write a program to to demonstrate State in react

- React components have a built-in state object.
- The state object is where you store property values that belong to the component.
- When the state object changes, the component re-renders.;

The state object is initialized in the constructor:

```
class Car extends React.Component {
  constructor(props)          {
super(props);     this.state =
{brand: "Ford"};
  }
```

The state object can contain as many properties as you like: class Car extends React.Component {

```
  constructor(props)
{         super(props);
this.state       =       {
brand:           "Ford",
model:    "Mustang",
color: "red",
      year: 1964
```

```
   };
  }
```

Using the state Object

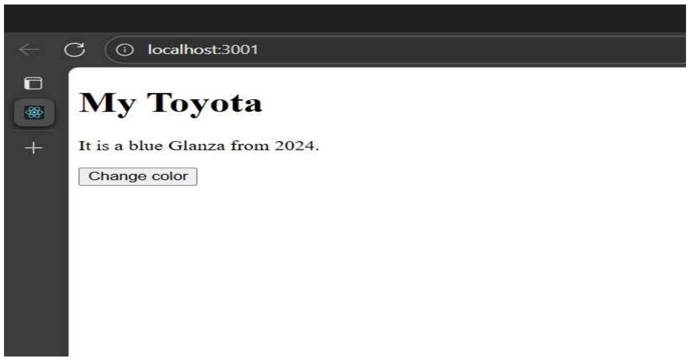
```
render()        {
return (
     <div>
       <h1>My {this.state.brand}</h1>
       <p>
         It          is          a
{this.state.color}
{this.state.model}
from {this.state.year}.
       </p>
     </div>
   );
```

- Refer to the state object anywhere in the component by using the this.state.propertyname syntax:
- When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).

Ex:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

class Car extends React.Component {
  constructor(props)                {
  super(props);
    this.state    =    {
brand:   "   Toyota",
model:  "  Glanza  ",
color: " blue",
      year: 2024
    };
  }
  changeColor = () => {
    this.setState({color: "cafe white"});
  }
  render()        {
return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It          is          a
{this.state.color}
{this.state.model}
from {this.state.year}.
        </p>                <button
type="button"
onClick={this.changeColor}
        >Change color</button>
      </div>
    );
  }
}
const    root    =    ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

//OUTPUT:



```
localhost:3001
My Toyota
It is a blue Glanza from 2024.
```