Dayananda Sagar University Bengaluru

MODULE 4—(refer notes)

React.JS basic,REST API, your first React Web Application

Introduction to React

- React is a JavaScript library created by Facebook.
- React is a tool for building UI components.
  - **What is React?**
  - React is a tool (or library) that helps you build websites and apps. It lets you build parts of a page and easily update them when something changes, like when you click a button.
- React is an open-source frontend JavaScript library which is used for building user interfaces especially for single page applications.
- It is used for handling view layers for web and mobile apps.
- ReactJS is a simple, feature rich, component based JavaScript UI library.
- It can be used to develop small applications as well as big, complex applications
- React versions The initial version, 0.3.0 of React is released on May, 2013 and the latest version, 17.0.1 is released on October, 2020.

*How does React Work?

React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

React only changes what needs to be changed!

React finds out what changes have been made, and React changes only what needs to be changed.

*Features

The salient features of React library are as follows:

- Solid base architecture
- Extensible architecture
- Component based library
- JSX based design architecture
- Declarative UI library

Benefits and Applications of React

Benefits

- Easy to learn
- Easy to adapt in modern as well as legacy application
- Faster way to code a functionality
- Availability of large number of ready-made component
- Large and active community

Applications

- Facebook, popular social media application
- Instagram, popular photo sharing application
- Netflix, popular media streaming application
- Code Academy, popular online training application
- Reddit, popular content sharing application

Dayananda Sagar University Bengaluru

installation of React.js

Step-by-Step Instructions:

Install Node.js and npm

Before you can use React, you need to have Node.js and npm (Node Package Manager) installed on your computer.

- Node.js is required to run JavaScript on the server side, and npm is the tool used to install libraries (like React) for JavaScript.
  - Download and install Node.js from the official website: https://nodejs.org/.

Once Node.js is installed, you can check if npm is installed by opening the command line (Command Prompt or Terminal) and running the following commands:

```
node -v
npm -v
```

  - These commands will show the version of Node.js and npm installed. If you see the version numbers, you have installed them successfully.

Install Create React App (Optional)

The **Create React App** tool is an easy way to set up a React project with a lot of useful features already configured for you (such as Webpack, Babel, etc.).

You can install **Create React App** globally so that you can use it from anywhere on your computer:

Open your command line (Command Prompt or Terminal) and run the following command:

```
npm install -g create-react-app
```

- **`-g`** means "globally", so it will be available in any folder on your computer.

Create a New React Project

Once **Create React App** is installed, you can use it to generate a new React app:

Navigate to the folder where you want to create your new project (e.g., `C:/Projects` or `Documents`).

```
cd path/to/your/folder
```

Run the following command to create a new React app:

```
npx create-react-app <appname>
```

- Replace `<appname>` with the name of your app (e.g., `my-app`).
- **`npx`** is a tool that comes with npm (npm 5.2+). It allows you to run packages without installing them globally, making it easier to use the latest version of Create React App each time.

For example, to create a project named `my-app`:

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

This will automatically:

- Set up a new folder named `my-app`
- Install React and other necessary libraries
- Set up everything you need to start developing a React app

4. **Navigate to Your Project Folder**

Once the project has been created, go into the newly created project folder by running:

```
cd my-app
```

5. Start the Development Server

```
npm start
```

- This will start a development server that you can access by opening your browser and navigating to http://localhost:3000.
- You should now see the default React app (a welcome page with the React logo).

The Directory Structure

After creating the React app, your project folder will look something like this:

```
my-app/
├── node_modules/
├── public/
│   └── index.html
├── src/
│   └── App.js
│   └── index.js
├── package.json
└── README.md
```

- **`src/`**: This is where you put your React code (JavaScript files).
- **`public/`**: Contains static files like `index.html`.
- **`package.json`**: Contains information about your project (like dependencies and scripts).

6. Edit Your First React Component

Now that your app is running, you can start editing it!

- Open the `src/App.js` file in any code editor (like VS Code).
- Make changes to the code in `App.js`, save the file, and your browser should automatically update the page.

Components in React

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. **Conceptually, components are like JavaScript functions.** They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Components – The Building Blocks

Think of React as a way to build things out of **lego blocks**. Each block is a **component**.

- A **component** is a small piece of a page (like a button, a header, or a message).
- You can combine many components to create a full page, just like you can build a big Lego house using small Lego blocks.

Example:

- You could have a **Button** component, a **Text** component, and a **Header** component.

JSX – Mixing JavaScript and HTML

React uses something called **JSX**, which looks like HTML, but it’s actually **JavaScript**.

- JSX lets you write HTML-like code inside your JavaScript.

Example:

jsx

```
const element = <h1>Hello, World!</h1>;
```

This is like saying: "I want a heading that says 'Hello, World!'"

PROPS

1. React props (properties) facilitate data transfer from parent to child components.
2. Data flows unidirectionally, ensuring a clear direction of information in React applications.
3. Props are immutable, meaning child components cannot modify the data received from parents.
4. Child components access props through their function parameter.
5. You can also pass JSX as props to another component.

Props – Passing Information

Think of **props** like **letters** you send between components.

- **Props** let you send information from one component to another.
- They help you pass messages from parent components (big pieces) to child components (small pieces).

Example:

jsx

```jsx
function Greeting(props) {
return <h1>Hello, {props.name}!</h1>;
}

function App() {
return <Greeting name="Alice" />;
}
```

In this example, we send the `name` **prop** from `App` to `Greeting`, and the message becomes "Hello, Alice!".

Updating the UI – React Does the Work for You

React is smart. When the **state** changes, React automatically updates the page, without you needing to tell it to.

- If you change something in the state (like clicking a button), React will **re-render** only the parts that need to change, making it super fast!

Example, this code renders “Hello, Sara” on the page:

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

const element = <Welcome name="Sara" />;
ReactDOM.render(
element,
document.getElementById('root') );
```

What happens in this example:

1. We call ReactDOM.render() with the <Welcome name="Sara" /> element.
2. React calls the Welcome component with {name: 'Sara'} as the props.
3. Our Welcome component returns a <h1>Hello, Sara</h1> element as the result.
4. React DOM efficiently updates the DOM to match <h1>Hello, Sara</h1>.

REST API

REST API (Representational State Transfer) is a way to interact with remote servers over the internet. It allows different software applications (like your React app) to communicate by sending requests (like **GET**, **POST**, **PUT**, **DELETE**) to access or manipulate data.

- In **React.js**, we often use **REST APIs** to fetch data from a server, such as getting a list of items, submitting a form, or updating information on a database.
- To communicate with a REST API in React, we typically use the `fetch()` function or libraries like **Axios** to make HTTP requests.

```
import React, { useEffect, useState } from 'react';

function App()
                    {
                      const [joke, setJoke] = useState('');

                        useEffect(     () => {
                        fetch('https://official-joke-api.appspot.com/random_joke')
                          .then(response => response.json())
                          .then(data => setJoke(data.setup + " - " + data.punchline));


                             },   []                );

  return <h1>{joke}</h1>;
                              }

export default App;
```

What this does:

1. `useState` holds the joke data.
2. `useEffect` runs when the component loads. It fetches the joke from the Joke API.
3. The joke is then displayed in an `<h1>` tag.

Output

```
Why don't skeletons fight each other? - They don't have the guts.
```

In this code:

1. **The API (https://official-joke-api.appspot.com/random_joke)** returns some **data** when we fetch it.

This **data** is a **JSON object**, like this:

```
{
"setup": "Why don't skeletons fight each other?",
"punchline": "They don't have the guts."
}
```

1. We then use that **data** inside

`.then(data => setJoke(data.setup + " - " + data.punchline))` to display the joke.