REACT JS

Module : 9 React JS Intro

(1) What is React JS ?

➔ ReactJS is a JavaScript library used for building reusable UI components. React is a JavaScript library created by a Facebook.

React is a tool for building UI Components.

Instead of manipulating the browser's DOM directly, React creates a **Virtual DOM** in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

React is a JavaScript library for building user interfaces.

React is used to build **Single-Page Application**.

React allows us to create Reusable UI Components.

(2) What is NPM in React JS ?

➔ NPM stands for Node Package Manager.

It's like an online store for code that helps React developers manage and use libraries and tools in their projects.

Node : Node.js is a plateform that lets you run JavaScript code outside of a web browser. It's commonly used in the React ecosystem for building and managing web applications on servers.

Package : A package is a bundle of code that serves a specific purpose. In React, you often need additional code to help with tasks like handling dates, making HTTP requests, or managing state. These packages can be easily found on NPM.

Manager : NPM is like a manager or organizer for these packages. It keeps track of which packages your React project depends on and makes it easy to install, update and remove them.

NPM is a handy tool that makes it easier for React developers to find, use and manage code from other developers, saving them time and effort in building web applications.

Example : if I want to use a special button component in my React app that someone else has created and shared, you can search for it on NPM, “add it to my cart” by installing it, and then use it in my project.

(3) What is Role of Node JS in React JS ?

➔ Node JS is like the backstange worker that helps make the show (your React app) run smoothly.

- **Server-Side Logic :** Node JS is a server-side JavaScript runtime environment. In React applications, most of the code runs in the user’s web browser (the client-side). However, there are tasks that are better handled on the server, like processing forms, handling user authentication and intercting with databases. Node JS allows you to write JavaScript code on the server side to manage these tasks.
- **APIs :** React apps often need to communicate with servers to fetch data or send data back. Node JS is excellent for creating APIs (Application Programming Interfaces) that enable your React app to request and exchange data with the server. It acts as the bridge between your React Front-End and the Server.
- **Build Tools** : Node JS used to run various build tools and development servers. For example, tools like Webpack and Babel, which are crucial for building and transpiling React code are typically run using Node JS.
- **Development Environment** : Node JS provides a runtime environment for developing and testing your React app locally. You can set up a Node JS server your React app during development, making it easy to see changes in real-time.
- **Dependency Management** : As mentioned in a previous response, Node JS also come with NPM, which is used to manage and install packages and libraries required for building and running React apps.

(4) What is CLI command in React JS ?

➔ React have its own CLI but currently they are only supporting creating an app (create-react-app).

- create-react-app used to generate the boilerplate version of a React application thru command line.
- npx create-react-app my-app or npm create-react-app my-app
- create-react-app has taken care of setting up the main structure of the application as well as a couple of developer settings.
- Most of what you see will not be visible to the visitor of your web app.
- React uses a tool called webpack which transforms the directories and files here into static assets. Visitors to your site are served those static assets.

(5) What is Components in React JS ?

➔ Components are independent and reusable bits of code.
They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Function Component :

- Here is the same example as above, but created using a Function component instead.
- A Function component also returns HTML and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand and will be preferred in this tutorial.

Example :

Create a Function component called Jainish :

```
Function Jainish()
{
    return
    <h1> Hi, I am Jainish. </h2>
};
```

Class Component :

- A class component must include the extends React.component statement. This statement creates an inheritance to React.component and gives your component access to React.component's functions.
- The component also requires a render() method, this method returns HTML.

Example :

Create a Class component called Patel :

```
Class Patel extends React.Component
{
    render()
    {
        return
        <h1> Hello, My Surname is Patel. </h1>
    }
};
```

(6) What is Header and Content Components in React JS ?

➔

Header Component :

- The Header component typically represents the top section of a web page or an application.
- It often contains elements like the site's logo, navigation menus, user profile information, and any other content that you want to display consistently at the top of the page.
- The Header component is reusable across different pages or sections of your application, allowing you to maintain a consistent look and feel throughout.

Example of Header Component :

```
Import React from ‘react’;


Const Header = () => {
return (
        <header>
                <nav>
                        <ul>
                                <li><a href=”#”> Home </a></li>
                                <li><a href=”#”> About </a></li>
                                <li><a href=”#”> Services </a></li>
                                <li><a href=”#”> Contact Us </a></li>
                        </ul>
                </nav>
        </header>
) };
Export default Header
```

Content Component :

- The Content component is a more generic term used to describe the main section of a web page or an application.
- It's where the primary content or functionality of the page is displayed.
- This could include text, image, forms, lists or any other type of content that makes up the core of the page's purpose.
- Content components are often designed to be flexible and reusable, allowing you to swap out different content while keeping the same layout and structure.

Example of Content Component :

```
Import React from 'react';

Const Content = () => {
return (
        <div className="content">
                <h1> Welcome to Our Website. </h1>
                <p> Lorem ipsum dolor sit amet, consecterur adipiscing elitinng ... </p>
                { /* Additional Content Elements. */ }
        </div>
) };
export default Content
```

(7) How to install React JS on Windows, Linux Operating System ? How to install NPM and How to check version of NPM ?

➔

Installing React JS and npm on Windows :

- Install Node.js : React.js requires Node.js, which includes npm. You can download the windows installer from the official Node.js website : https://nodejs.org/
- Visit the website and download the LTS (Long Term Support) version for Windows.
- Run the installer and follow the installation instructions.

Verify Installation

- After the installation was completed, then open a Command Prompt or Powershell window and enter the following commands to verify that Node.js and npm have been installed successfully :

  Node -v

  npm -v

Create a React App

- To Create a new React Application, you can use the following command in the Command Prompt or Powershell :
- npm create-react-app my-react-app(React App Folder Name)
- This will create a new directory named by my-react-app containing a basic React Project.
- Navigate to the App Directory : Use the following command to navigate to the app directory :
- cd my-react-app

Start the development server by running

- npm start
- This will start the React development server and you can access your app in a web browser at http://localhost:3000

Installing React.js and npm on linux

Install Node.is

- Open a terminal and use the following commands to install Node.js and npm using a package manager appropriate for your Linux distribution.
  Here, We'all use apt for Ubuntu/Debian-based systems and dnf for Fedora

For Ubuntu / Debian

- Sudo apt update
- Sudo apt install nodejs npm
  - For Fedora :
  - Sudo dnf install nodejs npm
- Verify Installation After installation, verify that Node.js and npm are installed bu running :
- node -v
- node -v

Create a React App

- Follow the same steps as mentioned for windows to create and navigate to a new React App.
- Start the Development Server : Similarly, start the development server using :
  - npm start
- Remember that package manager commands (like apt or dnf) might require administrative privileges, so use sudo as needed.
- Checking the Version of npm

- To check the version of npm, simply open a terminal and run :
- npm -v

(8) How to check version of React JS ?

➔ Three ways to find out the React version.

- Using package.json file
- Using command line
- Using the version property of default import from React

❖ Using package.json file :

- The package.json contains metadata about our project.
- It is created by default when we create our React project.
- We can create a react app using the command mentioned below.
- Npm create-react-app name_of_the_app

❖ Using the command line :

- We can easily check the React version by using the command mentioned below on our command line.
  - npm view react version

❖ Using the version property of default import from React :

- The default import from React library is an object that has a version property on it.
- We can use this property inside out JSX elements in our desired manner.
  - Import React from 'react';
  - Let a = React.version

(9) How to change in components of React JS ?

➔ **Update in state :**

- The state change can be from a prop or setState change to update a variable(say).
- The component gets the updated state and React re-renders the omponent to reflect the change on the app.

Update in prop :

- Likewise the change in prop leads to state change and state change leads to re-rendering of the component by React.

Re-rendering of parent component :

- Whenever the components render function is called, all its subsequent child components will re-render, regardless of whether their props have changed or not.

(10) How to Create a List View in React JS ?

➔ **Step 1 :**

- Create a list of elements in React in the form of an array ans store it in a variable.
- We will render this list as an unordered list element in the browser.

Step 2 :

- We will then traverse the list using the JavaScript map() function and updates elements to be enclosed between <li> elements.

Step 3 :

- Finally we will wrap this new list within <ul> elements and render it to the DOM.

Example :

```
Import React from 'react';
Import ReactDOM from 'react-dom';

const numbers = [ 1,2,3,4,5 ];
const updateNums = number.map((number) => {
    return <li> {number} </li>;
});

ReactDOM.render(
    <ul>
        { updateNums };
    </ul>,
    document.getElementByID('root')
);
```

Module : 10 List and Hooks

(1) Explain Life cycle in Class Component and functional component with Hooks ?

➔

Lifecycle in Class Components :

- **Mounting :** When a Class Component is created and instead into the DOM for the first time, it goes through these phases :
  - **Constructor** : This is like the creature's birthplace, where we set its initial characterstics and state.
  - **Render** : Here, we create the creature's appearance (UI) based on its current state and props.
  - **componentDidMount** : After the creature is born and shown to the world, this is where we can make it do things like fetching data or setting up timers.

- **Growth and Updates (Updating)**: As the creature grows and experiences changes, it goes through these phases:
  - **shouldComponentUpdate**: This is like the creature thinking about whether it should change based on new props or state.
  - **render**: If it decides to change, it updates its appearance (UI) in this step.
  - **componentDidUpdate**: After the change, it might want to do something special, like saving its current state.

- **Aging and Farewell (Unmounting)**: When the creature is no longer needed and removed from the DOM, it goes through:
  - **componentWillUnmount**: This is like the creature preparing for its farewell party, where it can clean up resources before it's gone.

Lifecycle in Functional Components with Hooks :

- **Birth and Growth (Mounting and Updating)**: Functional components with hooks have two main phases:

- **useState**: This is where you set up the creature's initial characteristics and state.
- **useEffect**: Similar to componentDidMount and componentDidUpdate in class components, this is where you can make your creature do things when it's born and whenever it changes. It's like a combination of those lifecycle methods.

❖ **Aging and Farewell (Unmounting)**: When the creature is no longer needed, you can use:

- **useEffect cleanup**: This is where you can clean up resources, similar to componentWillUnmount in class components.

Module : 11 React – Applying Redux

(1) What is Redux ?

➔ Redux is a State Management Library.

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to Read Data from a Redux Store, and Dispatch Actions to the Store to Update Data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model.

Redux was inspired by Flux. Redux studied the Flux studied the Flux architecture and omitted unnecessary complexity.

- Redux does not have Dispatcher concept.
- Redux has an only Store whereas Flux has many stores.
- The Action objects will be received and handled directly by Store.

(2) What is Redux Thunk used for ?

➔ Redux Thunk is a middleware for the Redux JavaScript Library. It allows you to return functions instead of actions, which allows for delayed actions. This is useful for handling actions that might not be synchronous.

Here are some use cases for Redux Thunk :

- Delayed actions : You can delay the dispatch of an action, or only dispatch if a certain condition is met.
- Working with promises : You can use Redux Thunk to make asynchronous API calls.
- Abstracting storage logic : Redux Thunk uses a pattern that abstracts storage logic from component to services, action builders and actions.

(3) What is Pure Component ? When to use Pure Component over Component ?

➔ Pure Component is the type of component which re-renders only when the props passed to it changes and not even if its parent component re-renders of if the shouldComponentUpdate() method is called. It is greatly used to enhance the performance of a web application.

Pure Components are also called “stateless components” or “dumb components”. They optimize performance by reducing unnecessary re-renders.

We can use Pure Components in the following scenarios :

- When a component’s output depends only on its state and props.
- When there are no side effects, such as network requests or interactions with the DOM.
- When children components are classes and are being passed only some of the props of their parents.

(4) What is the second argument that can optionally be passed to setState and what is its purpose ?

➔ The Second argument that can optionally be passed to setState is a callback function. This function is called after the state has been updated and the component has been re-rendered. This can be useful for performing any actions that need to be done after the state has been updated, such as updating the DOM or making an API call.

Generally we recommended using componentDidUpdate() for such logic instead.

Example of how to use a callback function with setState :

```
setState((prevState, props) => {
        // Updated the State
        return {
                count: prevState.count + 1,
                };
        }, () => {
                // Do something after the State has been updated.
                Console.log(‘The state has been updated.’);
        });
```