التكنولوجيا التطبيقية
APPLIED TECHNOLOGY
وزارة التربية والتعليم والتعليم الفني

مدارس we للتكنولوجيا التطبيقية

IT Department

Web Development and Programming

WORKSHOP

تطوير المواقع والبرمجيات

الصف الثالث

Unit 17 ReactJS

Web Development and Programming

Workshops

1st.

2024 - 2025

Unit 17 RectJS
الصف الثالث

Web Development and Programming
W o r k s h o p s

Unit 17

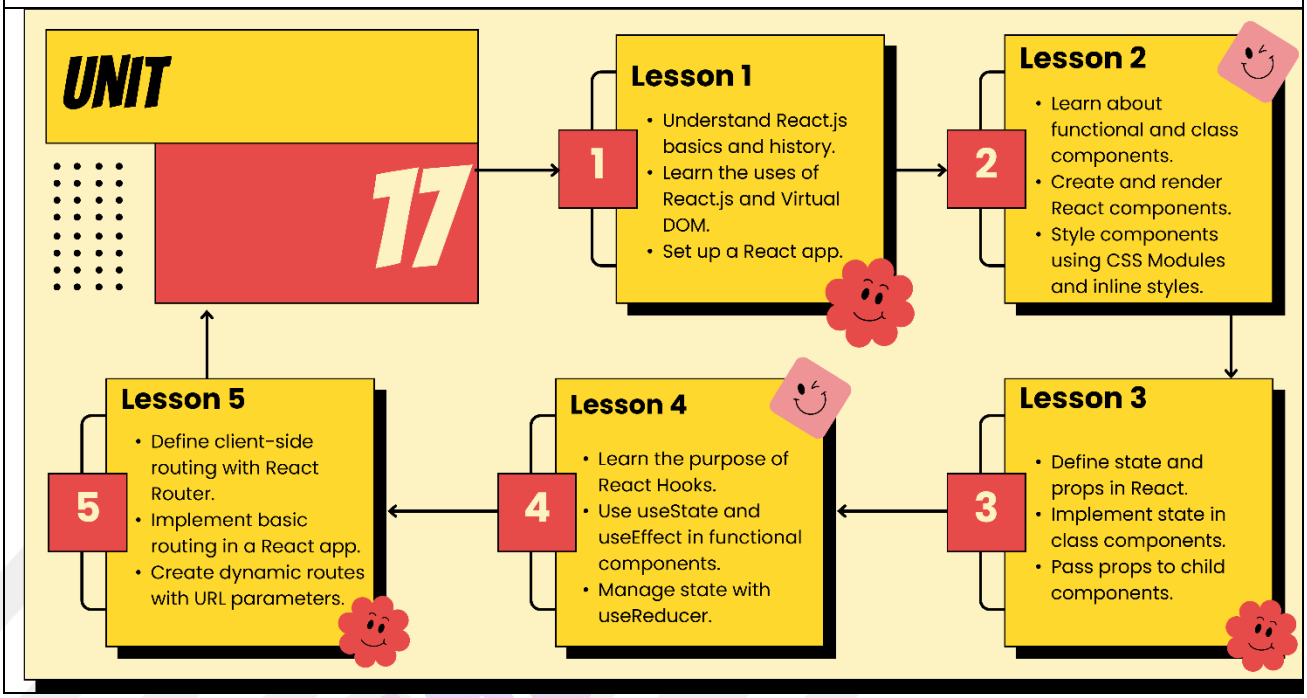
ReactJS

Unit	17
Name	ReactJS
Goals / Outcomes	**By the end of this Unit, students should be able to:** ➢ **Remembering** 1. Recall the basic concepts, history, and uses of React.js, including the Virtual DOM, React components, state, props, hooks, and routing. 2. Identify the advantages and differences of React components, Hooks, and React Router. ➢ **Understanding** 1. Explain the principles of React, including how it integrates with the Virtual DOM and the importance of state and props in building dynamic UIs. 2. Describe the use and differences between functional and class components, state management techniques, and the purpose of different Hooks (useState, useReducer, useContext, useEffect). 3. Understand client-side vs. server-side rendering and how React Router manages routing in single-page applications. ➢ **Applying** 1. Set up a React development environment and create basic React applications using create-react-app. 2. Implement and manage state in functional and class components using useState, useReducer, and useContext. 3. Create and render React components, establishing parent-child relationships and passing data through props.

	4. Use React Router to create a multi-route single-page application with navigation and dynamic routes. 5. Apply various styling methods (CSS Modules, inline styles, JavaScript objects) and integrate third-party libraries like Bootstrap. 6. Utilize useEffect for managing side effects and component lifecycle events. ➢ **Analyzing** 1. Analyze the role of the Virtual DOM in performance, and compare functional and class components. 2. Evaluate the differences and appropriate use cases for useState vs. useReducer. 3. Compare the advantages and implications of using SSR vs. CSR in React applications. 4. Examine the impact of different routing and state management techniques on application behavior and structure. ➢ **Evaluating** 1. Assess the suitability and impact of using React.js, Hooks, and React Router for different types of projects. 2. Evaluate the benefits of various state management approaches and the use of useContext for sharing state across components. 3. Judge the effectiveness of React Router in managing single-page applications and the pros and cons of SSR and CSR. ➢ **Creating** 1. Build a dynamic React application that utilizes components, state, props, and Hooks for interactivity and state management. 2. Design complex component hierarchies with effective state sharing using useContext.

	3. Develop a single-page application with multiple routes, dynamic navigation, and SSR using React Router. 4. Implement a state-sharing mechanism and lifecycle management using various Hooks (useState, useReducer, useContext, useEffect).	
Knowledge	**Code**	**Description**
	TPK20	**writing asynchronous code using various techniques**
	TPK21	**Basic routing and calling a controller method from a route**
	TPK22	**Analyze and solve common web applications tasks by writing PHP programs**
Skill	**Code**	**Description**
	TPC5.4	**Using UI Framework (React)**
	TPC5.5	**Creating an authentication system for your application**
	TPC5.7	**Using controllers and routes for APIs and URLs**

Unit Preface



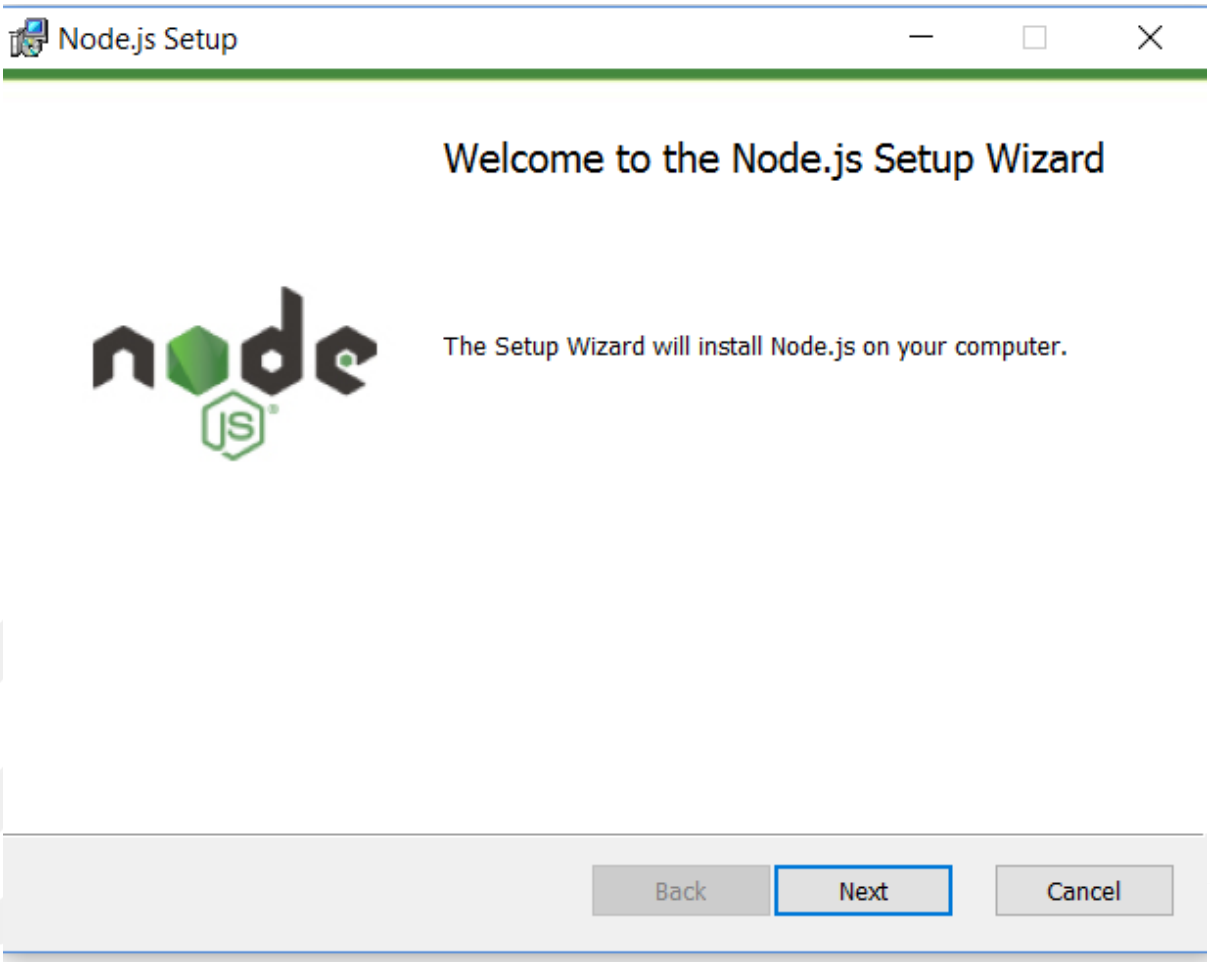
Workshop1

Question 1: How to install Nodejs and NPM?

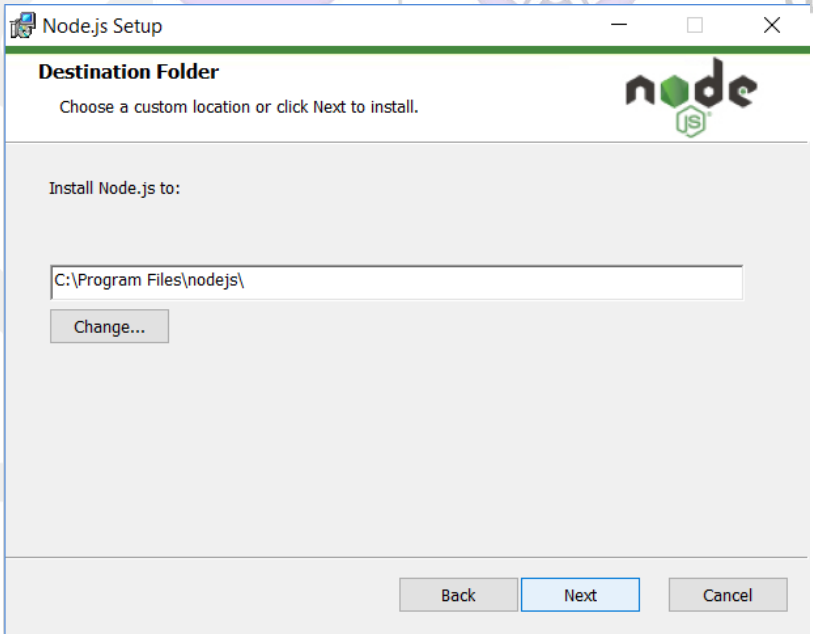
Step1: Download the Installer Official Node.js download page:

https://nodejs.org/en

Step2: double-click to install .msi binary file to initiate the installation process. Then give access to run the application

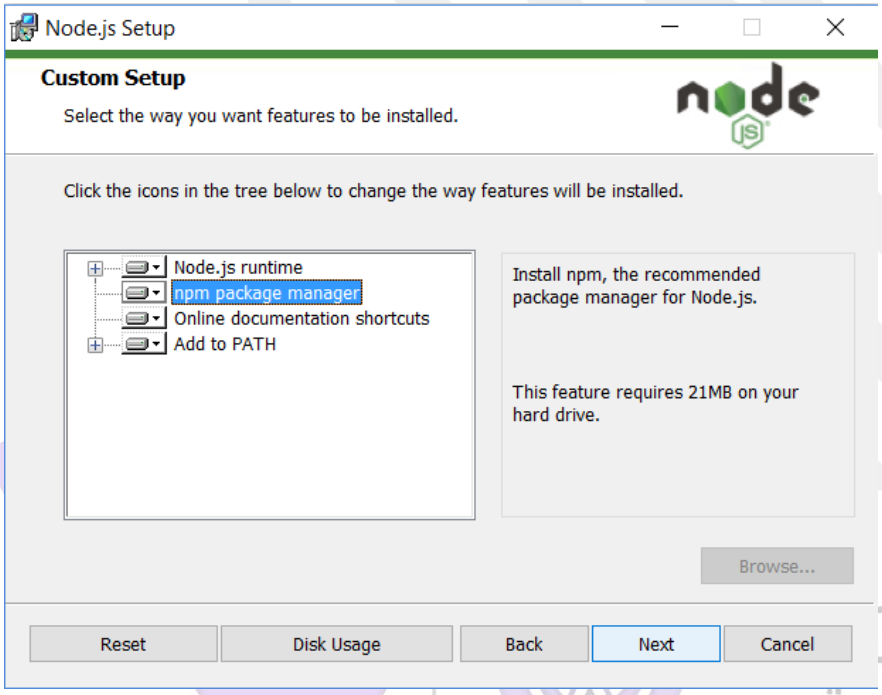


Choose the desired path where you want to install Node.js.

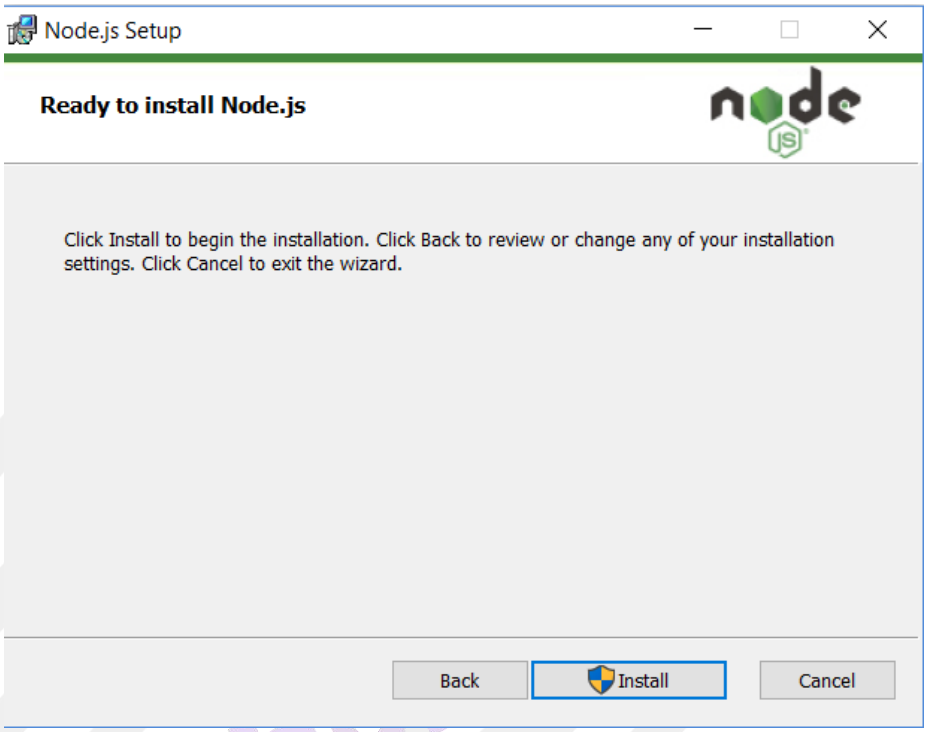


By clicking on the Next button, you will get a custom page setup on the screen. Make sure you choose npm package manager , not the default of Node.js runtime . This way, we can install Node and NPM simultaneously.

You should have 143MB of space to install Node.js and npm features.



Bang! The setup is ready to install Node and NPM. Let’s click on the Install button so hard!



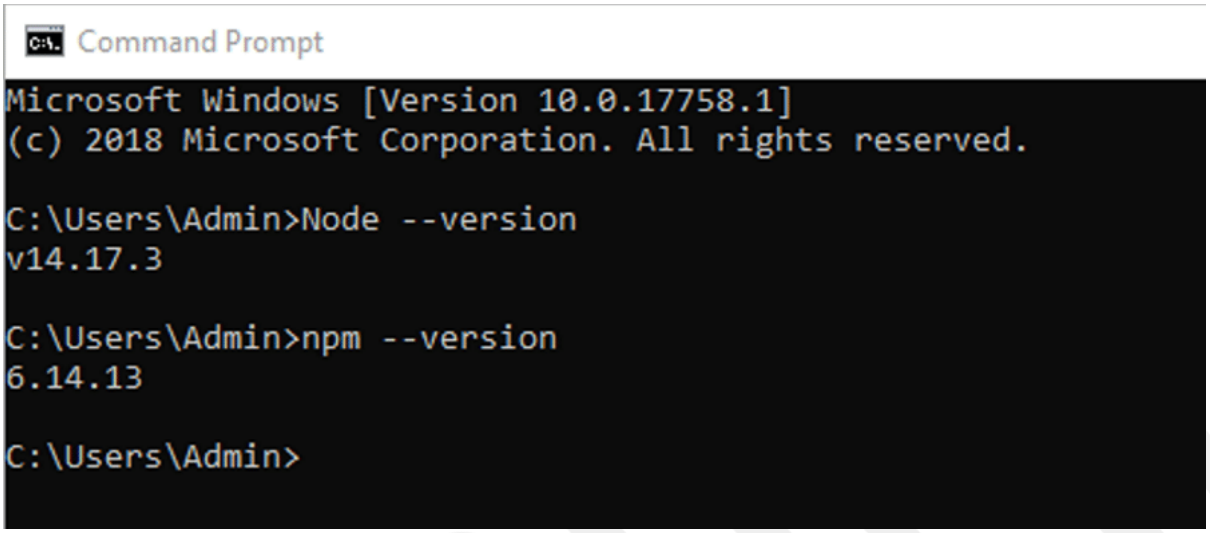
Step 3: Check Node.js and NPM Version

To confirm Node installation, type node -v command.

To confirm NPM installation, type npm -v command.

In Terminal:

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



Question 2: How to Create a React Project from Scratch

Step 1: Setting Up the Environment

Before getting started, make sure you have Node.js installed on your system. You can download the latest version from the official Node.js website (https://nodejs.org) and install it following the appropriate instructions for your operating system.

Step 2: Creating a New Project

- Open a terminal and navigate to the directory where you want to create your React project.
- Run the following command to create a new React project using Create React App, a tool that simplifies the initial setup:

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

This command will create a new folder called my-react-project with the basic structure of a React project.

Step 3: Running the Project

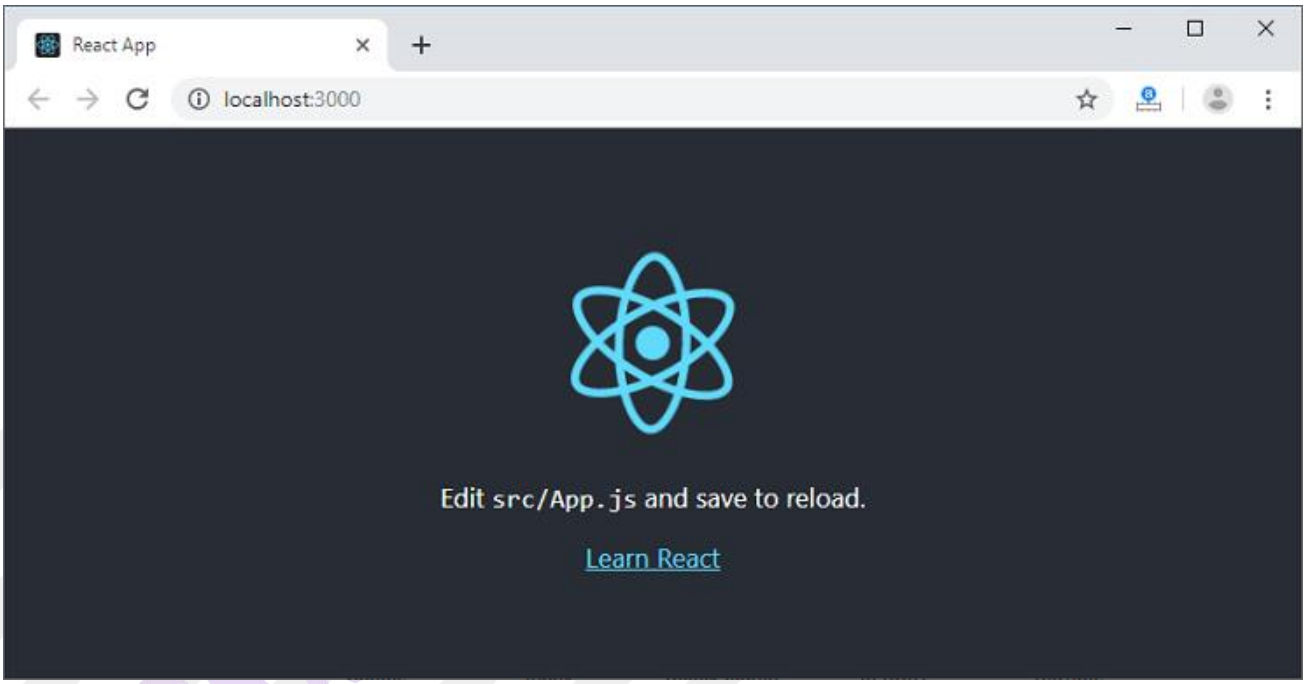
- Navigate the directory of your newly created project:

```
cd my-react-project
```

- Run the following command to start the application on a development server:

npm start

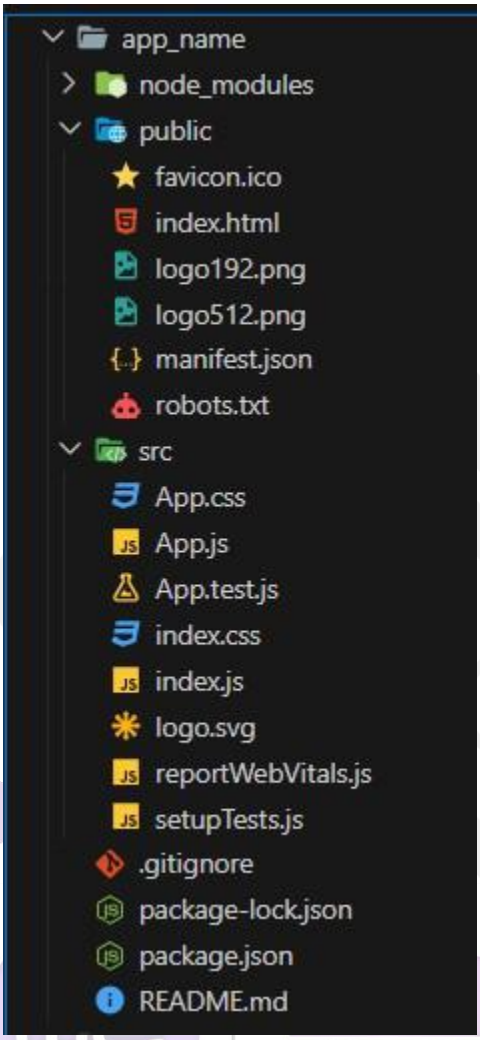
React will start the development server and automatically open your application in the browser.



Step 4: Exploring the Project Structure

Once your React project is up and running, it's important to understand its basic structure and main files. Here's an overview:

- src/index.js: This file is the entry point of your React application. It imports the root component and renders it to the DOM.
- src/App.js: This file contains the root component of your application. You can start building your user interface in this component or create additional components and use them here.
- src/components/: This folder is where you can store your reusable components. You can create subfolders to organize your components as needed.
- public/index.html: This file is the base HTML template for your React application. The entry point of your application will be included here.



Note: If the problem appears with Scripts id disabled

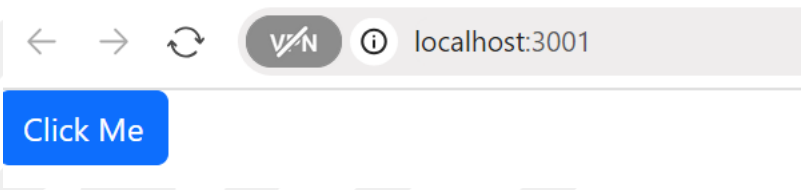
```
PS C:\Users\Acer\Desktop> npm create vite@latest
npm : File C:\Program Files\nodejs\npm.ps1 cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ npm create vite@latest
+ ~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Acer\Desktop>
```

Run terminal or powershell as administrator :

Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser

Workshop 2

Question 1: Describe the steps required to install and set up Bootstrap in a ReactJS project. Include the commands needed and explain how to use Bootstrap components in your project.



Answer:

1. **Install Bootstrap:** Start by installing Bootstrap through npm by running the following command in the terminal:

```
npm install bootstrap
```

This command installs Bootstrap into the React project so its styles are available for use.

2- Import Bootstrap CSS: Open the main entry file (usually index.js or App.js) and import Bootstrap's CSS:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Adding this line ensures that Bootstrap's CSS is applied throughout the project, so components and layouts follow Bootstrap's styling.

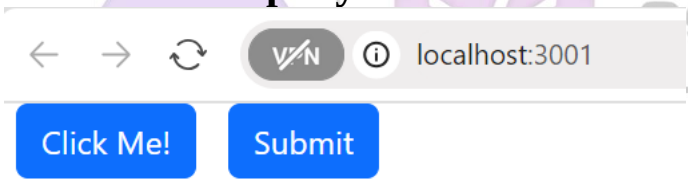
Using Bootstrap Components: You can now use Bootstrap classes in your JSX. For example, to create a button with Bootstrap styling, add the btn and btn-primary classes:

```
import React from 'react';
```

```
function App() {
  return (
    <div className="App">
      <button className="btn btn-primary">Click Me</button>
    </div>
  );
}
export default App;
```

When you run the project (using npm start), you should see a button styled with Bootstrap's primary color, indicating that Bootstrap is successfully installed and working.

Question 2: Describe the steps required to install and set up Bootstrap in a ReactJS project. After installation, create a React component that uses Bootstrap classes and takes `props` to display a dynamic message on a Bootstrap-style button.



Answer:

1. Install Bootstrap:

Install Bootstrap via npm by running this command in your terminal:

```
npm install bootstrap
```

This will add Bootstrap to your project's dependencies, making its styles available.

2. Import Bootstrap CSS:

Open the main file (usually `index.js` or `App.js`) and import Bootstrap's CSS:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

This import ensures that Bootstrap's CSS is applied to all components in the project.

3. Create a Button Component with Props:

Now, create a `ButtonComponent` that accepts a `label` prop to display a dynamic message on a Bootstrap-styled button:

```
import React from 'react';
function ButtonComponent({ label }) {
return <button className="btn btn-primary mx-2">{label}</button>;
}
export default ButtonComponent;
```

4. Using the Component:

Import `ButtonComponent` into `App.js` and pass a value for the `label` prop to display a custom message on the button:

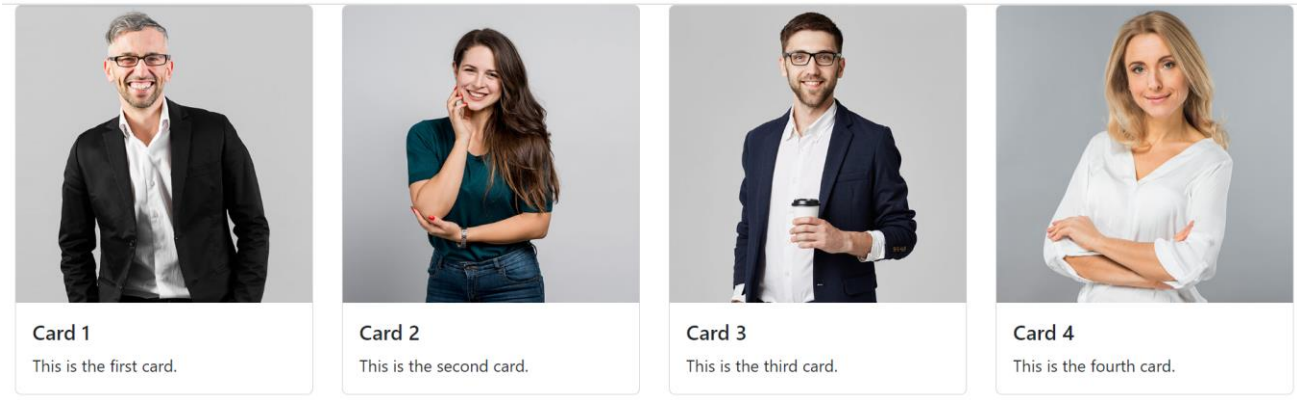
```
import React from 'react';
import ButtonComponent from './ButtonComponent';
function App() {
  return (
    <div className="App">
      <ButtonComponent label="Click Me!" />
      <ButtonComponent label="Submit" />
    </div>
  );
}
export default App;
```

5. Run the Project:

Use the command below to start the development server:
npm start

Now, each button displays a different message based on the value passed to the `label` prop, and the buttons are styled with Bootstrap's primary button class (`btn-primary`).

Question 3: Describe the steps to install Bootstrap in a ReactJS project and create a card component that uses props to display different images from the public folder and text. Include the Bootstrap setup, code to build a card component, and an example of how to render multiple cards with different images and text using props.



Answer:

1. Install Bootstrap: First, install Bootstrap using npm in the project directory:

```
npm install bootstrap
```

This command adds Bootstrap to your project.

2. Import Bootstrap CSS: Open `index.js` (or `App.js`) and import Bootstrap's CSS file:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

This line applies Bootstrap styles throughout your project.

3. Create a Card Component with Props: Next, create a `Card` component that takes `title`, `text`, and `image` as props. This component will use Bootstrap classes to style the card and display title, text and It will also display an image loaded from the public folder passed as props.

```
import React from 'react';
function Card({ title, text, image }) {
  return (
    <div className="card" style={{ width: '18rem' }}>
```

```
      <img src={`${process.env.PUBLIC_URL}/images/${image}`}
className="card-img-top" alt="card" />
      <div className="card-body">
        <h5 className="card-title">{title}</h5>
        <p className="card-text">{text}</p>
      </div>
    </div>
  );
}
export default Card;
```

Here, the Card component takes in title, description, and image as props. The image path uses process.env.PUBLIC_URL to ensure that images are served from the public folder, so you can add images to public/images/.

4. Using the Card Component: Import the `Card` component in `App.js` and render multiple cards with different images and text.

```
import React from 'react';
import Card from './Card';

function App() {
  return (
    <div className="App">
      <div className="d-flex justify-content-around">
        <Card
          title="Card 1"
          text="This is the first card."
          image="team-1.jpg"
        />
        <Card
          title="Card 2"
          text="This is the second card."
          image="team-2.jpg"
        />
        <Card
```

```
          title="Card 3"
          text="This is the third card."
          image=" team-3.jpg"
        />
        <Card
          title="Card 4"
          text="This is the fourth card."
          image="team-4.jpg"
        />
      </div>
    </div>
  );
}

export default App;
```

5. Run the Project: Start the development server with:

```
npm start
```

Now you should see three Bootstrap-style cards with different images and text.

Note: This question and answer not only cover the installation and usage of Bootstrap in a React project but also illustrate using props to create dynamic and reusable card components. This approach helps students practice component-based design and understand Bootstrap's styling in a practical way.

Workshop 3

Question 1: How to Work With Events In ReactJS?

Answer:

```
export default function App() {
  const myFunc = () => console.log('Welcome to Events')
  return (
    <div>
      <button className='btn btn-primary' onClick={myFunc}>
      Button 1 </button>
      <button className='btn btn-dark'>Button 2</button>
    </div>
  )}
```

Question 2: Create the Counter component and explain what happens each time the "Increment" button is clicked.

Describe how the count state variable is updated and displayed

Answer:

Code:

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

function Counter() {
  const [count, setCount] = useState(0); // Declare state variable 'count' with initial value of 0

  const handleIncrement = () => {
    setCount(count + 1); // Update state using setCount
  };
```

```
  return (
    <div>
      <p>Current count: {count}</p>
      <button onClick={handleIncrement}>Increment</button>
    </div>
  );
}
export default Counter;
```

Description:
Each time the "Increment" button is clicked, the handleIncrement function is called.
This function updates the count state variable by adding 1 to its current value using setCount(count + 1). The component re-renders, displaying the updated count in the paragraph (Current count: {count}).

Question 3: How would you toggle a boolean state in React, such as switching between showing and hiding content? Provide an example.

Answer:
To toggle a boolean state, you can use the state updater function with the opposite of the current state.

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

function ToggleContent() {
  const [isVisible, setIsVisible] = useState(true);

  const toggleVisibility = () => {
    setIsVisible(prevState => !prevState);
  };

  return (
    <div>
      <button onClick={toggleVisibility}>
```

```
      {isVisible ? 'Hide' : 'Show'} Content
    </button>
    {isVisible && <p>This content is visible.</p>}
  </div>
 );
}
export default ToggleContent;
```

Question 4: Using State For Form Input Handling

In a sign-up form, useState is commonly used to manage the values of user inputs like name, email, and password. When the user types into each input field, the state updates, allowing you to save or process the data as they type.

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

function SignupForm() {
 const [name, setName] = useState('');
 const [email, setEmail] = useState('');

 return (
  <div>
   <input
    type="text"
    value={name}
    onChange={(e) => setName(e.target.value)}
    placeholder="Enter your name"
   />
   <input
    type="email"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    placeholder="Enter your email"
   />
   <button onClick={() => console.log(`Name: ${name}, Email:
${email}`)}>Submit</button>
```

```
    </div>
  );
}
export default SignupForm;
```

Question 5: Create Like Button or Counter by using UseState

This example uses useState to manage a count, which is incremented each time the user clicks the "Like" button. This pattern is common in social media applications.

```
import React, { useState } from 'react';
function LikeButton() {
  const [likes, setLikes] = useState(0);

  return (
    <div>
      <button onClick={() => setLikes(likes + 1)}>Like</button>
      <p>{likes} {likes === 1 ? "Like" : "Likes"}</p>
    </div>
  );
}
export default LikeButton;
```

Question 6: Using useState can track the quantity of items in a shopping cart, updating the total when users add or remove items.

```
import React, { useState } from 'react';
function CartItem() {
  const [quantity, setQuantity] = useState(1);
  return (
    <div>
      <h2>Item Name</h2>
      <button onClick={() => setQuantity(quantity - 1)}
disabled={quantity === 1}>-</button>
```

```
      <span> {quantity} </span>
      <button onClick={() => setQuantity(quantity + 1)}>+</button>
    </div>
  );
}
export default CartItem;
```

Question 7: Creating Theme Toggler (Light/Dark Mode)
In applications with theme toggling, useState can switch between light and dark modes based on the user's preference.

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

function ThemeToggle() {
  const [isDarkMode, setIsDarkMode] = useState(false);

  return (
    <div style={{ background: isDarkMode ? '#333' : '#FFF', color:
isDarkMode ? '#FFF' : '#333' }}>
      <button onClick={() => setIsDarkMode(!isDarkMode)}>
        Switch to {isDarkMode ? "Light" : "Dark"} Mode
      </button>
    </div>
  );
}
export default ThemeToggle;
```

Workshop 4

Question 1: What is the useReducer hook in React, and how does it differ from the useState hook?

Answer:

The useReducer hook is a powerful tool in React for managing complex state within your components. It's particularly useful when:

1. **Multiple sub-values:** You need to manage multiple sub-values within a single state variable.
2. **Complex state logic:** Your state updates involve intricate logic or side effects.
3. **State that needs to be shared across multiple components:** You can use useContext in conjunction with useReducer to share state across components.

```
import { useReducer } from 'react';
const initialState = { count: 0 };
```

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}
```

```
function App() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
}
```

export default App

Question 2: Sharing state with React Context

Context allows you to natively share values between components without having to prop drill them. Let's learn how to build a context to handle our counter:

Add Employee Component
In App Compnent

```
import React from "react";
import Employee from "./Employee";
const user = { name:'Mohamed' } ;
export const UserContext = React.createContext()

export default function App() {
  return (
    <div>
      <UserContext.Provider value={user}>
        <Employee/>
```

```
      </UserContext.Provider>
    </div>
  );
}
```

In Employee Compnent using useContext Hook

```
import React ,{ useContext } from 'react'
import { UserContext } from './App'
export default function Employee() {
    const user = useContext(UserContext)
  return (    <h1> {user.name} </h1> )
}
```

Add city to Component

```
import React from "react";
import Employee from "./Employee";

const user = { name:'Mohamed' } ;
const city = { name:'Mansoura' } ;

export const UserContext = React.createContext()
export const CityContext = React.createContext()
```

```
export default function App() {
  return (
    <div>
      <UserContext.Provider value={user}>
      <CityContext.Provider value={city}>
        <Employee/>
      </CityContext.Provider>
      </UserContext.Provider>
    </div>
  );
}
```

```
import React ,{ useContext } from 'react'
import { UserContext , CityContext } from './App'
```

```
export default function Employee() {
    const user = useContext(UserContext)
    const city = useContext(CityContext)
  return (
      <h1> {user.name} From : {city.name} </h1>
  )
}
```

Question 3: Build a context to handle our counter

1 - Create New File CountProvider.jsx

```
import React , { useState , createContext } from "react";
export const CountContext = createContext();
```

```
export const CountStore =  ()=> {
const [count , setCount ] = useState(0);
const increment= ()=>{ setCount( (currentCount)=> currentCount + 1)}
const decrement= ()=>{setCount( (currentCount)=> currentCount - 1 )}
const reset = ()=> setCount( 0 )
return { count , increment , decrement , reset }
} // CountStore
```

```
export default function CountProvider(children) {
  return (
    <CountContext.Provider value={CountStore()}  {...children} />
  )
}
```

2 – In Index.js

```
import CountProvider from "./CountProvider";

    <CountProvider>
    <App />
    </CountProvider>
```

3 – In App.js

```
import { useContext } from "react";
import { CountContext } from "./CountProvider";
```

```
export default function App() {
const{count, increment,decrement, reset}= useContext(CountContext)
  return(
   <div>
    <h1>Count : {count} </h1>
```

```
    <button onClick={increment}> + </button>
    <button onClick={decrement}> - </button>
    <button onClick={reset}> Reset </button>
  </div>
 );
}
```

Question 4: Data Fetching from API Using useEffect from :
https://jsonplaceholder.typicode.com/

In Terminal : npm i axios

In App.js File :

```
import  axios   from "axios";
import {  useState, useEffect } from "react";

export default function App() {
  const [posts , setPosts  ] = useState([]);

  const getPosts = async ()=>{
    const response = await
axios.get(`https://jsonplaceholder.typicode.com/posts`);
    setPosts(response.data)
  }

  useEffect( ()=>{
    getPosts()
  } ,[])

  console.log(posts)
```

Using try & catch:

```
const getPosts = async () => {
    try {
      const response = await axios.get(
        `https://jsonplaceholder.typicode.com/posts`
      );
      setPosts(response.data);
    } catch (error) {
      console.log(error);
    }
  };
```

Display Data in Design :

```
return(
<div className="text-center py-5 row">
  {posts.map( function(post){
    return(  <div>
      <h3>{post.title}</h3>
      <p>{post.body}</p>
      </div>  )
  } )}
</div>
) } // return function
```

You can Display One Post:
https://jsonplaceholder.typicode.com/posts/1

```
import axios from "axios";
import { useState, useEffect } from "react";
export default function App() {
  const [post, setPost] = useState([]);
  const [id, setId] = useState('');

  const getPost = async () => {
    try {
      const response = await axios.get(
        `https://jsonplaceholder.typicode.com/posts/${id}`
      );
      setPost(response.data);
    } catch (error) {
      console.log(error);
    }
  };

  useEffect(() => {
    getPost();
  }, [id]);

  return(
  <div className="text-center py-5 row">
```

```
   <input type="text" onChange={ (e)=> setId(e.target.value)} />
      <h3>{post.title}</h3>
      <p>{post.body}</p>
  </div>
  ) // return
}
```


Workshop 5

Question: You have a React application with three pages: Home, About, and Contact. Using React Router, describe how you would set up routing to allow users to navigate between these pages. Include details on any components or imports you would use.

Answer: To set up routing between **Home**, **About**, and **Contact** pages in a React application using React Router, you would:

1. **Install React Router** if it's not already installed by running:

```
npm install react-router-dom
```

2. **Import necessary components from React Router:**

```
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';
```

3. **Create Components for each page (Home, About, Contact) if they're not created yet:**

```
function Home() { return <h2>Home Page</h2>; }
function About() { return <h2>About Page</h2>; }
function Contact() { return <h2>Contact Page</h2>; }
```

4. **Set up routing in the main component (e.g., App.js):**

```
function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link> |
```

```
      <Link to="/contact">Contact</Link>
    </nav>
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/contact" element={<Contact />} />
    </Routes>
  </Router>
 );
}
export default App;
```

In this setup:

- The Router component wraps the entire app to enable routing.
- The Link component is used for navigation, so clicking on "Home," "About," or "Contact" will take the user to the corresponding route without reloading the page.
- The Routes and Route components define each route's path and the component that should render for each path ("/" for Home, "/about" for About, and "/contact" for Contact).

This configuration allows users to navigate between the three pages smoothly using React Router.