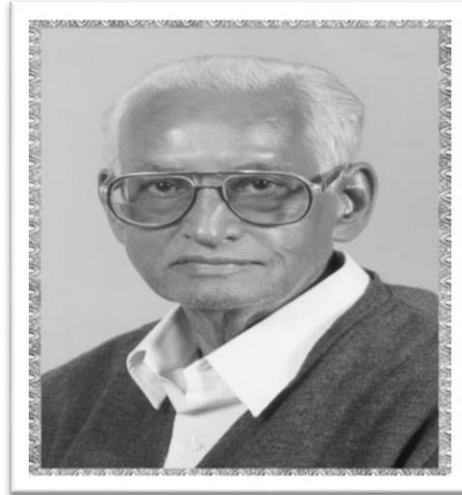


Shree H. N. Shukla College of It. & Mgmt. **(Affiliated To Saurashtra University)**



Lt. Shree Chimanbhai Shukla

MSCIT SEM-2 REACTJS

**Shree H.N.Shukla college2
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478**

**Shree H.N.Shukla college3
vaishali nagar
Near Amrapali Under Bridge,
Raiya road
Rajkot
Ph No:-0281 2440478**

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

Unit :2

Introduction to JSX & REACT JS

- **Introduction:** What is ReactJS? Installation or Setup, Hello World Program, Create a first app, folder structure
- **Components:** Creating components, Basic components, Nesting components, functional Component, class component
- **Introduction to JSX:** JSX Programs
- **Props:** ReactJS Props, React State, Destructuring Props and State, set State, methods as Props.



What is ReactJS?

ReactJS is an open-source JavaScript library used to create user interfaces in a declarative and efficient way. It is a component-based front-end library responsible only for the view layer of a **Model View Controller(MVC)** architecture. React is used to create modular user interfaces and promotes the development of reusable UI components that display dynamic data.



History of ReactJS

While building client-side application, a team of Facebook developers found that DOM is slow. Document Object Model (DOM) is an application programming interface(API) for HTML and XML documents. It defines the logical structure of documents and how a document is accessed and manipulated). To make it faster, React implements a virtual DOM which is basically a DOM tree representation in JavaScript, and React was invented.

ReactJS is updated so frequently, that it is quite difficult to navigate the version, which comes with new features every time, each time it comes with new features. The current stable version of ReactJS is **18.2.0** and released on June 14, 2022 and the first release was on May 29, 2013.

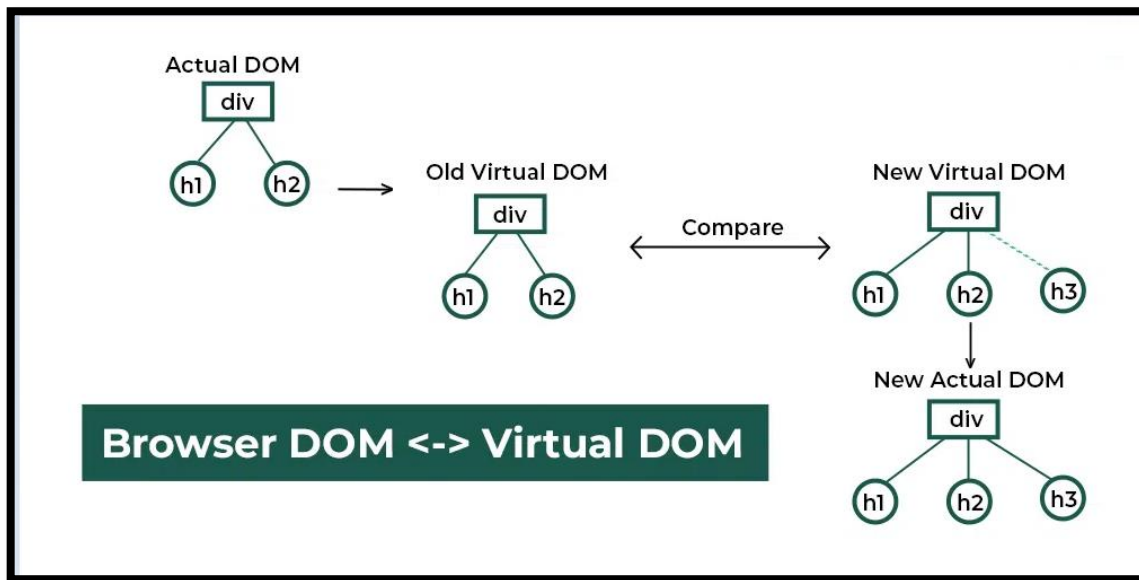
Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



How does ReactJS work?

React creates a virtual DOM in memory to update the browser's DOM. The virtual DOM will try to find the most efficient way to update the browser's DOM.



Unlike browser DOM elements, React elements are simple objects and are cheap to create. React DOM takes care of updating the DOM to match the React elements. The reason for this is that JavaScript is very fast and it's worth keeping a DOM tree in it to speed up its processing.

Although React was designed to be used in the browser, because of its design allows it to be used on the server with Node.js as well.

Installation Reactjs on Windows:

Step 1: Install Node.js installer for windows. Click on this [link](#). Here install the LTS version (the one present on the left). Once downloaded open NodeJS without disturbing other settings, click on the **Next** button until it's completely installed.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



Install the 14.18.1 LTS

Step 2: Open command prompt to check whether it is completely installed or not type the command →

`node -v`

Node Version is v14.15.3

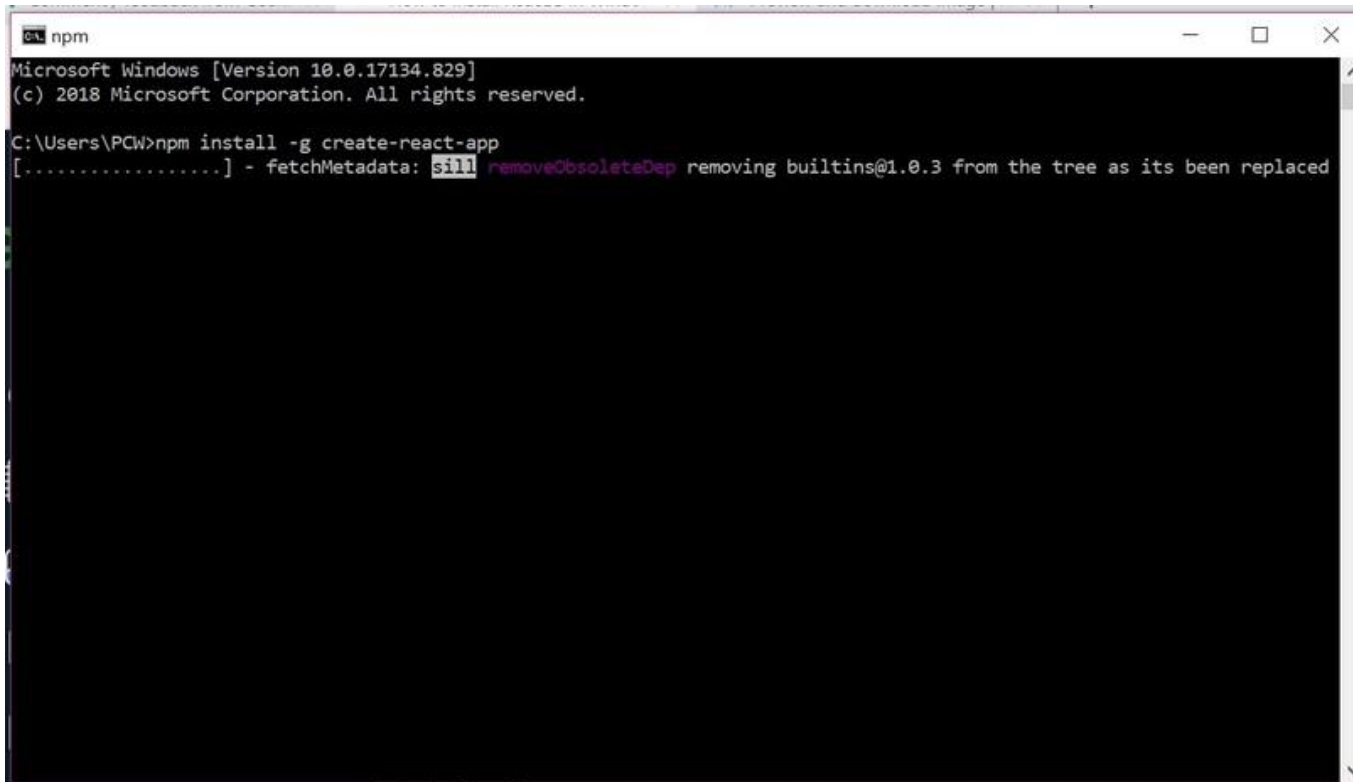
Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

If the installation went well it will give you the version you have installed

Step 3: Now in the terminal run the below command:

`npm install -g create-react-app`



```
npm
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

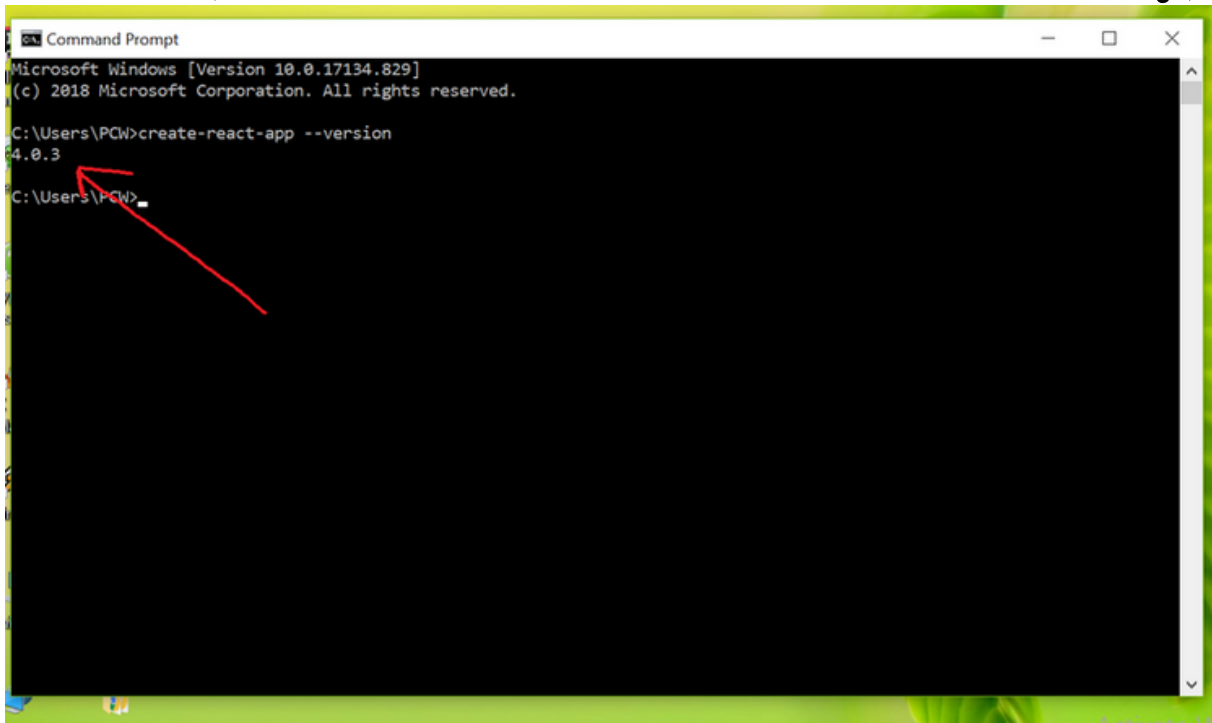
C:\Users\PCW>npm install -g create-react-app
[.....] - fetchMetadata: sill removeObsoleteDep removing builtins@1.0.3 from the tree as its been replaced
```

Installation will take few seconds

It will globally install react app for you. To check everything went well run the command `create-react-app --version`

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PCW>create-react-app --version
4.0.3
C:\Users\PCW>
```

version 4.0.3

If everything went well it will give you the installed version of react app

Step 4: Now Create a new folder where you want to make your react app using the below command:

`mkdir newfolder`

Note: The *newfolder* in the above command is the name of the folder and can be anything.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PCW>mkdir newfolder
```

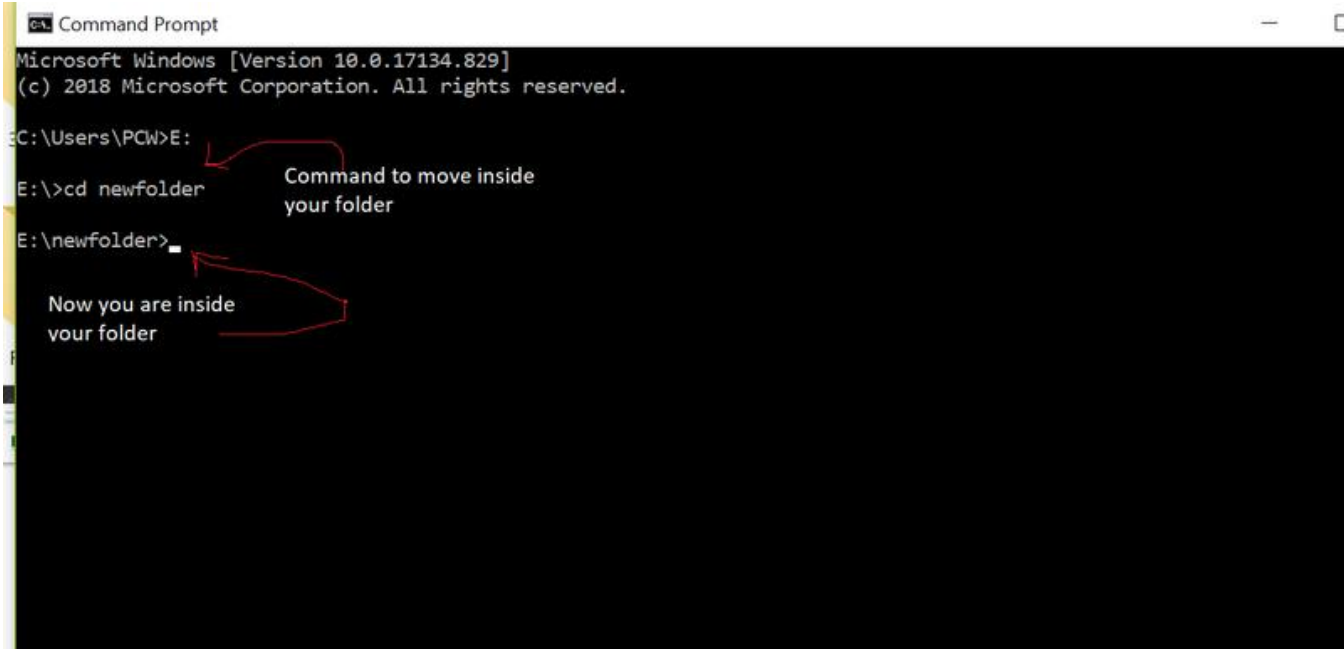
Enter Your Folder name in place of
newfolder

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

Move inside the same folder using the below command:

`cd newfolder` (your folder name)



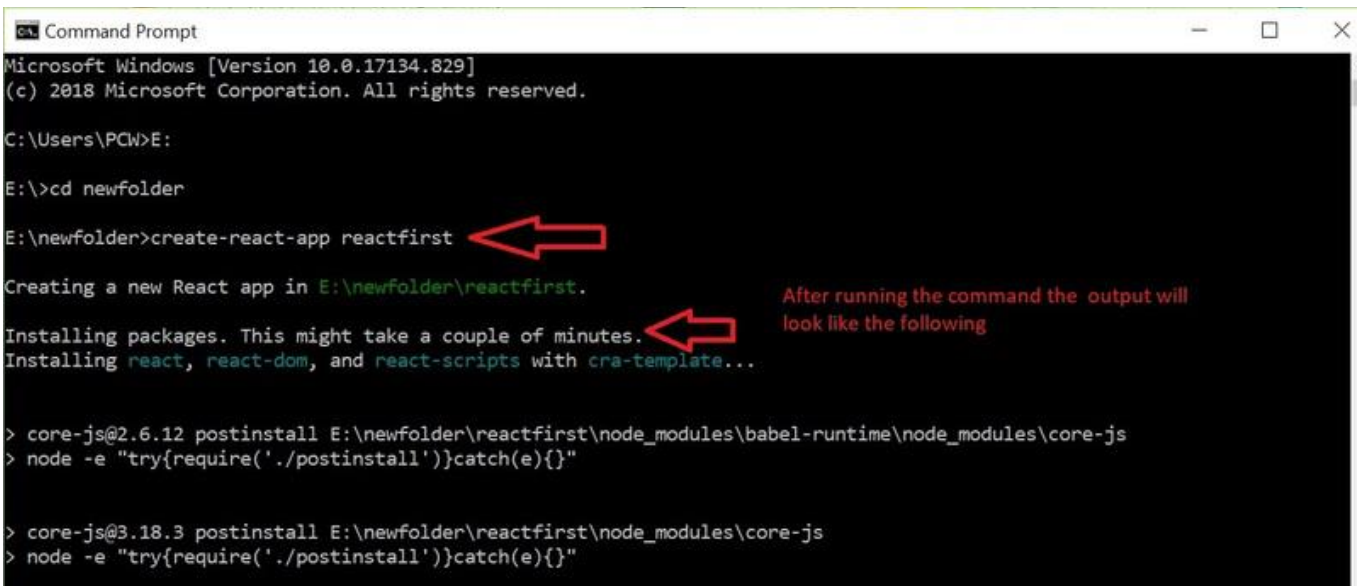
```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PCW>E:
E:\>cd newfolder
E:\newfolder>
```

Command to move inside your folder

Now you are inside your folder

Step 5: Now inside this folder run the command →
`create-react-app reactfirst YOUR_APP_NAME`



```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PCW>E:
E:\>cd newfolder
E:\newfolder>create-react-app reactfirst
Creating a new React app in E:\newfolder\reactfirst.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

> core-js@2.6.12 postinstall E:\newfolder\reactfirst\node_modules\babel-runtime\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

> core-js@3.18.3 postinstall E:\newfolder\reactfirst\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"
```

After running the command the output will look like the following

It will take some time to install the required dependencies

NOTE: Due to npm naming restrictions, names can no longer contain capital letters, thus type your app's name in lowercase.

Shree H. N. Shukla College of It. & Mgmt.

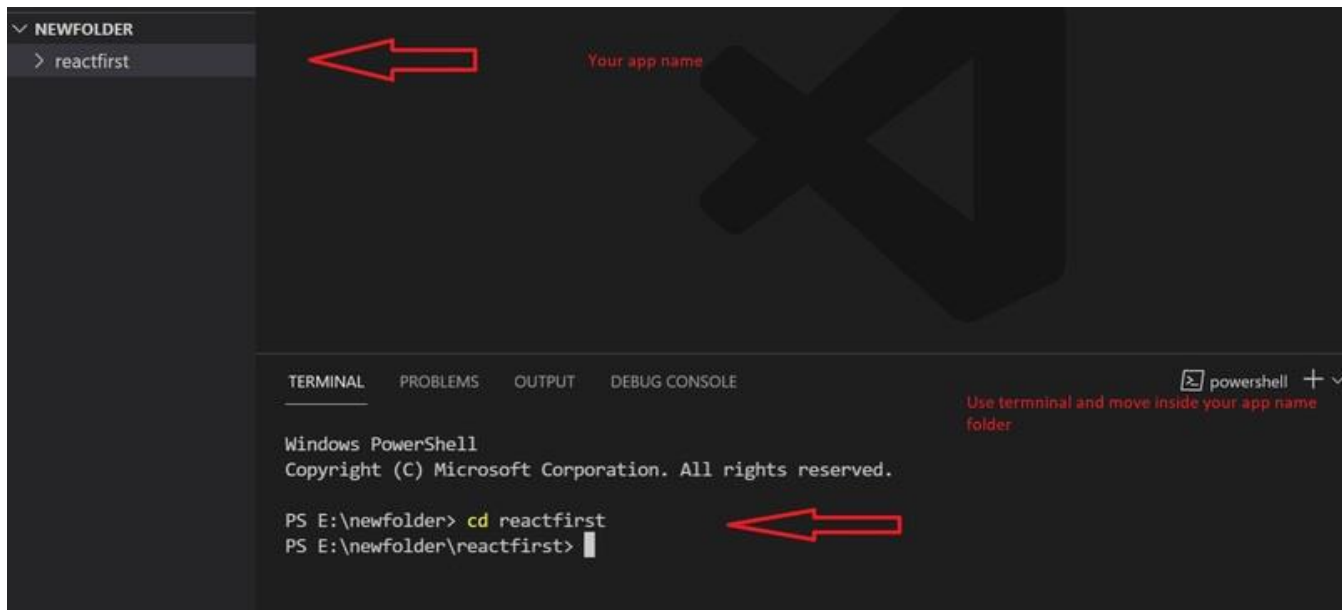
(Affiliated To Saurashtra University)

```
C:\Users\sa552>create-react-app FirstReact
Cannot create a project named "FirstReact" because of npm naming restrictions:

  * name can no longer contain capital letters

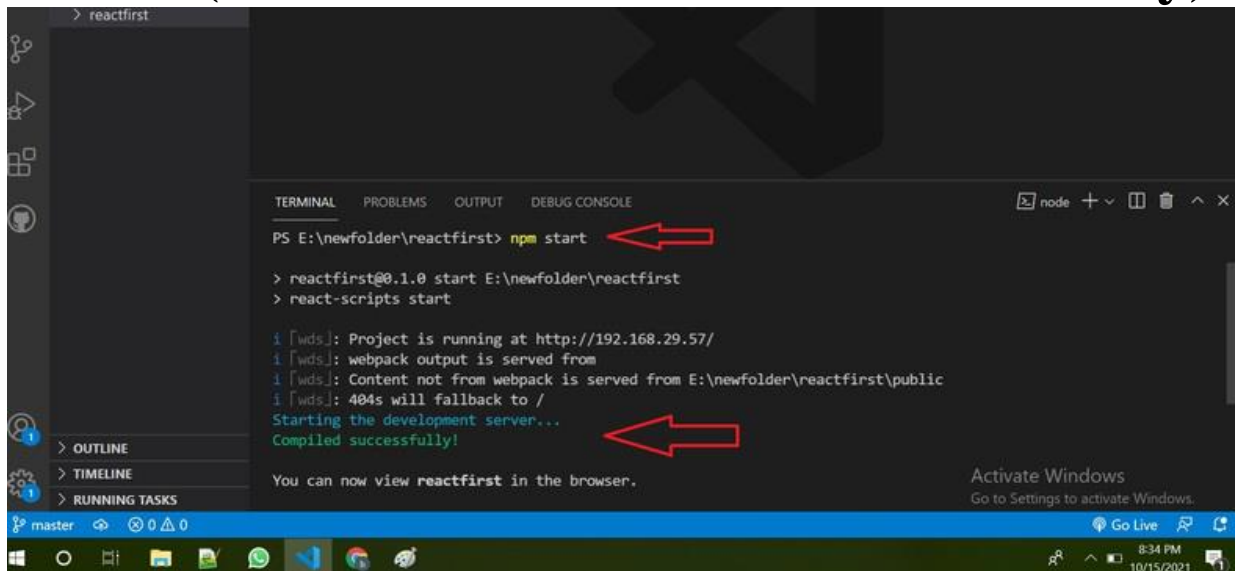
Please choose a different project name.
```

Step 6: Now open the IDE of your choice for eg. Visual studio code and open the folder where you have installed the react app **newfolder** (in the above example) inside the folder you will see your app's name **reactapp** (In our example). Use the terminal and move inside your app name folder. Use command **cd reactapp** (your app name)



Step 7: To start your app run the below command :
npm start

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)



```
> reactfirst  
PS E:\newfolder\reactfirst> npm start  
  
> reactfirst@0.1.0 start E:\newfolder\reactfirst  
> react-scripts start  
  
[wds]: Project is running at http://192.168.29.57/  
[wds]: webpack output is served from  
[wds]: Content not from webpack is served from E:\newfolder\reactfirst\public  
[wds]: 404s will fallback to /  
Starting the development server...  
Compiled successfully!  
  
You can now view reactfirst in the browser.
```

Once you run the above command a new tab will open in your browser showing React logo as shown below :



Congratulation you have successfully installed the react-app and are ready to build awesome websites and app

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)

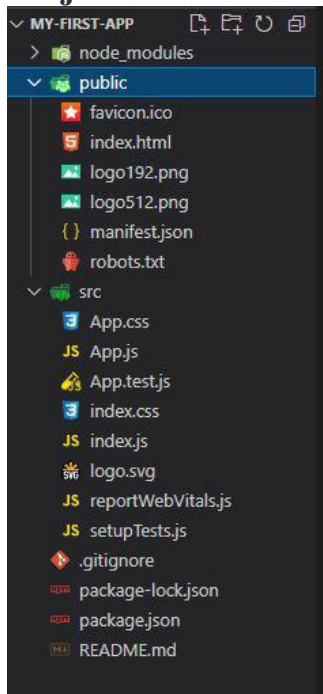
Steps to Create React Application:

Step 1: Create a react application using the following command

`npx create-react-app foldername`

Step 2: Change your directory to the newly created application using the following command
`cd foldername`

Project Structure:



Project file structure

The updated dependencies in **package.json** file will look like:

```
"dependencies": {  
  "react": "^18.2.0",  
  "react-dom": "^18.2.0",  
  "react-scripts": "5.0.1",  
  "web-vitals": "^2.1.4",  
}
```

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)

Example: Now inside **App.js** and write down the following code as shown below:

 **JavaScript**

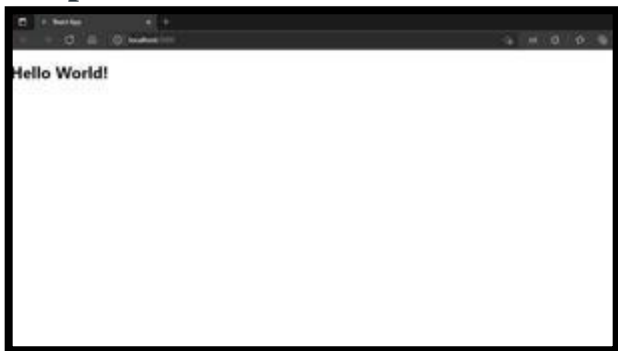
```
import React from 'react';  
import './App.css';  
function App() {  
  return (  
    <h1> Hello World! </h1>  
  );  
}
```

export default App;

Step to Run the application: Enter the following command to run the application.

npm start

Output:



Creating components in React is a fundamental aspect of building applications. Components are reusable and modular pieces of the user interface. Here's a step-by-step guide on how to create and use components in React:

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)

1. Class Components:

```
jsx

import React, { Component } from 'react';

class MyComponent extends Component {
  render() {
    return (
      <div>
        <h1>Hello, I am a class component!</h1>
      </div>
    );
  }
}

export default MyComponent;
```

2. Functional Components:

```
jsx

import React from 'react';

const MyFunctionalComponent = () => {
  return (
    <div>
      <h1>Hello, I am a functional component!</h1>
    </div>
  );
};

export default MyFunctionalComponent;
```

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

3. Using Props (Properties):

Props allow you to pass data from a parent component to a child component.

```
jsx

import React from 'react';

const Greeting = (props) => {
  return (
    <div>
      <h1>Hello, {props.name}!</h1>
    </div>
  );
};

export default Greeting;
```

Usage:

```
jsx

import React from 'react';
import Greeting from './Greeting';

const App = () => {
  return (
    <div>
      <Greeting name="John" />
      <Greeting name="Jane" />
    </div>
  );
};

export default App;
```

4. Using State:

State allows components to manage their own data.

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)

```
jsx

import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

5. Lifecycle Methods:

Class components have lifecycle methods, such as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)

```
jsx

import React, { Component } from 'react';

class LifecycleExample extends Component {
  componentDidMount() {
    console.log('Component did mount');
  }

  componentWillUnmount() {
    console.log('Component will unmount');
  }

  render() {
    return (
      <div>
        <h1>Lifecycle Example</h1>
      </div>
    );
  }
}

export default LifecycleExample;
```

6. Conditional Rendering:

You can conditionally render components based on certain conditions.

```
jsx

import React from 'react';

const ConditionalRender = ({ isLoggedIn }) => {
  return (
    <div>
      {isLoggedIn ? <p>Welcome, User!</p> : <p>Please log in.</p>}
    </div>
  );
};

export default ConditionalRender;
```

These are basic examples to get you started. React's documentation is a great **resource** for learning more about components, state, props, and other advanced concepts: [React Documentation](#).

Shree H. N. Shukla College of It. & Mgmt. (Affiliated To Saurashtra University)



ReactJS - Nested Components

In React, you can create nested components by composing them within each other. This allows you to break down your UI into smaller, reusable components, making your code more modular and easier to maintain.

Here's a simple example to illustrate the concept of nested components in React:

```
jsx

// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  return (
    <div>
      <h1>This is the Parent Component</h1>
      <ChildComponent />
    </div>
  );
};

export default ParentComponent;
```


Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

jsx

```
// ChildComponent.js
import React from 'react';
import GrandchildComponent from './GrandchildComponent';

const ChildComponent = () => {
  return (
    <div>
      <h2>This is the Child Component</h2>
      <GrandchildComponent />
    </div>
  );
};

export default ChildComponent;
```

jsx

```
// GrandchildComponent.js
import React from 'react';

const GrandchildComponent = () => {
  return (
    <div>
      <h3>This is the Grandchild Component</h3>
      <p>It can have its own content and functionality.</p>
    </div>
  );
};

export default GrandchildComponent;
```

In this example, ParentComponent includes ChildComponent, and ChildComponent includes GrandchildComponent. Each component can have its own state, props, and functionality, allowing you to encapsulate and manage different parts of your UI separately.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

To use these components in your main application file (e.g., App.js), you would import and render the ParentComponent:

```
jsx

// App.js
import React from 'react';
import ParentComponent from './ParentComponent';

const App = () => {
  return (
    <div>
      <h1>Welcome to My React App</h1>
      <ParentComponent />
    </div>
  );
};

export default App;
```

This is a basic example, and in a real-world scenario, your components would likely be more complex, with the ability to receive and manage props and state. Keep in mind that effective component composition can lead to a more maintainable and scalable React application.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



Introduction to JSX: JSX Programs

JSX, which stands for JavaScript XML, is an extension to the JavaScript language syntax. It allows developers to write HTML-like code directly within JavaScript, which can then be transformed into standard JavaScript by tools like Babel before being interpreted by the browser.

Here's a basic example of a JSX program:

```
jsx

// JSX code
const element = <h1>Hello, world!</h1>;

// Equivalent JavaScript code after compilation
const element = React.createElement('h1', null, 'Hello, world!');
```

In the JSX code above, `<h1>Hello, world!</h1>` looks like HTML, but it's actually JSX. When the code is compiled, it's transformed into regular JavaScript function calls using `React.createElement()`.

Here's a breakdown of how JSX works:

1 JSX Elements: JSX elements look similar to HTML tags, but they are actually syntactic sugar for function calls and object construction. In the example above, `<h1>` is a JSX element.

2 Attributes: JSX elements can have attributes just like HTML elements. For example, `` in JSX would be the equivalent of `` in HTML.

3 JavaScript Expressions: You can embed JavaScript expressions within JSX using curly braces `{ }`. For example:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

const name = 'John';
const element = <h1>Hello, {name}</h1>;
```

1 **.Nested JSX:** JSX elements can be nested inside one another, just like HTML elements. For example:

```
jsx

const element = (
  <div>
    <h1>Hello, world!</h1>
    <p>This is a paragraph.</p>
  </div>
);
```

1.**JSX as Expressions:** JSX expressions can be used wherever JavaScript expressions are valid. This means you can use JSX inside if statements, for loops, and assign them to variables or pass them as arguments to functions.

Self-Closing Tags: If a JSX tag is empty, you must close it with `/>`, similar to XML syntax.

```
jsx

const element = ;
```

Overall, JSX simplifies the process of creating and manipulating DOM elements in React applications, making the code more readable and maintainable. However, it's important to remember that JSX needs to be compiled into standard JavaScript before it can be executed in the browser.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



Props: ReactJS Props

In React.js, "props" (short for properties) are a way to pass data from a parent component to a child component. Props are read-only and are used to communicate and share information between components. They help make components more reusable and maintainable.

Here's an introduction to using props in React:

```
jsx
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  return (
    <div>
      <ChildComponent name="John" age={30} />
    </div>
  );
}

export default ParentComponent;
```

```
jsx
// ChildComponent.js
import React from 'react';

function ChildComponent(props) {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}

export default ChildComponent;
```

In this example, the ParentComponent renders the ChildComponent and passes two props:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

name with the value "John" and age with the value 30. The ChildComponent receives these props as an object argument (props) in its function, and it accesses the values using dot notation (props.name, props.age). This way, the ChildComponent can display the data passed down from its parent (ParentComponent).

Props can be of any data type, including strings, numbers, arrays, objects, functions, or even React elements. Additionally, you can pass down props as many levels deep as needed in your component hierarchy.



React State

In React.js, state is a fundamental concept used to manage the internal data of a component. State allows components to keep track of information that may change over time and to re-render when that data changes. Unlike props, which are passed down from parent components and are immutable within the component, state is managed internally by the component itself and can be updated using the setState method.

Here's an example of how you can use state in a React component:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

import React, { useState } from 'react';

function Counter() {
  // Define a state variable named "count" and a function to update it
  const [count, setCount] = useState(0);

  // Function to handle incrementing the count
  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      /* Button to trigger the incrementCount function */
      <button onClick={incrementCount}>Increment</button>
    </div>
  );
}

export default Counter;
```

In this example:

We import the useState hook from React, which allows us to use state in functional components.

We define a state variable named count using the useState hook, with an initial value of 0.

We define a function incrementCount that updates the count state by calling the setCount function with the new value (count + 1).

Inside the JSX, we display the current value of count and provide a button that, when clicked, triggers the incrementCount function.

Whenever the state is updated using setCount, React re-renders the component, updating the UI with the new state data.

State in React components is local and encapsulated, meaning it is only accessible and modifiable within the component where it's defined. This encapsulation ensures that each component manages its own state independently, leading to a more predictable and maintainable application structure.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)



Destructuring Props and State

In React.js, destructuring is a technique used to extract values from objects or arrays, which can make your code cleaner and more concise. Destructuring is often used with both props and state in React components to simplify accessing their values.

Destructuring Props:

When a component receives props, you can destructure them directly in the function parameters or within the function body. Here's how you can destructure props in a functional component:

```
JSX

import React from 'react';

function MyComponent({ prop1, prop2 }) {
  // Destructure props within the function body
  return (
    <div>
      <p>Prop 1: {prop1}</p>
      <p>Prop 2: {prop2}</p>
    </div>
  );
}

export default MyComponent;
```

In this example, { prop1, prop2 } is used to directly destructure prop1 and prop2 from the props object in the function parameters. This allows you to use prop1 and prop2 directly within the component without referencing props.prop1 and props.prop2.

❖ Destructuring State:

Similarly, you can destructure state variables in a functional component using the useState hook:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

import React, { useState } from 'react';

function MyComponent() {
  // Destructure state variables
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={incrementCount}>Increment</button>
    </div>
  );
}

export default MyComponent;
```

In React.js, destructuring is a technique used to extract values from objects or arrays, which can make your code cleaner and more concise. Destructuring is often used with both props and state in React components to simplify accessing their values.

Destructuring Props:

When a component receives props, you can destructure them directly in the function parameters or within the function body. Here's how you can destructure props in a functional component:

jsx

Copy code

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

```
function MyComponent({ prop1, prop2 }) {
  // Destructure props within the function body
  return (
    <div>
      <p>Prop 1: {prop1}</p>
      <p>Prop 2: {prop2}</p>
    </div>
  );
}
```

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
);  
}
```

export default MyComponent;

In this example, { prop1, prop2 } is used to directly destructure prop1 and prop2 from the props object in the function parameters. This allows you to use prop1 and prop2 directly within the component without referencing props.prop1 and props.prop2.

Destructuring State:

Similarly, you can destructure state variables in a functional component using the useState hook:

jsx

Copy code

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

```
function MyComponent() {  
  // Destructure state variables  
  const [count, setCount] = useState(0);  
  
  const incrementCount = () => {  
    setCount(count + 1);  
  };  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={incrementCount}>Increment</button>  
    </div>  
  );  
}
```

export default MyComponent;

In this example, [count, setCount] is used to destructure the state variable count and the function setCount from the array returned by useState. This allows you to directly use count as the state variable and setCount as the function to update it.

Using destructuring in React components can help make your code more readable and maintainable by reducing verbosity and improving clarity. However, it's important to use

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

destructuring judiciously and consider the readability and maintainability of your code when deciding whether to destructure props and state.



setState

In React.js, the `setState` function is used to update the state of a component. It is primarily used within class components but can also be used within functional components when utilizing the `useState` hook.

Class Components:

In class components, `setState` is a method of the component's instance, and it is used to update the component's state. It takes an object as an argument, which represents the new state values that you want to set. Here's an example:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

import React, { Component } from 'react';

class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  incrementCount = () => {
    // Update count state using setState
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.incrementCount}>Increment</button>
      </div>
    );
  }
}

export default MyComponent;
```

In this example, `setState` is called within the `incrementCount` method to update the count state whenever the button is clicked.

Functional Components (using `useState` hook):

In functional components, `setState` equivalent is `useState` hook's setter function. Here's an example:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

import React, { useState } from 'react';

function MyComponent() {
  // Initialize count state using useState hook
  const [count, setCount] = useState(0);

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={incrementCount}>Increment</button>
    </div>
  );
}

export default MyComponent;
```

In this example, `setCount` is called within the `incrementCount` function to update the count state whenever the button is clicked. Unlike the class component's `setState`, `setCount` doesn't directly take an object representing the new state. Instead, it updates the state with the new value passed as an argument.

Using `setState` (or its equivalent in functional components) is the recommended way to update the state in React components, as it ensures that React will re-render the component with the updated state values.



methods as Props.

Passing methods as props in React allows you to communicate between components by passing functions as props from parent components to child components. This is a common pattern used in React applications for handling events or triggering actions in parent components from child components.

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

Here's an example of how you can pass a method as a prop from a parent component to a child component:

ParentComponent.js:

```
jsx

import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  // Method to handle a button click
  const handleClick = () => {
    console.log('Button clicked in ParentComponent');
  };

  return (
    <div>
      {/* Pass handleClick method as a prop to ChildComponent */}
      <ChildComponent onClickHandler={handleClick} />
    </div>
  );
}

export default ParentComponent;
```

ChildComponent.js:

Shree H. N. Shukla College of It. & Mgmt.

(Affiliated To Saurashtra University)

```
jsx

import React from 'react';

function ChildComponent(props) {
  return (
    <div>
      {/* Button that triggers the handleClick method passed from the p
      <button onClick={props.onClickHandler}>Click me</button>
    </div>
  );
}

export default ChildComponent;
```

In this example:

ParentComponent defines a method called handleClick that logs a message to the console.

ParentComponent renders ChildComponent and passes handleClick as a prop named onClickHandler.

ChildComponent receives onClickHandler as a prop and attaches it to the onClick event of a button.

When the button is clicked in ChildComponent, it triggers the handleClick method defined in ParentComponent.

By passing methods as props in React, you can enable child components to interact with their parent components, allowing for a more flexible and modular architecture. This pattern is commonly used for handling user interactions, form submissions, or any other actions that require communication between components.