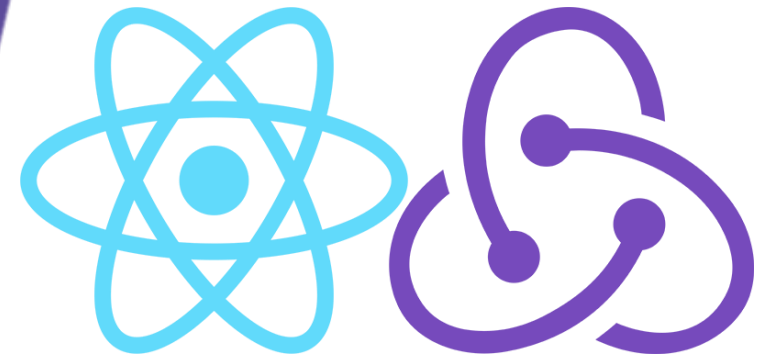


React JS Course



About Us

The world's largest provider of classroom and online training courses

- ✓ World Class Training Solutions
- ✓ Subject Matter Experts
- ✓ Highest Quality Training Material
- ✓ Accelerated Learning Techniques
- ✓ Project, Programme, and Change Management, ITIL® Consultancy
- ✓ Bespoke Tailor Made Training Solutions
- ✓ PRINCE2®, MSP®, ITIL®, Soft Skills, and More



Course Syllabus

Module 1: Introduction to ReactJS	5
Module 2: ReactJS Environment Setup	12
Module 3: Emerging JavaScript	21
Module 4: Functional Programming with JavaScript	33
Module 5: Overview of Pure React	39
Module 6: React with JSX	51
Module 7: Basics of Prop, State and Component Tree	56
Module 8: Enhancing Components	61
Module 9: Introduction to Redux	68



Course Syllabus

Module 10: React Router	85
Module 11: Handling Events	90
Module 12: Conditional Rendering	96
Module 13: Lists and Keys	102
Module 14: Forms	110
Module 15: : Lifting State Up	122
Module 16: Composition vs Inheritance	130
Module 17: Thinking In React	138

Module 1: Introduction to ReactJS

Introduction to ReactJS

- ReactJS is a web-based JavaScript library that creates interactive elements on websites
- This library is developed by Facebook and is used for handling the view layer for mobile and web applications
- It allows us to build reusable User Interface (UI) components
- React offers better rendering performance and simpler programming models
- React implements in the one-way reactive data flow, reducing the boilerplate, and making it easier to reason than traditional data binding

Introduction to ReactJS

(Continued)

- ReactJS is declarative, highly flexible, and efficient for developing simple, scalable, and fast front-end for mobile and web apps
- Developers can create large-scale applications with the JavaScript library named as ReactJS without reloading the page where data is reflected in real-time
- Through continuous improvement, the web development of ReactJS provides several possibilities such as real-time updates, server-side rendering, different rendering goals such as Android, iOS, and so on

Introduction to ReactJS

(Continued)

- One of the selling points of React is that it can also render on the server using Node. It can power native apps using its framework called React Native
- It implements one-way reactive data flow, which reduces the boilerplate making its comprehension easier than traditional data binding
- It uses a concept called Virtual DOM (Document Object Model), that renders subtrees of nodes on changes in state. It does the least amount of DOM manipulation possible to keep your components up-to-date

Introduction to ReactJS

Features of ReactJS

1.

Virtual DOM

2.

JSX (JavaScript Syntax Extension)

3.

One Way Data Binding

4.

React Native

5.

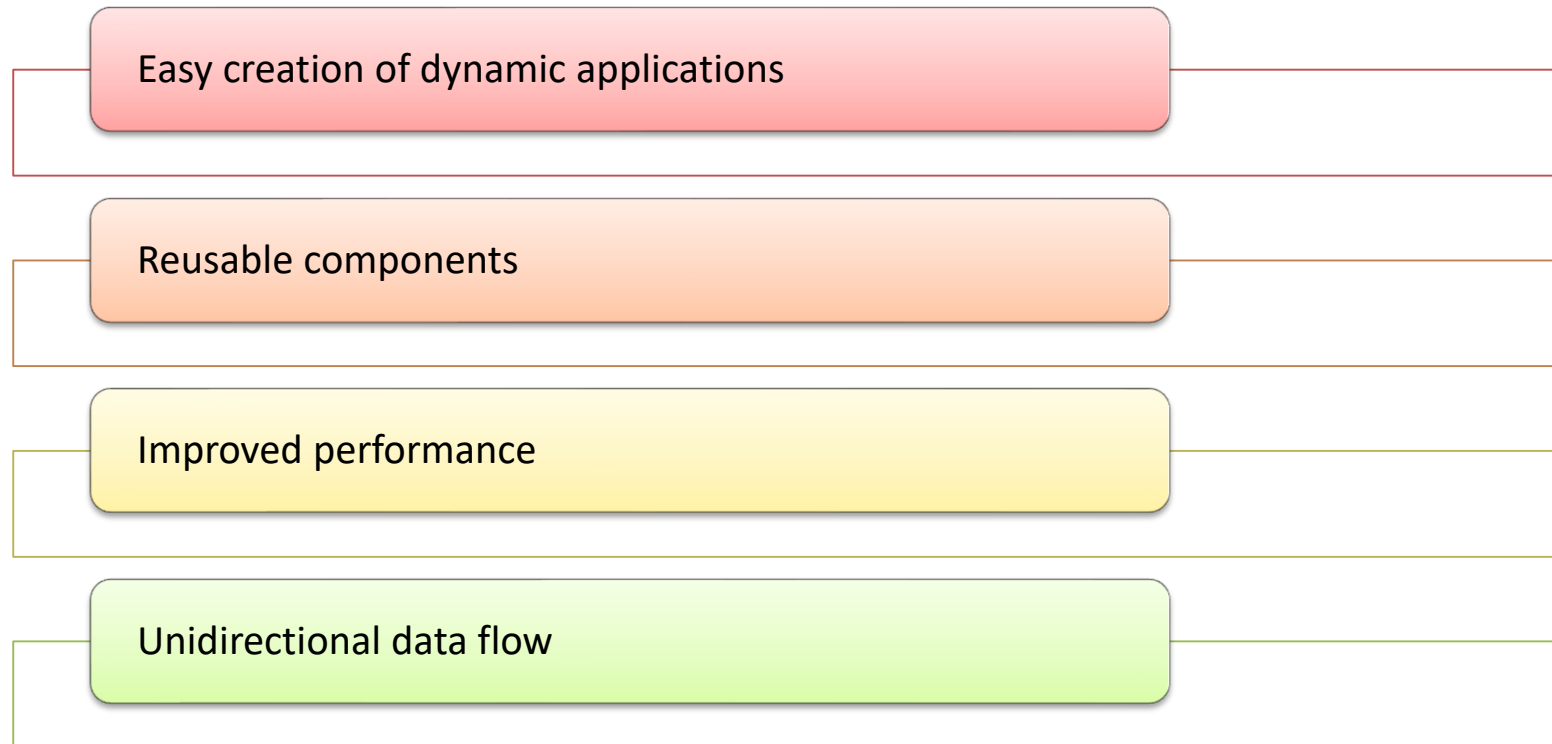
Declarative

6.

Component

Introduction to ReactJS

Why is React so popular?



Introduction to ReactJS

(Continued)

Small learning curve

It can be used for the development of both web and mobile apps

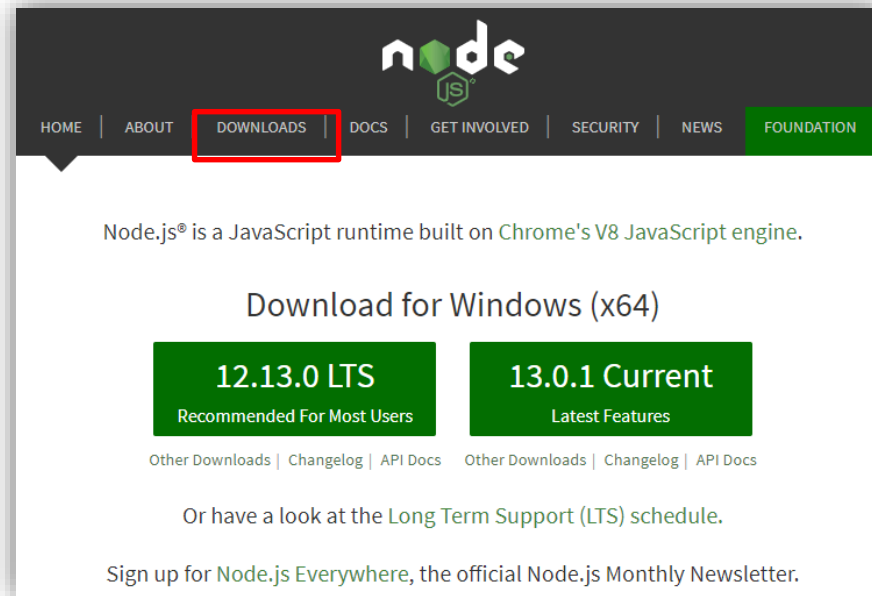
Dedicated tools for easy debugging

Module 2: ReactJS Environment Setup

ReactJS Environment Setup

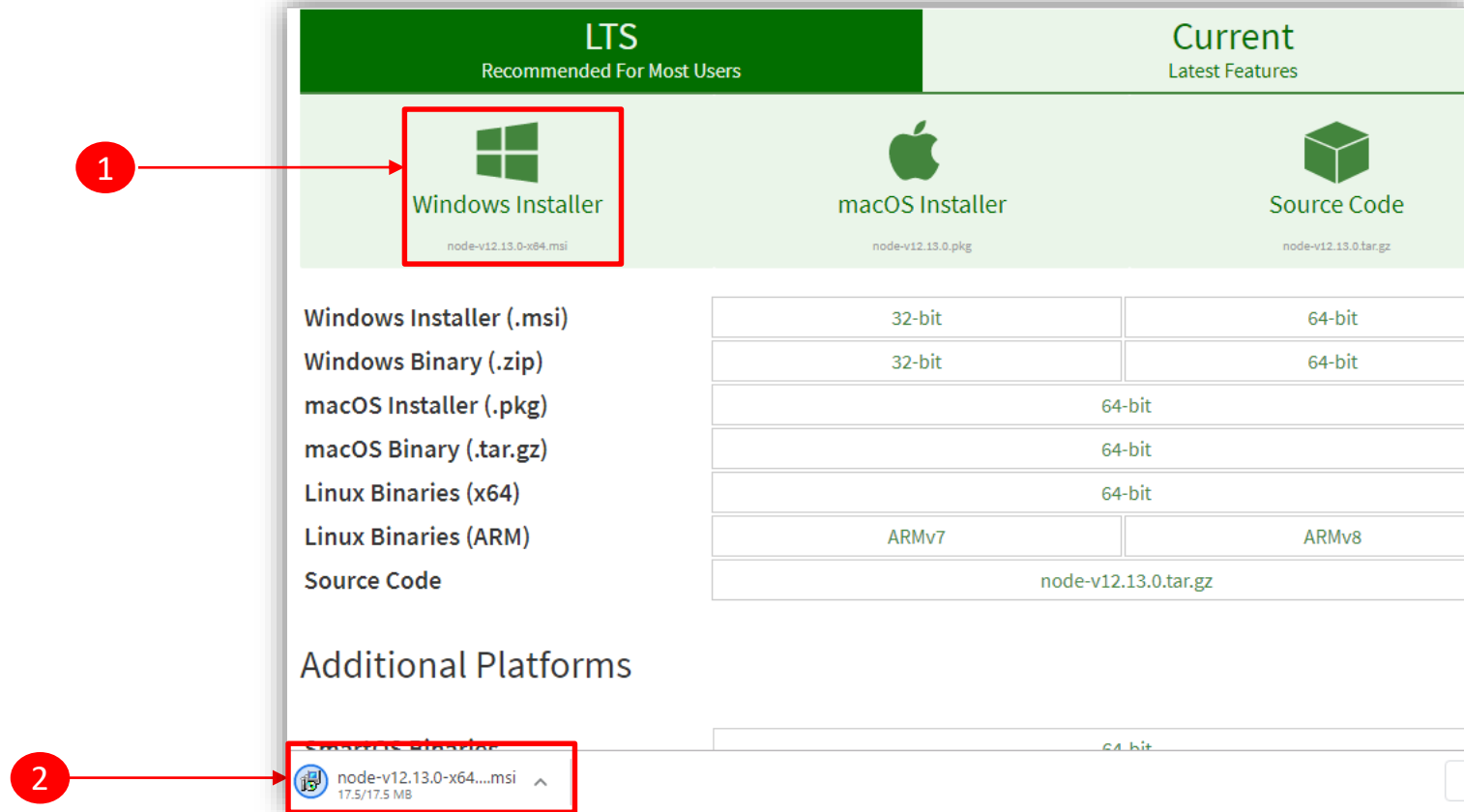
ReactJS Environment Setup through NPM

Step 1: To install ReactJS Environment Setup, type the url: www.nodejs.org and click on **Downloads**



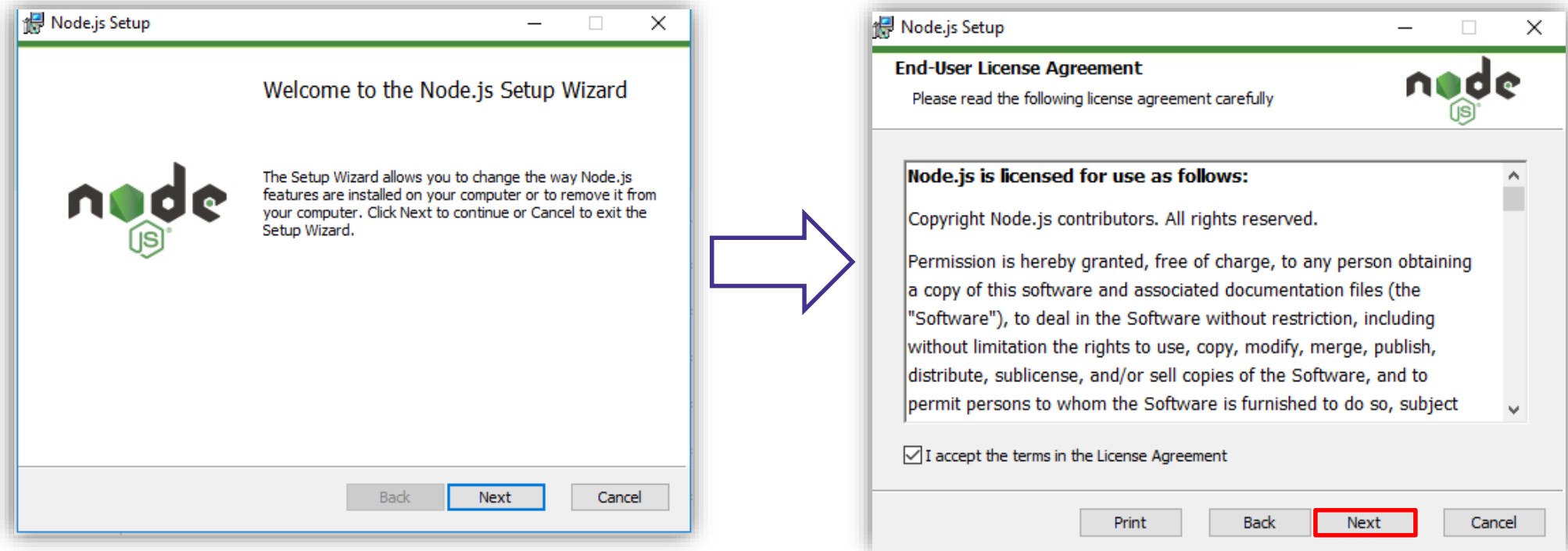
ReactJS Environment Setup

Step 2: Click on the Windows Installer and then click on the setup of the nodejs



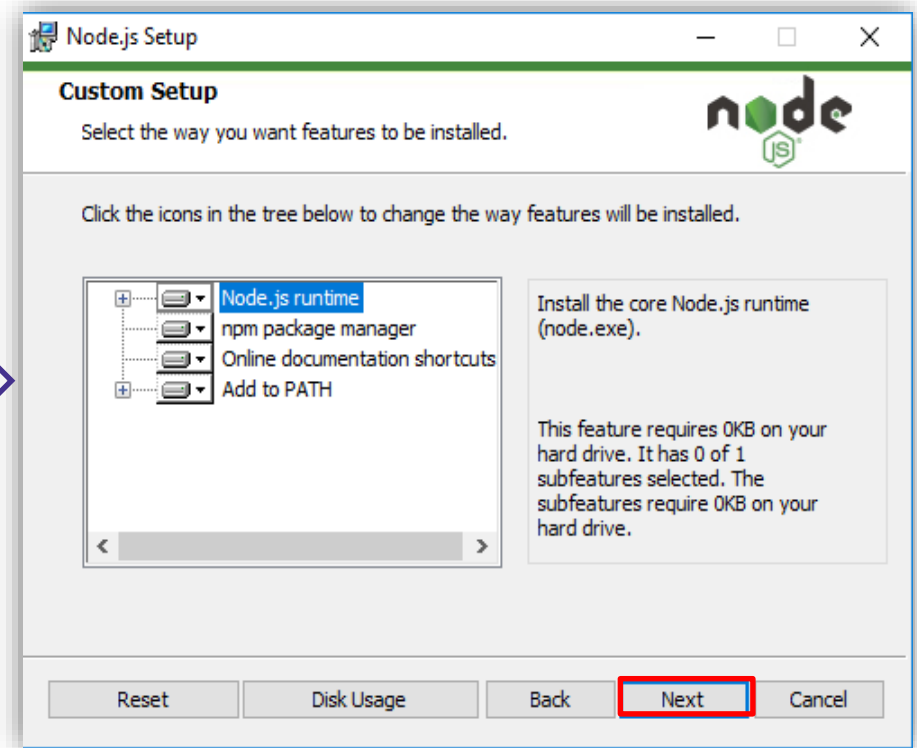
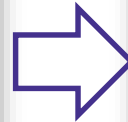
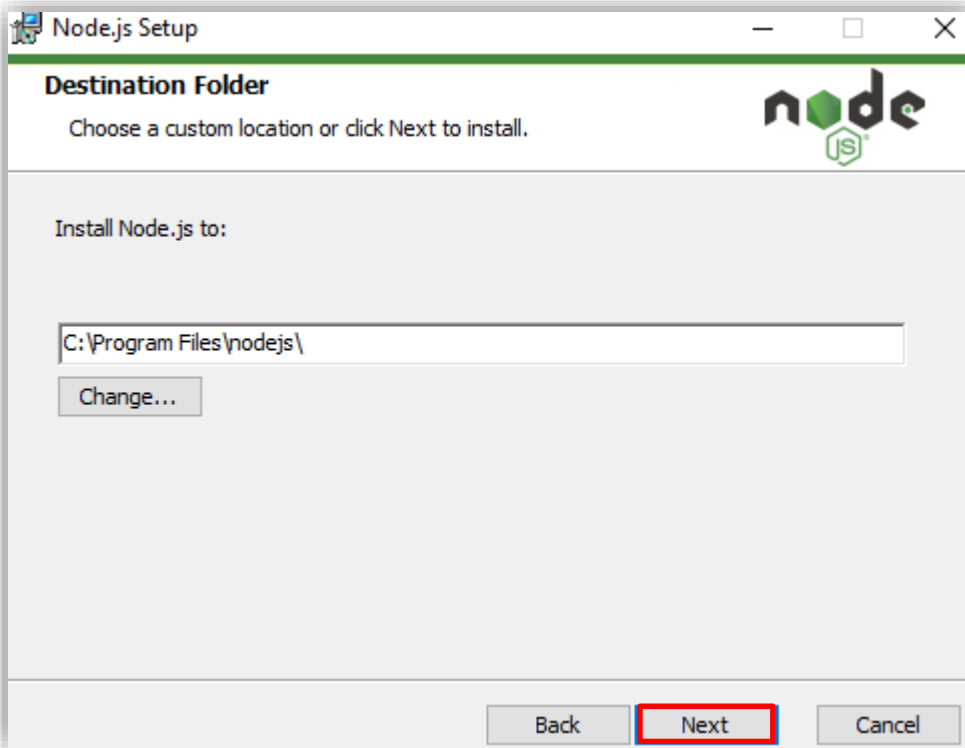
ReactJS Environment Setup

Step 3: Click on the **Next** button. Click on the checkbox and click on the **Next** button



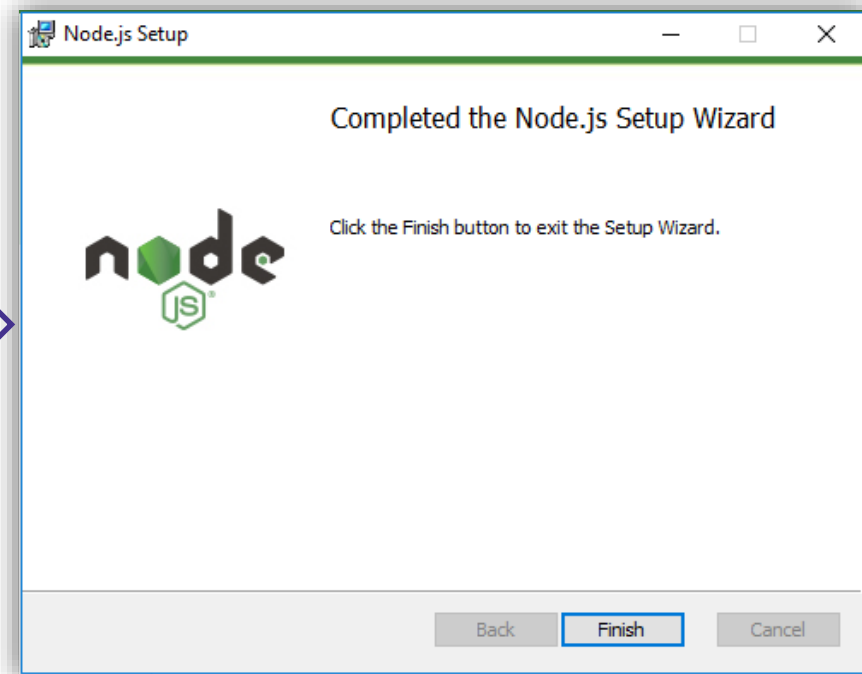
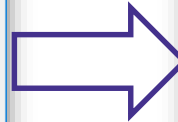
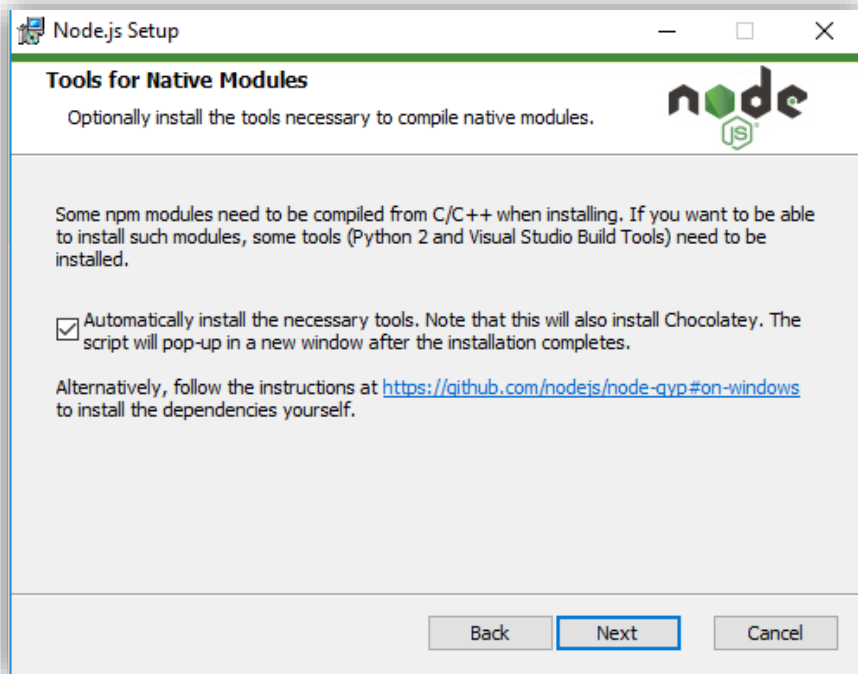
ReactJS Environment Setup

Step 4: Click on the **Next** button on both of the windows



ReactJS Environment Setup

Step 5: Click on the **Next** button and then click the **Finish** button to exit the **Setup Wizard**



ReactJS Environment Setup

Step 6: Open the command prompt by typing the **cmd** in the search bar. Type the **mkdir** command to create the new directory named as **reactdemo**. To move into the created directory, type the **cd reactdemo** command

```
C:\Users\Training>mkdir reactdemo  
C:\Users\Training>cd reactdemo
```

Step 7: Now, we install the package globally named as **create-react-app**

```
C:\Users\Training\reactdemo>npm install -g create-react-app  
C:\Users\Training\AppData\Roaming\npm\create-react-app -> C:\Users\Training\AppData\Roaming\npm\node_mod  
+ create-react-app@3.2.0  
added 91 packages from 45 contributors in 9.954s
```

ReactJS Environment Setup

Step 8: Give the name of the application as **my_app**

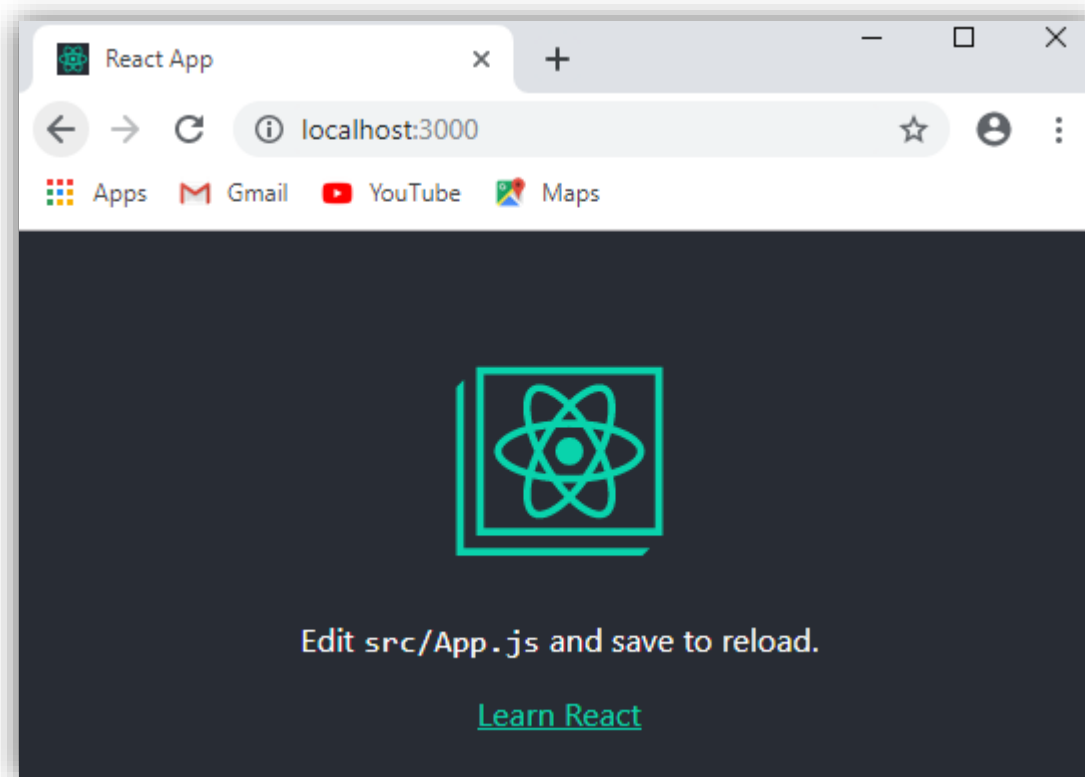
```
C:\Users\Training\reactdemo>create-react-app my_app
```

Step 9: Firstly, go inside the **my_app** directory and type the command **npm start** to start the development server

```
C:\Users\Training\reactdemo>cd my_app  
C:\Users\Training\reactdemo\my_app>npm start_
```

ReactJS Environment Setup

Step 10: The ReactJS environment setup is created



Module 3: Emerging JavaScript

Emerging JavaScript

- JavaScript was released in 1995, and since then it has gone through many changes and emerged accordingly
- In the beginning, its main aim was to add interactive elements to a website in a simple way and after some time it became more powerful and reliable with DHTML and AJAX
- Now, with JavaScript has become a language that is used to build full-stack applications with Node.js
- European Computer Manufacturers Association (ECMA) is incharge of applying changes to JavaScript
- The most recent major update to the specification was approved in June 2015¹ and is called by many names: ECMAScript 6, ES6, ES2015, and ES6Harmony

Emerging JavaScript

- The following are the features and functions which opt to emerging javaScript

Declaring Variables in ES6

Arrow Functions

Transpiling ES6

ES6 Objects and Arrays

Promises

Classes

ES6 Modules

CommonJS

Emerging JavaScript

Declaring Variables in ES6

- Before the emergence of ES6, using var keyword was the only way to declare a variable
- Now, we have different options which helps in improving the functionality



Emerging JavaScript

Arrow Functions

- Arrow functions are a useful new feature of ES6
- One can create functions without using the function keyword by using these functions

For Example:

```
var lordify = firstname => `${firstname} of Canterbury`
```

- Now have a whole function declaration on one line with the arrow

Emerging JavaScript

Transpiling ES6

- ES6 is not supported by all web browsers and converting the ES6 code to ES5 code prior of running it on web browser is the only method to be sure that your ES6 code will work. The process is known as transpiling
- Babel is one of the most popular tools for transpiling
- Example: ES6 code before Babel Transpiling

```
const add = (x=5, y=10) => console.log(x+y);
```

Emerging JavaScript

(Continued)

- After we run the transpiler on this code, here is what the output will look like:

```
"use strict";

var add = function add() {
  var x = arguments.length <= 0 || arguments[0] ===
undefined ?
    5 : arguments[0];
  var y = arguments.length <= 1 || arguments[1] ===
undefined ?
    10 : arguments[1];
  return console.log(x + y);
};
```

Emerging JavaScript

(Continued)

- In order to run the code in strict mode, the transpiler added a “use strict” declaration
- By using the inline Babel transpiler, you can transpile JavaScript directly in the browser. You just include the browser.js file, and any scripts with type="text/babel"

```
<script  
  src="https://cdnjs.cloudflare.com/ajax/libs/babel-  
core/5.8.23/browser.js">  
</script>  
<script src="script.js" type="text/babel">  
</script>
```

Emerging JavaScript

ES6 Objects and Arrays

- ES6 introduces us with the new options to work with objects and arrays and for scoping the variables
- The features consist of de-structuring assignment, object literal enhancement, and the spread operator

Promises

- Promises helps us to make sense out of asynchronous behaviour

Emerging JavaScript

(Continued)

- While making an asynchronous request, one of two things can occur: everything goes as we hope or there is possibility of an error
- There may be various different types of successful or unsuccessful requests. Promises helps us to simplify back to a simple pass or fail

Classes

- In Javascript, there are no official classes. However, ES6 introduce us with the concept of class declaration
- Functions in ES6 are objects, and inheritance is carried out with the help of prototype

Emerging JavaScript

ES6 Modules

- A JavaScript *module* can be defined as a piece of reusable code which can be easily integrated into other JavaScript files
- JavaScript modules are stored in separate files, one file per module
- With ES6, Javascript supports modules by itself

Emerging JavaScript

CommonJS

- CommonJS can be defined as the module pattern that is supported by all versions of Node.js
- With CommonJS, JavaScript objects are exported using `module.exports`
- CommonJS does not support the **import** statement

Module 4: Functional Programming with JavaScript

Functional Programming

Introduction

- Functional Programming (FP) is a paradigm of programming with some specific techniques
- It treats computation as the evaluation of mathematical functions
- It is the process of building software by composing pure functions and avoiding mutable data, shared state, and side effects

Functional Programming

- The following are some examples of Functional Programming:

1. Pure functions:

- A key concept of functional programming is that functions should not have side effects and should not depend on external state
- It means a function should take certain input and return certain output without modifying or accessing any value outside the function

Functional Programming

2. Higher Order Functions:

- Higher Order Functions are functions that take other functions as parameters
- The following are some examples of Higher order Functions:
 - **Filter:** It is a method of arrays. It accepts a test function as argument which should return a boolean, and returns a new array with only the elements for which the test function returned true

Functional Programming

Example of Filter

```
C: > Users > Training > Documents > JS 2.js > [?] numbers
1 function isEven(x){
2     return x % 2 === 0;
3 }
4
5 const numbers = [2,56,11,34,22,890,786];
6 const evenNumbers = numbers.filter(isEven);
7 console.log(evenNumbers);
```

Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Program Files\nodejs\node.exe --inspect-brk=28256
Debugger listening on ws://127.0.0.1:28256/ab0f6e01-5
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
> Array(6) [2, 56, 34, 22, 890, 786]
```

Functional Programming

- **Lambda function:** While defining short functions, lambda function allows us to define anonymous functions in a more compact way:
(/*arguments*/) => { /*value to return*/ }
- **Example:**

C: > Users > Training > Documents > JS 2js > ...

```
1 const numbers = [34,90,45,33,765,56,453];  
2 const isEven = (x) => x % 2 === 0  
3 const evenNumbers = numbers.filter(isEven);  
4 console.log(evenNumbers);
```

Output

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Program Files\nodejs\node.exe --inspect-brk=26355  
Debugger listening on ws://127.0.0.1:26355/7bf7002c-4  
For help, see: https://nodejs.org/en/docs/inspector  
Debugger attached.  
> Array(3) [34, 90, 56]
```

Module 5: Overview of Pure React

Overview of Pure React

Page Setup

- We need to include two libraries to work with React in the browser: React and ReactDOM
- React is the library to create views
- ReactDOM is the library provides you to render the User Interface (UI) in the browser
- We also need an HTML element for the UI to be rendered by ReactDOM

Overview of Pure React

The Virtual DOM

- Hypertext Markup Language (HTML) is simply a collection of instructions that are followed by browser while constructing the document object model (DOM)
- When the browser loads HTML and renders the user interface, the elements that make up an HTML document become DOM elements
- Websites have traditionally been made up of independent HTML pages. The browser would request and load different HTML documents when the user browsed these pages
- AJAX's invention brought us a single-page application or SPA

Overview of Pure React

(Continued)

- Whole web apps could now run out of a single page and depend on JavaScript to update the user interface because browsers could request and load tiny bits of data using Asynchronous JavaScript and XML (AJAX)
- The browser first loads an HTML document in an Single Page Application (SPA)
- Users are actually staying on the same page as they navigate through the web. JavaScript destroys and builds a new user interface as the user interacts with the application

Overview of Pure React

(Continued)

- It might seem like you are navigating from page to page, but you are still on the same HTML page, and JavaScript is doing the heavy lifting
- The DOM API is a set of objects that JavaScript can use for browser interaction to modify the DOM
- React is a library that is designed for us to update the browser DOM
- We do not interact directly with the DOM API with React. Instead, we are dealing with a Virtual DOM, or React will use a set of instructions to build the UI and interact with the browser

Overview of Pure React

(Continued)

- The virtual DOM consists of React elements, which appear conceptually identical to HTML elements, but are actually JavaScript objects
- Working directly with JavaScript objects is much faster than operating with the DOM API
- We make modifications to a JavaScript object, the virtual DOM, and React renders those changes for us using the DOM API as efficiently as possible

Overview of Pure React

React Elements

- The browser DOM consists of DOM elements. In the same way, the React DOM consists of React elements
- DOM elements and React elements may look similar, but in fact, they are quite different
- A React element is a definition of what the actual DOM element should appear
- Also, React elements are the instructions on how to build the browser DOM
- React elements are the instructions in the browser that React will use to build the UI

Overview of Pure React

ReactDOM

- ReactDOM includes the tools in the browser to render React elements
- ReactDOM is where we will find the render method as well as the `renderToStaticMarkup` and `renderToString` methods that are used on the server
- Using `ReactDOM.render`, we can render a React element including its children to the DOM
- The virtual DOM is a tree of React elements that all come from a single root element

Overview of Pure React

Constructing Elements with Data

- The significant advantage of React is its ability to isolate information from UI elements
- Since React is simply JavaScript, we can add JavaScript logic to help us in building the component tree of React
- If we create a list of child elements by iterating through an array, React likes to have a key property for each of those elements
- React uses the key property to help it update the DOM efficiently

Overview of Pure React

Components of React

- Each user interface consists of parts. In React, each of these parts is known as a component
- React Components enable us to reuse the same DOM structure for different sets of data
- There are three different ways to create components:
 - i. `createClass`
 - ii. ES6 classes
 - iii. stateless functional components

Overview of Pure React

DOM Rendering

- As we can pass data to our components as props, we are able to separate the data of our application from the logic that is used to build the UI
- This provides us with an isolated set of data that is much easier to work with and handle than the DOM
- When we change any of the values in this separate dataset, we change the state of our application
- Imagine storing all data in a single JavaScript object in your application

Overview of Pure React

(Continued)

- Every time a change is made to this JavaScript object, you could send it to a component as props and rerender the User Interface (UI)
- We can render the DOM using ReactDOM.render method
- ReactDOM.render has to work smart to work in a reasonable amount of time
- Instead of emptying and rebuilding of the entire DOM, ReactDOM.render leaves the existing DOM in place and only applies the minimum amount of modifications necessary to mutate the DOM

Module 6: React with JSX

React with JSX

Introduction

- React uses JSX instead of standard JavaScript for templating
- The following are some pros that come with it:
 - It is faster because it performs optimisation while compiling code to JavaScript
 - Templates can be written in an easier and faster way if you are familiar with HTML
 - It is also **type-safe**, and during compilation, most of the errors can be identified

React with JSX

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">\
  <title>reactapp</title>
  <link rel="stylesheet" href="app.css">
</head>
<body>
  <h1>Live Server Extension- updated</h1>
```

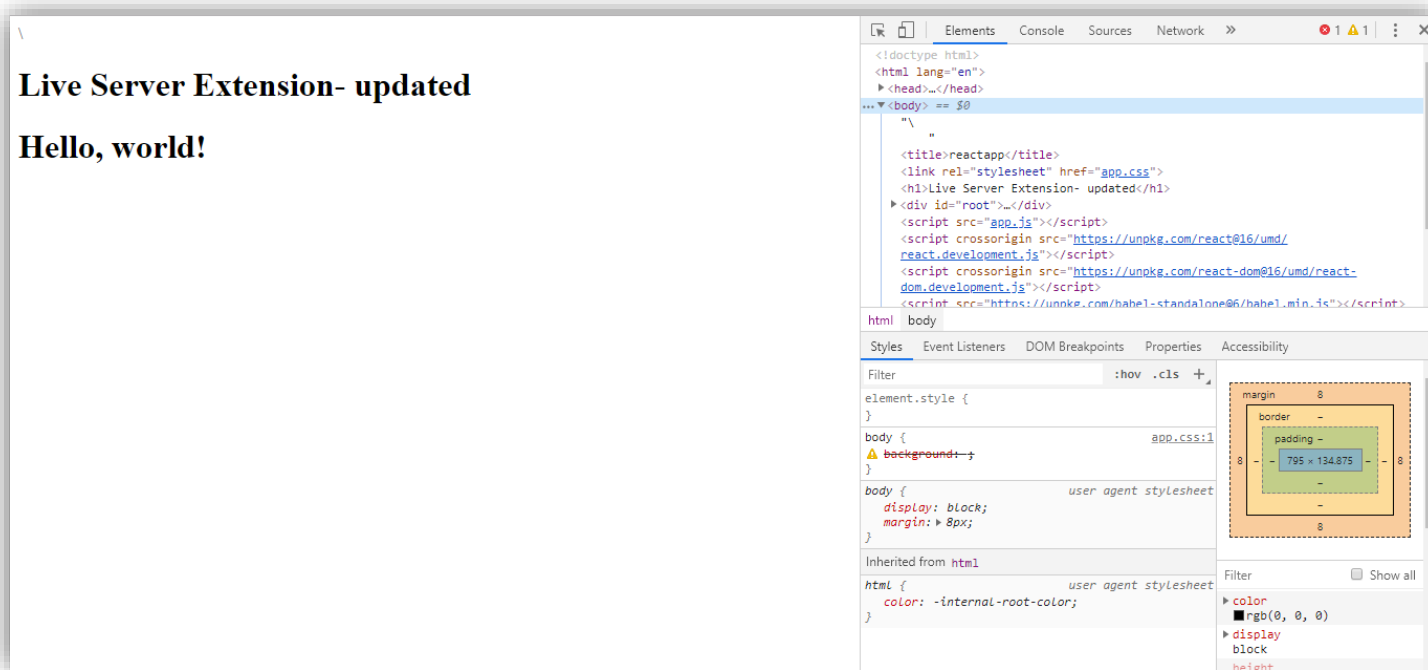
React with JSX

(Continued)

```
<div id="root"></div>
  <script src = 'app.js'></script>
  <script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
  <script type="text/babel">
    var element = <h1>Hello, world!</h1>;
    ReactDOM.render(
      element,
      document.getElementById('root')
    );
  </script>
</body>
</html>
```

React with JSX

Output



Module 7: Basics of Prop, State and Component Tree

Basics of Prop, State and Component Tree

- React provides two primary mechanisms to provide data to components. These are:



- The **Props** are read-only and permit a parent component to transfer attributes to a child component
- The **State** is local and encapsulated inside the component and can change anytime in the component's lifecycle

Property Validation

- Properties validation is a helpful method to force the accurate usage of the components
- During the development, it will help to avoid future bugs and problems
- It makes the code more readable because we can see the usage of every component
- React components use the unique property PropTypes which helps to catch bugs by validating data types of values that pass through props, however, it is not required to defining components with prototypes

Property Validation

- **App.propTypes** is used for props validation in react component
- You will get the warning on JavaScript console when some of the props are passed with an invalid type
- After specifying the validation patterns, you will set the **App.defaultProps**

Syntax:

```
class App extends React.Component {  
  render() {}  
}  
Component.propTypes = { /*Definition */};
```

React State Management

- State management has always been an essential yet somewhat terrible aspect for working with React
- From the Complaints about Redux making it difficult to keep components self-contained to testing with the new Context API, most are continually assessing for the correct answer to pick for their team
- React Contains different ways of managing state in an application
- Defining a state inside a component is the most simple way
- The component's state is the same as the props that are passed to a component, a plain JavaScript object containing information that impacts the manner in which a component is rendered

Module 8: Enhancing Components

Enhancing Components

Components Lifecycle

- Each React Component has its own lifecycle that can be defined as the series of methods that are invoked in the various stages of the component's existence
- There are four stages of a React Component's Lifecycle:
 - **Initialisation:** In this stage, the component is constructed with the given Props and default state. It is done in the constructor of a Component Class
 - **Mounting:** It is the stage of rendering the JSX that is returned by the render method itself

Enhancing Components

(Continued)

- **Updating:** It is the stage when the component's state is updated, and the application is repainted
- **Unmounting:** Unmounting is the final step of the component lifecycle where the component is removed from the page

Enhancing Components

JavaScript Library Integration

- Frameworks like Angular and jQuery have their own tools for accessing data, rendering the UI, handling routing, modeling state, and more
- On the other hand, React is simply a library for creating views, so we may need to work with other libraries in JavaScript

Higher-Order Components

- A higher-order component (HOC) is simply a function that takes a React component as an argument and returns another React component

Enhancing Components

(Continued)

- Higher-order components wrap the incoming component with a class that has functionality or maintains state
- These components are the best way to reuse functionality through React components
- It allows us to wrap a component with another component
- These components are a great way to summarise the details of how component state or lifecycle are managed

Enhancing Components

Managing State Outside of React

- Managing state outside of React refers to not using React state or setState in your applications
- It reduces the need for class components, if any
- It is more easy to keep your components stateless mostly, if you are not using state
- Stateless functional components are easier to understand as well as test
- They fit into strictly functional applications in a very nice manner as they are pure functions

Enhancing Components

Basics of Flux

- Flux is a design pattern which was developed by Facebook and was designed to keep the flow of data in particular direction
- Web development architecture was influenced by variations of the MVC design pattern, prior to the introduction of Flux
- Flux is complementary to MVC which is a completely unique design pattern that complements the functional approach
- Flux gives us a way to design web applications that complements how React works. In particular, Flux provides a way to provide the data that React will use to create a UI

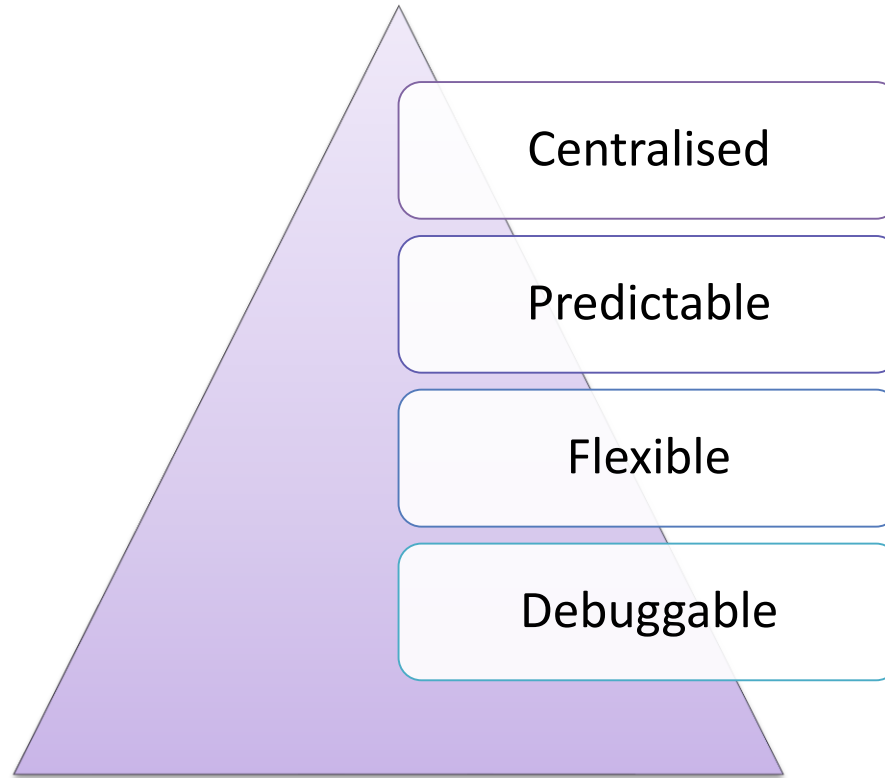
Module 9: Introduction to Redux

Introduction to Redux

- Redux was created by Dan Abramov in June 2015
- Redux is a predictable state container for JavaScript apps and a very useful application for organising application state
- Redux got popular very quickly because of its simplicity, small size and excellent documentation
- However, Redux is most often used together with React

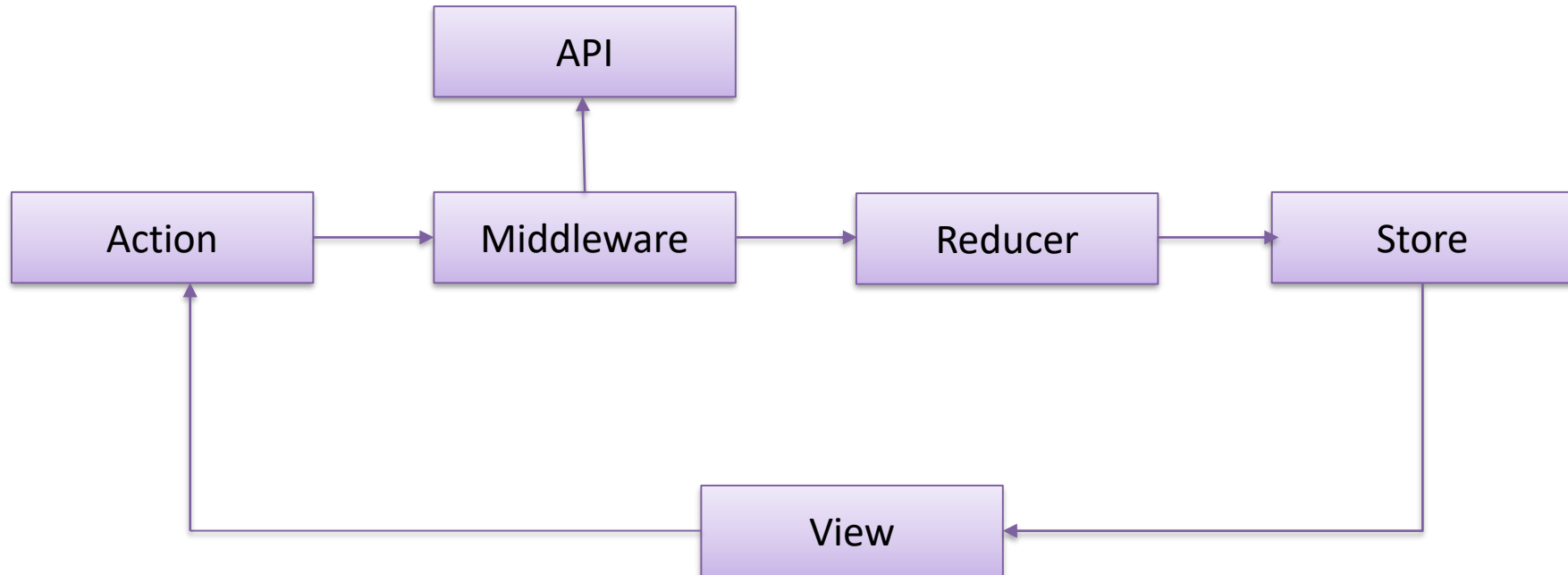
Introduction to Redux

- The following are the characteristics of Redux:



Introduction to Redux

- Redux is an architecture with a unidirectional data flow that makes it simple to build applications that are easy to test and maintain



State and Actions

Introduction to State

- State (also called the state tree) is a broad term, but in Redux API it usually refers to the single state value that is maintained by the store and returned by `getState()`
- It represents the entire state of a Redux application, which is often an extremely nested object
- **Syntax:**

```
type State = any
```


State and Actions

Introduction to Actions

- An action is a plain object that represents the purpose of changing the state
- Actions are the only method to access data into the store. Any data, whether from UI (User Interface) events, network callbacks, or other sources like Web Sockets, needs to be dispatched as actions eventually
- **Syntax:**

```
type Action = Object
```

Action Creators

Introduction to Actions Creators

- Action objects are JavaScript literals and action creators are functions that create and return these literals
- Action Creators are used to simplify the task of dispatching actions, when we only need to call a function and send it the required data
- The best thing regarding action creators is that to create an action, all the necessary logic can be encapsulated

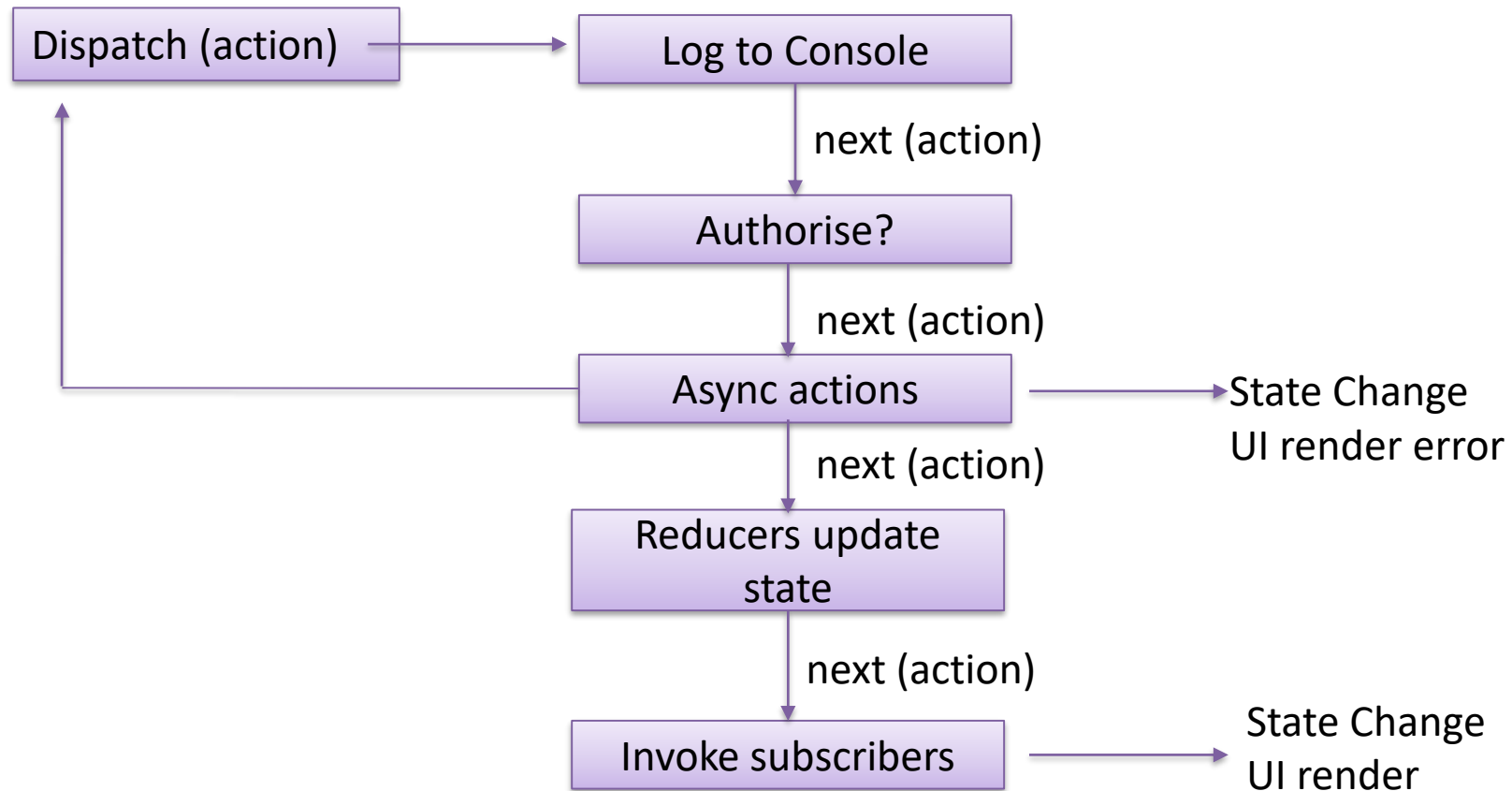
Middleware

Introduction to Middleware

- Redux also has middleware and it acts on the store's dispatch pipeline
- In the process of dispatching an action, redux middleware consists of a series of functions that are executed in a row

Middleware

- The following figure describes the execution of middleware functions sequentially



Overview of React Redux

Introduction

- React is a component-based front end framework tool that connects several segments of the web page
- In react, props i.e. short for properties are used in a component that permits the usage of non-static variables
- Using props, we can pass these variables into a number of other components i.e. child components from the parent component

Introduction to React Redux

(Continued)

- The state's presence in react permits defining its own variables of a component
- With some components in our application, we can pass these states as props to the child components
- But as the number of components rises in the application in agreement with its large objective, we need to pass these state to other components that are located far away from each other in the component tree
- At this moment, the typical way of transferring state as props turns out to be complex as not every component is a parent to the component requiring the state

Introduction to React Redux

(Continued)

- This brings the requirement for **react-redux** in our react application
- React-Redux which is a state management tool makes it easier to pass these states from one component to another regardless of their position in the component tree and therefore prevents the complexity of the application

Introduction to React Redux

Presentational Versus Container Components

- The react components are divided into two types and each will have different characteristics:
 - i. **Presentational Components:** These components are concerned with generation of some markup to be outputted
 - They do not manage any type of state, excluding the state related to the presentation
 - ii. **Container Components:** These components are typically concerned with the “backend” operations
 - They might handle the state of several sub-components. They might wrap many presentational components. They might interface with Redux

Introduction to React Redux

The React Redux Provider

- The `< Provider />` makes the Redux store available for all nested components that are wrapped in the `connect()` function
- As any React component in a React-Redux app can be connected, most applications would render a `< Provider >` at the top level, with the entire app's component tree inside it
- Usually, a connected component can not be used unless it is nested within a `< Provider>`

Introduction to React Redux

React Redux Connect

- The `connect()` function is used to connect a React component to a Redux store
- It provides its connected component with the pieces of the data it requires from the store, and the functions it can use to dispatch actions to the store
- It does not alter the component class passed to it; instead, it returns a new, connected component class that wraps the component you passed in

Introduction to React Redux

React Redux Connect

```
function connect(mapStateToProps?, mapDispatchToProps?, mergeProps?, options?)
```

- The **mapStateToProps** and **mapDispatchToProps** deals with the state and dispatch of Redux store respectively.
- State and dispatch will be supplied as the first argument to your **mapStateToProps** or **mapDispatchToProps** functions

Introduction to React Redux

(Continued)

connect() Parameters

The following are four different parameters (Optional):

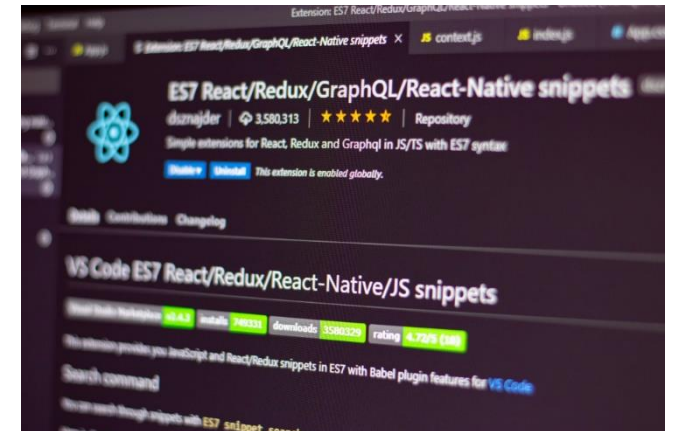
- i. mapStateToProps?: Function
- ii. mapDispatchToProps?: Function | Object
- iii. mergeProps?: Function
- iv. options?: Object

Module 10: React Router

React Router

Introduction to Routing

- Routing is a process of defining endpoints for the request of your clients
- These endpoints work in conjunction with the location of the browser and history objects
- They are used to identify requested content to load JavaScript and provide the appropriate user interface



React Router

Introduction to React Router

- React is a great tool to creating single-page apps
- React Router is used in larger projects and can save quite a bit of headache when the scope of an app gets more extensive and complex

Installation of Set Up:

- You can use npm to install the React Router

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

Nesting Routes

- Route components are used with content that should be displayed when particular URLs are matched
- This feature enables us to organise our web applications into eloquent hierarchies that facilitate the reuse of content
- The React Router provides us to compose Route components anywhere in our application because our root element is the HashRouter

Router Parameters

- The facility to set up routing parameters is another most useful feature of the React Router
- Routing parameters can be considered as the variables that obtain their values from the URL
- These are incredibly useful for data-driven web applications to filter content and control display requirements
- It provides the following features:
 - Adding Colour Details Page
 - Moving Colour Sort State to Router

Module 11: Handling Events

Handling Events

- React events are not named using lowercase. CamelCase letters are used to name React events
- A function is passed as the event handler With JSX (JavaScript XML) instead of a string
- For instance, the HTML:

```
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

- Is somewhat In React:

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

Handling Events

(Continued)

- In the following example, we have used only one component and attaching an **onChange event**. The **onChange event** will trigger the **changeText** function, which returns the company name

```
1  import React, { Component } from 'react';
2  class App extends React.Component {
3      constructor(props) {
4          super(props);
5          this.state = {
6              companyName: ''
7          };
8      }
9      changeText(event) {
10         this.setState({
11             companyName: event.target.value
12         });
13     }
14     render() {
15         return (
16             <div>
```

Handling Events

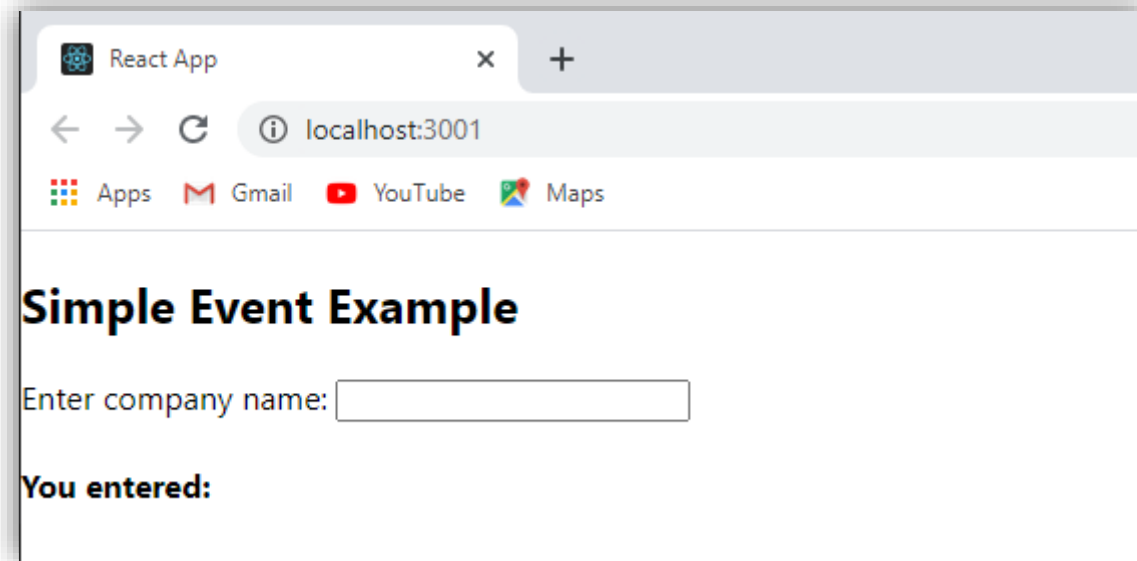
(Continued)

```
17     <h2>Simple Event Example</h2>
18     <label htmlFor="name">Enter company name: </label>
19     <input type="text" id="companyName" onChange={this.changeText.bind(this)}>
20     <h4>You entered: { this.state.companyName }</h4>
21   </div>
22   );
23 }
24 }
25 export default App;
26
```

Handling Events

(Continued)

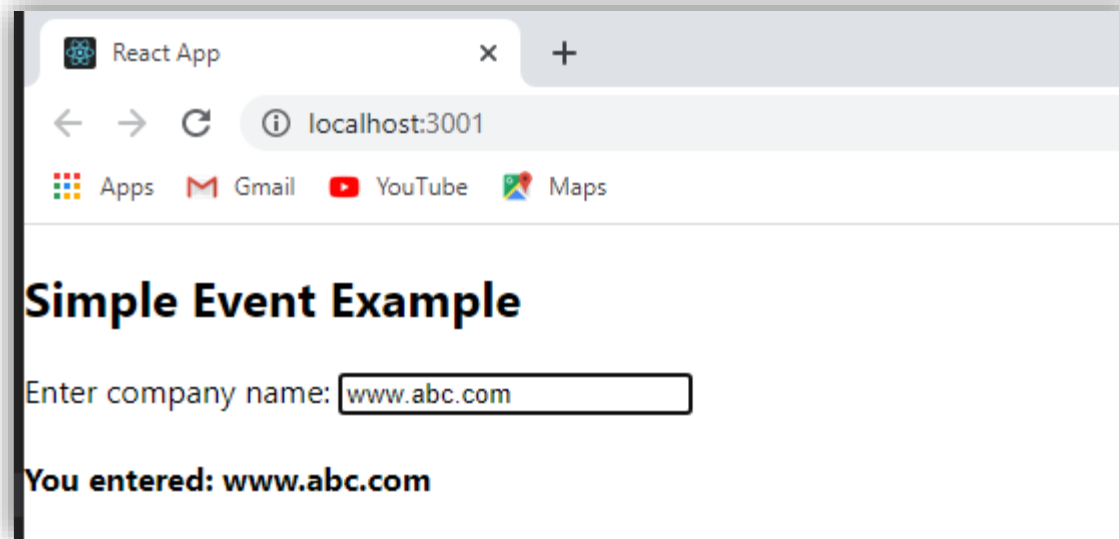
- **Output:**



Handling Events

(Continued)

- After writing the name in the textbox, you will get the output as shown below:



Module 12: Conditional Rendering

Conditional Rendering

- Conditional rendering is defined as the ability to render various user interface (UI) markup if a condition is true or false
- In React, it permits us to render different components or elements based on a condition
- In ReactJS, you can render only the required components based on specific given conditions by using the logical `&&` operator or by using the if-else statements of JavaScript
- When the given condition is satisfied, then React will match the UI and then update it accordingly

Conditional Rendering

Using if-else Conditions

```
.vscode > hello-world > src > JS App.js > [🔍] Form
1  import React, { useState } from 'react';
2  const App = () => {
3    const [isLogin, setIsLogin] = useState(true);
4    return (
5      <div>
6        <div>Username:User1</div>
7        <div onClick={() => setIsLogin((prev) => !prev)}>
8          {isLogin ? <Form login /> : <Form logout />}
9        </div>
10     </div>
11   );
12 };
13
14 const Form = ({ login, logout }) => {
15   return (
16     <div>
17       {login ? (
18         <>
19           <input placeholder="Email" />
20           <input placeholder="Password" />
```

Conditional Rendering

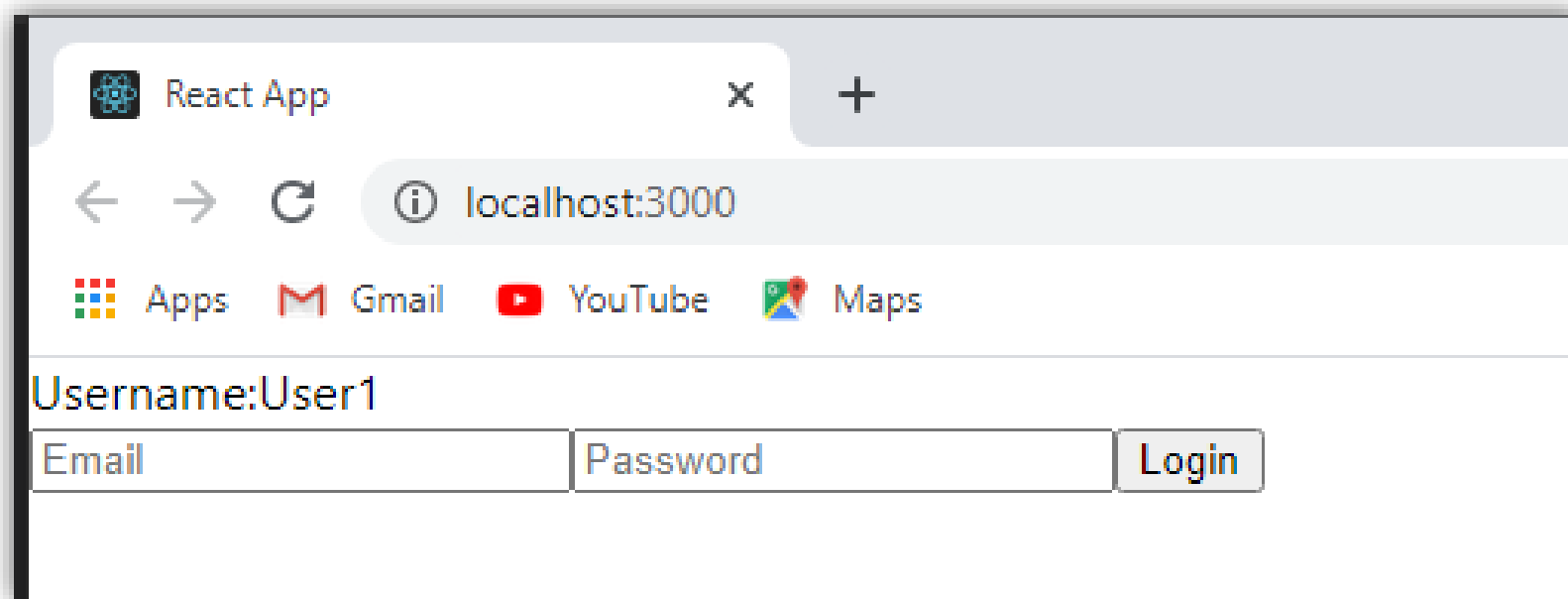
(Continued)

```
21     <button>Login</button>
22   </>
23   ) : null}
24   {logout ? (
25     <button>Logout</button>
26   ) : null}
27 </div>
28 );
29 };
30 export default App;
31
```

Conditional Rendering

(Continued)

- **Output:**



Conditional Rendering

Using logical &&

```
.vscode > hello-world > src > JS App.js > ...
1  import React, { useState } from 'react';
2  const App = () => {
3    const [isLoggedIn, setIsLoggedIn] = useState(true);
4    return (
5      <div>
6        <div>Username: User1</div>
7        <div onClick={() => setIsLoggedIn((prev) => !prev)}>
8          {isLoggedIn ? <Form login /> : <Form logout />}
9        </div>
10     </div>
11   );
12 };
13
14 const Form = ({ login, logout }) => {
15   return (
16     <div>
17       {login && (
18         <>
19           <input placeholder="Email" />
20           <input placeholder="Password" />
```

Conditional Rendering

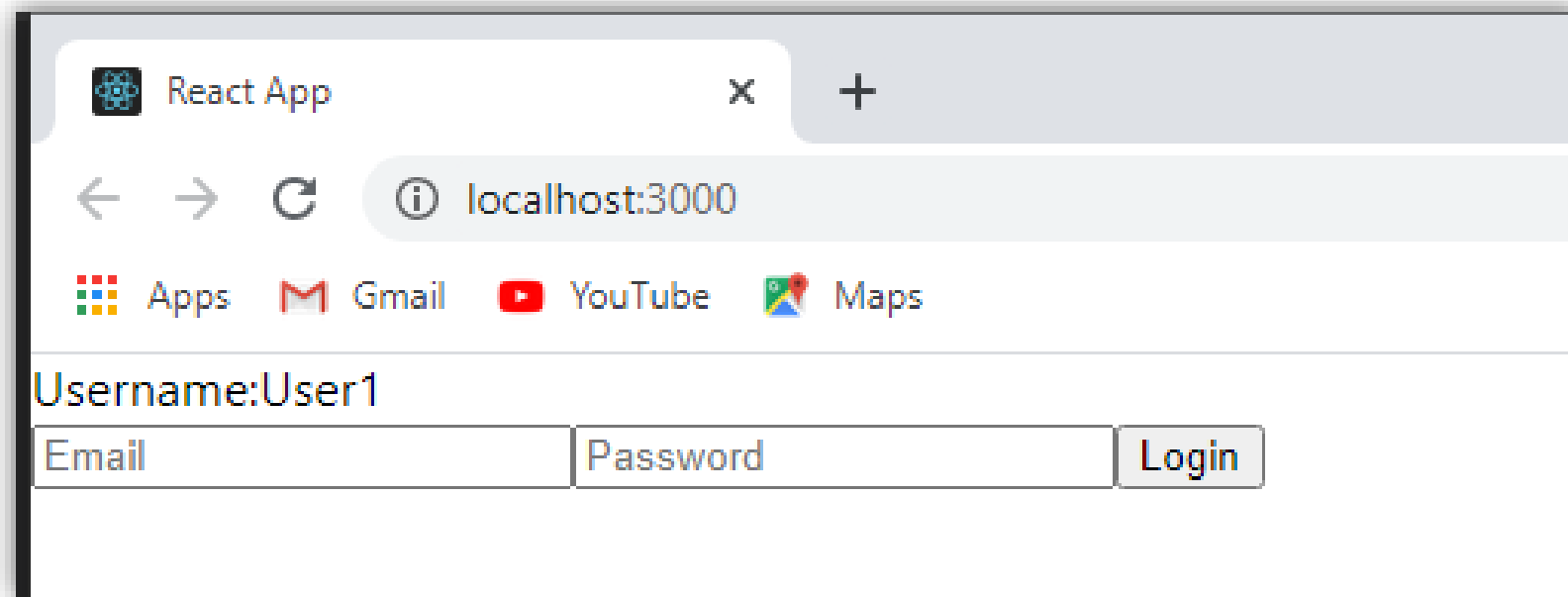
(Continued)

```
21     <button>Login</button>
22   </>
23   )}
24   {logout && (
25     <button>Logout</button>
26   )}
27 </div>
28 );
29 };
30 export default App;
31
```

Conditional Rendering

(Continued)

- **Output:**



Module 13: Lists and Keys

Lists and Keys

Lists

- Lists are used for displaying data in an ordered format. Also, lists can be used for displaying menus on websites
- In React, Lists can be created as same as we create lists in JavaScript. For traversing the lists, the `map()` function is used

Lists and Keys

(Continued)

- The **map()** function takes an array of numbers and multiply their values with 5 in the following example. Moreover, the new array returned by **map()** is assigned to the variable **multiplyNums** and log it

```
var numbers = [11, 12, 13, 14, 15];  
const multiplyNums = numbers.map((number)=>{  
  |   | return (number * 5);  
});  
console.log(multiplyNums);
```

Lists and Keys

Keys

- It is a unique identifier. In React, we use it for identifying which terms have updated, altered, or eliminated from the Lists. It is useful when the users change the lists or when components are created dynamically
- Moreover, it also helps for determining which components in a collection needs to be re-rendered rather than re-rendering the whole set of components every time
- For giving the elements a stable identity, keys must be given inside the array. The best approach to select a key as a string which uniquely point out the items in the list

Lists and Keys

(Continued)

For example

```
const stringLists = [ 'Jack', 'Mark', 'Joe', 'Oliver', 'Sam' ];  
  
const updatedLists = stringLists.map((strList)=>{  
  <li key={strList.id}> {strList} </li>;  
});
```

Lists and Keys

(Continued)

- The item **index** can be assigned as a key to the lists if there are no stable IDs for rendered items. Have a look on the following example:

```
const stringLists = [ 'Jack', 'Mark', 'Joe', 'Oliver', 'Sam' ];  
  
const updatedLists = stringLists.map((strList, index)=>{  
  <li key={index}> {strList} </li>;  
});
```

Module 14: Forms

Forms

- These are an integral part of any modern web application. It enables the users for interacting with the application and collect information from the users
- Forms can perform various tasks which depend on your business demands and logic, like the user's **authentication, adding user, filtering, searching, ordering, booking** etc
- **Buttons, text fields, radio buttons, checkbox** etc., can be contained by a form

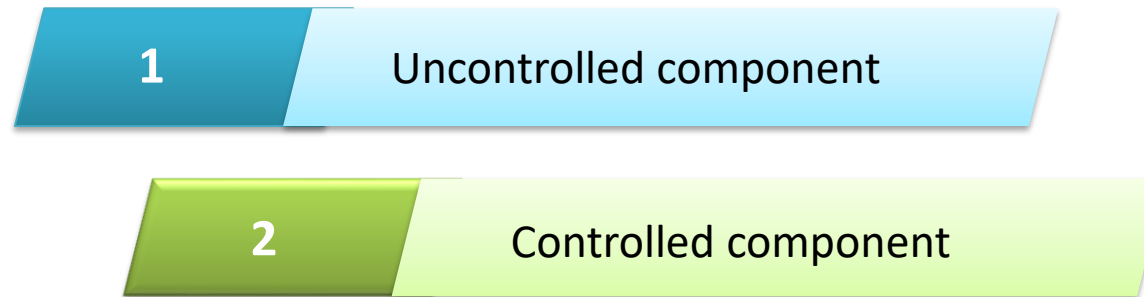
Creating Form

- For building a form, a stateful, reactive approach is offered by React. Instead of the DOM, The components usually handles the React form
- Usually, the form is implemented with the usage of controlled components in React

Forms

(Continued)

- In React, there are mainly two types of form:



Forms

(Continued)

1. Uncontrolled component

- The uncontrolled input is as same as inputs of the traditional HTML form. The DOM itself handles the form data
- Here, the HyperText Markup Language elements maintain their own state updated when the input value changes
- For writing an uncontrolled component, you need to use a ref for getting form values from the DOM
- Alternatively, for every state update, there is no need to write an event handler. A ref can be used for accessing the input field value of the form from the DOM

Forms

(Continued)

```
.vscode > hello-world > src > JS App.js > ...
1  import React, { Component } from 'react';
2  class App extends React.Component {
3      constructor(props) {
4          super(props);
5          this.updateSubmit = this.updateSubmit.bind(this);
6          this.input = React.createRef();
7      }
8      updateSubmit(event) {
9          alert('You have entered the UserName and CompanyName successfully. ');
10         event.preventDefault();
11     }
12     render() {
13         return (
14             <form onSubmit={this.updateSubmit}>
15                 <h1>Uncontrolled Form Example</h1>
16                 <label>Name:
17                 |   <input type="text" ref={this.input} />
18                 </label>
19                 <label>
```

Forms

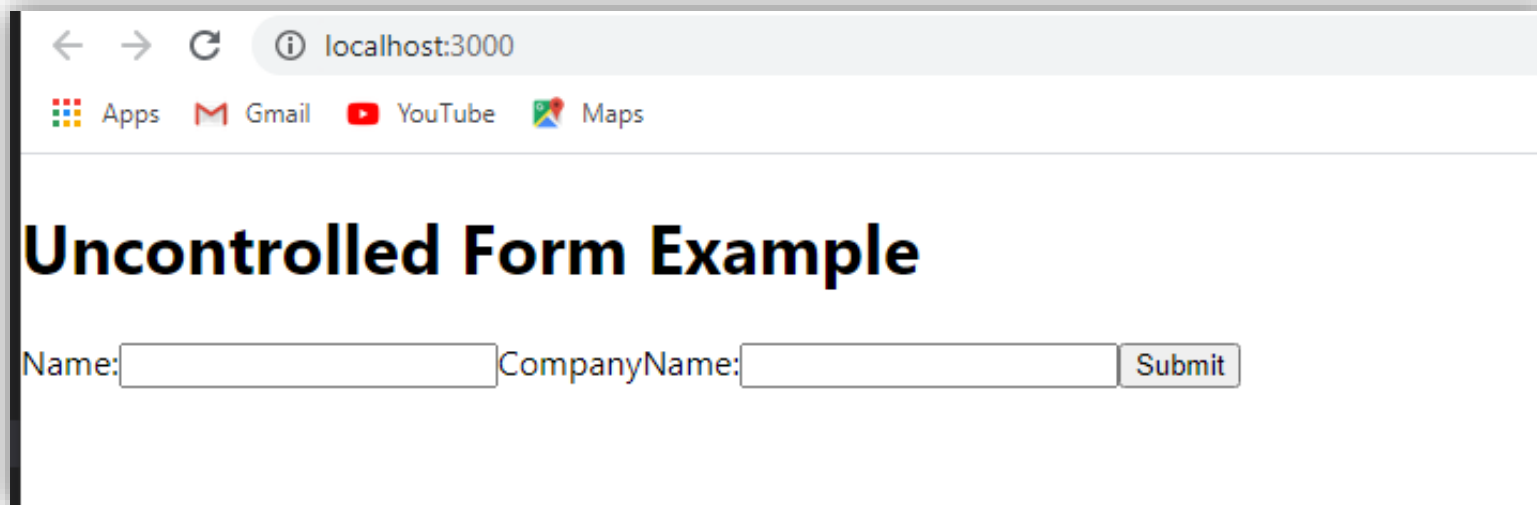
(Continued)

```
20         companyName:  
21         <input type="text" ref={this.input} />  
22     </label>  
23     <input type="submit" value="Submit" />  
24 </form>  
25 );  
26 }  
27 }  
28 export default App;
```

Forms

(Continued)

- **Output:**



A screenshot of a web browser window. The address bar shows 'localhost:3000'. Below the address bar are links for 'Apps', 'Gmail', 'YouTube', and 'Maps'. The main content area has the heading 'Uncontrolled Form Example' in a large, bold, black font. Below the heading is a form with two text input fields. The first field is preceded by the label 'Name:' and the second by 'CompanyName:'. To the right of the second input field is a 'Submit' button. The form is uncontrolled, meaning the inputs are not managed by the component's state.

Forms

2. Controlled Component

- In HTML, form elements typically manage their own state and update it as per the input of the user
- Instead of the DOM, the component handles the input form element in the controlled component
- Here, the mutable state is kept in the state property and will be updated only with **setState()** method
- Controlled components have functions which direct the data passing into them on each **onChange** event, instead of grabbing the data only once, for example, when you click a submit button

Forms

(Continued)

- Then, this data is saved to state as well as updated with **setState()** method. This makes component have better control over the form elements and data
- A controlled component takes its current value via props and notifies the changes via callbacks like an **onChange** event
- A parent component "controls" these changes by managing its own state and handling the callback, and after that, passing the new values as props to the controlled component. It is known as a "dumb component." as well

Forms

(Continued)

```
.vscode > hello-world > src > JS App.js > ...
1  import React, { Component } from 'react';
2  class App extends React.Component {
3      constructor(props) {
4          super(props);
5          this.state = {value: ''};
6          this.handleChange = this.handleChange.bind(this);
7          this.handleSubmit = this.handleSubmit.bind(this);
8      }
9      handleChange(event) {
10         this.setState({value: event.target.value});
11     }
12     handleSubmit(event) {
13         alert('You have submitted the input successfully: ' + this.state.value);
14         event.preventDefault();
15     }
16     render() {
17         return (
18             <form onSubmit={this.handleSubmit}>
19                 <h1>Controlled Form Example</h1>
20                 <label>
```

Forms

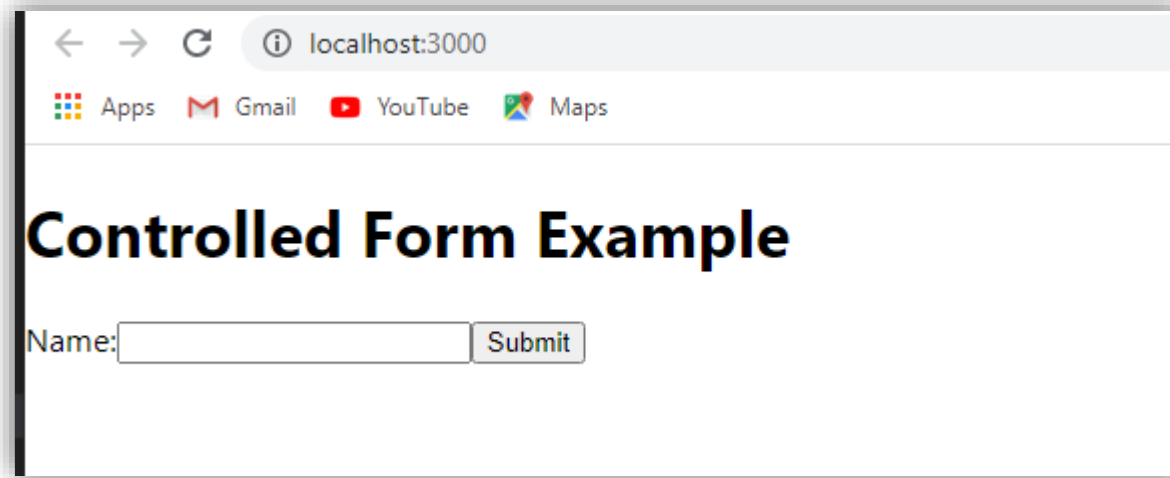
(Continued)

```
21      Name:
22      <input type="text" value={this.state.value} onChange={this.handleChange} />
23    </label>
24    <input type="submit" value="Submit" />
25  </form>
26  );
27  }
28  }
29  export default App;
30
```


Forms

(Continued)

- **Output:**



A screenshot of a web browser window. The address bar shows 'localhost:3000'. Below the address bar are icons for 'Apps', 'Gmail', 'YouTube', and 'Maps'. The main content area displays the title 'Controlled Form Example' in a large, bold, black font. Below the title, there is a form with the label 'Name:' followed by a text input field and a 'Submit' button.

Module 15: Lifting State Up

Lifting State Up

- Usually, between different components, there will be a need to share the state. To move the state to the common parent of the two components is the common approach to share the state between two components. In React.js, this approach is known as **lifting state up**
- Changes in the state reflect relevant components at the same time with the shared state. For shared state components, this is a single source of truth

Lifting State Up

Example

- We have an App component containing **PlayerContent** and **PlayerDetails** component. The player name buttons are displayed by **PlayerContent**. **PlayerDetails** displays the details in one line
- The state for both the component is contained by the app component. The selected player is displayed once we click on one of the player buttons

Lifting State Up

(Continued)

App.js

```
.vscode > hello-world > src > JS App.js > ...
1  import React from 'react';
2  import PlayerContent from './PlayerContent';
3  import PlayerDetails from './PlayerDetails';
4  import './App.css';
5  class App extends React.Component {
6      constructor(props) {
7          super(props);
8          this.state = { selectedPlayer:[0,0], playerName: ''};
9          this.updateSelectedPlayer = this.updateSelectedPlayer.bind(this);
10     }
11     updateSelectedPlayer(id, name) {
12         var arr = [0, 0, 0, 0];
13         arr[id] = 1;
14         this.setState({
15             playerName: name,
16             selectedPlayer: arr
17         });
18     }
19 }
```

Lifting State Up

(Continued)

```
19     render () {
20       return (
21         <div>
22           <PlayerContent active={this.state.selectedPlayer[0]}
23             clickHandler={this.updateSelectedPlayer} id={0} name="David"/>
24           <PlayerContent active={this.state.selectedPlayer[1]}
25             clickHandler={this.updateSelectedPlayer} id={1} name="Steve"/>
26           <PlayerDetails name={this.state.playerName}/>
27         </div>
28       );
29     }
30   }
31   export default App;
32
```

Lifting State Up

(Continued)

PlayerContent.js

```
.vscode > hello-world > src > JS PlayerContent.js > ...
1  import React , { Component } from 'react';
2  class PlayerContent extends Component {
3      render () {
4          return (
5              <button onClick={() => {this.props.clickHandler(this.props.id, this.props.name)}}
6                  style={{color: this.props.active? 'red': 'blue'}}>{this.props.name}
7              </button>
8          );
9      }
10 }
11 export default PlayerContent;
12
```

Lifting State Up

(Continued)

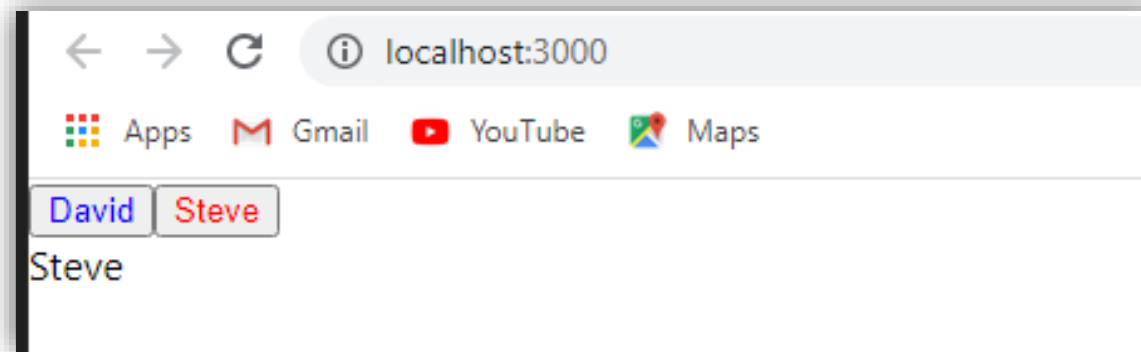
PlayerDetails.js

```
vscode > hello-world > src > JS PlayerDetails.js > ...  
1  import React, { Component } from 'react';  
2  class PlayerDetails extends Component {  
3      render () {  
4          return (  
5              <div>{this.props.name}  
6              </div>  
7          );  
8      }  
9  }  
10 export default PlayerDetails;
```


Lifting State Up

(Continued)

- **Output**



Module 16: Composition vs Inheritance

Composition vs Inheritance

- Composition and inheritance are the ways to use various components together in React.js
This supports code reuse
- React suggest using composition instead of inheritance as much as possible, and inheritance should be used in special cases only
- For instance, we have a component to input username

Composition vs Inheritance

Inheritance

```
1 class UserNameForm extends React.Component {  
2     render() {  
3         return (  
4             <div>  
5                 <input type="text" />  
6             </div>  
7         );  
8     }  
9 }  
10 ReactDOM.render(  
11     < UserNameForm />,  
12     document.getElementById('root'));  
13
```

Composition vs Inheritance

(Continued)

- This is simple used to just input the name
- We will have two more components to create as well as update the username field
- By using inheritance, we will make it like –

```
1 class UserNameForm extends React.Component {  
2   render() {  
3     return (  
4       <div>  
5         <input type="text" />  
6       </div>  
7     )  
9   }  
10 }
```

Composition vs Inheritance

(Continued)

```
7      );  
8    }  
9  }  
10  class CreateUserName extends UserNameForm {  
11    render() {  
12      const parent = super.render();  
13      return (  
14        <div>  
15          {parent}  
16          <button>Create</button>  
17        </div>  
18      )  
19    }  
20  }  
21  class UpdateUserName extends UserNameForm {  
22    render() {  
23      const parent = super.render();  
24      return (  
25        <div>  
26          {parent}
```

Composition vs Inheritance

(Continued)

```
27         <button>Update</button>
28     </div>
29 )
30 }
31 }
32 ReactDOM.render(
33   (<div>
34     < CreateUserName />
35     < UpdateUserName />
36   </div>), document.getElementById('root')
37 );
38
```

- Using **super.render()**, we extended the UserNameForm component and extracted its way in a child component

Composition vs Inheritance

Composition

```
1  class UserNameForm extends React.Component {
2      render() {
3          return (
4              <div>
5                  <input type="text" />
6              </div>
7          );
8      }
9  }
10 class CreateUserName extends React.Component {
11     render() {
12         return (
13             <div>
14                 < UserNameForm />
15                 <button>Create</button>
16             </div>
17         )
18     }
19 }
20 class UpdateUserName extends React.Component {
```


Composition vs Inheritance

(Continued)

```
21   render() {  
22     return (  
23       <div>  
24         < UserNameForm />  
25         <button>Update</button>  
26       </div>  
27     )  
28   }  
29 }  
30 ReactDOM.render(  
31   (<div>  
32     <CreateUserName />  
33     <UpdateUserName />  
34   </div>), document.getElementById('root')  
35 );  
36
```

- Use of composition is easier than inheritance and simple to maintain the complexity

Module 17: Thinking in React.js

Thinking in React.js

- A direction has been provided by the React community for thinking in React way and creating big, fast and scalable applications
- React has reached many platforms and broadly utilised as a popular JavaScript UI interface library

Step 1: Creating a simple mock service

- Suppose we want to make a server call and retrieve data. We can create a mock service for starting with and building a component in order to retrieve and display data
- Here we can include JSON (JavaScript Object Notation) processing in the component and evaluating the anticipated outcome

Thinking in React.js

(Continued)

Step 2: Split the functionality into smaller components

- The first Thing React recommend is for creating a smaller understandable design of boxes
- The association between the different components and the passing of data flow will be represented by these boxes
- The making of components must be on the basis of the principle of single responsibility. That means a single component must manage a single task
- Creating a component hierarchy will make the tasks of the developers more straightforward

Thinking in React.js

(Continued)

Step 3: Start by making a simple static version if possible

- We can create a static version and build the app thereon with smaller components and mock service
- Creating a static version must not utilise state modifications. It must play with the passage of props and rendering Ui only
- In React, keeping one-way data flow makes it modular and straightforward. For making it more precise, the difference and use case of props and state must be understood properly

Thinking in React.js

(Continued)

Step 4: Knowing the minimum representation of the User Interface state

- To make the app interactional, the state must be able to trigger from the relevant component
- Knowing the needed mutable state is essential for building the right app

Step 5: Know where the state must live

- The state can be shared between the many child components. Lifting the state up may be utilised for making communication between the various components

Thinking in React.js

(Continued)

- Using state management libraries such as **Redux** resolves various problems originating from the state
- The one-way **downflow** of data to components is suggested by React

Step 6: If required, add two-way data flow

- In form handling, the controlled component is the two-way data binding example

*the*knowledgeacademy



get it

Congratulations

The World's Largest Global Training Provider

✉ theknowledgeacademy.com

🌐 info@theknowledgeacademy.com



/The.Knowledge.Academy.Ltd



/TKA_Training



/the-knowledge-academy



/TheKnowledgeAcademy