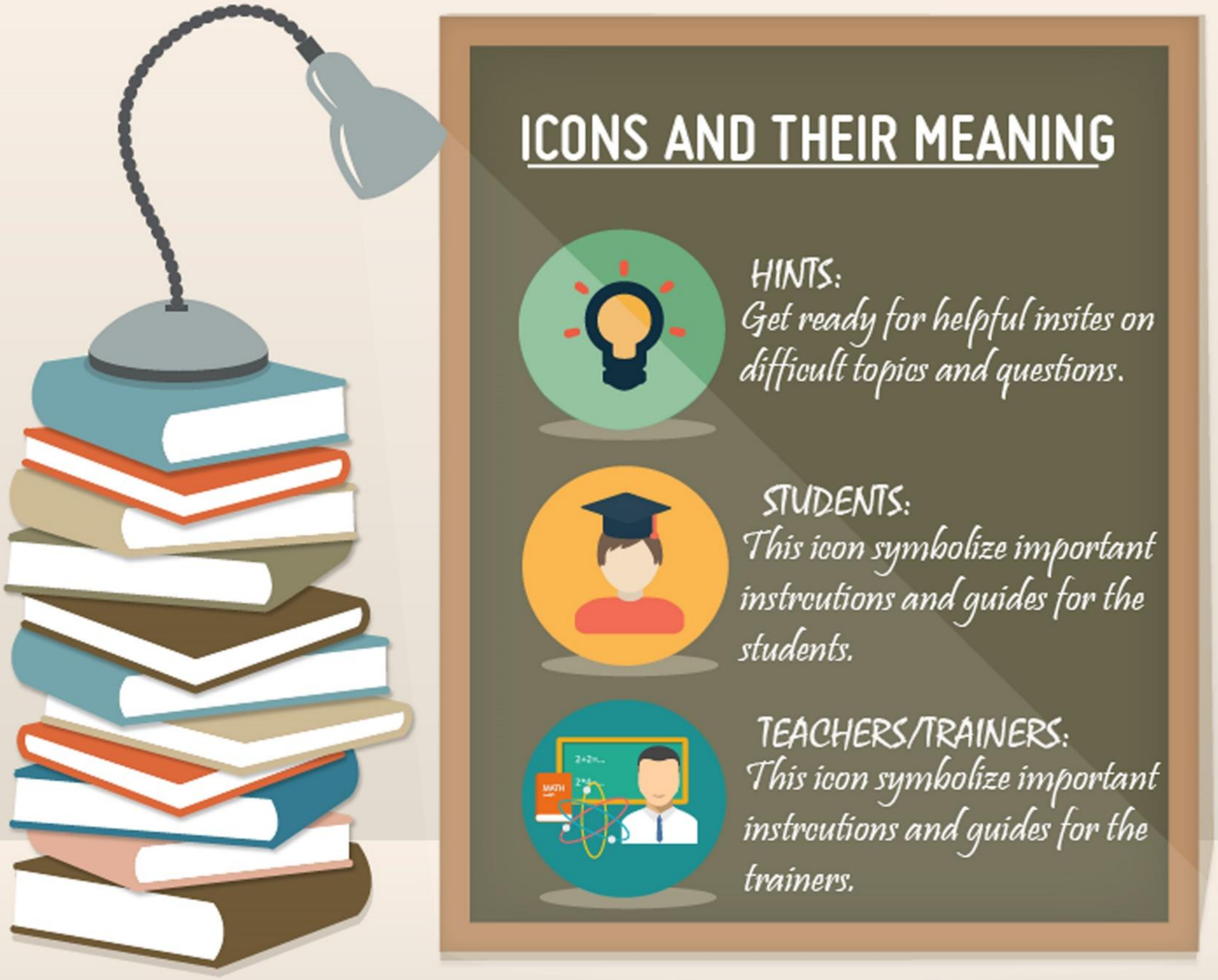
Full Stack Module-VIII

Manual V8.3

ANUDIP FOUNDATION


ICONS AND THEIR MEANING
HINTS:
Get ready for helpful insites on difficult topics and questions.
STUDENTS:
This icon symbolize important instrcutions and guides for the students.
TEACHERS/TRAINERS:
This icon symbolize important instrcutions and guides for the trainers.

Lesson No	Lesson Name	Practical Duration (Minutes)	Theory Duration (Minutes)	Page No
1	React Introduction	120	nil	03
2	React Essential Features and Syntax	120	nil	06
3	React Components, Props and State	60	60	10
4	Styling Components	60	60	13
5	Debugging React Apps	60	60	16
6	React Component	60	60	19
7	HTTP Requests/Ajax Calls	60	60	22
8	React Routing and Form Validation	60	60	27

Total Duration: __ Hours

Lesson 01: React Introduction (120 minutes)

Objective: After completing this lesson you will be able to learn about : • Intro • What and why react • VS code	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 120 minutes	**Practical Duration:** nil
Total Duration: 120 minutes	

Introduction

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components".

What and why react?

React creates a VIRTUAL DOM in memory. React allows developers to create large web applications which can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in application. This corresponds to view in the MVC template. It can be used with a combination of other JavaScript libraries or frameworks, such as Angular JS in MVC. Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM. React only changes what needs to be changed!

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

Basic Advantages of React

Simplicity

ReactJS is just simpler to grasp right away. The component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it. React uses a special syntax called JSX which allows you to mix HTML with JavaScript. This is not a requirement; Developer can still write in plain JavaScript but JSX is much easier to use.

Easy to learn

Anyone with a basic previous knowledge in programming can easily understand React while Angular and Ember are referred to as 'Domain specific Language', implying that it is difficult to learn them. For react you just need basic knowledge of CSS and HTML.

Native Approach

React can be used to create mobile applications (React Native). And React is a diehard fan of reusability, meaning extensive code reusability is supported. So at the same time we can make IOS, Android and Web application.

Data Binding

React uses one-way data binding and an application architecture called Flux controls the flow of data to components through one control point – the dispatcher. It's easier to debug self-contained components of large ReactJS apps.

Performance

React does not offer any concept of a built-in container for dependency. You can use Browserify, Require JS, EcmaScript 6 modules which we can use via Babel, ReactJS-di to inject dependencies automatically.

Testability

ReactJS applications are super easy to test. React views can be treated as functions of the state, so we can manipulate with state we pass to the ReactJS view and take a look at the output and triggered actions, events, functions, etc.

VS Code in React

The VS (Visual Studio) Code editor supports React.js IntelliSense and code navigation out of the box.

Reviewing the chapter

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces
- It lets you compose complex UIs from small and isolated pieces of code called "components"
- React creates a VIRTUAL DOM in memory

Testing your skills

1. ________ are small and isolated pieces of code

a) Array, b) components, c) library, d) None of the above

2. React creates a VIRTUAL ¬¬¬___ in memory

a) DOM, b) ROM, c) RAM, d) Drive

3. ________ allows you to mix HTML with JavaScript

a) JSK, b) JKS, c) JSX, d) JXS

4. The full form of DOM is

a) Disc Operating Model, b) Document Object Model, c) Dynamic Original Model, d) None of the above

5. Which of the following are domain specific Language

a) React, b) CSS, c) HTML, d) only (b) and (c)

Lesson 02: React Essential Features and Syntax (120 minutes)

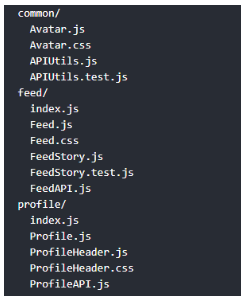
Objective: After completing this lesson you will be able to learn about : • Directory structure • Web pack • Babel • JSX	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 120 minutes	**Practical Duration:** nil
Total Duration: 120 minutes	

Directory Structure

When using React, you will never find any option on how you put files into folders. However, there are a few strategies that can be considered for structuring projects.

Grouping by features or routes

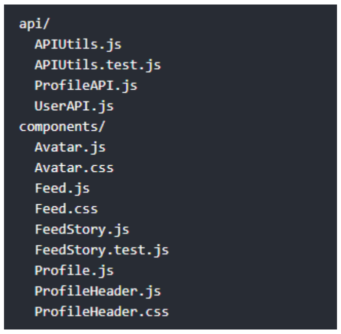
One common way to structure projects is locate CSS, JS, and tests together inside folders grouped by feature or route.



```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```

Grouping by file type

Another popular way to structure projects is to group similar files together. Have a look:



```
api/
  APIUtils.js
  APIUtils.test.js
  ProfileAPI.js
  UserAPI.js
components/
  Avatar.js
  Avatar.css
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
```

You will also witness those who prefer to go a bit further; they separate components into different folders depending on their role in the application.

Better to stay away from too much nesting

This is necessary. There are many pain points associated with deep directory nesting in JavaScript projects. It becomes harder to write relative imports between them, or to update those imports when the files are moved. Unless you have a very compelling reason to use a deep folder structure, consider limiting yourself to a maximum of three or four nested folders within a single project.

Web pack

Web pack is a popular module bundling system built on top of Node.js. It can handle not only combination and minification of JavaScript and CSS files, but also other assets such as image files (spriting) through the use of plugins. It ingests raw React components for producing JavaScript code that (almost) every browser can understand.

Web pack can be used as an alternative to Cassette or ASP.NET Combination and Minification.

In order to use Webpack with ReactJS.NET's server-side rendering, it is suggested that you create a separate bundle ("entry point") containing only the code required to perform server-side rendering. Any components you would like to render server-side must be exposed globally so that ReactJS.NET can access them. This can be done by assigning properties to global. Even though global is not available in the browser, Webpack will add a shim that makes any assigned properties available in the global Javascript scope.

Installing Web pack

Let’s install it by running:

npm i webpack --save-dev

You will also need webpack-cli. Pull it in with:

npm i webpack-cli --save-dev

Next up add the webpackcommand inside package.json:

```
"scripts": {
  "build": "webpack --mode production"
}
```

At this point there is no need to define a configuration file for webpack. Older webpack's version did automatically look for a configuration file. Since version 4 that is no longer the case: you can start developing straight away.

Babel

Babel is a JavaScript compiler. It is a tool chain used primarily for converting ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

Below are listed the following features of Babel:

- Transform syntax
- Poly fill features that are missing in your target environment (through @babel/polyfill)
- Source code transformations (codemods)

JSX

JSX is a syntax extension to JavaScript. It produces React "elements". Although there is no hard and fast rule to use it but there are several benefits associated with the use of JSX:

- It is faster because it performs optimization while compiling code to JavaScript.
- It is also type-safe and most of the errors can be caught during compilation.
- It makes it easier and faster to write templates, if you are familiar with HTML.

Reviewing the chapter

In this chapter, we discussed about directory structures in React, the ways to group files, and also focused on web pack installation, babel, and jsx.

Testing your skills

1. Web pack is a popular_________ bundling system built on top of Node.js

a) Array, b) components, c) library, d) None of the above

2. Web pack can be used as an alternative to

a) Cassette, b) ASP.NET, c) both (a) and (b), d) None of the above

3. Babel is a ______compiler

a) JavaScript, b) ASP.net, c) C++, d) None of the above

4. Which of these are features of Babel

a) poly fill missing features, b) transform syntax, c) source code transformations, d) all of the above

5. Which of the following are features of JSX

a) slow performer, b) difficult to write templates, c) type safe, d) all of the above

Lesson 03: React Components, Props and State (120 minutes)

Objective: After completing this lesson you will be able to learn about : • Props and States • Events • Manipulating the state	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

Props and States

React controls the data flow in the components with state and props. The data in states and props are used to render the Component with dynamic data.

Features and use of Props in React JS

- In ReactJS we use props to send data to components.
- In ReactJS every component is treated as a pure javascript function.
- In ReactJS props are equivalent to parameters of a pure javascript function.
- Props are immutable since these are developed in the concept of pure functions. In pure functions we cannot change the data of parameters. So, also cannot change the data of a prop in ReactJS.

Features and use of States in React JS

- State is like a data store to the ReactJS component. It is mostly used to update the component when user performed some action like clicking button, typing some text, pressing some key, etc.
- React.Component is the base class for all class based ReactJS components. Whenever a class inherits the class React.Component it’s constructor will automatically assigns attribute state to the class with intial value is set to null. we can change it by overriding the method constructor.
- In many cases we need to update the state. To do that we have to use the method setState and directly assigning like this.state = {'key': 'value'} is strictly prohibited.

Events in ReactJS

An event is an action that occurs as a result of the user action or system generated event, such as clicking mouse, pressing a key, loading of webpage, or file being created or modified etc.

An event handler is a callback function which is invoked when specified event occurs. An event handler allows programmers to control the execution of the program when specified event occurs.

In React, event handling is same as handling events on DOM elements; however, there are certain syntactic differences:

- Function is passed as event handler instead of a string.
- Events are named in camel case instead lowercase.
- In React, we must call preventDefault explicitly to prevent default behavior instead of return false.

Manipulating the State

The component state in React can be manipulated through the use of certain functions. The setState() function is the most popular one to do so. Let’s see an example where we add a new method “updateMsg()” to a code:

```
updateMsg() {
this.setState({
msg: "ReactJS"
});
}
```

Practical Session

Instructions: Create the code....using an editor....and save it with HTML extension.

1. Create a program that will feature a feedback form and once the user inputs and submits the form, will display a thank you message. Use the setState function.

```
submit(){
this.setState(function(prevState, props){
return {showForm: !prevState.showForm}
});
}
```

2. Create a program using setState to display a welcome message.

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

- React controls the data flow in the components with state and props. The data in states and props are used to render the Component with dynamic data.
- An event handler is a callback function which is invoked when specified event occurs. An event handler allows programmers to control the execution of the program when specified event occurs.

Testing your skills

1. In ReactJS every component is treated as a pure ______function

a) Array, b) javascript, c) library, d) None of the above

2. ________is the base class for all class based ReactJS components

a) React.Component, b) State.Component, c) React.Base, d) React.Main

3. ________ is a callback function

a) React, b) State, c) Prop, d) Event Handler

4. _______ are developed in the concept of pure functions

a) Props, b) States, c) Events, d) None of the above

5. In React, we must call ______explicitly to prevent default behavior

a) return false, b) return default, c) preventDefault, d) None of the above

Lesson 04: Styling Components (120 minutes)

Objective: After completing this lesson you will be able to learn about : • Inline styles • Limitations • Reactstrap	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 120 minutes	**Practical Duration:** nil
Total Duration: 120 minutes	

Inline styles

In React, inline styles are not specified as a string. Instead they are specified with an object whose key is the camelCased version of the style name, and whose value is the style's value, usually a string.

Have a look at an example:

```
var divStyle = {
  color: 'white',
  backgroundImage: 'url(' + imgUrl + ')',
  WebkitTransition: 'all', // note the capital 'W' here
  msTransition: 'all' // 'ms' is the only lowercase vendor prefix
};

ReactDOM.render(<div style={divStyle}>Hello World!</div>, mountNode);
```

Limitations/disadvantages of inline styling

- Duplication of CSS properties
- CSS properties will be limited to a component scope only, so there is zero reusability
- No pseudo-classes, pseudo-element, media queries, keyframe animations, etc.
- It is hard to maintain or edit/update, and lot of inline CSS can reduce the code readability
- It hampers the performance, on each re-rendering the style object will be recomputed

Reactstrap

Reactstrap is a good library well suited to our needs. If you are working on some React project and want to make your life easier when designing the user interface using Bootstrap, we have good news for you.

Today, it's possible to use the latest version of Bootstrap as React components. We are talking about Reactstrap. This library will help you to work in the Reactjs way while using the awesome, latest and easy to use Bootstrap v4 features.

The Reactstrap implements all of the Bootstrap 4 components as React components, so they can be easily incorporated into your application. It doesn't depend on jQuery, so your code will be kept as clean as possible. However, it does relies on Tether (a dependency of Bootstrap) to produce efficient and nice tooltips.

Practical Session

Instructions: Create the code....using an editor....and save it with HTML extension.

1. Create a program that will insert an object with the styling information

```
import React from 'react';

import ReactDOM from 'react-dom';

class MyHeader extends React.Component {

render() {

return (

<div>

<h1 style={{color: "red"}}>Hello Style!</h1>

<p>Add a little style!</p>

</div>

);

}

}

ReactDOM.render(<MyHeader />, document.getElementById('root'));
```

2. Create another program that will insert an object with the student information

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

- In React, inline styles are not specified as a string. Instead they are specified with an object whose key is the camelCased version of the style name.
- In React, inline styles are not specified as a string but as objects.

Testing your skills

1. In React, inline styles are specified as

a) Array, b) string, c) object, d) None of the above

2. Which of the following cannot be used in inline styling

a) pseudo class, b) pseudo element, c) media query, d) All of the above

3. ________percentage of reusability for CSS components when using inline styling

a) Zero, b) Two, c) Five, d) One Hundred

4. The Reactstrap doesn’t depend on

a) C#, b) JQuery, c) ASP.Net, d) None of the above

5. Reactstrap depend on________ to produce efficient and nice tooltips

a) JQuery, b) CSS, c) Tether, d) None of the above

Lesson 05: Debugging React Apps (120 minutes)

Objective: After completing this lesson you will be able to learn about : • React error messages • Error boundaries	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

React error messages and error boundaries

To handle error messages, React has introduced a new concept, React Error Boundaries.

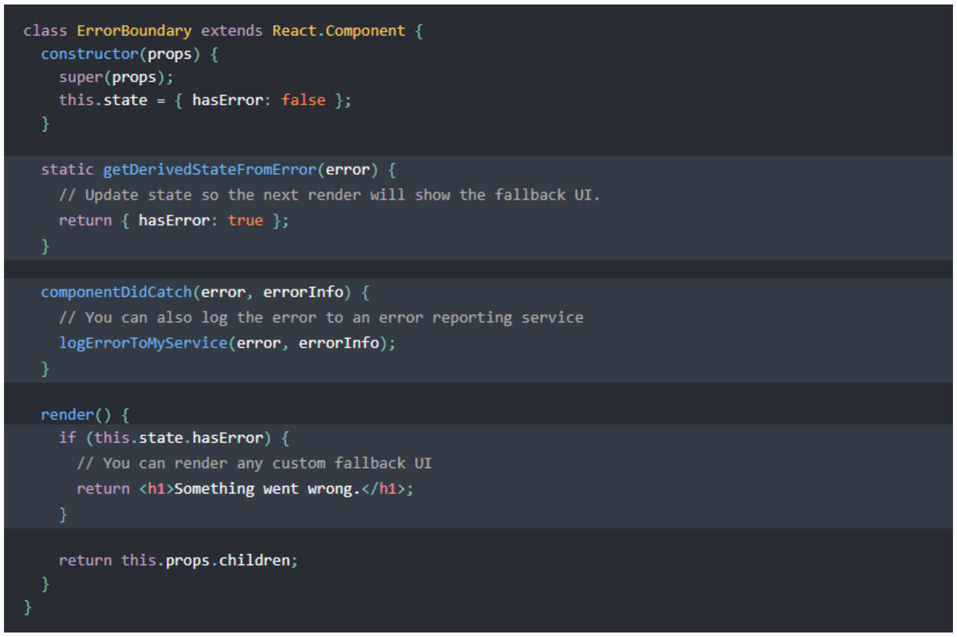
Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

However, you need to keep in mind that error boundaries do not catch errors for:

- Event handlers
- Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)
- Server side rendering
- Errors thrown in the error boundary itself

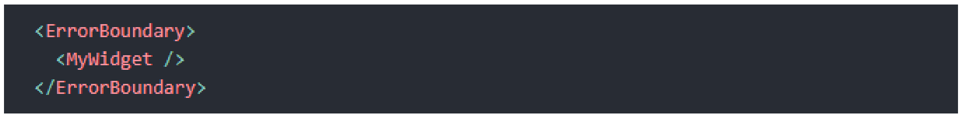
A class component becomes an error boundary if it defines either (or both) of the lifecycle methods static getDerivedStateFromError() or componentDidCatch().

Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown. Use componentDidCatch() to log error information. Focus on the below example:



```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

After that, you can use it as a regular component. Have a look below:



```
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

Error boundaries work like a JavaScript catch {} block, but only for components! Only class components can be error boundaries. In practice, most of the time you'll want to declare an error boundary component once and use it throughout your application. It also needs to be noted that the error boundaries only catch errors in the components below them in the tree. An error boundary can't catch an error within itself. If an error boundary fails trying to render the error message, the error will propagate to the closest error boundary above it. This, too, is similar to how catch {} block works in JavaScript.

Practical Session

Instructions: Create the code....using an editor....and save it with HTML extension.

1. Create a program that will store your error messages in the Redux store and display them.

render () {

return (

<div>

```
{this.props.errors && <div>Error Message</div>}

<h1>Your Amazing Content</h1>

</div>

);

}
```

2. Create another program that will store your error messages in the Redux store and display them.

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

In this chapter, we discussed about different error boundaries and ways to manage errors in React.

Testing your skills

1. Which of the following are Asynchronous code

a) setTimeout, b) requestAnimationFrame, c) callbacks, d) all of the above

2. Error boundaries are React components that catch _________ errors

a) JavaScript, b) Java, c) ASP.Net, d) C++

3. Error boundaries do not catch errors for

a) Event handlers, b) Server side rendering, c) Asynchronous code, d) All of the above

4. Use _________to log error information

a) getDerivedStateFromError(), b) componentDidCatch(), c) both (a) and (b), d) None of the above

5. Only ________components can be error boundaries

a) Class, b) Array, c) HTML, d) None of the above

Lesson 06: React Component (120 minutes)

Objective: After completing this lesson you will be able to learn about : • Pure components • Higher order components	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

Pure components

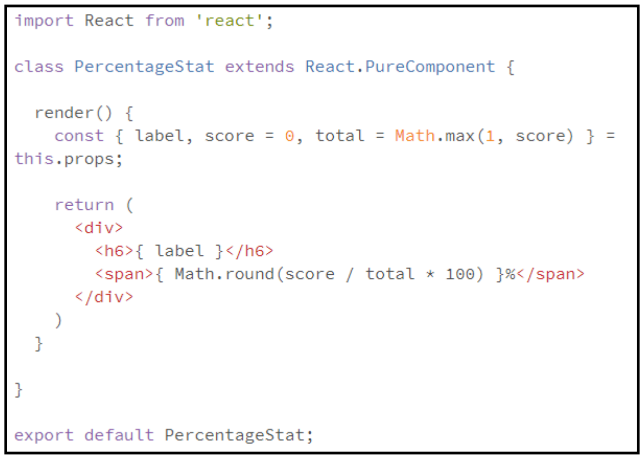
A React component can be considered pure if it renders the same output for the same state and props. For class components like this, React provides the PureComponent base class. Class components that extend the React.PureComponent class are treated as pure components.

Pure components have some performance improvements and render optimizations since React implements the shouldComponentUpdate() method for them with a shallow comparison for props and state.

Based on the concept of purity in functional programming paradigms, a function is said to be pure if:

- its return value is only determined by its input values.
- its return value is always the same for the same input values.

A sample example



```
import React from 'react';

class PercentageStat extends React.PureComponent {

  render() {
    const { label, score = 0, total = Math.max(1, score) } =
this.props;

    return (
      <div>
        <h6>{ label }</h6>
        <span>{ Math.round(score / total * 100) }%</span>
      </div>
    )
  }

}

export default PercentageStat;
```

Higher order components

A higher-order component is a function that takes a component and returns a new component. A higher-order component (HOC) is the advanced technique in React.js for reusing component logic. Higher-Order Components are not part of the React API. They are the pattern that emerges from React's compositional nature. The component transforms props into UI, and a higher-order component converts a component into another component. The examples of HOCs are Redux's connect and Relay's createContainer.

A sample example

Creating a react.js project

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

cd my-app
npm start
```

Creating a new file inside src folder called HOC.js

```
// HOC.js

import React, {Component} from 'react';

export default function Hoc(HocComponent){
    return class extends Component{
        render(){
            return (
                <div>
                    <HocComponent></HocComponent>
                </div>

            );
        }
    }
}
```

Now, include this function into the App.js file.

```
// App.js

import React, { Component } from 'react';
import Hoc from './HOC';

class App extends Component {

  render() {
    return (
      <div>
        Higher-Order Component Tutorial
      </div>
    )
  }
}
App = Hoc(App);
export default App;
```

Practical Session

Instructions: Create the code....using an editor....and save it with HTML extension.

1. Create a Class component called Car

```
class Car extends React.Component {
render() {
return <h2>Hi, I am a Car!</h2>;
}
}
```

2. Create a Function component called Car

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

In this chapter, we learned about the pure components and high order components in React while also focusing on coding samples.

Testing your skills

1. What are the features of a pure function

a) return value determined by input values, b) return value is same as the input value, c) both (a) and (b), d) None of the above

2. The full form of HOC is

a) Higher Order Class, b) Higher Order Component, c) Heavily Ordained Components, d) None

3. Which of the following are HOCs

a) Redux's connect, b) Relay's createContainer, c) both (a) and (b), d) None of the above

4. _______ is used for reusing component logic

a) HOC, b) Pure Component, c) Prop, d) None of the above

Lesson 07: HTTP Requests/Ajax Calls (120 minutes)

Objective: After completing this lesson you will be able to learn about : • HTTP requests • Interceptors • Making AJAX calls	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

HTTP Requests

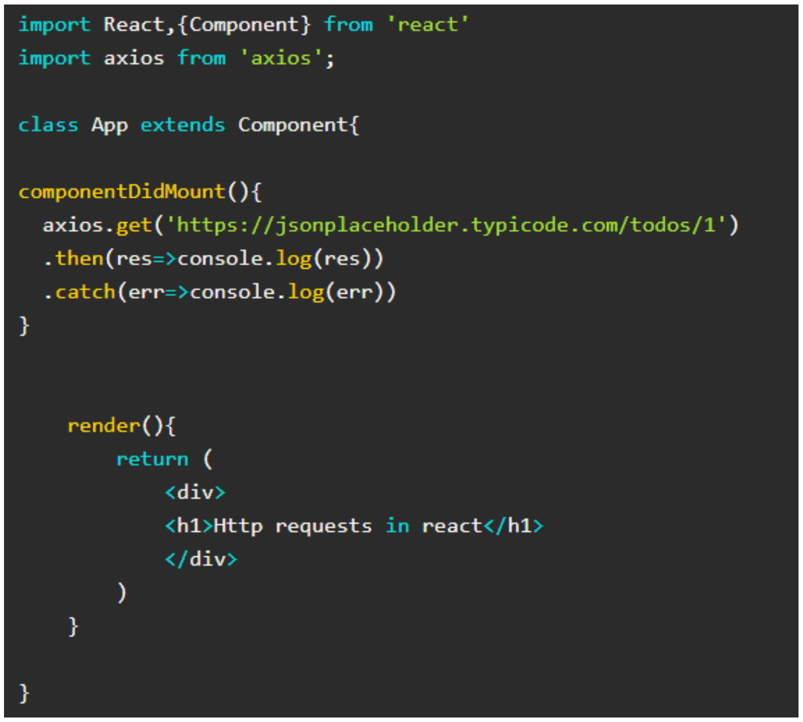
We are using axios http library to fetch the data from the backend.

Axios: It's a Promise based HTTP client for the browser and node.js.

First of all, we need to install the axios library by using the npm package manager.

npm i axios

Once you successfully installed the Axios, let's make some API calls to the server.



```
import React,{Component} from 'react'
import axios from 'axios';

class App extends Component{

componentDidMount(){
  axios.get('https://jsonplaceholder.typicode.com/todos/1')
  .then(res=>console.log(res))
  .catch(err=>console.log(err))
}


    render(){
        return (
            <div>
            <h1>Http requests in react</h1>
            </div>
        )
    }

}
```

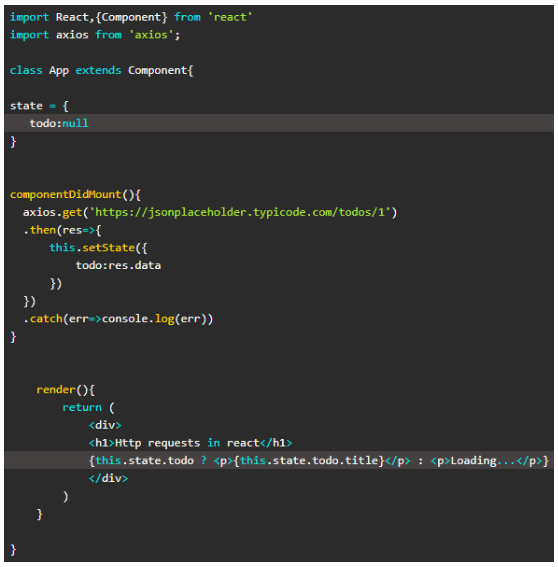
In the above code, we make the http request inside the componentDidMount() life cycle method and log the response in the console.

Why do we need componentDidMount()?

In the react componentDidMount lifecycle method is invoked immediately after a component is connected to the browser's dom. So that it is the correct place to make the http requests.

Let's store the data inside the state instead of logging in the browser.

Full example code



```
import React,{Component} from 'react'
import axios from 'axios';

class App extends Component{

state = {
   todo:null
}


componentDidMount(){
  axios.get('https://jsonplaceholder.typicode.com/todos/1')
  .then(res=>{
      this.setState({
          todo:res.data
      })
  })
  .catch(err=>console.log(err))
}


    render(){
        return (
            <div>
            <h1>Http requests in react</h1>
            {this.state.todo ? <p>{this.state.todo.title}</p> : <p>Loading...</p>}
            </div>
        )
    }

}
```

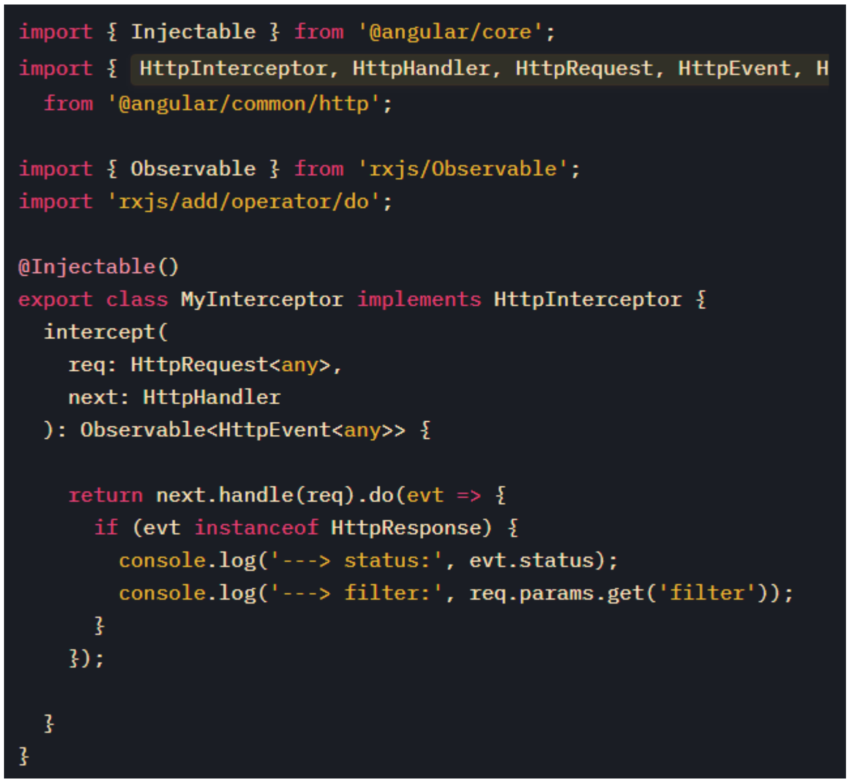
Interceptors

Interceptors provide a mechanism to intercept and/or mutate outgoing requests or incoming responses. They are very similar to the concept of middleware with a framework like Express, except for the frontend. Interceptors can be really useful for features like caching and logging.

To implement an interceptor, you'll want to create a class that's injectable and that implements HttpInterceptor. The class should define an intercept method to correctly implement HttpInterceptor. The intercept method takes two arguments, req and next, and returns an observable of type HttpEvent.

- req is the request object itself and is of type HttpRequest.
- next is the http handler, of type HttpHandler. The handler has a handle method that returns our desired HttpEvent observable.

Below is a basic interceptor implementation. This particular interceptor simply uses the RxJS do operator to log the value of the request's filter query param and the http event status to the console. The do operator is useful for side effects such as this:



```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpHandler, HttpRequest, HttpEvent, H
  from '@angular/common/http';

import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/do';

@Injectable()
export class MyInterceptor implements HttpInterceptor {
  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {

    return next.handle(req).do(evt => {
      if (evt instanceof HttpResponse) {
        console.log('---> status:', evt.status);
        console.log('---> filter:', req.params.get('filter'));
      }
    });

  }
}
```

Making an AJAX call

You can use any AJAX library you like with React. Some popular ones are Axios, jQuery AJAX, and the browser built-in window.fetch.

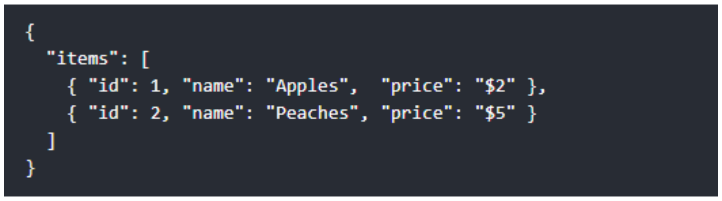
Where in the component lifecycle should I make an AJAX call?

You should populate data with AJAX calls in the componentDidMount lifecycle method. This is so you can use setState to update your component when the data is retrieved.

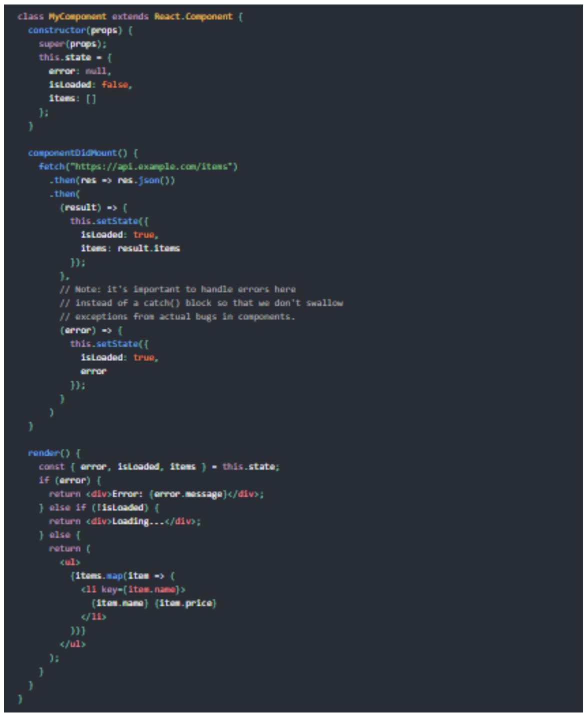
Example: Using AJAX results to set local state

The component below demonstrates how to make an AJAX call in componentDidMount to populate local component state.

The example API returns a JSON object like this:



```
{
  "items": [
    { "id": 1, "name": "Apples",  "price": "$2" },
    { "id": 2, "name": "Peaches", "price": "$5" }
  ]
}
```



```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      items: []
    };
  }

  componentDidMount() {
    fetch("https://api.example.com/items")
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            items: result.items
          });
        },
        // Note: it's important to handle errors here
        // instead of a catch() block so that we don't swallow
        // exceptions from actual bugs in components.
        (error) => {
          this.setState({
            isLoaded: true,
            error
          });
        }
      )
  }

  render() {
    const { error, isLoaded, items } = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
        <ul>
          {items.map(item => (
            <li key={item.name}>
              {item.name} {item.price}
            </li>
          ))}
        </ul>
      );
    }
  }
}
```

Practical Session

Instructions: Create the code….using an editor….and save it with HTML extension.

1. Create a simple POST request by AJAX

```
<!DOCTYPE html>

<html>

<body>

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};
xhttp.open("POST", "demo_post.asp", true);
xhttp.send();
}
</script>
</body>
</html>
```

2. Create another POST request by AJAX

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

In this chapter, we learned about HTTP Request, basics of interceptors, and making AJAX calls.

Testing your skills

1. _________ is a Promise based HTTP client for the browser and node.js

a) Axios, b) Bios, c) AJAX, d) None of the above

2. _____ method is invoked immediately after a component is connected to the browser's DOM

a) componentMount, b) componentDidMount, c) componentConnect, d) Component

3. _________ provide a mechanism to intercept and/or mutate outgoing requests or incoming responses

a) Interceptors, b) Module, c) HTTP, d) None of the above

4. ________ is the http handler

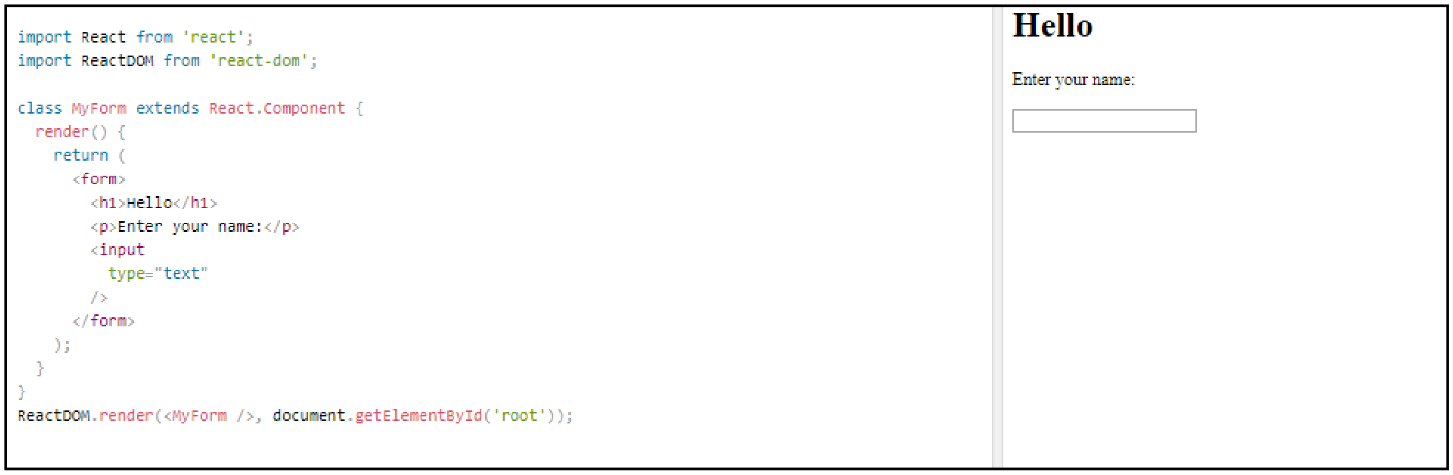
a) HTTP, b) Req, c) Next, d) None of the above

Lesson 08: React Routing and Form Validation (120 minutes)

Objective: After completing this lesson you will be able to learn about : • Handling Form Submission • Custom Form Validation	**Materials Required:** • Computer With Windows XP and above • Stable Internet connection
Self- Learning Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

Similar to that of HTML, React also uses forms to allow users to interact with the web page.

Below is a sample example that shows the coding to add forms in React (*observe the output at the right*):



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

class MyForm extends React.Component {
  render() {
    return (
      <form>
        <h1>Hello</h1>
        <p>Enter your name:</p>
        <input
          type="text"
        />
      </form>
    );
  }
}
ReactDOM.render(<MyForm />, document.getElementById('root'));
```

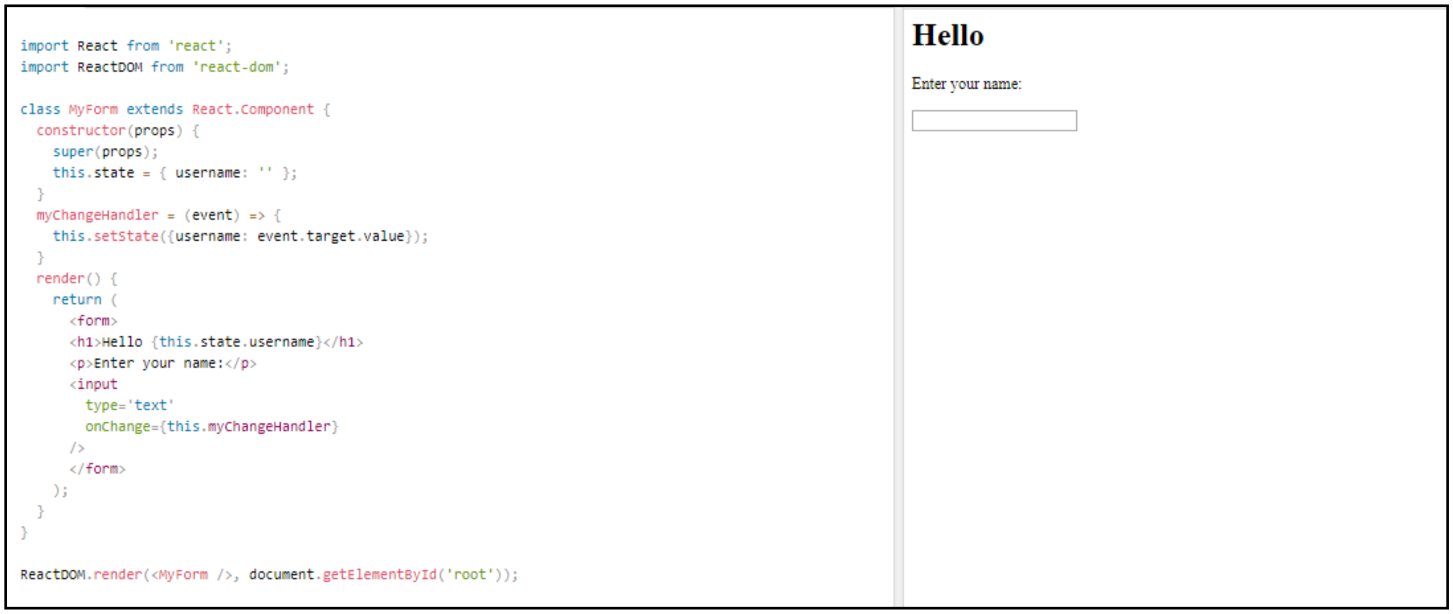
Handling forms

Handling forms in React can be described as handling the data when it gets submitted or changes the value. In HTML, form data is usually handled by the DOM. In React, form data is usually handled by the components.

When the data is handled by the components, all the data is stored in the component state.

You can control changes by adding event handlers in the onChange attribute.

Below is an example that adds an event handler to the onChange attribute while updating the state:

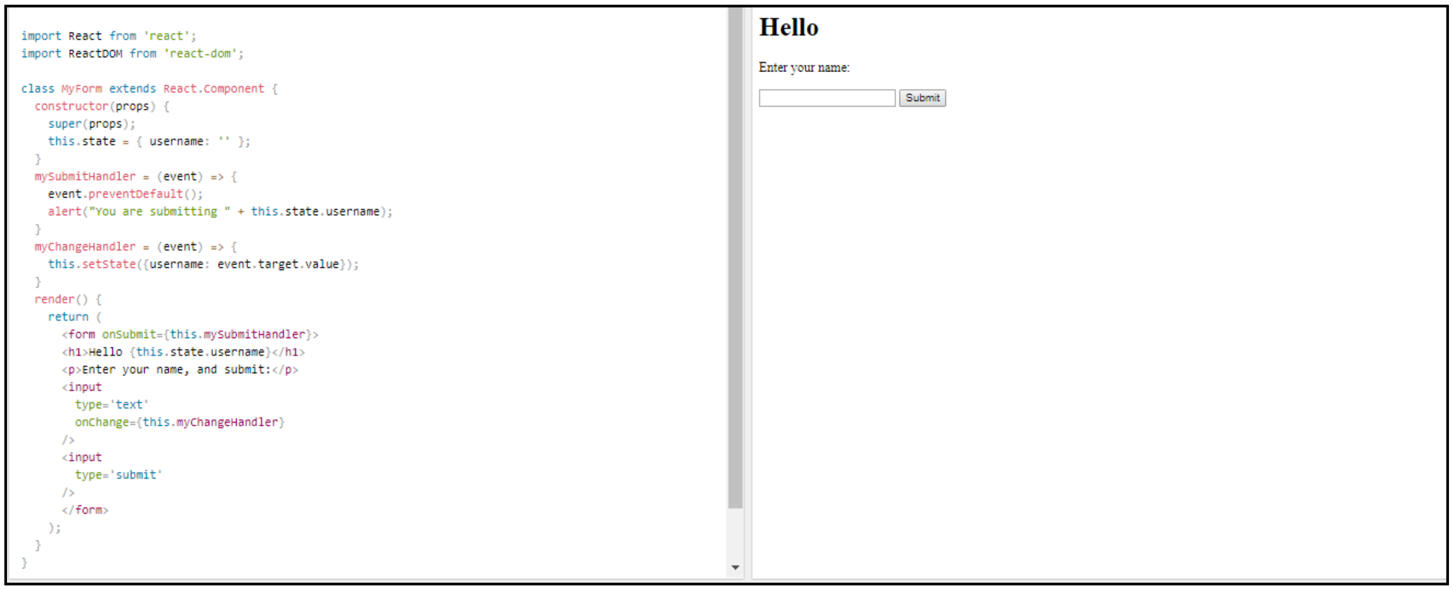


```
import React from 'react';
import ReactDOM from 'react-dom';

class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  myChangeHandler = (event) => {
    this.setState({username: event.target.value});
  }
  render() {
    return (
      <form>
      <h1>Hello {this.state.username}</h1>
      <p>Enter your name:</p>
      <input
        type='text'
        onChange={this.myChangeHandler}
      />
      </form>
    );
  }
}

ReactDOM.render(<MyForm />, document.getElementById('root'));
```

```
Hello

Enter your name:
```

Submitting Forms

You can control the submit action by adding an event handler in the onSubmit attribute. Have a look at the below example with output at the right:



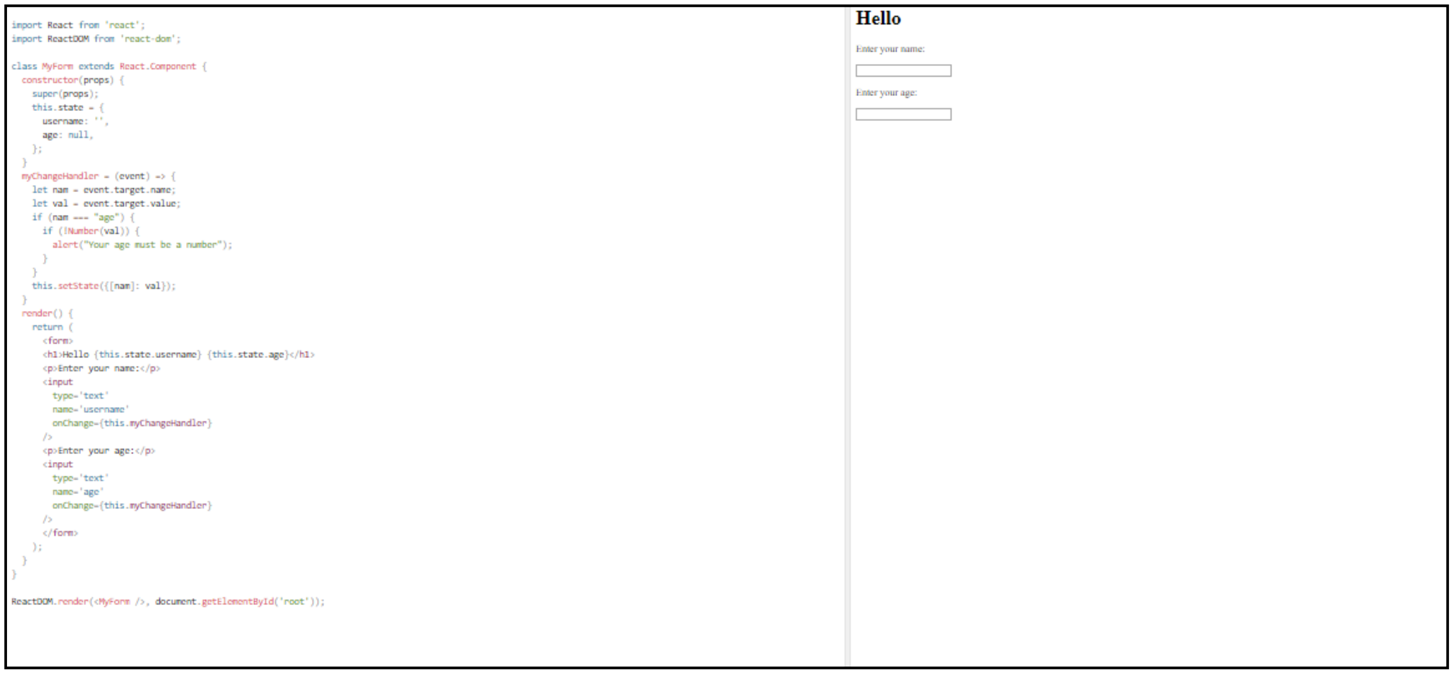
```
import React from 'react';
import ReactDOM from 'react-dom';

class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  mySubmitHandler = (event) => {
    event.preventDefault();
    alert("You are submitting " + this.state.username);
  }
  myChangeHandler = (event) => {
    this.setState({username: event.target.value});
  }
  render() {
    return (
      <form onSubmit={this.mySubmitHandler}>
      <h1>Hello {this.state.username}</h1>
      <p>Enter your name, and submit:</p>
      <input
        type='text'
        onChange={this.myChangeHandler}
      />
      <input
        type='submit'
      />
      </form>
    );
  }
}
```

```
Hello

Enter your name:
                    Submit
```

Custom form validation

You can validate form input when the user is typing or you can wait until the form gets submitted.

Let’s focus on an example (*When you fill in your age, you will get an alert if the age field is not numeric*):



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

class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      age: null,
    };
  }
  myChangeHandler = (event) => {
    let nam = event.target.name;
    let val = event.target.value;
    if (nam === "age") {
      if (!Number(val)) {
        alert("Your age must be a number");
      }
    }
    this.setState({[nam]: val});
  }
  render() {
    return (
      <form>
      <h1>Hello {this.state.username} {this.state.age}</h1>
      <p>Enter your name:</p>
      <input
        type='text'
        name='username'
        onChange={this.myChangeHandler}
      />
      <p>Enter your age:</p>
      <input
        type='text'
        name='age'
        onChange={this.myChangeHandler}
      />
      </form>
    );
  }
}

ReactDOM.render(<MyForm />, document.getElementById('root'));
```

Practical Session

Instructions: Create the code....using an editor....and save it with HTML extension.

1. Create a program to add forms in React (take reference from sample examples shown in the chapter)

2. Create a program that allows for custom form validation.

Note: After saving the code, click the created HTML file to view the output.

Reviewing the chapter

In this chapter, we learned about handling forms in React along with the procedure to submit a form and validating the same.

Testing your skills

1. _________ uses forms to allow users to interact with the web page

a) React, b) HTML, c) both (a) and (b), d) None of the above

2. In ______ form data is usually handled by the DOM

a) HTML, b) AJAX, c) React, d) None of the above

3. In ______ form data is usually handled by the components

a) HTML, b) AJAX, c) React, d) None of the above

4. You can control changes by adding event handlers in the _____________attribute

a) onEvent, b) onChange, c) onControl, d) changeEvent

5. You can control the submit action by adding an event handler in the ___________attribute

a) onAdd, b) eventSubmission, c) submitControl, d) onSubmit