CitiusTech

React JS

Version 1.0

May 2019

Agenda

- **What is ReactJS?**
- The Virtual DOM
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

What is React JS? – Plethora of definitions

- A declarative, efficient & flexible JavaScript library for building user interfaces
- An open-source JavaScript library used for building user interfaces specifically for Single Page Applications & for handling the view layer for web & mobile apps
- An engine for building composable user interfaces using JavaScript & (optionally) XML
- A UI library developed at Facebook to facilitate the creation of interactive, stateful & reusable UI components
  - Real-world entity
  - Relation-based tables
  - Isolation of data and application
  - Less redundancy
  - Consistency
  - Query Language
  - ACID Properties
  - Multiuser and Concurrent Access
  - Multiple views
  - Security
- React wraps an imperative API with a declarative one

Why React JS? – The million dollar question (1/2)

Reactive Rendering is Simple

- SPAs constantly fetch new data & transform parts of the DOM as the user interacts
- As interfaces grow more complex, it becomes difficult to examine the current state of the application & make necessary changes on the DOM to update it
- Many JavaScript frameworks use data-binding to tackle this increasing complexity & keep the interface in sync with state. This approach comes with disadvantages in maintainability, scalability & performance
- Reactive rendering is easier to use than traditional data binding. It lets us write in a declarative way how components should look & behave. When the data changes, React conceptually renders the whole interface again
- React uses an in-memory, lightweight representation of the DOM called virtual DOM for re-rendering the entire UI and hence is much faster

Why React JS? – The million dollar question (2/2)

Component-oriented Development using Only Js

- React components are written in plain JavaScript, instead of template languages or the HTML directives traditionally used for web application UIs
- Templates now can be limiting because they dictate the full set of abstractions that are allowed to use to build the UI
- React's use of a full-featured programming language to render views is a big advantage to the ability to build abstractions
- By being self-contained & using a markup with its corresponding view logic, React components lead to separation of concerns
- React assumes that display logic & markup are highly cohesive; they both show the UI & encourage the separation of concerns by creating discrete, well-encapsulated & reusable components for each concern

Agenda

- What is ReactJS?
- **The Virtual DOM**
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

The Virtual DOM (1/2)

- The Virtual DOM is a React Object unlike the DOM, which is a browser object. It's an abstraction of the HTML DOM or browser DOM
- Extremely fast as compared to the browser DOM
- Can produce 200, 000 virtual DOM nodes per second
- Created completely from scratch on every state change
- Because manipulating the in-memory representation of the DOM is faster & more efficient than manipulating the real DOM, when the state of the application changes (as the result of an user interaction or data fetching), React quickly compares the current state of the UI with the desired state & computes the minimal set of real DOM mutations to achieve it

The Virtual DOM (2/2)

- ReactJS does not update the Real DOM directly, but it updates the Virtual DOM
- At any given time, React maintains two virtual DOMs, one with the updated state & the other with the previous state
- React uses a diff algorithm to compare both the Virtual DOMs to find the minimum number of steps to update the Real DOM
- This is how it works
  - Whenever the state of the data model changes, React re-renders the UI to a virtual DOM representation
  - React then calculates the difference between the two virtual DOM representations: the previous virtual DOM representation that was computed before the data was changed & the current virtual DOM representation that was computed after the data was changed. This difference is what actually needs to be changed in the real DOM
  - React then updates only what needs to be updated in the real DOM
    - This process is fast
    - As a React developer, you don't need to worry about what actually needs to be re-rendered. React allows you to write your code as if you were re-rendering the entire DOM every time your application's state changes

Agenda

- What is ReactJS?
- The Virtual DOM
- **Introduction to JSX**
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

Introducing JSX (JavaScript XML)

- A syntax extension for JavaScript, written to be used with React
- JSX is not valid JavaScript & web browsers cannot read it
- If a JavaScript file contains JSX code, then that file needs to be compiled, which means that before the file reaches a web browser, a JSX compiler will translate any JSX into regular JavaScript
- JSX produces React elements
- Instead of artificially separating technologies by putting markup & logic in separate files, React separates concerns with loosely coupled units called components that contain both. This is achieved using JSX

Introducing Babel

- A transpiler for JavaScript best known for its ability to turn ES6 (the next version of JavaScript) into code that runs in a browser
- Created by an Australian developer named Sebastian McKenzie
- Can handle all of the new syntax that ES6 brings, along with built-in support for React's JSX extensions
- Has the greatest level of compatibility with the ES6 specifications
- Lets you use virtually all new features of ES6, without sacrificing backward compatibility for older browsers

https://babeljs.io/docs/plugins/transform-react-jsx/

Understanding JSX (1/7)

- A basic unit of JSX is called a JSX element
- JSX elements must be a part of a .js file
- JSX elements are treated as JavaScript expressions
  - They can go anywhere that JavaScript expressions can go
  - JSX element can be saved in a variable, passed to a function, stored in an object or array, etc.

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Understanding JSX (2/7)

- JSX elements can have attributes, just like HTML elements
  - JSX attribute is written using HTML-like syntax: a name, followed by an equals sign, followed by a value. The value should be wrapped in quotes

```
<div mycustomattribute="something" />


        const catImage = ( <img src="#" alt="a cat"
        height: {styles.height} width: {styles.width} /> )
```

Understanding JSX (3/7)

- JSX elements can be nested inside other JSX elements

```
<a href="http://www.example.com"><h1>Click Me</h1></a>
```

- To make the JSX more readable, line breaks & indentation can be used

```
<a href="http://www.example.com">
    <h1>
        Click Me
    </h1>
</a>
```

- If a JSX expression takes up more than one line, then it must be wrapped in parentheses

```
(
            <a href="http://www.example.com">
                <h1>
                    Click Me
                </h1>
            </a>
)
```

Understanding JSX (4/7)

- A JSX expression must have exactly one outermost element
- The first opening tag & the final closing tag of a JSX expression must belong to the same JSX element
- If a JSX expression has multiple outer elements, wrap the JSX expression in a

```
const paras = (
    <div>
        <p>Para1</p>
        <p>Para2</p>
    </div>
)
```

Understanding JSX (5/7)

- To render a JSX expression means to make it appear onscreen
- ReactDOM.render() method is the most common way to render JSX
  - It takes a JSX expression, creates a corresponding tree of DOM nodes & adds that tree to the DOM
- ReactDOM.render() is a part of React JS JavaScript library
- ReactDOM.render()'s first argument should be a JSX expression
  - The first argument is appended to whatever element is selected by the second argument

```
ReactDOM.render(<h1>Hello World</h1>,
    document.getElementById('app'));
```

Understanding JSX (6/7)

- ReactDOM.render()'s first argument should evaluate to a JSX expression, it doesn't have to literally be a JSX expression
- The first argument could also be a variable, so long as that variable evaluates to a JSX expression

```
const toDoList = (
    <ol>
        <li>Learn React</li>
        <li>Become a developer</li>
    </ol>
)

ReactDOM.render(toDoList,
    document.getElementById('app'));
```

Understanding JSX (7/7)

- ReactDOM.render() only updates DOM elements that have changed
- If you render the exact same thing twice in a row, the second render will do nothing

```
const hello = "<h1>Hello World</h1>";

//This will add "Hello World to the screen"
ReactDOM.render(hello,document.getElementById('app'));

//This will not render anything
ReactDOM.render(hello,document.getElementById('app'));
```

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX
- **Guidelines for Planning a React Application**
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

Guidelines for planning a React application

- There are two simple guidelines which need to be followed when planning a React application
  - Each React component should represent a single user interface element in the web application. It should encapsulate the smallest element possible that can potentially be reused
  - Multiple React components should be composed into a single React component. Ultimately, the entire user interface should be encapsulated in one React component

Creating a React JS Application from scratch (1/4)

- **Pre-Requisites**
  - Node JS
  - React JS Library
  - Babel compiler for browser
- Install React JS Library

```
D:\reactjs>npm install -g react react-dom
```

- Install babel transpiler for browser compilation

```
D:\reactjs>npm install -g @babel/standalone
```

- Create a folder which holds the React application

Creating a React JS Application from scratch (2/4)

- Create a folder named scripts inside the application folder
- Copy the following files inside the scripts folder
  - react.development.js from

    C:\Users***username***\AppData\Roaming\npm\node_modules\react\umd

  - react-dom.development.js from

    C:\Users***username***\AppData\Roaming\npm\node_modules\react-dom\umd

  - babel.js from

    C:\Users***username***\AppData\Roaming\npm\node_modules\@babel\standalone

Creating a React JS Application from scratch (3/4)

- Creating a React Component & render it
  - A React component is defined as a plain JavaScript class inheriting from an abstract class named React.Component
  - The HTML markup to be returned (JSX) must be defined inside a render() method

```
class HelloWorld extends Component
{
    render()
    {
        return (
            <div>
                <h2>Hello from React JS</h2>
                <h2>HelloWorldComponent.js</h2>
            </div>
        );
    }
}

ReactDOM.render(<HelloWorldComponent/>,document.getElementById('app'));
```

Creating a React JS Application from scratch (4/4)

- The main HTML page
  - This will include the React JS libraries & the babel compiler library for the browser
  - It will have a placeholder (mounting point) where the component will be rendered
  - It will also include the component's JS file

```
<html>
    <head>
        <script src="./scripts/react.development.js"></script>
        <script src="./scripts/react-dom.development.js"></script>
        <script src="./scripts/babel.min.js"></script>
    </head>
    <body>
        <div id="app">
            <script type="text/babel" src="./HelloWorldComponent.js">
            </script>
        </div>
    </body>
</html>
```

Creating a React JS Application using CRA (1/3)

- Create React App (CRA) is a tool built by developers at Facebook to help build React applications
  - Provides a basic scaffolding for a react application
- Saves one from time-consuming setup & configuration
- Simply run one command & CRA sets up the tools needed to start a React project
- What comes inside the CRA box?
  - ES6 modules including support for React & JSX
  - Hot module reloading — edit something in components & the app refreshes without even a reload
  - Linting — syntax error highlighting
  - Readable errors — errors that make much more sense as a developer
  - Ability to import CSS & images directly from JavaScript
  - A production build script that sets up the app ready for deployment

Creating a React JS Application using CRA (2/3)

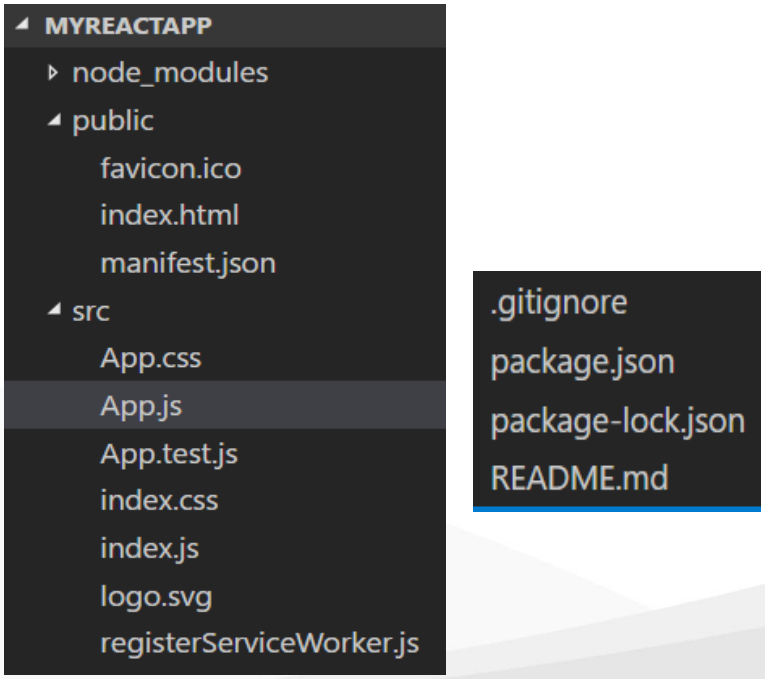
- Install the tool using npm

```
D:\>npm install -g create-react-app
```

- Create a React app

```
D:\>create-react-app myreactapp
```

- The application structure appears as shown below



Creating a React JS Application using CRA (3/3)

- Public/index.html is the main HTML file that includes React code & application & provides a context for React to render.
  - Specifically, it includes a div that the React app will show up inside
- The package.json file stores the lists of dependencies for the application
- The node_modules directory is where all of the dependencies get built/stored
- The src directory stores all of the modifiable code
  - index.js stores the main render() call from ReactDOM. It imports the App.js component & tells React where to render it
    - react is for React components & react-dom is for rendering components in the DOM
  - index.css stores the base styling for the application
  - App.js is a sample React component
  - App.css stores styling targeting that component specifically
  - logo.svg is just the React logo
- Use npm start from the command line to start the development server & to compile & serve the app

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX
- Guidelines for Planning a React Application
- **Working with Components**
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

Creating nested components (1/3)

- When using CRA, the App.js is the main component which is rendered first
- Create components in a separate .js files
- Include the markup of each child component inside the JSX of the main component inside App.js
- Each time React encounters the markup of one component inside another, it calls the render() method of the child component

Creating nested components (2/3)

GroceryListComponent

- The code snippet below uses the **GroceryItemComponent** inside the **GroceryListComponent.**

```
import React,{Component} from 'react';
import GroceryItemComponent from './groceryitemcompponent';

class GroceryListComponent extends Component
{
    render()
    {
        return (
            <div>
                <GroceryItemComponent/>
            </div>
        );
    }
}
```

Creating nested components (3/3)

GroceryItemComponent

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

class GroceryItemComponent extends Component
{
    render()
    {
        return (
            <div>
                <span>Item1</span>
                <span>Item2</span>
                <span>Item3</span>
            </div>
        );
    }
}
```

Passing data to components – props (1/3)

- Props are immutable pieces of data that are passed into child components from parents
  - They are the inputs to components. If we think of a component as a function, we can think of props as parameters
- They are single values or objects containing a set of values that are passed to React Components on creation using a naming convention similar to HTML-tag attributes
- Within React Component definition, these attributes then become properties attached to the React's native props object within component's constructor or inside the render() method
- Can be made explicit in components stating which props can be used, which ones are required & which types of values they accept
  - This can be done by declaring propTypes on the component class
  - They are a way to validate the values that are passed in through props

Passing data to components – props (2/3)

- The following code snippet declares local variables inside a component class & then uses the variable values to pass props to the <EmployeeComponent/>

```
class App extends Component
{
    render()
    {
        let name = "ABC";
        let age = 40;
        let address = {
            city:"Mumbai",
            state:"MAH"
        };

        return (
            <div>
                <EmployeeComponent name={name} age={age}
                            address={address}/>
            </div>
        );
    }
}
```

Passing data to components – props (3/3)

```
class EmployeeComponent extends Component
{
        render()
        {
                return
                        <h3>Employee Component</h3>
                        <h4>Name: {this.props.name}</h4>
                        <h4>Age: {this.props.age}</h4>
                         <h4>Address:</h4>
                         <h5>City: {this.props.address.city}</h5>
                         <h5>State: {this.props.address.state}</h5>
                         <h5>Pincode: {this.props.address.pincode}</h5>
        }
}
```

Property Validation – PropTypes (1/2)

- React PropTypes export a range of validators that can be used to make sure the data received is valid.
  - By default, all of the PropTypes are optional, but they can be chained with isRequired to ensure a warning is shown if the prop isn't provided

Validator	Description
PropTypes . array	Prop must be an array.
PropTypes . Bool	Prop must be a Boolean value (true/false
PropTypes . Func	Prop must be a function
PropTypes . number	Prop must be a number (or a value that can be parsed Into a number)
PropTypes . Object	Prop must be an object.
PropTypes . string	Prop must be a string

Property Validation – PropTypes (2/2)

- The sample code snippet uses propTypes to validate properties of an Employee component.

```
import React,{Component} from 'react';
import PropTypes from 'prop-types';
class EmployeeComponent extends Component { }

EmployeeComponent.propTypes = { name:PropTypes.string.isRequired,
        age:PropTypes.number.isRequired,
        address:PropTypes.shape
        {
                city:PropTypes.string,
                state:PropTypes.string,
                pincode:PropTypes.number.isRequired
        }.isRequired
}
export default EmployeeComponent;
```

Default Prop values

- A default value is used in case none is provided
- Use the defaultProps object to provide a default value for a property

```
EmployeeComponent.defaultProps = { place:'AIROLI' };
```

Stateless v/s Stateful (1/4)

- To react means to switch from one state to another
- There must be a state & the ability to change that state
- By default, React elements are stateless
  - Their sole purpose is to construct & render virtual DOM elements
- A React element must encapsulate some state
  - Some user action will trigger a change of state
  - Every state is represented by a different React element
- A React component is therefore a kind of a state machine

Stateless v/s Stateful (2/4)

- The render() function is responsible for telling React how to render a React component
  - As with any other function, before it returns a value, it can choose what value to return

```
import React,(Component) from 'react';

class StatefulComponent extends Component
{
        render() {
                //encapsulate some state here
                var elementState  = {sHidden:true ;}

                //render depending on the state
                if(elementState.isHidden == true)
                {
                        //don't render anything
                        return null;
                }
                return <h2>StatefulComponent rendered<h2>
        }
}
export default StatefulComponent;
```

Stateless v/s Stateful (3/4)

- There are two ways to pass data to a render() function using the React API
  - this.props
  - this.state

```
import React,(Component) from 'react.;
class StatefulComponent extends Component
{
        render() {
        //render depending on value of
isHidden props          if(this.props.isHidden ==
true) {
                        return( <h2)Component
will be
        hidden</h2> );
        }
        return <h2>StatefulComponent
rendered!!< /h2>
}
export default StatefulComponent;
```

```
<StatefulComponent
isHidden={false}/>
<StatefulComponent isHidden={true}/>
```

render() must not worry about the state & it shouldn't mutate the state or access the real DOM. Hence, the state is passed from outside & accessed inside **render()** using **this.props.**

This makes our component "STATELESS"

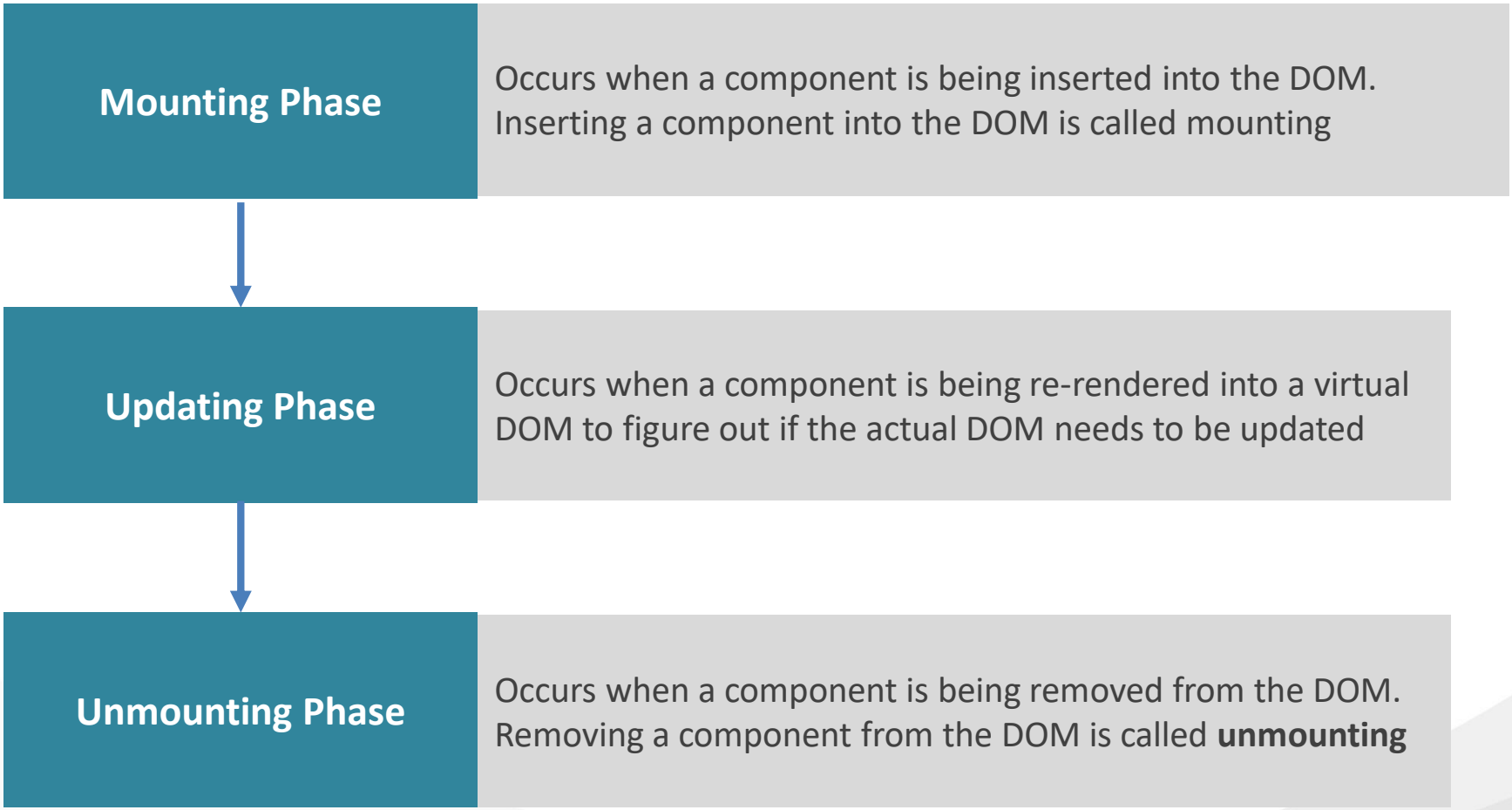
Stateless v/s Stateful (4/4)

- React components are composable
  - There can be a hierarchy of React components
  - Each component can manage its own state
- It’s NOT easy to figure out what the last child component in the hierarchy will render if the top component in the hierarchy updates its state
- The idea is to separate the components into two concerns: how to handle the user interface interaction logic & how to render data
  - Stateful components should be at the top of the components' hierarchy. They encapsulate all of the interaction logic, manage the user interface state, & pass that state down the hierarchy to stateless components, using props
  - Stateless components receive state data from their parent components via this.props & render that data accordingly

Component Lifecycle (1/6)

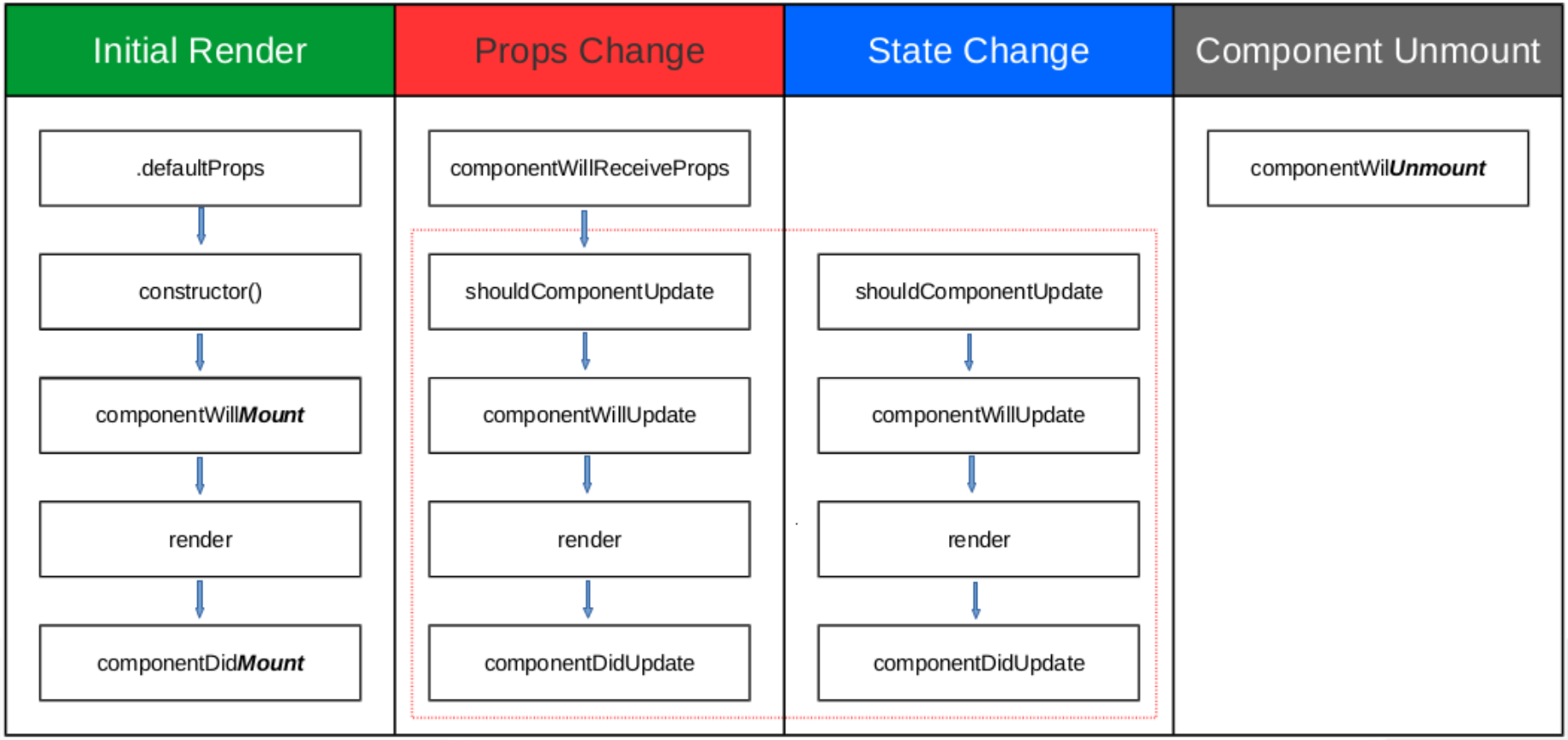
- Each React component goes through a process, during which it is rendered
- This process is called the component's lifecycle
- React provides a number of methods that we can use to get notified when a certain stage in a component's lifecycle process occurs
  - These methods are called the component's lifecycle methods
  - They are called in a predictable order
- All the React component's lifecycle methods can be grouped into three phases:
  - Mounting
  - Updating
  - Unmounting

Component Lifecycle (2/6)



Component Lifecycle (3/6)

The predictable order



SOURCE: https://medium.com/@re6exp/react-js-simple-lifecycle-schema-511e0523923f

Component Lifecycle (4/6)

- Class Constructor
  - Called once & for the first time before any other method
  - Must receive props as an argument & call super with props passed to it so that props are available
  - Can be used to set the initial state of the component
- componentWillMount()
  - Invoked immediately before React inserts a component into the DOM
  - Executed once in a lifecycle of a component & before first render()
  - Used for initializing the states or props
- componentDidMount()
  - Executed once in a lifecycle of a component & after the first render()
  - Best place for initializing other JavaScript libraries that need access to that DOM

Component Lifecycle (5/6)

- componentWillReceiveProps(nextprops)
  - Invoked first in the component lifecycle's updating phase
  - Called when a component receives new properties from its parent component
  - Is an opportunity to compare the current component's properties using this.props object with the next component's properties using the nextprops object
    - Based on this comparison, we can choose to update the component's state using this.setState() function, which will not trigger an additional render
- shouldComponentUpdate(nextprops, nextstate)
  - Allows to decide whether the next component's state should trigger the component's re-rendering
  - Returns a Boolean value, which by default is true
  - Great place to prevent re-rendering of the component based on some condition
- componentWillUpdate(nextprops, nextstate)
  - Called immediately before React updates the DOM
  - setState() cannot be used here
  - Can be used as a preparation stage before the DOM update

Component Lifecycle (6/6)

- componentDidUpdate(previousprops, previousstate)
  - Called immediately after React updates the DOM
  - Can be used to interact with the updated DOM or perform any post-render operations
  - Updating cycle ends after this method is called
  - A new cycle is started when a component's state is updated or a parent component passes new properties
  - Good place to do network requests as long as you compare the current props to previous props
- componentWillUnmount()
  - Invoked immediately before a component is unmounted & destroyed
  - Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount()

Stateful Component(s) (1/4)

- Stateful components are the most appropriate place for an application to handle the interaction logic & manage the state
  - This makes it easy to reason out how the application works
  - This reasoning plays a key role in building maintainable web apps
- React stores the component's state in this.state
- The initial state can be set by reading value(s) from props as well

Stateful Component(s) (2/4)

- The following code snippet creates a stateful component which creates a flag for hiding or showing a component.

```
class StatefulComponent extends Component
{

		//get the state from props & initialize the state
		constructor(props)
		{
			//call base constr with 'props'
			super(props);
			this.state ={ isHidden:this.props.isHidden };
		}
		render()
		{
			//NOW CHECK THE STATE!!
			if(this.state.isHidden == true)
			{
				return( <h2>Component's STATE IS HIDDEN</h2>
			}
			return  <h1>Component's STATE IS NOT HIDDEN</h1>
		}
}
```

Stateful Component(s) (3/4)

- Difference between this.props & this.state
  - this.props stores read-only data that is passed from the parent. It belongs to the parent & cannot be changed by its children. This data should be considered immutable
  - this.state stores data that is private to the component. It can be changed by the component. The component will re-render itself when the state is updated
- There is a common way of informing React of a state change using setState(data) function
  - The data represents the next state
- React calls the component's render() function every time a component's state is updated. It re-renders the entire virtual DOM every time the render() function is called

Stateful Component(s) (4/4)

- When this.setState() function is called & passed a data object that represents the next state, React merges that next state with the current state
- During the merge, React overwrites the current state with the next state. The current state that is not overwritten by the next state becomes part of the next state

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX
- Guidelines for Planning a React Application
- Working with Components
- **Event Handling**
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

Event Handling (1/3)

- React events are named using camelCase, rather than lowercase
- With JSX, a function is passed as the event handler, rather than a string

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


```
<button onClick={activatelasers)>Activate Lasers</button>
```

Event Handling (2/3)

- React implements a synthetic event system that brings consistency & high performance to React applications
  - It achieves consistency by normalizing events so that they have the same properties across different browsers & platforms
  - It achieves high performance by automatically using event delegation. React doesn't actually attach event handlers to the DOM nodes themselves. Instead, React listens for all the events at the top level using a single event listener & delegates them to their appropriate event handlers
- Event handlers are passed instances of SyntheticEvent, a cross-browser wrapper around the browser's native event, which works identically across all browsers
- More on SYNTHETIC EVENTS - https://reactjs.org/docs/events.html

Event Handling (3/3)

Passing args to event handler(s)

- The sample code snippet shows how event arguments can be passed to an event handler function of a React component.

```
render()
   {
      return (
         <div>
            <button onClick={this.HandleClick(this,"button1")}>Button 1</button>
            <button onClick={this.HandleClick(this,"button2")}>Button 1</button>
         </div>
      );
   }
   HandleClick(eventdata,eventdsource)
   {
      console.log("Event data: " +eventdata);
      console.log("Event source: " +eventsource.target.innerText);
   }
```

Setting state with event handling – An example

```
class CounterComponent extends Component
{
   constructor(props)
   {
      this.state = {
         count:this.props.count,
         time: new Date().toLocaleTimeString()
      }
   }
   render()
   {
      return (
         <div>
            <h3>Counter Component</h3>
            <h4>Count: {this.state.count} at
{this.state.time}</h4>
            <button
onClick={this.Increment.bind(this)}>Increment</button>
         </div>
      );
   }
}
```

```
<CounterComponent count={1}/>
<CounterComponent count={10}/>
```

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- **React Rendering**
- Using AJAX with ReactJS
- Routing in ReactJS
- Refs and DOM
- References

Conditional Rendering (1/3)

- In React, distinct components that encapsulate behavior can be rendered depending on the state of the application
  - This works the same way conditions work in JavaScript
  - Use JavaScript operators like if or the conditional operators to create elements representing the current state, & let React update the UI to match them
- Example:
  - A GreetingComponent can render a UserGreetingComponent or a GuestUserComponent depending on whether a user is logged in
  - The GreetingComponent internally stores the login status in its state

Conditional Rendering (2/3)

Inline if with && operator

- Expressions can be embedded within JSX by wrapping them in curly braces
  - This includes the usage of the JavaScript && logical operator
  - This can be handy for conditionally including an element

```
render()
  {
    let messages = this.props.messages;

    return (
      <div>
          messages.length > 0 &&
            <h2>You have {messages.length} messages</h2>
      </div>
    );
  }
```

Conditional Rendering (3/3)

Inline if..else

- This uses the JavaScript condition ? true : false operator

```
render()
   {
      let messages = this.props.messages;

      return (
         <div>
               {
                  messages.length > 0 ?
                  (
                     <h3>You have {messages.length} messages</h3>
                  )
                  :
                  (
                     <h3>There are no messages</h3>
                  )
               }
         </div>
      );
   }
```

Lists & Keys (1/4)

- The map() function in JavaScript can be used to transform an array & obtain a new array

```
const numbers = [1,2,3,4,5,6];
const doubled = numbers.map((number)
=> number * 2);
console.log(doubled);pro
```

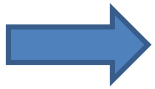
- In React, an array can be transformed into a list of elements
- This can be used to build a collection of elements which can then be included in JSX using curly braces

```
let productlist = [

{productid:1,pname:'P1',price:600.00
},

{productid:2,pname:'P2',price:700.00
},

{productid:3,pname:'P3',price:800.00
}
];
```



export default productlist;

Lists & Keys (2/4)

```
<ul>
  {
    productlist.map((product) =>
      <li>
        {product.productid},
        {product.productname},
        {product.price}
      </li>
    )
  }
</ul>
```

When this code is run, a warning is displayed stating that **a key should be provided for list items**

Lists & Keys (3/4)

- A key is a special string attribute which needs to be included when creating lists of elements
- Keys help React identify which items have changed, are added, or are removed
- Keys should be given to the elements inside the array to give the elements a stable identity
- The best way to pick a key is to use a string that uniquely identifies a list item among its siblings

```
const numbers = [1,2,3,4,5];
const listitems = numbers.map((number) =>
   <li key={number.toString()}>number</li>
);
```

```
const toDoItems = toDos.map((toDo)
=>
   <li key={toDo.id}>{toDo.text}</li>
);
```

Lists & Keys (4/4)

- Keys only make sense in the context of the surrounding array
- Example
  - If a ProductComponent component is extracted, it is recommended to keep the key on the <ProductComponent/> elements in the array rather than on the root <li> element in the ProductList itself

```
class ProductComponent extends
Component
{
    render()
    {
        return (

<li>{this.props.value}</li>
        );
    }
}
```

```
<ul>
    {
        productlist.map((product)
=>
            <ProductComponent
key={product.productid}

value={product.productname}/>
        )
    }
</ul>
```

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- **Using AJAX with ReactJS**
- Routing in ReactJS
- Refs and DOCM
- References

Using AJAX with React JS (1/4)

- React itself doesn't have any of it's own APIs for fetching data
- It simply renders components, using data from only two places: props & state
- To use some data from the server, the data needs to be fetched into the components' props or state
- To fetch that data from the server, there is a need for an HTTP library
  - Can choose from a variety of libraries like jQuery AJAX, AXIOS, fetch etc.

Using AJAX with React JS (2/4)

- Install the AXIOS library using npm
  - A Promise based HTTP client for the browser & Node.js

```
D:\myreactapp>npm install axios --save
```

- Import the axios module

```
import axios from 'axios';
```

- Make the AJAX call inside the componentDidMount() function & assign the results obtained to the component's state or props

Using AJAX with React JS (3/4)

POST Operation (a)

```
render()
    {
        return (
            <div>
                <h4>POST operation - Add a new country</h4>
                <p>
                    Country id:
                    <input type="text" id="countryid"/>
                </p>

                <p>
                    Country name:
                    <input type="text" id="countryname"/>
                </p>
                <button onClick={this.AddCountry.bind(this)}>Add
Country</button>
            </div>
        );
    }
```

Using AJAX with React JS (4/4)

POST Operation (b)

```
AddCountry()
    {
        let countryid = document.getElementById('countryid').value;
        let countryname = document.getElementById('countryname').value;
        let country = {CountryId:countryid,CountryName:countryname};

        axios.post("http://localhost:9999/api/Countries", country)
            .then((response) =>
            {
                alert(response.data);
            })
            .catch((error) =>
            {
                alert(error.response.data.Message));
            };
    }
```

Styles in React (1/2)

- React provides the capacity to write inline styles using JavaScript
  - Inline styles are specified as a JavaScript object
  - Style names are camel cased in order to be consistent with DOM properties
  - Don’t specify pixel units - React automatically appends the correct unit behind the scenes

```
class Hello extends Component
{
        render()
        {
                let divStyle = { width: 100, height: 30, padding: 5,backgroundColor:
'#ee9900'
        }
        return <div style={divStyle}>Hello World</div>
}
```

Styles in React (2/2)

- Styles can be grouped into CSS Modules & then referenced inside a component class using import

```
.divStyle
{
        color:orange;
        background-image:
url('./images/reactlogo.jpg');
        height:200px;
        border-style:solid;
        text-align:center;
}

.h1style
{
        color:red;
        font-family:Verdana;
}
```

```
import './reactstyles.css';

class StyledComponent extends
Component
{
    render()
    {
        return (
            <div
className="divStyle">
                <h1
className="h1style">...</h1>
            </div>
        );
    }
}
```

Controlled Components (1/2)

- In HTML, form elements such as <input>, <textarea> & <select> typically maintain their own state & update it based on user input
  - An input form element whose value is controlled by React is called a controlled component

```
class NameForm extends
React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange =
this.handleChange.bind(this);
    this.handleSubmit =
this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value:
event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: '
+ this.state.value);
    event.preventDefault();
  }
```

```
 render() {
    return (
      <form
onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text"
value={this.state.value}
         onChange={this.handleChange}
/>
        </label>
        <input type="submit"
value="Submit" />
      </form>
    );
  }
}
```

Controlled Components (2/2)

- With a controlled component, every state mutation will have an associated handler function. This makes it straightforward to modify or validate user input.

```
render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value}
onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
```

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- **Routing in ReactJS**
- Refs and DOM
- References

Routing in React JS (1/6)

- Routing is the process of keeping the browser URL in sync with what's being rendered on the page
  - React Router lets you handle routing declaratively
  - The declarative routing approach allows you to control the data flow in your application
- react-router is the standard routing library for React
  - This library comprises three packages: react-router, react-router-dom, & react-router-native
  - react-router is the core package for the router
  - Use react-router-dom when building a website
  - Use react-router-native when building a mobile app development environment using React Native

Routing in React JS (2/6)

Setting up the router

- Install the routing library

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

- Create one or more routes

```
<Router>
  <Route exact path="/" component={Home}/>
  <Route path="/category" component={Category}/>
  <Route path="/login" component={Login}/>
  <Route path="/products" component={Products}/>
</Router>
```

Routing in React JS (3/6)

Router Types

- There are two types of routers from the React Router API:
  - <BrowserRouter>
  - <HashRouter>
- The primary difference between them is evident in the URLs that they create

```
// <BrowserRouter>
http://example.com/about

// <HashRouter>
http://example.com/#/about
```

Routing in React JS (4/6)

Adding routing to a React App

- Wrap the <BrowserRouter> component around the App component

```
/* Import statements */
import React from 'react';
import ReactDOM from 'react-dom';

/* App is the entry point to the React code.*/
import App from './App';

/* import BrowserRouter from 'react-router-dom' */
import { BrowserRouter } from 'react-router-dom';

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

Routing in React JS (5/6)

Links and Routes

- The <Route> component is the most important component in React router
  - It renders some UI if the current location matches the route's path
  - Should have a prop named path, & if the pathname is matched with the current location, it gets rendered
- The <Link> component, on the other hand, is used to navigate between pages.
  - It's comparable to the HTML anchor element
  - Use <Link> to navigate to a particular URL and have the view re-rendered without a browser refresh

```
<nav className="navbar navbar-light">
          <ul className="nav navbar-nav">

           /* Link components are used for linking to other views */
            <li><Link to="/">Homes</Link></li>
            <li><Link to="/category">Category</Link></li>
            <li><Link to="/products">Products</Link></li>

          </ul>
        </nav>
```

Routing in React JS (6/6)

Switch component

- When multiple <Route>s are used together, all the routes that match are rendered inclusively

```
<Route exact path="/" component={Home}/>
<Route path="/products" component={Products}/>
<Route path="/category" component={Category}/>
<Route path="/:id" render = {()=> (<p> I want this text to show up for all
routes other than '/', '/products' and '/category' </p>)}/>
```

- If the URL is /products, all the routes that match the location /products are rendered
- So, the <Route> with path :id gets rendered along with the Products component
- With <Switch>, only the first child <Route> that matches the location gets rendered

```
<Switch>
      <Route exact path="/" component={Home}/>
      <Route path="/category" component={Category}/>
       <Route path="/products" component={Products}/>
    </Switch>
```

Agenda

- What is ReactJS?
- The Virtual DOM
- Introduction to JSX (JavaScript XML)
- Guidelines for Planning a React Application
- Working with Components
- Event Handling
- React Rendering
- Using AJAX with ReactJS
- Routing in ReactJS
- **Refs and DOM**
- References

Refs & the DOM

- Refs provide a way to access DOM nodes or React elements created in the render() method
- Creating refs:
  - Can be created using React.createRef() & attached to React elements via the ref attribute
  - Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component

```
class MyComponent extends
React.Component
{
  constructor(props) {
    super(props);
    this.myRef = React.createRef();
  }
  render() {
    return <div ref={this.myRef} />;
  }
}
```

- When a ref is passed to an element in render, a reference to the node becomes accessible at the current attribute of the ref

```
const node = this.myRef.current;
```

Summary

- React is a pure JS library for creating user-interfaces using JavaScript
- React manipulates the virtual DOM instead of the actual DOM which makes it much more optimized when rendering
- React is component-based
- React components can be stateful or stateless
- React uses props which allows external input to be passed to it
- React re-renders the component when the internal state changes
- React in itself does not have any libraries for making HTTP calls, but it relies on external libraries to do so
- React routing is a way of navigating between components.
- React uses refs to access DOM elements

References

- https://babeljs.io/docs/plugins/transform-react-jsx/
- https://reactjs.org/docs/events.html
- https://reactjs.org/docs/reconciliation.html#recursing-on-children5
- https://medium.com/@re6exp/react-js-simple-lifecycle-schema-511e0523923f

Thank You

CitiusTech Markets

CitiusTech Services

CitiusTech Platforms

Accelerating Innovation

CitiusTech Contacts

Email ct-univerct@citiustech.com

www.citiustech.com