WEB TECHNOLOGIES

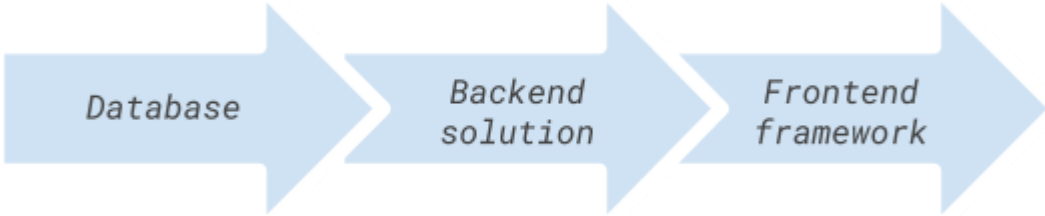
Web Development Stack

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

Web Development Stack

Introduction

- A set of tools typically used in tandem to develop web apps.
- Refers to the technologies that individual developer specializes in and use together to develop new pieces of software.
  - Front End web development stack
  - Back End web development stack
  - Full web development stack
  - Web technology sets that include all the essential parts of a modern app:
    - **frontend framework**,
    - **backend solution** and
    - **database (relational or document-oriented)**



Web Development Stack

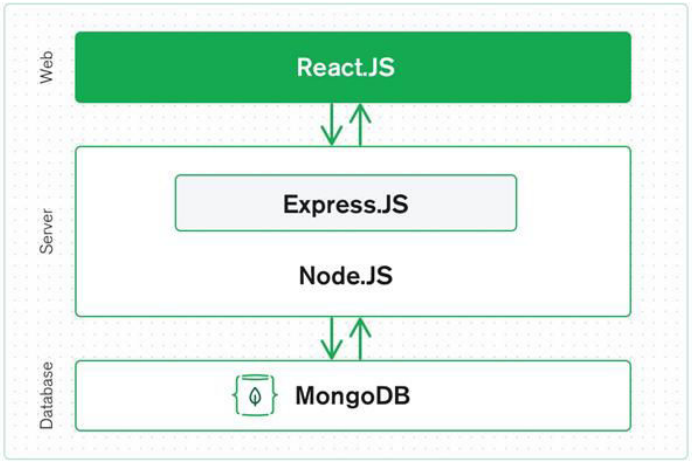
Full web development Stack

- As the web development world is continually changing, its technology stacks too change
- Top web development stacks
  - MEAN
  - MERN
  - Meteor.js
  - Flutter
  - The serverless Technology stack
  - The LAMP technology stack
  - Ruby on Rails Tech Stack

Web Development Stack

MERN - Introduction

- Stands for **MongoDB, Express, React, Node**
  - MongoDB - document database
  - Express(.js) - Node.js web framework
  - React(.js) - a client-side JavaScript framework
  - Node(.js) - the premier JavaScript web server



- Allows you to easily construct a **3-tier architecture** (frontend, backend, database) entirely using JavaScript and JSON MERN

Web Development Stack
React.JS

- Front End Library (The top tier of MERN Stack)
- The declarative JavaScript Library for creating dynamic client-side applications
- Build up complex interfaces through
    - simple Components,
    - connect them to data on your backend server, and
    - render them as HTML
- Strengths:
    - Handling stateful, data-driven interfaces
    - Great support for forms, error handling, events, lists, etc.

MERN Stack
Node.JS and Express.JS

- Server side Framework (The middle tier of MERN Stack)
- Express.js server-side framework, running inside a Node.js server
- Has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses
- In turn use Node.js MongoDB drivers, either via callbacks for using Promises, to access and update data in your MongoDB database

MERN Stack
MongoDB

- Database Server (The bottom tier of MERN Stack)
- Application data store(user profiles, content, comments, uploads, events, etc.)
- JSON documents created in the front end can be stored directly in MongoDB for later retrieval through Node.JS and Express.JS

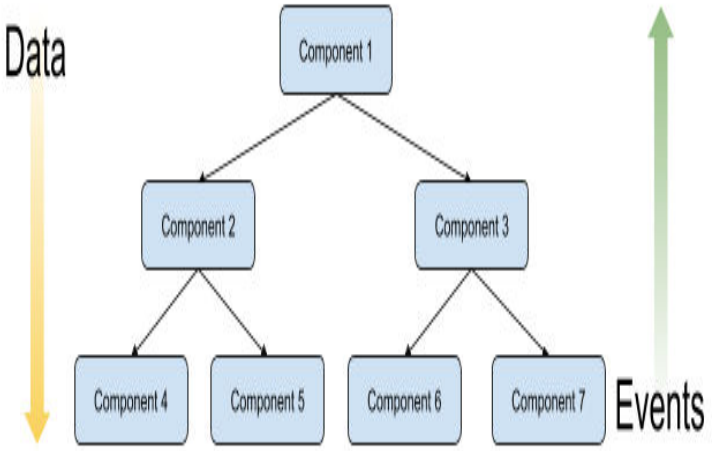
React.JS
Introduction

- An open-source **JavaScript library used for building user interfaces** specifically for **single-page applications.**
- Handles the **view layer for web and mobile apps.**
- Allows developers to create **large web applications that change dynamically**, without the need of reloading.
- Allows to **create reusable UI components.**
- **Works only on user interfaces** in the application.
- First created **by Jordan Walke**, a software engineer working for Facebook. First dep on Facebook's newsfeed in 2011 and on Instagram.com in 2012.

React.JS
Why React?

- **Properties**
  - Declarative, Simple, Component based, Supports server side, Mobile support, Extensive, Fast , Easy to learn

- **Single way data flow**
- In React JS we cannot make any change to a component directly if needed we provide a callback.
- Actions flow up and properties flow down
  - The property of passing a value in the component and rendering it as HTML tags properties



Why React?

PES UNIVERSITY ONLINE

Virtual DOM

- React maintains a in-memory copy of the actual DOM known as the ***virtual DOM***
- Manipulating this virtual DOM is extremely fast
- React takes care of updating the real DOM when the time is right
- Compares the changes between your virtual DOM and the real DOM
- Figuring out which changes actually matter, and making the least amount of DOM changes

React.JS

How to create a React App?

- Two ways:
  - **Using node package manager(npm):**
    - To setup a build environment for React involves use of npm (node package manager), webpack, and Babel
  - **Directly importing Reactjs library in HTML Code.**
  - Defined in two .js files (**React** and **ReactDOM**)
    - <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
    - <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin> </script>
    - The files differ for development and production.

React.JS
React Component

- A user control that has **code to represent visual interfaces and data**
- An isolated piece of code which can be reused in one or the other module
- Contains a root component in which other subcomponents are included
- 2 types of components in React.js
  - **Stateless Functional Component**
  - **Stateful Class Component**

React.JS

React Component Example codes

- **Stateless Functional Component**

```
function Demo(props) {
  return <h1> Welcome to REACT JS, {props.Name} </h1>;
}
```

- **Stateful Class Component**

```
class HelloClass extends React.Component
        {       render()
                {    return React.createElement('h1', null, 'welcome to REACT JS');        }
        }

class Demo extends React.Component{
            render(){
                    return <h1> Welcome to REACT JS, {props.Name} </h1>;    }
}
```

React.JS

React Component – Few key terms

- **Placeholder**: A place where the component will load
- **Create component**: Stateless or stateful
- **Call Component: ReactDOM.render()** is responsible for rendering a React component.
  - The first parameter: is a component class name.
  - Second parameter: is the destination where the component is to be rendered.

  In our case, we render component inside the div id='root'.

  - Example: ReactDOM.render(

```
React.createElement(HelloClass, null, null),
document.getElementById('root'));
```

React.JS
What is JSX?(JAVASCRIPT XML)

- JavaScript Extension Syntax used to **mix HTML with JavaScript**
- Uses HTML syntax to create elements and components.
- Has tag name, attributes, and children.
- JSX compiles the code into pure JavaScript which can be understood by the browser.
- Include the library: <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"> </script>

```
<script type="text/babel">
        ReactDOM.render(<h1>Welcome    to REACTJS</h1>,
          document.getElementById('root'));
```

React.JS
What is Babel?

- A JavaScript compiler that can translate markup or programming languages into JavaScript.
- Available for different conversions.
- React uses Babel to convert JSX into JavaScript.
- Please note that <script type="text/babel"> is needed for using Babel

```
<script type="text/babel">
        ReactDOM.render(<h1>Welcome    to REACTJS</h1>,
           document.getElementById('root')
        );
```

React.JS

Visuals Defined using JSX vs Pure JavaScript

JSX

```
ReactDOM.render(
    <div>
        <h1>Batman</h1>
        <h1>Iron Man</h1>
        <h1>Nicolas Cage</h1>
        <h1>Mega Man</h1>
    </div>,
destination );
```

JavaScript

```
ReactDOM.render(
    React.createElement ( "div", null,
        React.createElement ( "h1", null, "Batman" ),
        React.createElement ( "h1", null, "Iron Man" ),
        React.createElement ( "h1", null, "Nicolas Cage" ),
        React.createElement ( "h1", null, "Mega Man" ),
    destination);
```

React.JS
Demonstration

- Using React, print welcome to REACTJS on the web page
- Use JSX and NonJSX both

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 8026726622

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Components and Properties

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

React.JS – Components and Properties

Agenda

- Introduction to Components
- Types of components
- First Stateful component
- Parameterized components
- Components with child element
- Usage of props.child
- Introduction to Properties
- Validating and setting property values
- Transferring properties

React.JS – Components and Properties

Introduction to Components

- Defines the **visuals and interactions** that make up what the end user percieve when they use you app.
- Need for Components
  - There may be elements which are similar
  - A need to make a change in one will reflect changes in multiple places
  - Similar to functions, if we could write code related to the element in one place, changes can be minimized
- Solution - Reusable piece of JavaScript code that output (via JSX) HTML elements

React.JS – Components and Properties

Types of Components

- **Stateless Functional Component**
  - Includes simple JavaScript functions and immutable properties, i.e., the value for properties cannot be changed.
  - Use hooks to achieve functionality for making changes in properties using JS.
  - Used mainly for UI.

- **Stateful Class Component**
  - Classes which extend the Component class from React library.
  - The class component must include the render method which returns HTML.

React.JS – Components and Properties

First Stateful Component

Creation of a component:

```
class HelloWorld extends React.Component
{       render()
        {   return <p>Hello, componentized world!</p>; }
}
```

Calling a component: Add the JSX in the render method with a element with the tag name as the Component name

```
ReactDOM.render(  <HelloWorld/>,
          document.querySelector("#container")
);
```

React.JS – Components and Properties

Parameterized components

- Accepting the properties in the component and passing these properties while calling the components
- **Creation of a component:**

```
class HelloWorld extends React.Component {
                render() {  return <p>Hello, {this.props.greetTarget}!</p>;          }
        }
```

- **Calling a component:**

```
ReactDOM.render(
    <div>
        <HelloWorld greetTarget="Batman"/>  <HelloWorld greetTarget="Iron Man"/>
  </div>,
  document.querySelector("#container")
);
```

React.JS – Components and Properties

Components with Child elements

- **Creation of a component:**

```
class Buttonify extends React.Component { render()
                    { return(
                    <div> <button type={this.props.behavior}>{this.props.children}
                    </button>
                     </div> ); }
                      }
```

- **Calling a component:**

```
ReactDOM.render(
          <div>
             <Buttonify behavior="submit">SEND DATA</Buttonify>
          </div>,          document.querySelector("#container")          );
```

React.JS – Components and Properties

Usage of props.children

- Used to display what you include data between the opening and closing JSX tags when invoking a component.

- A special property that is passed to components automatically

- Can have one element, multiple elements, or none at all.

- It’s value is respectively a single child node, an array of child nodes or undefined.

React.JS – Components and Properties

Introduction to Properties

- Properties are ways in which **React components can be customized.**
- Props are arguments passed into React components and are passed via HTML attributes.
- Properties are immutable and are same as what attributes are to HTML elements.
- Their most basic use is in the form of attributes, in JSX.
- Are actually passed to a constructor and can be accessed by **this.props**
- Design decision is to use the Attributes specified in hyphen-case in HTML as camelCase in JSX.
- 2 steps to add properties to components
  - Make the function of your component read the props from the props parameter
  - When rendering the component, add the prop to the component using the attribute

React.JS – Components and Properties

Validating and Setting the property values

- Props validation is a tool that will help the developers to avoid future bugs and problems.
- Validating the type of properties
  - Possible by providing any of the types in **React.PropTypes.**

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

- Setting the default property values
  - Can be overridden by properties passed in initialization using **getDefaultProps()** method
  - Must return an object. The contents of that object are up to us
- Coding examples

React.JS – Components and Properties

Transferring Properties

- Sending properties down the structural tree while working with multiple components in a hierarchy
- The properties have to flow down from its parent to every child component that lies on the intended path
- **transferPropsTo():**
  - Passes all the properties of a parent component to a child component unless we explicitly set the value of a child component's property.
  - ReactComponent transferPropsTo(ReactComponent targetComponent)
- Coding examples

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 9886703973

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Styling the Components and Complex Components

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

React.JS – Styling the Components

Agenda

- Introduction
- Inline CSS
- Usage of props.children
- Demo code

React.JS – Styling the Components

Introduction

- Different ways to style React Components
  - Inline CSS - using style attribute
  - CSS in JS - Using Libraries JSS and Styled Components
  - CSS modules – css loader and Sass & SCSS
  - Stylable

React.JS – Styling the Components

Inline CSS

- In inline styling we create objects of style and render it inside the components in style attribute using the React technique.
- In JSX, JavaScript expressions and objects are written inside curly braces the styling is written inside two sets of curly braces {{}}.
- Example:

```
class MyHeader extends React.Component {
  render() {    return (
                        <h1 style={{color: "red"}}>Hello Style!</h1>
                         <p>Add a little style!</p>      );  }}
```

- camelCased Property Names

```
<h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
```

React.JS – Styling the Components

Inline CSS continued..

- Creation of JavaScript object which contains the css properties and values in the form of key value pair.
- Passing this as a value to style attribute
- Example:

```
class Letter extends React.Component {
    render() {var letterStyle = {
                padding: 10,
                margin: 10,
            backgroundColor:"#ffde00",
            color: "#333“,
            display: "inline-block“,
            fontFamily: "monospace",
            fontSize: 32,
            textAlign: "center“
            };
    return(
        <div style={letterStyle}>
          {this.props.children}
        </div>       ); }
```

React.JS – Styling the Components

Demo

- Create the following letters using react components

- Additionally, apply styles to them differently by passing styling properties

React.JS – Complex Components

Agenda

- Introduction
- Approach followed
- Complex Component creation
- Sub component creation
- Properties from parent to sub component
- Modified parent and sub components
- Transferring properties

React.JS – Complex Components

Introduction

- Combining components to create the complex one
- Advantage is composability
- Approach
  - Identify the major visual elements
  - Breaking them into individual components

React.JS – Complex Components

Approach followed

tiger | Facts, Information, & Habitat ...
britannica.com

- The complex component consists of **Image rendering, caption rendering and link rendering** on the web page

React.JS – Complex Components

Creation of complex component

```
class SrchResult extends React.Component {
    render() {
        return (
            <div>
                <ResImage/>
                <ResCaption/>
                <ResLink/>
            </div>
        );
    }
}
```

React.JS – Complex Components

Creation of sub components

```
class ResImage extends React.Component
{
    render() {
        return (
            <div> <img src="tiger.jpg">
            </img> </div> );
    }
}
```

```
class ResLink extends React.Component {
    render() {
        return (
            <div> <a
            href="https://www.britannica.com/animal/tiger">
            britannica.com </a> </div> );
    }
}
```

```
class ResCaption extends React.Component {
    render() {
        return (
            <div> <p>tiger | Facts, Information and
            Habitat…</p> </div> );
    }
}
```

React.JS – Complex Components

Properties from parent to sub component

To avoid hard coding, properties have to be passed from parent Components to it's sub components

- src
- href
- linktext
- caption

<SrchResult src="tiger.jpg" href="https://www.britannica.com/animal/tiger"
linktext="britannica.com" caption="tiger | Facts, Information and Habitat… " />

React.JS – Complex Components

Modified Parent and sub components

```
class SrchResult extends React.Component {
   render() {return ( <div>
            <ResImage {...this.props}/>    <ResCaption {...this.props}/>  <ResLink {...this.props}/>
                    </div>  );  }
   }
```

```
class ResImage extends React.Component {
     render() {
          return (
               <div> <img src={this.props.src}>
               </img> </div>
          );   }   }
```

```
class ResLink extends React.Component {
     render() {
          return (
               <div> <a href={this.props.href}> {this.props.linktext}
               </a> </div>  );
     }   }
```

```
class ResCaption extends React.Component {
     render() {
          return (
               <div> <p>{this.props.caption}</p> </div>
          );   }   }
```

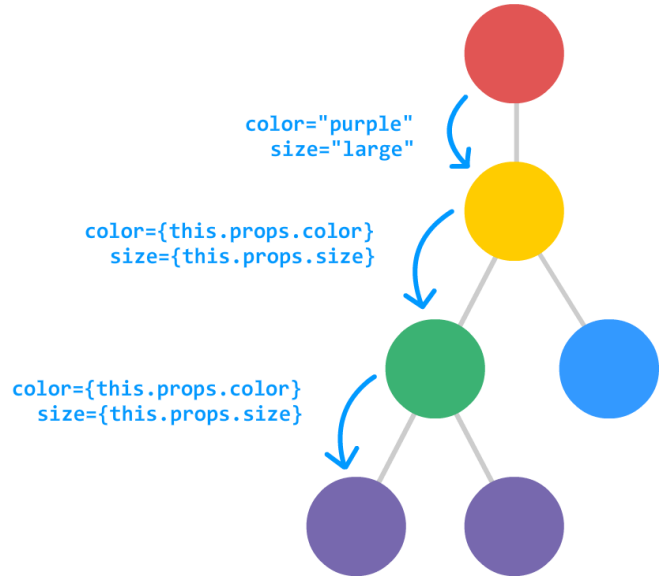
React.JS – Complex Components
Transferring Properties

- Transferring properties was not a tedious as there was only one level of hierarchy

- If there are multiple levels of components, transferring properties will be very cumbersome

- To overcome that, we use the **spread operator (…)**

<Display {...this.props}/>



THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 80 2672 6622

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Component States and LifeCycle Methods

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

Component States and LifeCycle Methods

Agenda

- Introduction to states
- Conventions of using states
- Demo of differences between props and states
- Introduction to Life cycle
- Life cycle methods in brief
- Demo

Component States and LifeCycle Methods

States - Introduction

- State is used with React Component Classes to make them dynamic.
- It enables the component to keep track of changing information in between renders.
- An instance of React Component Class can be defined as an object of a set of observable properties that control the behavior of the component
- An object that holds some information that may change over the lifetime of the component and to control the behavior after each change
- Can only be used in class components

Component States and LifeCycle Methods

Conventions of using states

- When using State, we need the state of a component to always exist i.e., we need to set an initial state.
- **initial state**, should define the State in the constructor of the component's class.

```
class MyClass extends React.Component {
        constructor(props)  {
            super(props);  this.state = { attribute : "value" };    }
}
```

- State should **never be updated explicitly**. Use **setstate()**
  - Takes a single parameter and expects an object which should contain the set of values to be updated.
  - Once the update is done the method implicitly calls the render() method to repaint the page.
- **State updates are independent**

Component States and LifeCycle Methods

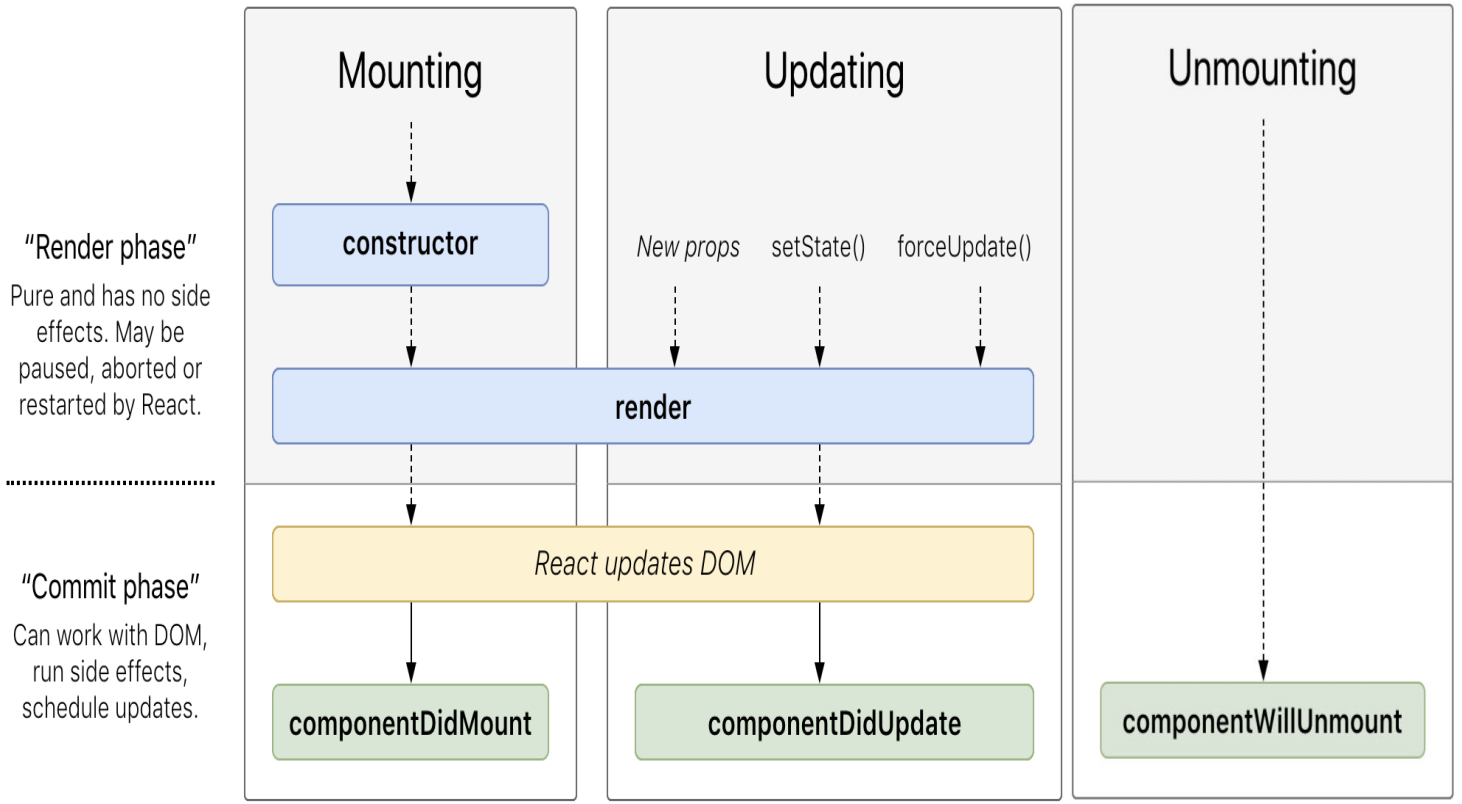
Demo of differences : props and state

- Generate the Digital clock using reactJS

Component States and LifeCycle methods

Introduction to Life Cycle

- The series of events that happen from the starting of a React component to its ending.
- Every component in React should go through the following lifecycle of events.
  - **Mounting** - Birth of the Component
  - **Updating**- Growing of component
  - **Unmounting**- End of the component



Component States and LifeCycle methods

Methods

- **componentWillMount**()
  - Executed before rendering, on both the server and the client side

- **constructor()**
  - Called before the component is mounted.
  - Implementation requires calling of **super(props)** before further moving. Otherwise, this.props it will be undefined in the constructor, which can lead to a major error in the application.
  - Supports **Initializing the state** and **Binding our component**

- **render()**
  - Most useful life cycle method as it is the only method that is required
  - Handles the rendering of component while accessing ***this.state*** and **this.props**

Component States and LifeCycle methods

Methods continued..

- **componentDidMount()**
  - The best place to initiate API calls in order to fetch data from remote servers
  - Use setState which will cause another rendering but It will happen before the browser updates the UI. This is to ensure that the user won't see the intermediate state
  - AJAX requests and DOM or state updates should occur here
  - Also used for integration with other JavaScript frameworks like Node.js and any functions with late execution such as **setTimeout** or **setInterva**l

- **componentWillReceiveProps()**
  - Allows us to match the incoming props against our current props and make logical
  - We get our current props by calling this.props and the new value is the nextProps argument passed to the method
  - It is invoked as soon as the props are updated before another render method is called

Component States and LifeCycle methods

Methods continued..

- **shouldComponentUpdate()**
  - Allows a component to exit the Update life cycle if there's no reason to use a replacement render.
  - It may be a no-op that returns true. Means while updating the component, we'll re-render.

- **componentWillUpdate()**
  - Called just before the rendering

- **componentDidUpdate()**
  - Is invoked immediately after updating occurs. Not called for the initial render.
  - Will not be invoked if **shouldcomponentUpdate() returns false**.

- **componentWillUnmount()**
  - Called when a component is being removed from the DOM

Component States and LifeCycle methods

Demo of diff lifecycle methods

PES UNIVERSITY ONLINE

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 80 2672 6622

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Stateless Components

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

Stateless Components

Agenda

- Introduction
- Demo
- Key differences

Stateless Components

Introduction

- A component that has **no internal state management** in it
- Can be **written as functions that just return the JSX** element
- Simple **functional components without having a local state**
- Usually associated with how a concept is presented to the user
- Properties are passed like regular parameters. The props are displayed like **{props.name}**
- Hook in react is available to add state behavior in functional component

Stateless Components

Demo

- Sample code

```
function Demo(props) {
  return <h1> Welcome to REACT JS, {props.Name} </h1>;  }
```

- Code to render such components remains the same.

```
ReactDOM.render(<Demo/>, destination )
```

Stateless Components

Key differences - Stateless and Stateful components

- **Stateless component**
  - Simple functional component
  - It takes an input (props) and returns the output (react element)
  - No internal state management in it
  - Also known as Presentational components/Dump components
- **Stateful component**
  - Always a class component
  - It is created by extending the React.Component class.
  - Dependent on it's state object and can change it's own state. The component re-renders based on changes to it's state
  - Also known as Smart or container Components

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 9886703973

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – refs and keys

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

refs and keys

Agenda

- Introduction to refs
- Create and use refs
- Callback refs
- Introduction to keys
- Keys - Sample code
- Map function
- Using map with key property

refs and keys

Introduction to refs

- Provides a **way to access DOM nodes or React elements created in the render method**
- Used to return a reference to the element
- Good use cases for refs to be used:
  - Managing focus, text selection, or media playback
  - Triggering imperative animations
  - Integrating with third-party DOM libraries

refs and keys

How to create and use refs?

- **Creating refs:**
  - **React.createRef()** method is used to attach React elements via the **ref attribute**.
  - Refs are 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} />;
  }
}
```

- **Accessing refs:**
  - 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;
```

refs and keys

Callback refs

- Gives more fine-grain control over when refs are set and unset
- The function receives the React component instance or HTML DOM element as its argument, which can be stored and accessed elsewhere.
- Coding example

```
class CustomTextInput extends React.Component {
  constructor(props) {
    super(props);

    this.textInput = null;

    this.setTextInputRef = element => {
      this.textInput = element;
    };

    this.focusTextInput = () => {
      // Focus the text input using the raw DOM API
      if (this.textInput) this.textInput.focus();
    };
  }

  componentDidMount() {
    // autofocus the input on mount
    this.focusTextInput();
  }
```

```
render() {
  // Use the `ref` callback to store a reference to the text input DOM
  // element in an instance field (for example, this.textInput).
  return (
    <div>
      <input
        type="text"
        ref={this.setTextInputRef}
      />
      <input
        type="button"
        value="Focus the text input"
        onClick={this.focusTextInput}
      />
    </div>
  );
```

refs and keys

Introduction to keys

- Utilized to identify **specific virtual DOM elements** that have changed, added, or removed.
- Keys should be given to the elements inside the array to give the elements a stable identity
- **Keys only make sense in the context of the surrounding array**
- Not specifying the key property will display a warning on the console:

Warning: Each child in an array or iterator should have a unique "key" prop.

refs and keys

Keys - Sample codes

- When returning a list of elements

```
function Stuff () {
    return (
    [
      <p>Batman</p>,
      <p>Ironman</p>,
      <p>Spiderman</p>
    ]
    );
};
```

- **Specify key property** as follows

```
function Stuff () {
    return (
    [
      <p key="1">Batman</p>,
      <p key="2">Ironman</p>,
      <p key="3">Spiderman</p>
    ]
    );
};
```

refs and keys

Map function

- Map is a data collection type where data is stored in the form of key-value pairs
- The value stored in the map must be mapped to the key
- The map is a JavaScript function that can be called on any array
- With the map function, we map every element of the array to the custom components in a single line of code
- Sample code:

```
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map((number) => number * 2);

console.log(doubled);
```

refs and keys

Using map with key property

- The code will **result in error**, as the list of items does not have the key property

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

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root') );
```

- Modified by **adding key property**

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
            <li key={number.toString()}> {number}
            </li> );
            return ( <ul>{listItems}</ul> );
          }
const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />
  document.getElementById('root') );
```

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 9886703973

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Event Handling

Prof. Sindhu R Pai
Department of Computer Science and Engineering

ReactJS - Event Handling

Agenda

- Introduction
- Synthetic Event objects
- Requirement
- Event Registration
- Demo
- Programming assignments

ReactJS - Event Handling

Introduction

- Events make the **web app interactive and responsive** to the user.
- React event handling system is known as **Synthetic Events.**
- Almost same as the DOM events. A few differences in the syntax
  - With JSX in **ReactJs, you pass a function as event handler** and in **DOM element we pass function as string**

```
// event handling in ReactJs element
<input id="inp" name="name" onChange={onChangeName} />
```

```
// event handling in DOM element
<input id="inp" name="name" onchange="onChangeName()"/>
```

  - In **DOM elements the event name is in lowercase** while in **ReactJs it is in camelCase**.
  - In ReactJs, **you cannot prevent default behavior by returning false from the event,** need to explicitly call preventDefault of the event.

ReactJS - Event Handling

Synthetic Event objects

- The event object passed to the event handlers are **SyntheticEvent object**
- **A wrapper** around the DOMEvent object
- The event handlers are registered at the time of rendering
- Returning false does not prevent the default browser, use e.preventDefault() or e.stopPropogation() accordingly
- **Properties**
  - boolean bubbles
  - DOMEventTarget currentTarget
  - number eventPhase
  - DOMEvent nativeEvent
  - boolean cancelable
  - boolean defaultPrevented
  - boolean isTrusted
  - string type

ReactJS - Event Handling

Requirement

- Clicking on the + button, the value of our counter must be incremented by one
- Involves:
  - Listen for the click event on the button
  - Implement the event handler where we react to the click and increase the value of **this.state.count** that counter relies on

ReactJS - Event Handling

Event Registration

- In React, you listen to an event by specifying everything inline in your JSX itself
- Specify both the **event** you are listening for and the **event handler** that will get called **inside the markup**

ReactJS - Event Handling

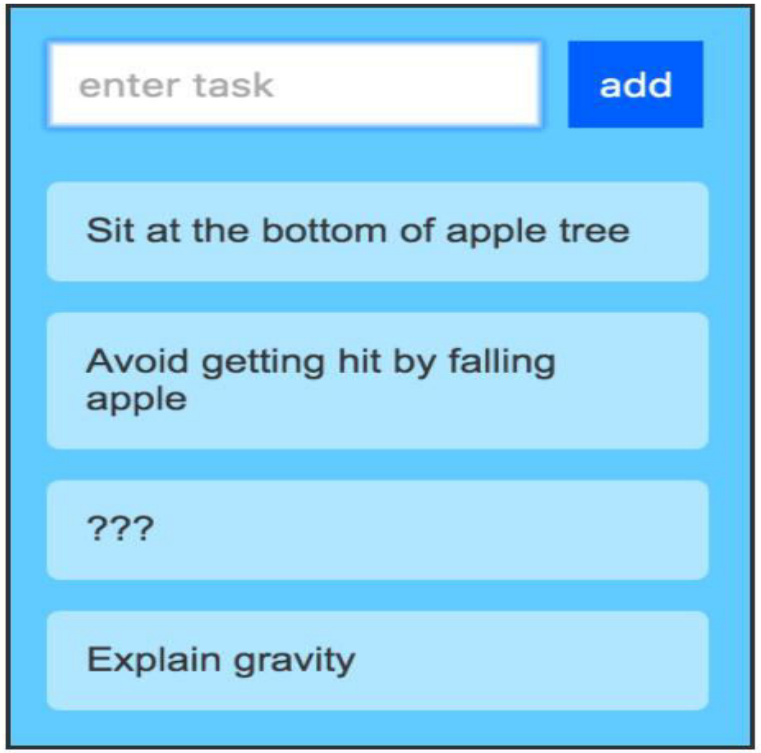
Demo

- Demo of event handling

ReactJS - Event Handling

Programming assignment

- Build an awesome todo list as shown.



THANK YOU

Sindhu R Pai
Department of Computer Science and Engineering

sindhurpai@pes.edu
+91 8277606459

www.kirupa.com

WEB TECHNOLOGIES

React JS – Form Handling

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

ReactJS - Form Handling

Agenda

- Introduction
- Uncontrolled Components
- Controlled components
- Sample codes
- Handling multiple inputs

ReactJS - Form Handling

Introduction

- Deals with **how to handle the data when the input values are changed or when the form is submitted**
- Control these changes by adding **event handlers** to **onChange** and **onSubmit** respectively
- Two different ways
  - **Uncontrolled components :** Data handled by DOM
  - **Controlled components:** Data handled by Components
    - When the data is handled by the components, all the data is stored in the **component state**
    - Form data is usually handled by the components.

ReactJS - Form Handling

Uncontrolled components

- Use a ref to get form values from the DOM
- Use the defaultValue property to specify initial value in React
- <input **defaultValue="Bob"** type="text" **ref={this.input}** />

ReactJS - Form Handling

Controlled components

- Has two aspects:
  - Have functions to govern the data going into them on every **onChange event,** rather than grabbing the data only once
    - Examples: When a user clicks a submit button. This 'governed' data is then saved to state
  - Data displayed by a controlled component is received through **props** passed down from it's parent/container component.
- Value attribute is set on form element, the displayed value will always be **this.state.value**, making the React state the **source of truth.**
- handleChange runs on every keystroke to update the React state, **the displayed value will update as the user types.**

ReactJS - Form Handling

Sample codes related to Controlled components

- text inputs, number inputs, radio inputs, checkbox inputs, text areas, selects

- Sample codes:
  - <textarea> Default Value </textarea> can be changed to

    <textarea value={this.state.value} />

  - <select>

    <option select value = “optionselected”> Default Value </option>

    </select> can be changed to

    <select value = {this.state.value}>

    <option value = “optionselected”>Default Value</option>

    </select>

ReactJS - Form Handling

Handling Multiple inputs

- First case: Single input

```
<label>
  First name
  <input
    type="text"
    onChange={handleChange}
  />
</label>
```

```
function handleChange(evt) {
  console.log("new value", evt.target.value);
}
```

```
import React from "react";
function Form() {
  const [state, setState] = React.useState({
    firstName: ""
  })
  return (
    <form>
      <label>
        First name
        <input
          type="text"
          value={state.firstName}
          onChange={handleChange}
        />
      </label>
    </form>
  );
}
```

```
function handleChange(evt) {
  setState({ firstName: evt.target.value });
}
```

ReactJS - Form Handling

Handling Multiple inputs contd

- Second case: Multiple input

```
function Form() {
  const [state, setState] = React.useState({
    firstName: "",
    lastName: ""
  })
  return (
    <form>
      <label>
        First name
        <input
          type="text"
          name="firstName"
          value={state.firstName}
          onChange={handleChange}
        />
      </label>
      <label>
        Last name
        <input
          type="text"
          name="lastName"
          value={state.lastName}
          onChange={handleChange}
        />
      </label>
    </form>
  );
}
```

```
function handleChange(evt) {
  const value = evt.target.value;
  setState({
    ...state,
    [evt.target.name]: value
  });
}
```

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 80 2672 6622

sindhurpai@pes.edu
+91 8277606459

WEB TECHNOLOGIES

React JS – Form Handling

Prof. Vinay Joshi and Prof. Sindhu R Pai
Department of Computer Science and Engineering

ReactJS - Form Handling

Agenda

- Introduction
- Uncontrolled Components
- Controlled components
- Sample codes
- Handling multiple inputs

ReactJS - Form Handling

Introduction

- Deals with **how to handle the data when the input values are changed or when the form is submitted**
- Control these changes by adding **event handlers** to **onChange** and **onSubmit** respectively
- Two different ways
  - **Uncontrolled components :** Data handled by DOM
  - **Controlled components:** Data handled by Components
    - When the data is handled by the components, all the data is stored in the **component state**
    - Form data is usually handled by the components.

ReactJS - Form Handling

Uncontrolled components

- Use a ref to get form values from the DOM
- Use the defaultValue property to specify initial value in React
- <input **defaultValue="Bob"** type="text" **ref={this.input}** />

ReactJS - Form Handling

Controlled components

- Has two aspects:
  - Have functions to govern the data going into them on every **onChange event,** rather than grabbing the data only once
    - Examples: When a user clicks a submit button. This 'governed' data is then saved to state
  - Data displayed by a controlled component is received through **props** passed down from it's parent/container component.
- Value attribute is set on form element, the displayed value will always be **this.state.value**, making the React state the **source of truth.**
- handleChange runs on every keystroke to update the React state, **the displayed value will update as the user types.**

ReactJS - Form Handling

Sample codes related to Controlled components

- text inputs, number inputs, radio inputs, checkbox inputs, text areas, selects

- Sample codes:
  - <textarea> Default Value </textarea> can be changed to

    <textarea value={this.state.value} />

  - <select>

    <option select value = “optionselected”> Default Value </option>

    </select> can be changed to

    <select value = {this.state.value}>

    <option value = “optionselected”>Default Value</option>

    </select>

ReactJS - Form Handling

Handling Multiple inputs

- First case: Single input

```
<label>
  First name
  <input
    type="text"
    onChange={handleChange}
  />
</label>
```

```
function handleChange(evt) {
  console.log("new value", evt.target.value);
}
```

```
import React from "react";
function Form() {
  const [state, setState] = React.useState({
    firstName: ""
  })
  return (
    <form>
      <label>
        First name
        <input
          type="text"
          value={state.firstName}
          onChange={handleChange}
        />
      </label>
    </form>
  );
}
```

```
function handleChange(evt) {
  setState({ firstName: evt.target.value });
}
```

ReactJS - Form Handling

Handling Multiple inputs contd

- Second case: Multiple input

```
function Form() {
  const [state, setState] = React.useState({
    firstName: "",
    lastName: ""
  })
  return (
    <form>
      <label>
        First name
        <input
          type="text"
          name="firstName"
          value={state.firstName}
          onChange={handleChange}
        />
      </label>
      <label>
        Last name
        <input
          type="text"
          name="lastName"
          value={state.lastName}
          onChange={handleChange}
        />
      </label>
    </form>
  );
}
```

```
function handleChange(evt) {
  const value = evt.target.value;
  setState({
    ...state,
    [evt.target.name]: value
  });
}
```

THANK YOU

Vinay Joshi and Sindhu R Pai
Department of Computer Science and Engineering

vinayj@pes.edu
+91 80 2672 6622

sindhurpai@pes.edu
+91 8277606459