



React Components

SENG 4640

Software Engineering for Web Apps

Winter 2023

Sina Keshvadi

Thompson Rivers University

Review

- React allows us to create custom components and insert them into the VirtualDOM in our HTML page
- This allows for selective rendering and modular development of dynamic behavior

React Components

- Components are JavaScript objects based off the **React.Component** prototype
- Components define properties, event-based state variables, and callback functions
- A component's **render ()** function is used to render its HTML
- VirtualDOM manages each component's lifecycle and calls its **render ()** function as needed

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.render(
    <HelloReact />,
    document.getElementById("divHello")
  )
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```


Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);
</script>
```

Creating Custom Components – ES6

- ES6 is a more recent version of JavaScript syntax
- We can define a **class** instead of a single object

```
<div id="container"></div>
<script type="text/babel">
  class HelloReact extends React.Component {
    render() {
      return (
        <h1>Hello React!</h1>
      );
    }
  }
  ReactDOM.
  createRoot(document.getElementById("container")).
  render(<HelloReact />);

</script>
```

React Component Attributes

- **Properties**

- Attributes and values that are set when the component is created
- Should never be modified after initialization

React Component Attributes

- **Properties**

- Attributes and values that are set when the component is created
- Should never be modified after initialization

- **State**

- Attributes and values that represent the current state of the component, based on what it does/represents
- Can be modified during the component's lifecycle

React Component Attributes

- **Properties**

- Attributes and values that are set when the component is created
- Should never be modified after initialization

- **State**

- Attributes and values that represent the current state of the component, based on what it does/represents
- Can be modified during the component's lifecycle

- Both properties and state can be used when rendering the component

Component Properties

- Should always be assigned upon object creation, never modified afterward
- Component accesses its properties through **this.props**

Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```


Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```

Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```

Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```

Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```

Component Properties

- Component accesses its properties through **this.props**

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    class HelloUser extends React.Component{
      render(){
        return(
          <h1>Hello {this.props.name}!</h1>
        );
      }
    }

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<HelloUser name="Maria" />);

  </script>
</body>
```

Component State

- The set of variables that can change during the component's lifecycle
- Should be initialized in the **constructor**
- Component accesses its state through **this.state**

Page Visit Counter using React

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```



```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```



```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

```
<body>
  Visited <span id="container"></span>
  <script type="text/babel">
    class TimesVisited extends React.Component {
      constructor (props){
        super (props);
        var timeViewed = 0;
        if(localStorage.timeViewed){
          timeViewed = localStorage.timeViewed;
        }
        timeViewed ++;
        this.state = {numViews:timeViewed};
        localStorage.timeViewed = timeViewed;
      }

      render (){
        return <b>{this.state.numViews}</b>;
      }
    };

    ReactDOM.
    createRoot(document.getElementById("container")).
    render(<TimesVisited />);

  </script>
</body>
```

Component Lifecycle

- The React VirtualDOM invokes callback functions on components during their lifecycle
- These functions fall into three categories:
 - Mounting
 - Updating
 - Unmounting
- You can optionally implement these for controlling the component

Component Lifecycle: Mounting

- Called when a component is being created and added to the VirtualDOM
- **constructor:** creates component, initializes state based on properties
- **componentWillMount:** invoked before component is added to VirtualDOM
- **componentDidMount:** invoked after component has been added to VirtualDOM and has been rendered

Component Lifecycle: Updating

- Called when a component's props or state is changing and the component is re-rendered
- **componentWillReceiveProps:** invoked before receiving new props, e.g. when its parent component re-renders
- **shouldComponentUpdate:** can be used to determine whether to re-render
- **componentWillUpdate:** invoked before re-rendering after change to state
- **componentDidUpdate:** invoked after being re-rendered

Component Lifecycle: Unmounting

- Called when a component is being removed from the VirtualDOM
- **componentWillUnmount**: invoked before component is removed from VirtualDOM and destroyed

Summary

- React components are JavaScript objects that can be used as HTML elements in the VirtualDOM
- A component's **render ()** function is used to render its HTML
- A component's **properties** are assigned when it is created
- A component's **state** can change during its lifecycle
- A component's **lifecycle functions** are invoked depending on relevant activities