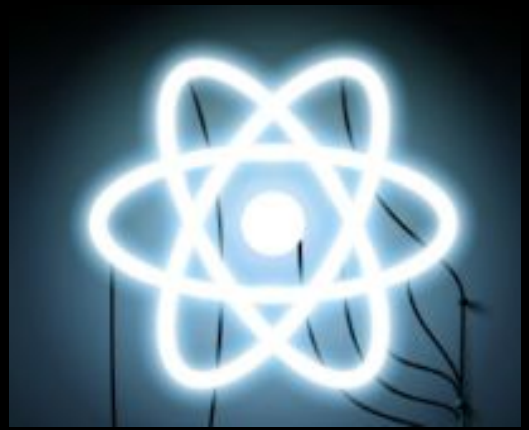
Session 2.1

Intro to React Components



Topics

- Components
  - Functional vs Class
- Props

Video - Recap VDOM

https://youtu.be/pKYiKbf7sP0

Components

React Components - UI Tree

Single view of UI is divided into logical pieces. The starting component becomes the root and the rest of the components become branches and sub-branches.

React Components

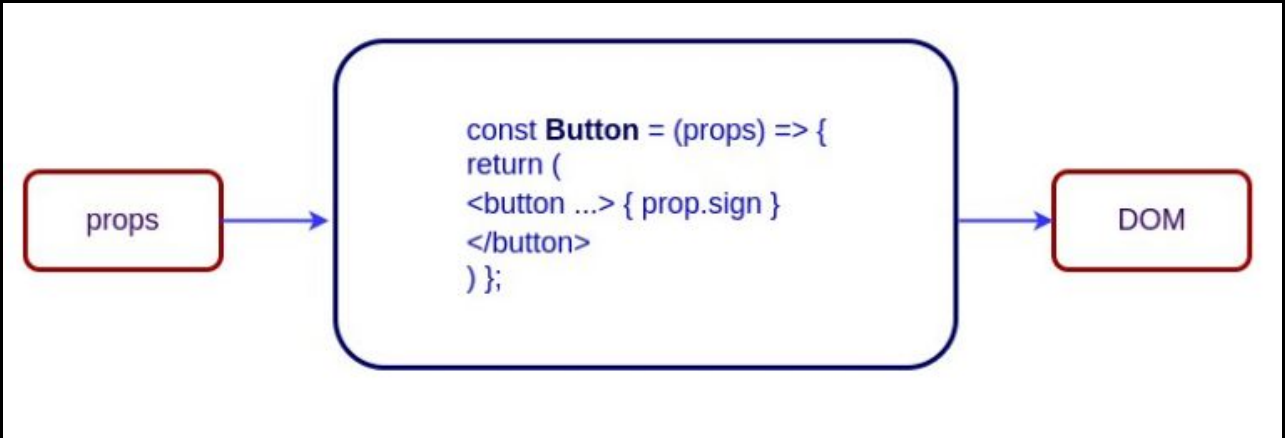
- React has a component-based architecture.
- Every component is required to have a render method. A component will be rendered on the DOM
- ReactDOM.render method renders the component on the DOM.
- ReactDOM.render uses the component name and DOM node id as arguments.
- The "HTML" that is rendered isn't actually HTML, but JSX
- React takes these JS object and will form a "virtual DOM"

```
<!-- the element to be rendered -->
        <div id="root"></div>

    <script type="text/babel">
        ReactDOM.render( <h1>Hello, world!</h1>, document.getElementById('root') );</script>
    </body>
```

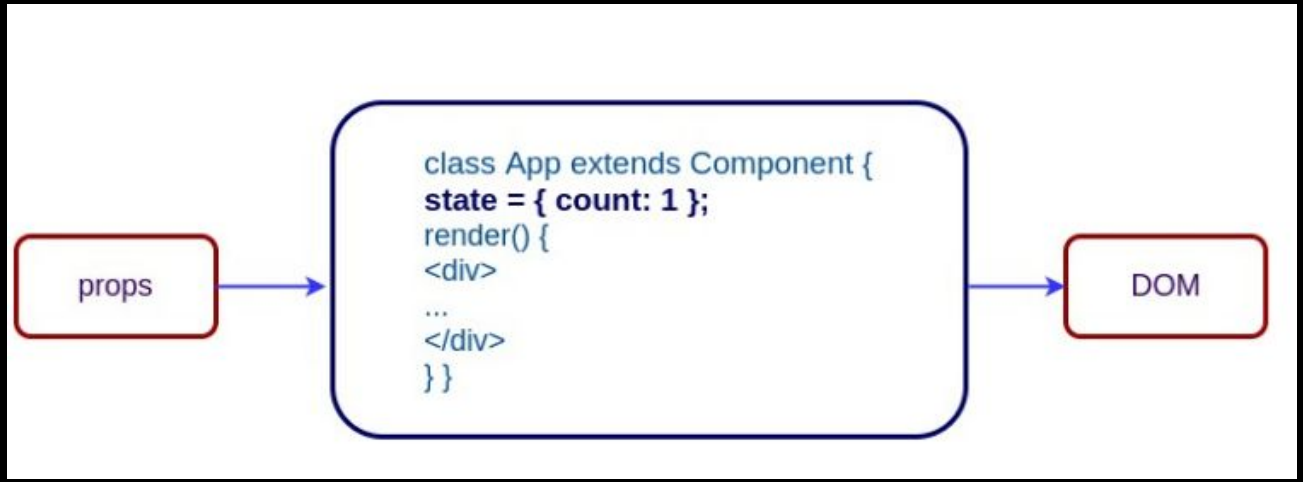
Functional Components

- A React component can be two types: either a class component or a functional component.
- Functional components are just JavaScript functions. They take an optional input named props.
- Functional components are considered stateless components.



Class Components

- Class components offer more features and with more features comes more baggage.
- The primary reason to choose class components over functional components is that can have state.
- Class components are considered "stateful" components.



Class Components cont..

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

    render() {
        return(
            <div>
                Hello {props}
            </div>
        )
    }
}
```

- There are two ways to create a class component:
  - traditional way using React.createClass()
  - ES6 syntax to extend React.Component
- Class components can exist without state too.
- Best practices is class components should always call the base constructor with super() and props.
- The constructor is optional, if there is no state.
- However when using a construtor, super() must be used and is not optional

Function and Class Components

- The simplest way to define a component is to write a JavaScript function
- This function is a valid React component because it accepts a single "props" object argument with data and returns a React element.
- You can also use an ES6 class to define a component
- These two components are equivalent from React's point of view.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Stateful Components

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

```
class UserListContainer extends React.Component {
  constructor() {
    super()
    this.state = { users: [] }
  }
```

- There is another popular way of classifying components. The criteria is simple, the component has state or it doesn't.
- Stateful components are alway class components.
- Stateful components have a state that gets initialized in the constructor.
- The this keyword refers to the instance of the current component.

Stateless Components

- You can use either function or class for creating stateless components.
- Unless you need a lifecycle hook in your components, you should go for stateless functional components.
- There is a a lot of benefits to use stateless functional components
  - easy to write and understand
  - easy to test
  - avoid the this keyword altogether
- However, there are no performance benefits to using stateless functional component over class components

```
const HelloWorld = ({name}) => {
  const sayHi = (event) => {
    alert(`Hi ${name}`);
  };
```

Rendering a Component

- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Video - React Dev Tools

https://youtu.be/TRPfZ4INN9w

Props

Props

- The components are very useful, if all do is render the same thing.
- React allows props (properties) to be passed to components with a syntax similar to HTML attributes.
- Props help components converse

Props can passed in through JSX

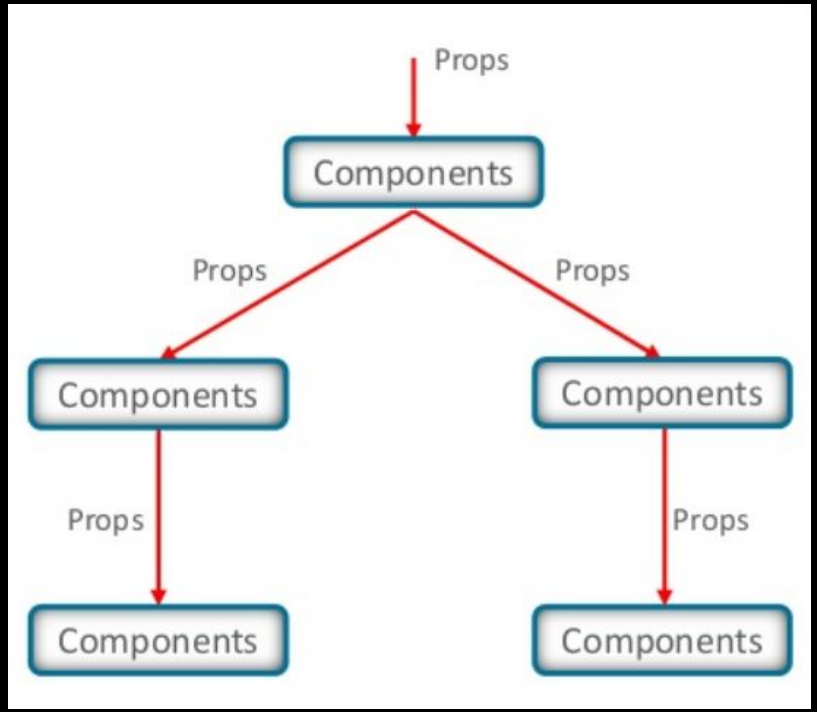
```
<Greeter name="Al Pacino" />
```

Props can be accessed using this.props

```
class Greeter extends React.Component {
  render() {
    const {name} = this.props;
    return (
      <div>Hello {name}!</div>
    );
  }
}
```

Props Data Flow

1. Work similar to HTML attributes
2. Data flows downwards from the parent component
3. Uni-directional data flow
4. Props are immutable i.e pure
5. Can set default props



Conditional Default Props

- If the props.value is not passed in JSX, the component will be rendered without a value.
- The props.value in the component will be undefined.
- Default Props for the component can be set to use a default value for when the prop is not passed.

```
class Greeter extends React.Component {
  render() {
    return (
      <div>Hello {this.props || 'World'}!</div>
    );
  }
}
```

Defaulting with defaultProps

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

// Specifies the default values for props:
Greeting.defaultProps = {
  name: 'Stranger'
};

// Renders "Hello, Stranger":
ReactDOM.render(
  <Greeting />,
  document.getElementById('example')
);
```

- Functional and class components can set default properties
- Add a static property named defaultProps to the component itself to default the prop values.

Type checking with propTypes

```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

- As you application grows, you can catch a lot of bugs with typechecking.
- JavaScript extensions like Flow or TypeScript can be used to typecheck your whole application.
- React has some-built in typechecking abilities. To run typechecking on the props for a component, you can assign the special propTypes property.

Validation with propTypes

```
import PropTypes from 'prop-types';

MyComponent.propTypes = {
  // You can declare that a prop is a specific JS type.
  // are all optional.
  optionalArray: PropTypes.array,
  optionalBool: PropTypes.bool,
  optionalFunc: PropTypes.func,
  optionalNumber: PropTypes.number,
  optionalObject: PropTypes.object,
  optionalString: PropTypes.string,
  optionalSymbol: PropTypes.symbol,
```

- PropTypes exports a range of validators that can be used to make sure the data you recieve is valid.
- When an invalid value is provided for a prop, a warning will be shown in the JavaScript console.

```
// You can chain any of the above with `isRequired` to make sure a warning
// is shown if the prop isn't provided.
requiredFunc: PropTypes.func.isRequired,

// A value of any data type
requiredAny: PropTypes.any.isRequired,
```

JavaScript Expressions as Props

- You can pass any JavaScript expression as a prop, by surrounding it with {}
- For MyComponent, the value of props.foo will be 10 because the expression 1 + 2 + 3 + 4 gets evaluated.

```
<MyComponent foo={1 + 2 + 3 + 4} />
```

- if statements and for loops are not expressions in JavaScript, they can't be used in JSX directly.

```
function NumberDescriber(props) {
  let description;
  if (props.number % 2 == 0) {
    description = <strong>even</strong>;
  } else {
    description = <i>odd</i>;
  }
  return <div>{props.number} is an {description} number</div>;
}
```

Spread Attributes

- If you already have props as an object, and you want to pass it in JSX, you can use ... as a “spread” operator to pass the whole props object. These two components are equivalent:

```
function App1() {
  return <Greeting firstName="Ben" lastName="Hector" />;
}

function App2() {
  const props = {firstName: 'Ben', lastName: 'Hector'};
  return <Greeting {...props} />;
}
```

Video - Props

https://youtu.be/ICmMVfKjEuo