How React works

- The React application is made of components
- React takes the components and creates a JavaScript representation of the DOM (Virtual DOM)
- React renders the DOM based on the Virtual DOM
- Whenever data changes within a component, a new Virtual DOM is rendered and compared to the current Virtual DOM
- The changes are updated in the browser

CDN

React

```
<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js">
</script>


<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
```

Babel

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

```
<script type="text/babel">
    ...
</script>
```

Components

```
<script type="text/babel"> javascript
    class App extends React.Component {
        render() {
          return (
            // jsx
          )
    }
}
```

`render()`

- Renders the template to the element

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

- Render App to #app

`{ }`

- Treat whatever is inside of `{ }` as JavaScript

State

- State of data/UI of component
- A JSON object storing local data of the component
- Changing the state of a component makes it re-render

```
state = {
  name: 'react',
  isCool: true,
}
```

```
{ this.state.name } // react
```

Events

```
handleClick(e){
  console.log(e.target);
}
```

```
<button onClick={this.handleClick}>click</button>
```

`this`

```
handleClick(e){
  console.log(this.state);

}
```

- Error, `this` is out of scope, undefined
- We must manually bind `this`
- Use an arrow function

```
handleClick = e => {
  console.log(this.state);
}
```

- Arrow functions bind `this` to whatever `this` is outside the function
- For regular functions, `this` refers to the object that called the function
- For `=>` functions, `this` is bound *lexically*, using the enclosing function scope as its `this` value

Changing State

```
this.state.name = 'cool' // no!
```

- Do not change the state directly

`setState`

```
this.setState({
    name: 'epic',
});
```

- Pass in an object representing the state
- Asynchronous

Forms

```
 <form onSubmit={this.handleSubmit}>
    <input type="text" onChange={ this.handleChange }/>
    <button>Submit</button>
</form>
```

- Remember to `e.preventDefault()` form submits

Single Page Applications

- React apps are SPAs
- Only one HTML page is served on the browser
- React controls what the user sees

Components Continued

- Components can be defined by functions or classes

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

- No state
- Receive data from props
- UI components

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

- State
- Lifecycle hooks
- Container components

Props

- Pass data from parent components to child components

```
<Component name="Jay JO" likes="biking" drift="true"></Component>
```

```
{ props.name }, { props.likes }, { props.drift }
```

Lists

Container

```
state = { // for class component
  members: [
    {},
    ...
  ]
}
```

```
class
<Component prop={this.state.members}/>
function
<Component prop={members}/>
```

Component

```
const memberList = props.members.map(member => {
    return (
        //html
    );
});

{ memberList }
```

Conditional Output

- Use the Ternary operator

```
const coolList = members.map(member => {
    return member.cool == 'true' ? (
        // html
    ) : null;
});
```

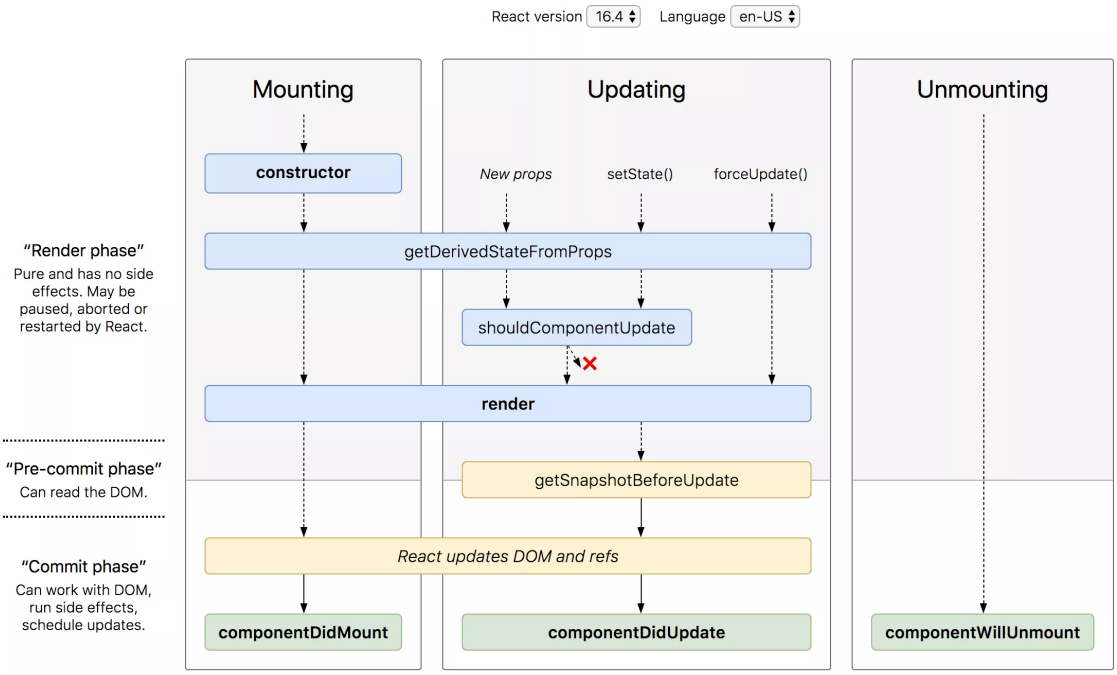
Functions as Props

- ex. accessing state of parent component
- Put a function to set state in the parent component and pass it as a prop to the child component
- `this.props.function()`

```
<button onClick={ () => {function(id)} }>x</button>
```

- Must be an arrow function or it will fire immediately

Lifecycle Methods



Mounting

- Mounting: Putting elements into the DOM
- 4 methods called in this order upon mount
- `render()` is required, others are optional

Constructor

- Called upon component instantiation
- Set up initial values such as `state`

getDerivedStateFromProps

- Called before rendering elements
- Set state based on props

render

- Renders JSX to the DOM

componentDidMount()

- Fires when component first mounts
- Good for grabbing external data (ex. Firebase)

Updating

- Updating data, self-explanatory

shouldComponentUpdate

getSnapshotBeforeUpdate

componentDidUpdate

Unmounting

componentWillUnmount

- Component is about to removed from the DOM