React

Prof. Dr. Mohamed Amine Chatti

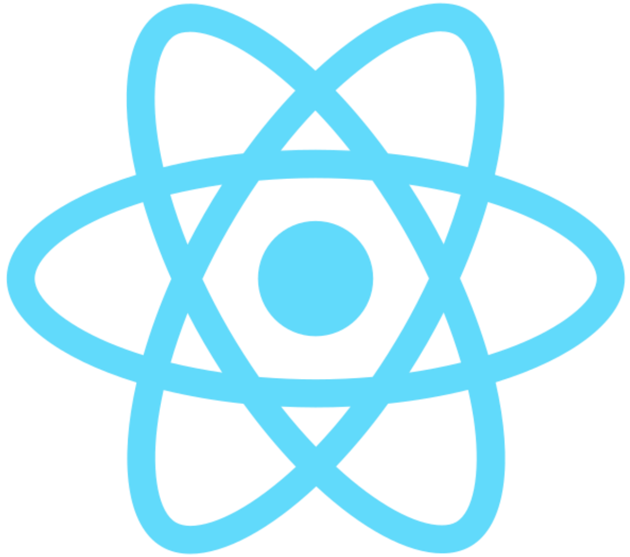
M.Sc. Shoeb Joarder

Social Computing Group, University of Duisburg Essen

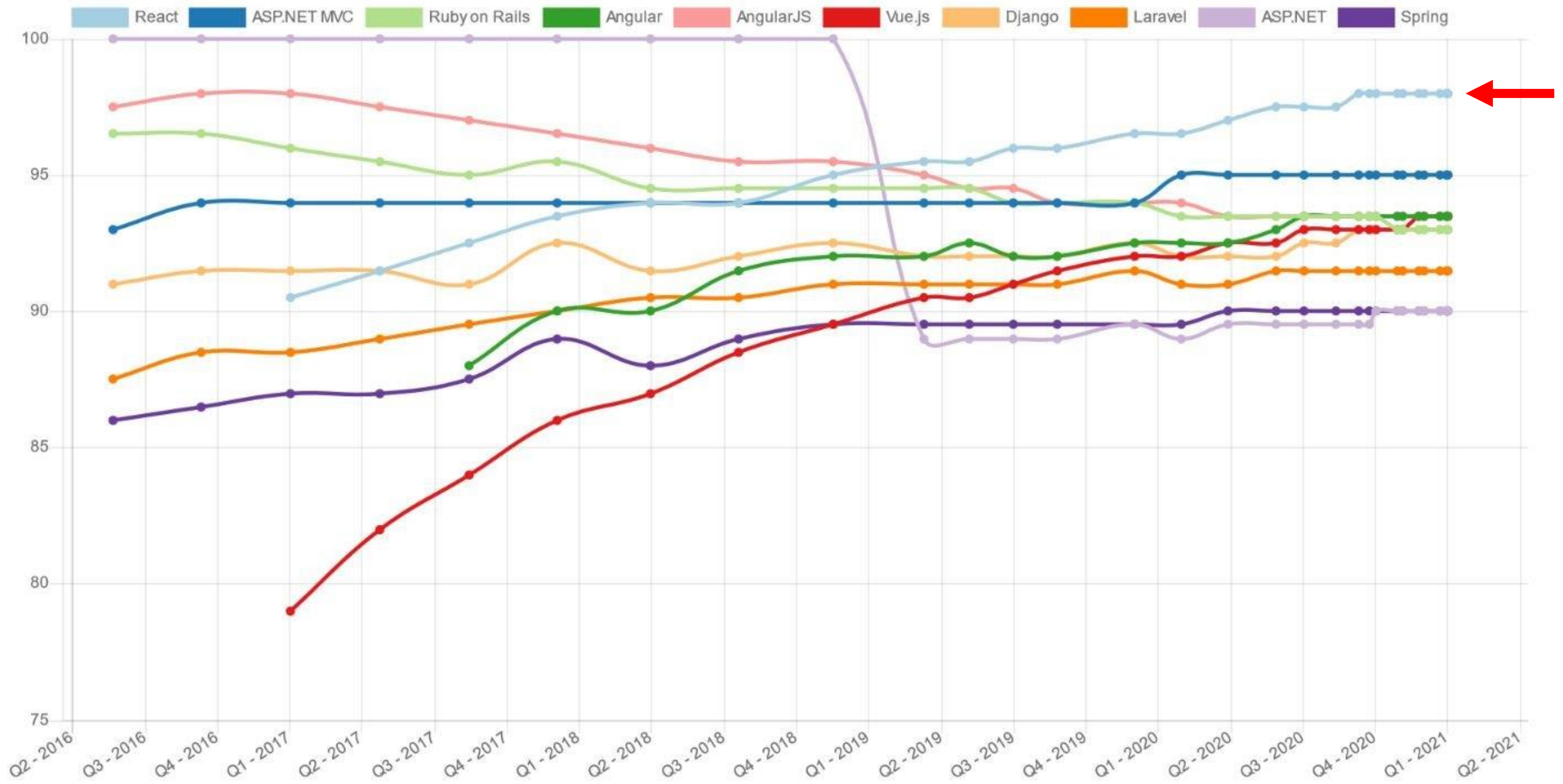
www.uni-due.de/soco

What is React?

- React is a JavaScript (JS) library
- Developed by Facebook in 2011
- Most popular front-end JS library in the industry (for now)
- Builds beautiful, fast and interactive User Interface (UI) for front-end applications



React's Popularity

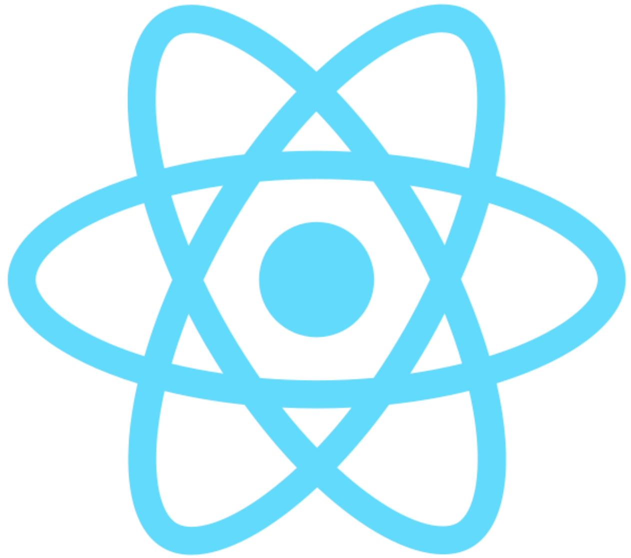


React's Popularity based on GitHub & StackOverflow scores in the year 2021

http://hotframeworks.com/

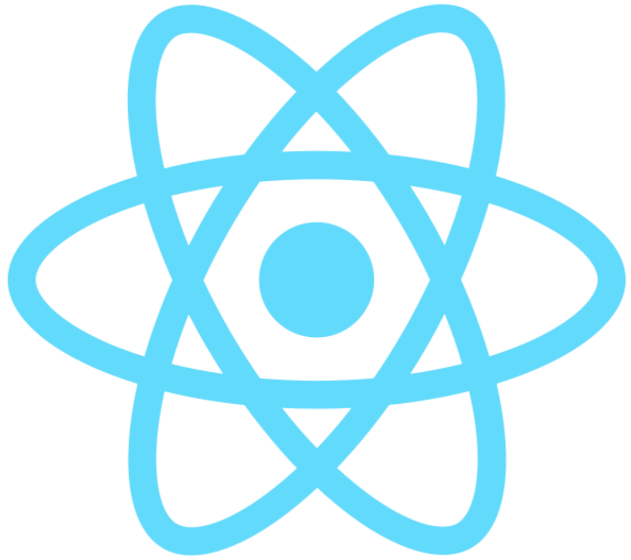
Why use React?

- Front-end development in JavaScript with ECMAScript 6 (ES6)
- React consists of
  - Independent,
  - Isolated &
  - Reusable components
- Very interactive UIs
- Virtual Document Object Model (DOM)
  - Single page web application

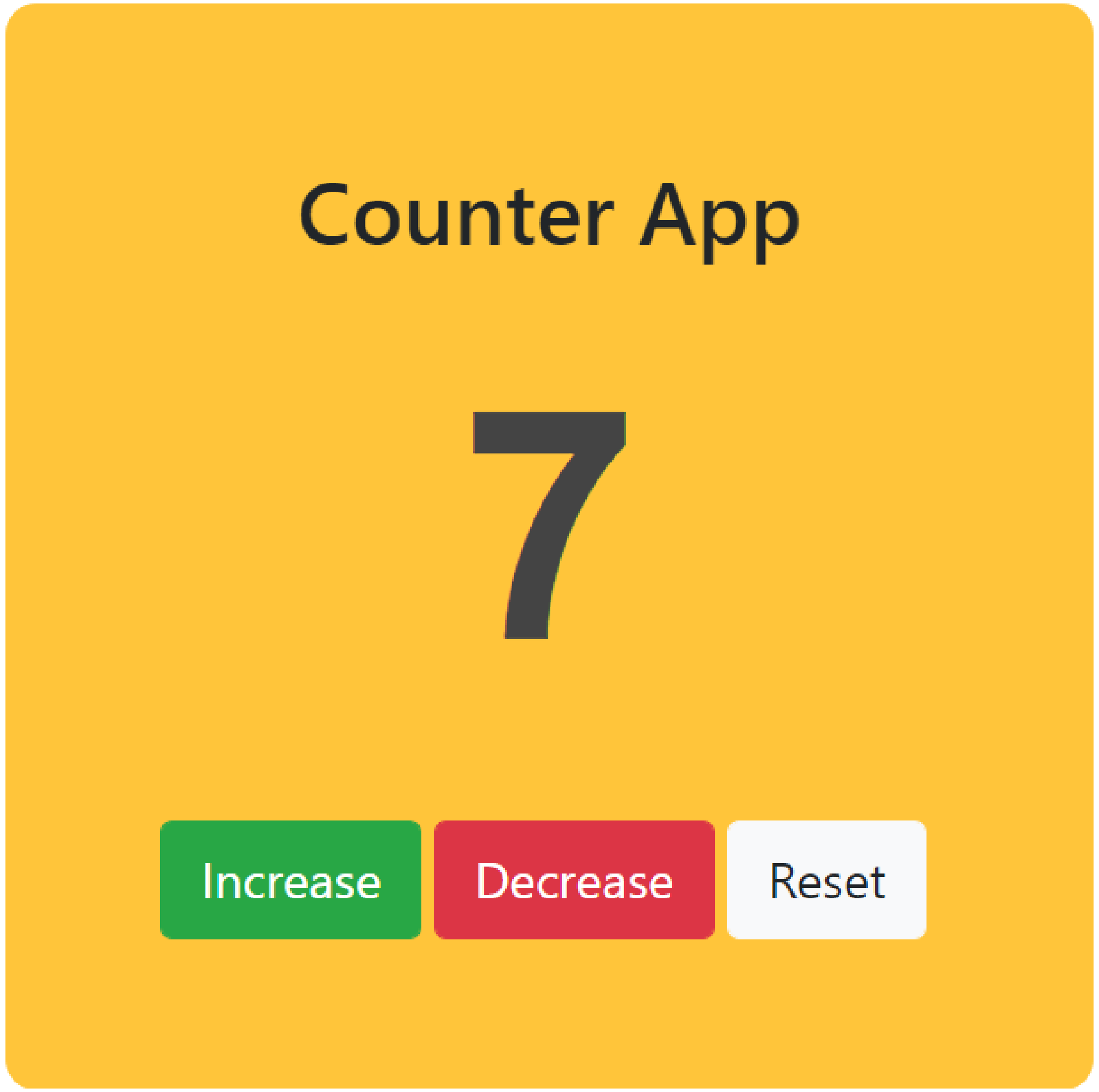


What should you know?

- Fundamentals of JavaScript
  - Objects, Arrays, Conditionals etc.
- Knowledge about HTML and CSS
- Additional knowledge necessary to learn that comes from the latest ES6 standard
  - Classes
  - Destructuring
  - Higher order array methods
    - Map, forEach, spread operator etc.
  - Arrow functions
  - Fetch API and promises



Simple Counter



Objectives

Part 1

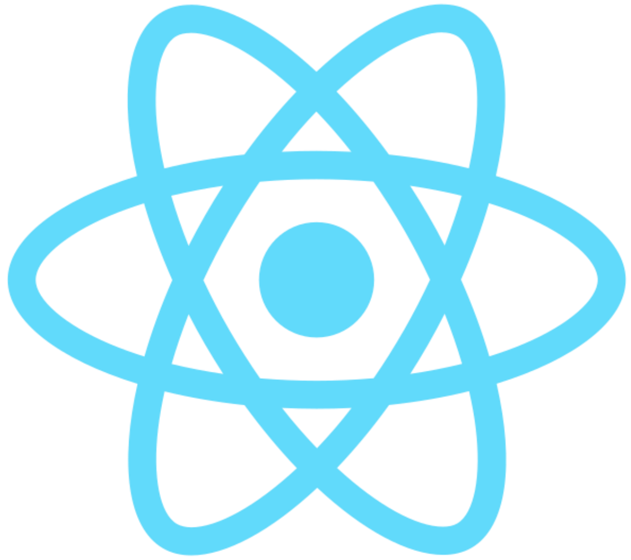
- Component
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event handlers
- Props
- Data binding

Part 2

- Component lifecycle
- React Router
- Redux
- Discussion
- Installation Guide
- Project Demo

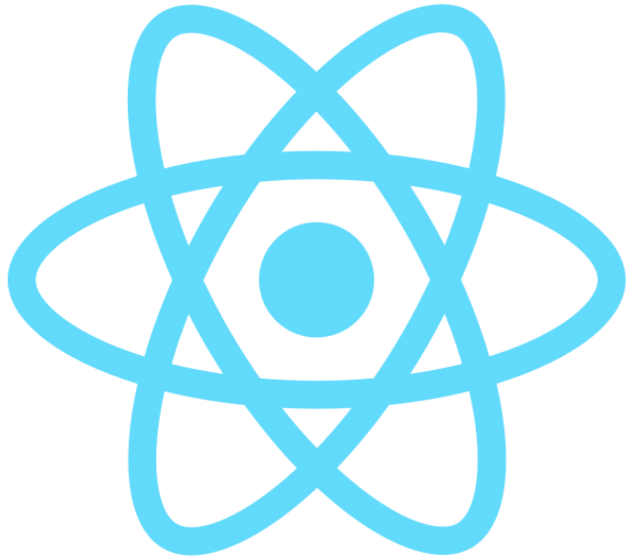
Objectives: React (Part 1)

- Component
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event handlers
- Props
- Data binding



Objectives: React (Part 1)

- **Component**
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event Handlers
- Props
- Data binding

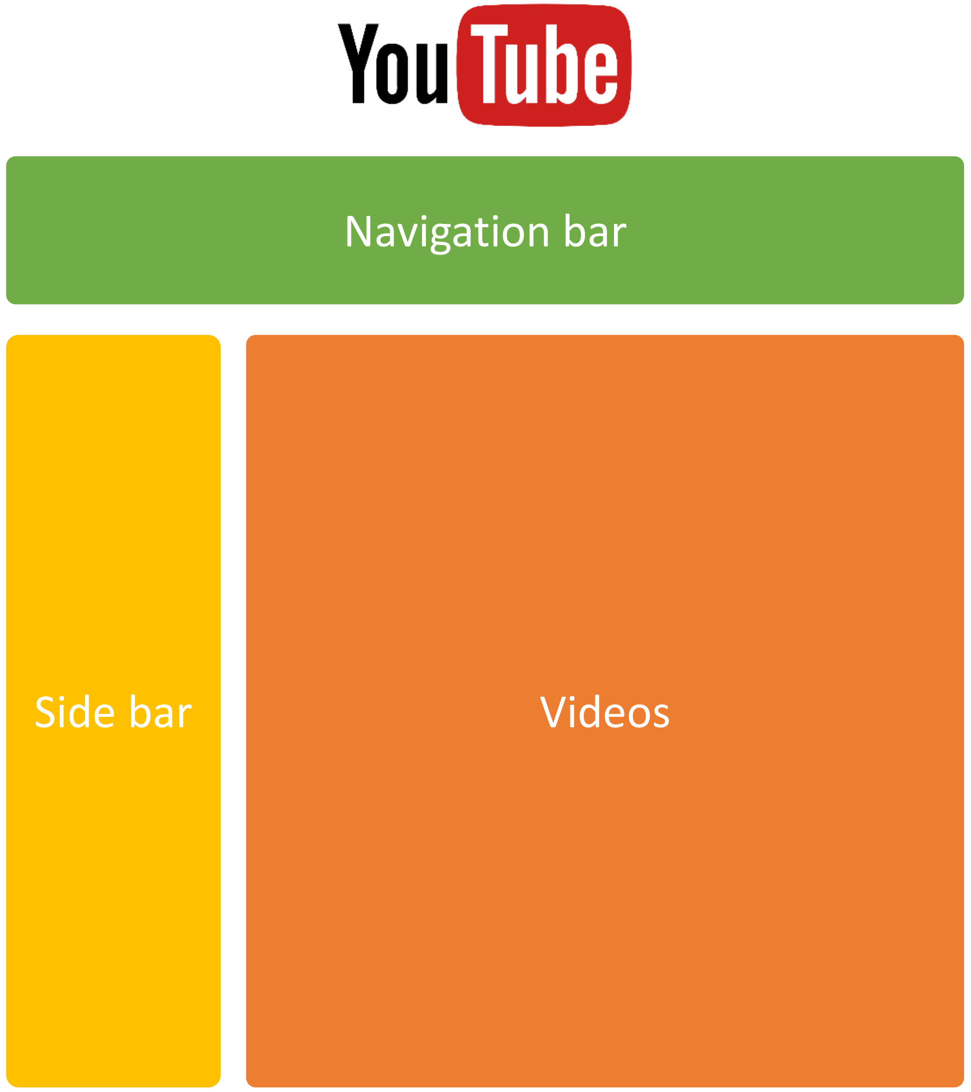


Component

- A piece of UI in React application
- Independent, isolated and reusable
- Compose multiple components together to build a complex UI

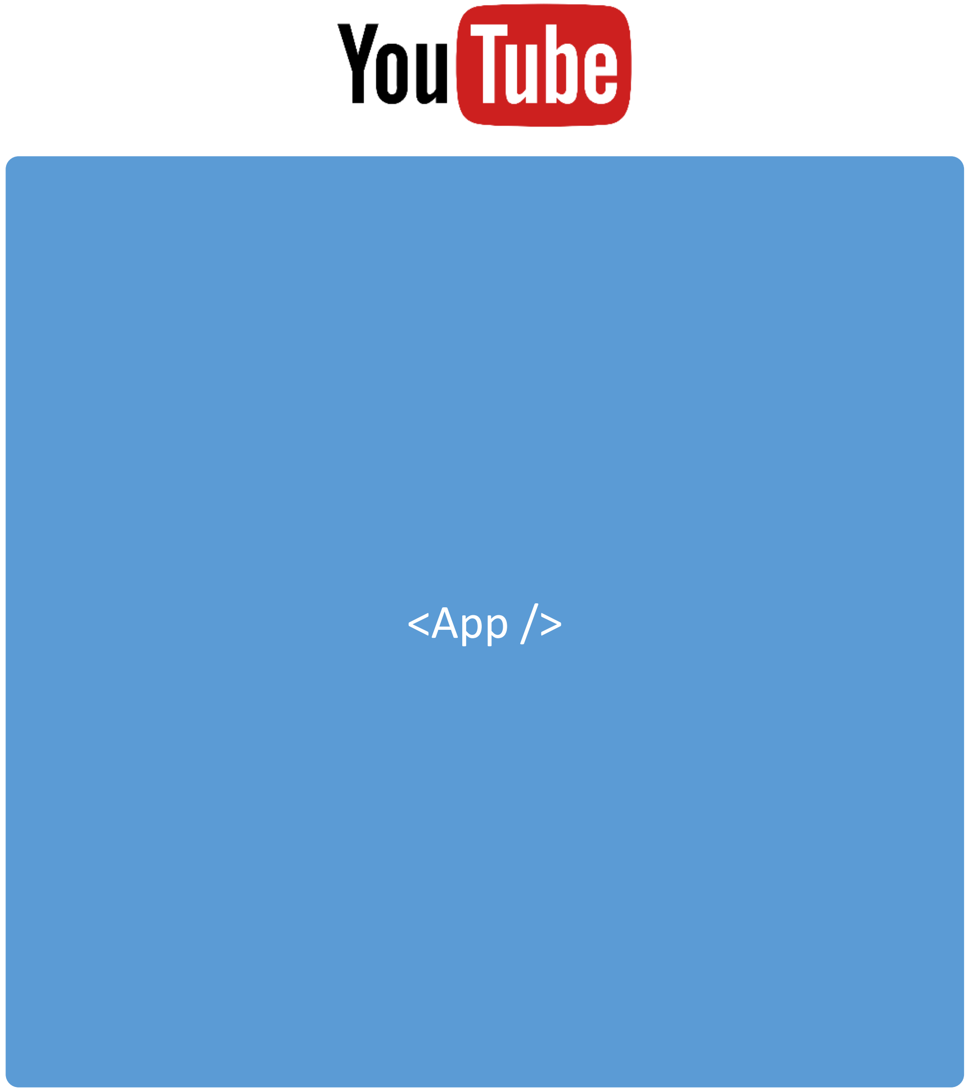
Component

- A piece of UI in React application
- Independent, isolated and reusable
- Compose multiple components together to build a complex UI
- Example: YouTube website →



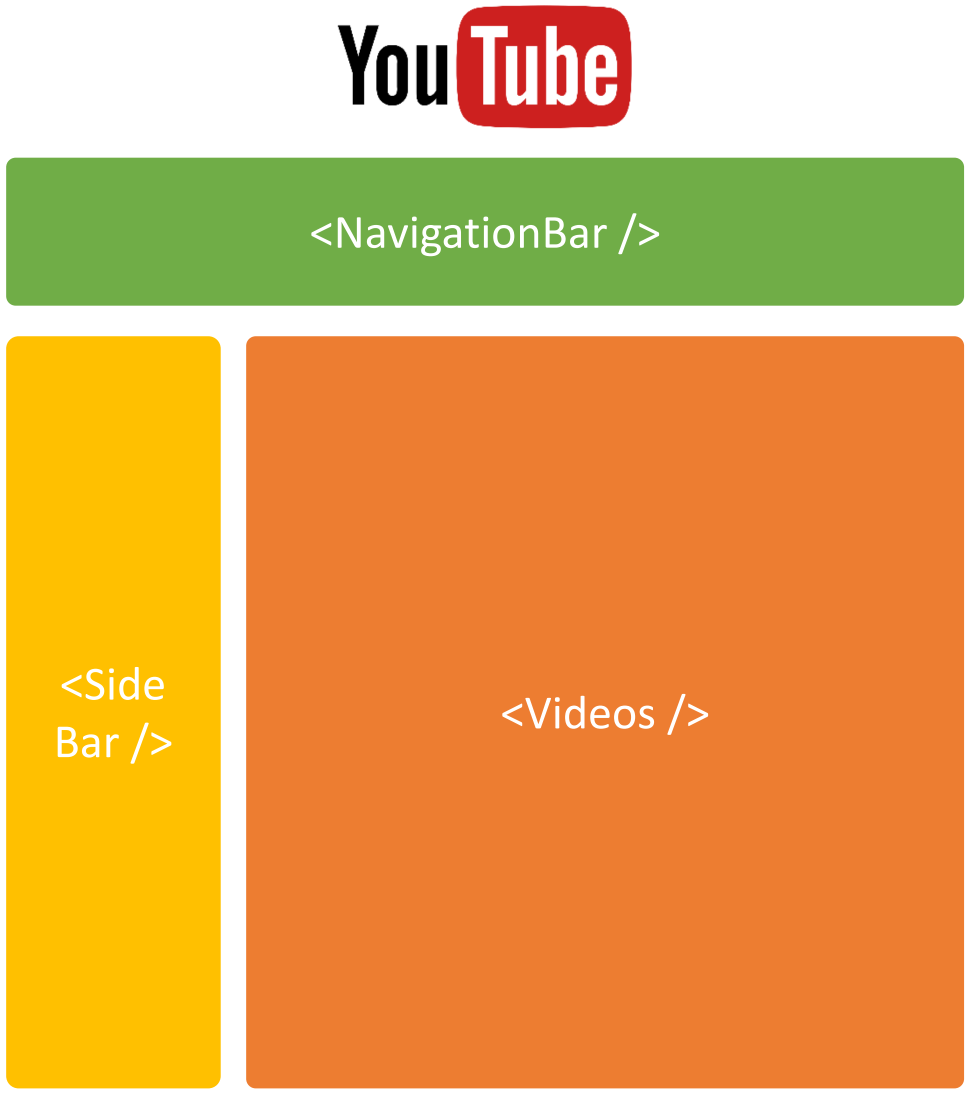
Component

- A piece of UI in React application
- Independent, isolated and reusable
- Compose multiple components together to build a complex UI
- Example: YouTube website →



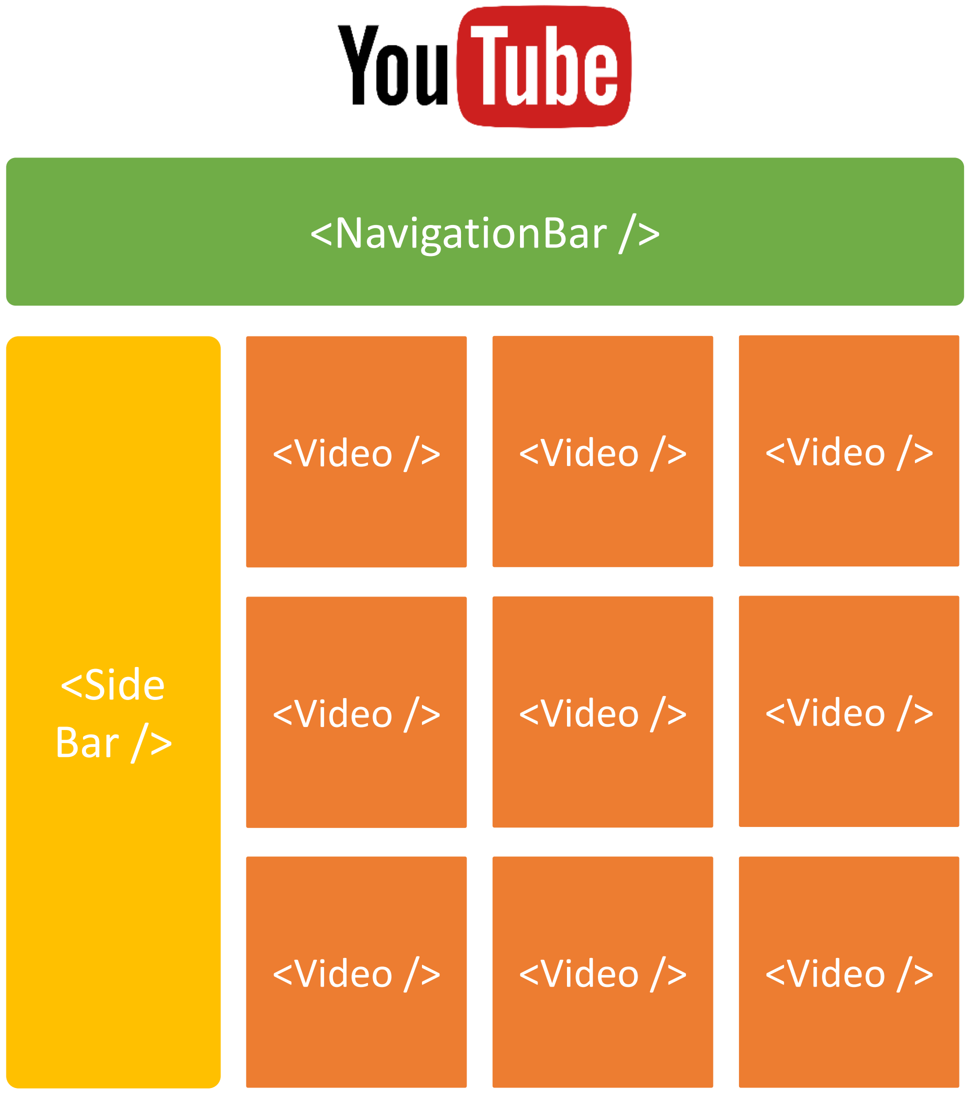
Component

- A piece of UI in React application
- Independent, isolated and reusable
- Compose multiple components together to build a complex UI
- Example: YouTube website →



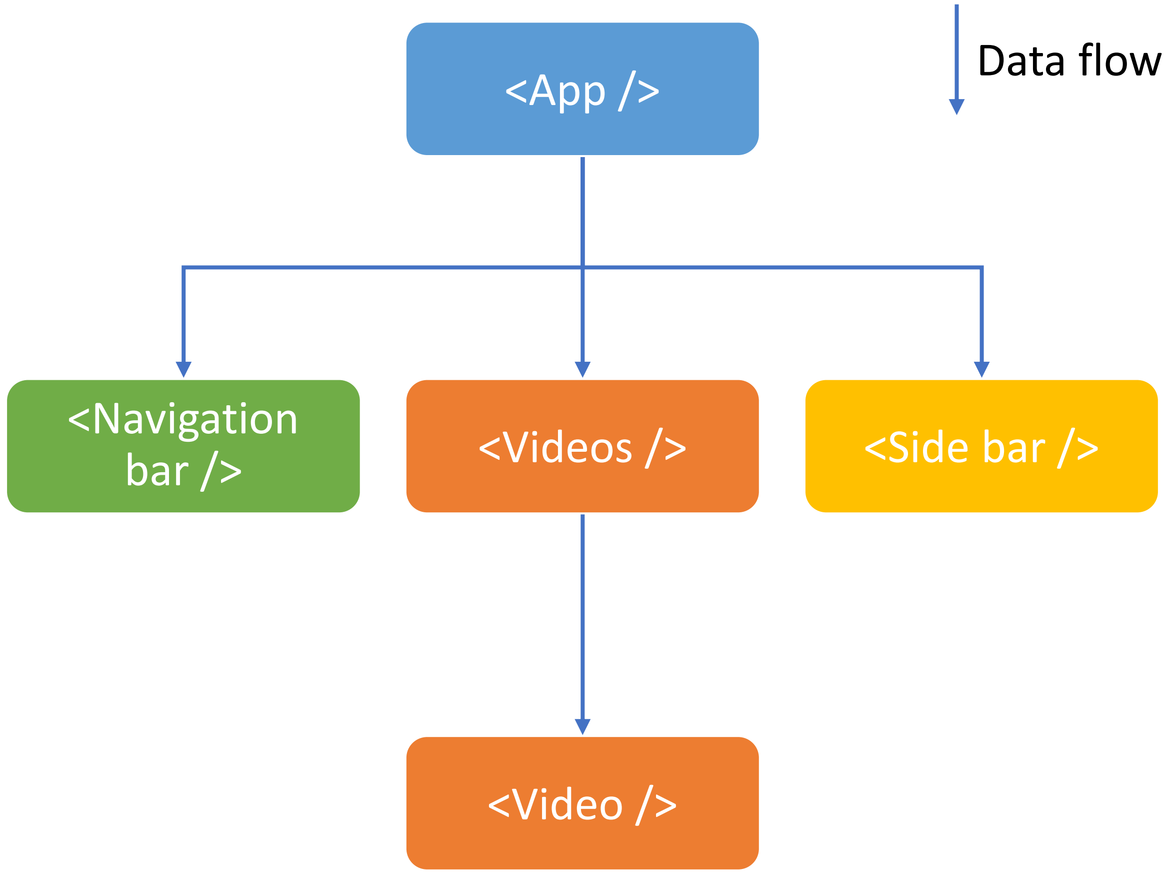
Component

- A piece of UI in React application
- Independent, isolated and reusable
- Compose multiple components together to build a complex UI
- Example: YouTube website →



Component

- The root component is called the "App" component
- App can have child components
  - Forming a tree of components
- Data always flow from top to down



Component

- Two types of components
  - Class Component
  - Functional Component

```
// Class App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
    };
  }
  render () {
    return (
        ...
    );
  }
}
```

```
// Functional App component
function App (props) {
  return (
        ...
  );
}
```

Component

- Two types of components
  - Class Component
  - ~~Functional Component~~

```
// Class App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
    };
  }
  render () {
    return (
        ...
    );
  }
}
```

```
// Functional App component
function App (props) {
  return (
      ...
  );
}
```

Component

- Two most important features in class component

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Component

- Two most important features in class component
- **State**
  - Hold the data to display when rendered

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Component

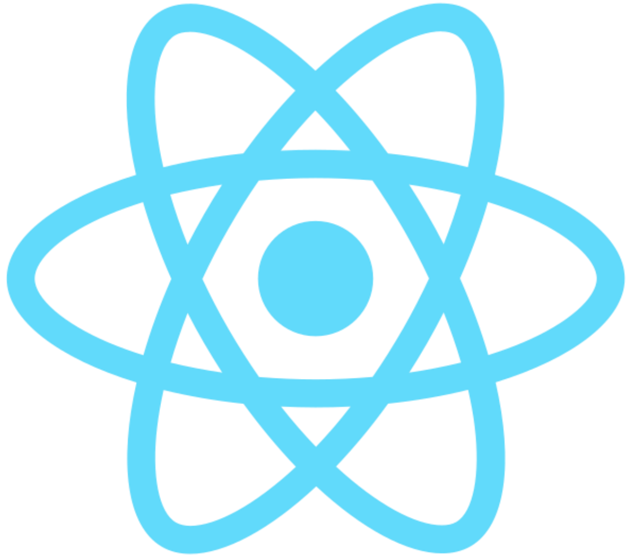
- Two most important features in class component
- State
  - Hold the data to display when rendered
- **Render method**
  - Describes what the UI should appear

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Demo

Objectives: React (Part 1)

- Component
- **States**
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event handlers
- Props
- Data binding



State

- Updateable JavaScript objects
  - Contain values
  - Can change due to user actions or system events
- Used in Class and Functional components
  - Advanced technique called Hooks in Functional Component
- When state changes, components re-renders automatically*

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
      name: "Counter App",
    };
  }
  render () {
    console.log(this.state);
    console.log(this.state.count);
    console.log(this.state.name);
    return (
      ...
    );
  }
}
```

*using setState method

State conventions

1. Initialize the state
2. Access the state
3. Change the state

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
       count: 7,
       name: "Counter App",
    };
  }
  render () {
    console.log(this.state);
    console.log(this.state.count);
    console.log(this.state.name);
    return (
       ...
    );
  }
}
```

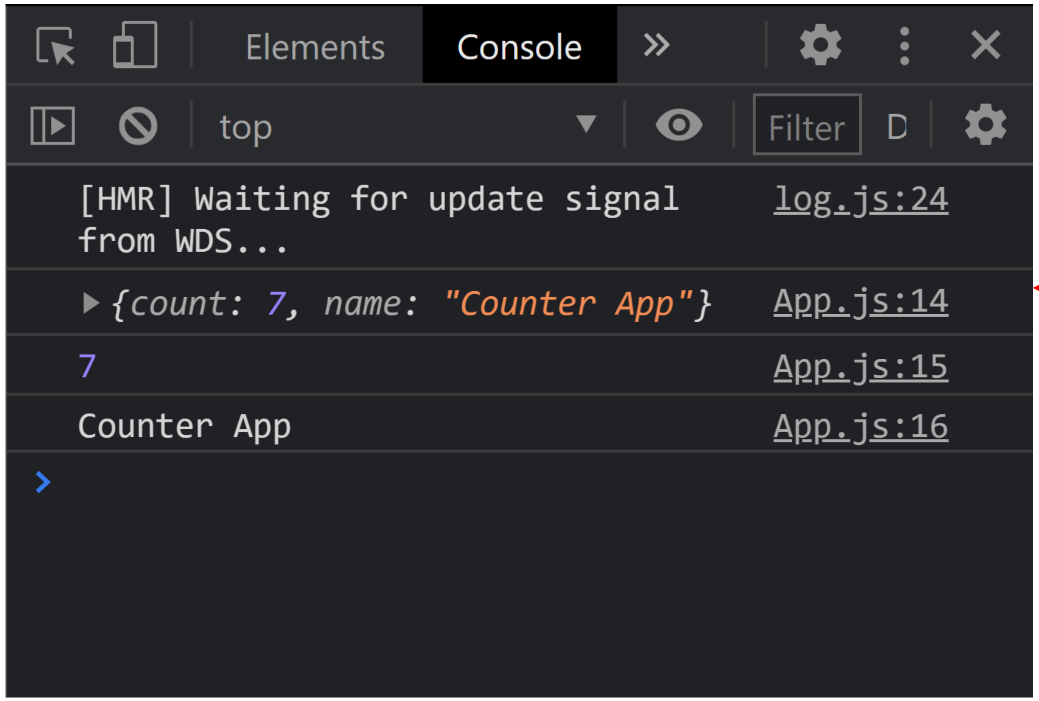
States conventions

1. **Initialize the state**
2. Access the state
3. Change the state

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
      name: "Counter App",
    };
  }
  render () {
    console.log(this.state);
    console.log(this.state.count);
    console.log(this.state.name);
    return (
      ...
    );
  }
}
```

States conventions

1. Initialize the state
2. **Access the state**
3. Change the state



```
[HMR] Waiting for update signal        log.js:24
from WDS...
▸{count: 7, name: "Counter App"}       App.js:14
7                                      App.js:15
Counter App                            App.js:16
>
```

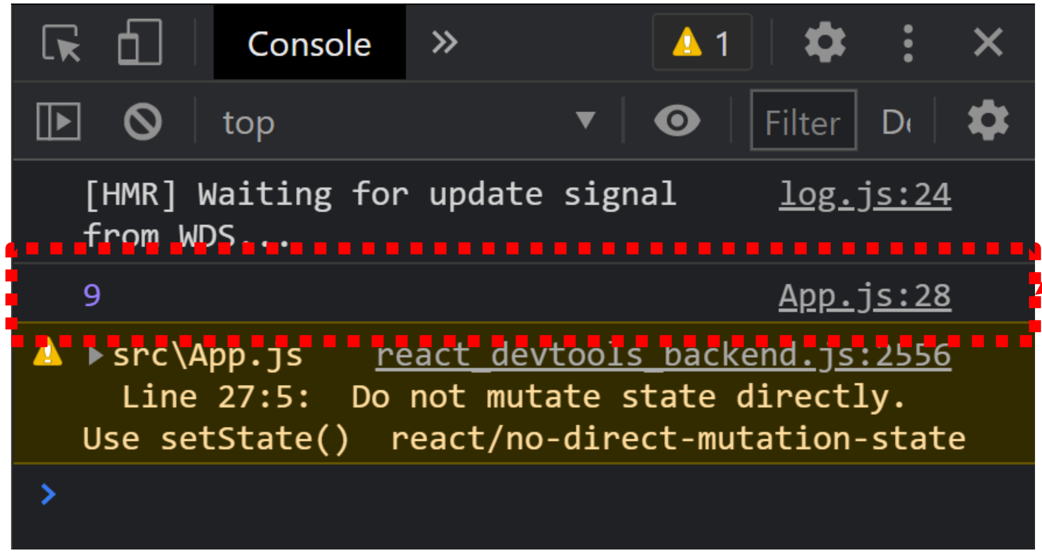
Right click inside white space of Google Chrome and click inspect

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
      name: "Counter App",
    };
  }
  render () {
    console.log(this.state);
    console.log(this.state.count);
    console.log(this.state.name);
    return (
      ...
    );
  }
}
```

Demo

States conventions

1. Initialize the state
2. Access the state
3. **Change the state**



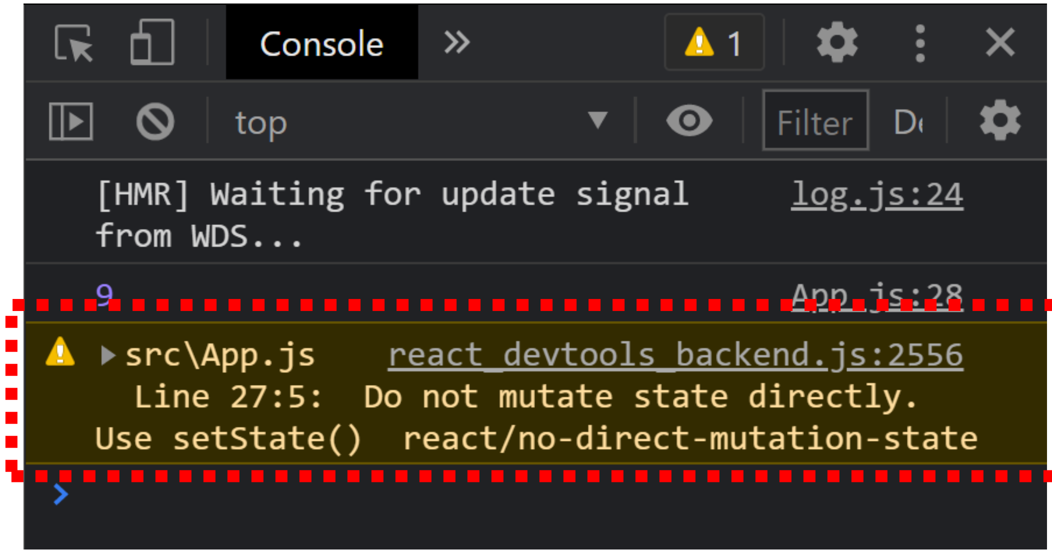
```
[HMR] Waiting for update signal         log.js:24
from WDS...
9                                       App.js:28
⚠ ▸src\App.js          react_devtools_backend.js:2556
  Line 27:5:  Do not mutate state directly.
Use setState()  react/no-direct-mutation-state
>
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
      name: "Counter App",
    };
  }
  render () {
    this.state.count = 9;
    console.log(this.state.count);
    return (
      ...
    );
  }
}
```

Right click inside white space of Google Chrome and click inspect

States conventions

1. Initialize the state
2. Access the state
3. **Change the state**



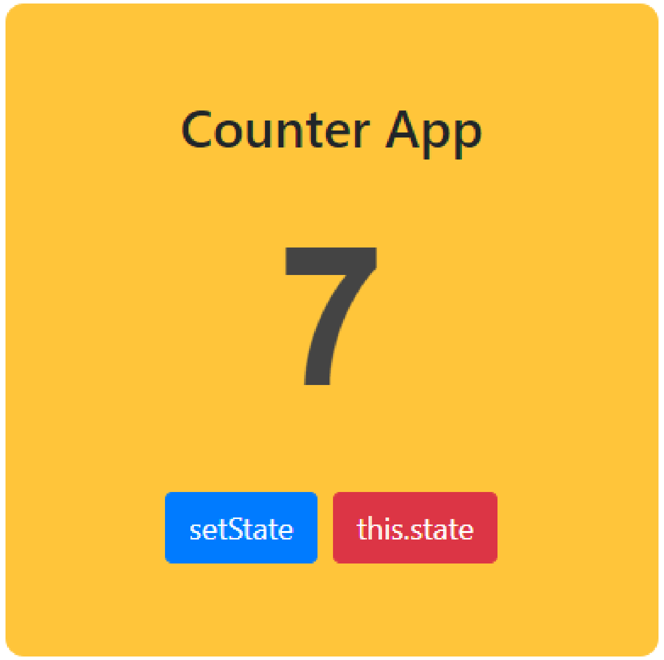
```
[HMR] Waiting for update signal      log.js:24
from WDS...
9                                    App.js:28
⚠ ▸src\App.js      react_devtools_backend.js:2556
   Line 27:5:  Do not mutate state directly.
 Use setState()  react/no-direct-mutation-state
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
       count: 7,
       name: "Counter App",
    };
  }
  render () {
    this.state.count = 9;
    console.log(this.state.count);
    return (
       ...
    );
  }
}
```

Right click inside white space of Google Chrome and click inspect

States conventions

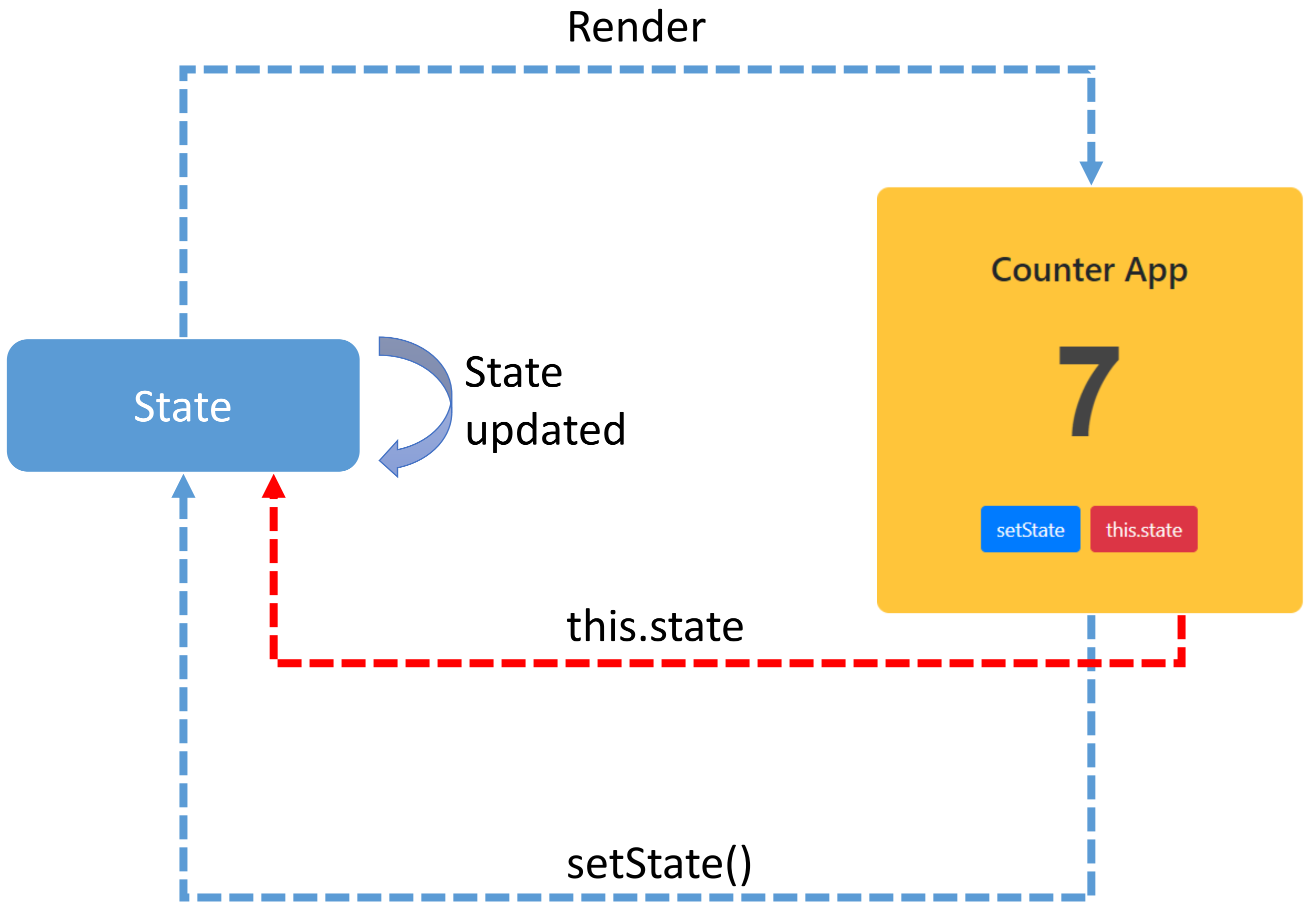
1. Initialize the state
2. Access the state
3. **Change the state**



```
// App Component
// The proper way to change state
// is using "setState method"
handleSetState() {
  this.setState({
    count: this.state.count + 1
  })
}

// The wrong way to change state is
// assigning value using this.state
handleThisState() {
  this.state.count =
       this.state.count + 1;
}
```

States conventions

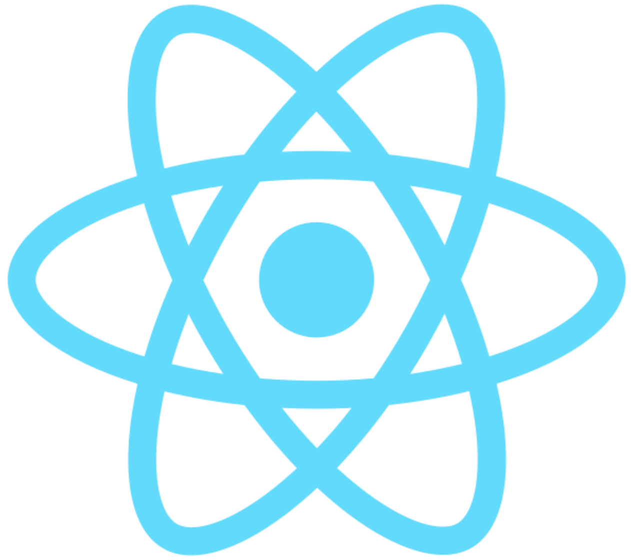
1. Initialize the state
2. Access the state
3. **Change the state**
   - setState method
     - Updates the state and renders the view
   - this.state
     - Only mutates the state



Demo

Objectives: React (Part 1)

- Component
- States
- **React Element and Virtual DOM**
- JavaScript XML (JSX)
- Event handlers
- Props
- Data binding



Recap: Component

- Two most important features in class component
- **State**
  - Hold the data to display when rendered
- **Render method**
  - Describes what the UI should appear

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Render method

- Two most important features in class component
- State
  - Hold the data to display when rendered
- **Render method**
  - Describes what the UI should appear

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Render method

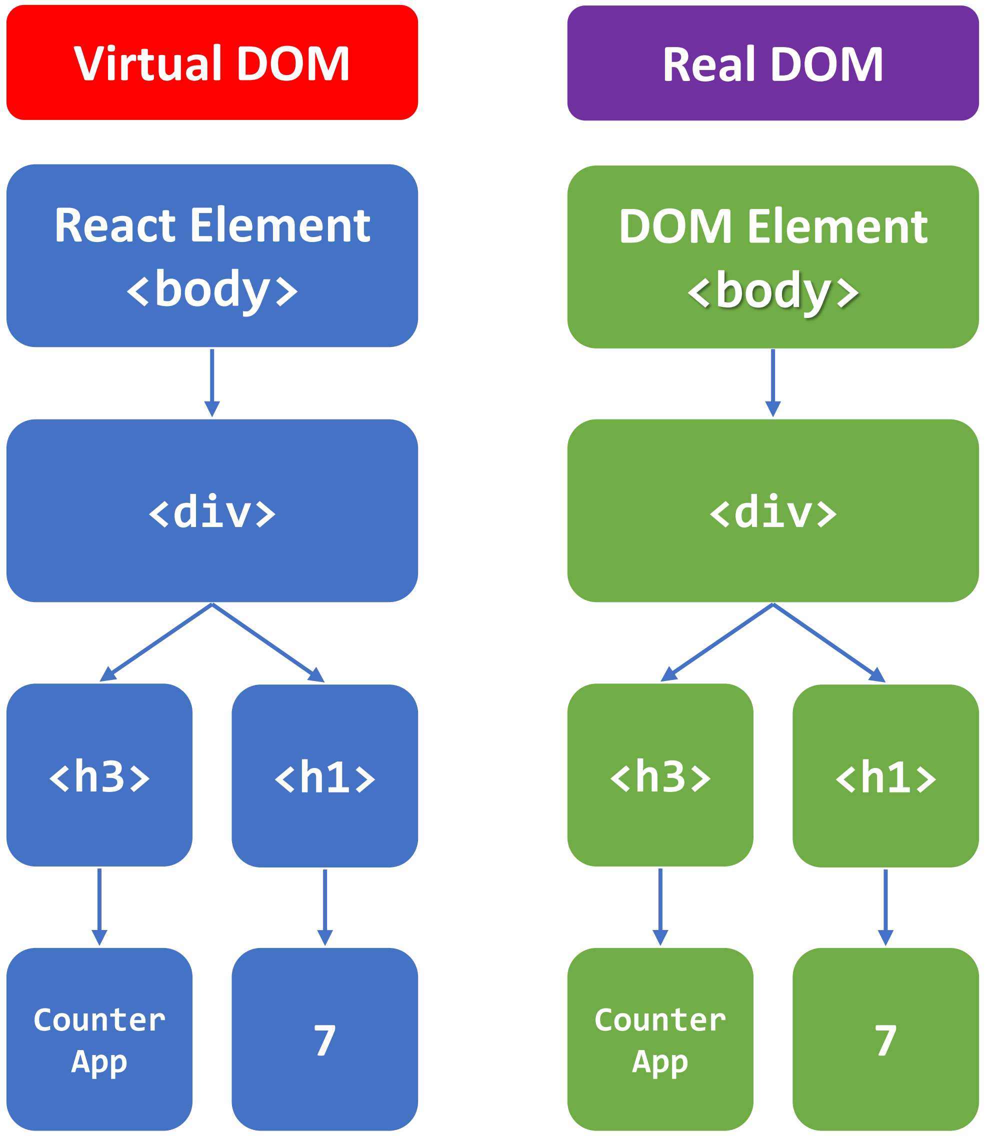
- Two most important features in class component
- State
  - Hold the data to display when rendered
- **Render method**
  - Describes what the UI should appear
  - Output → **React Element**

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

React Element

- React Element is a plain JavaScript object and keeps a copy of Real DOM

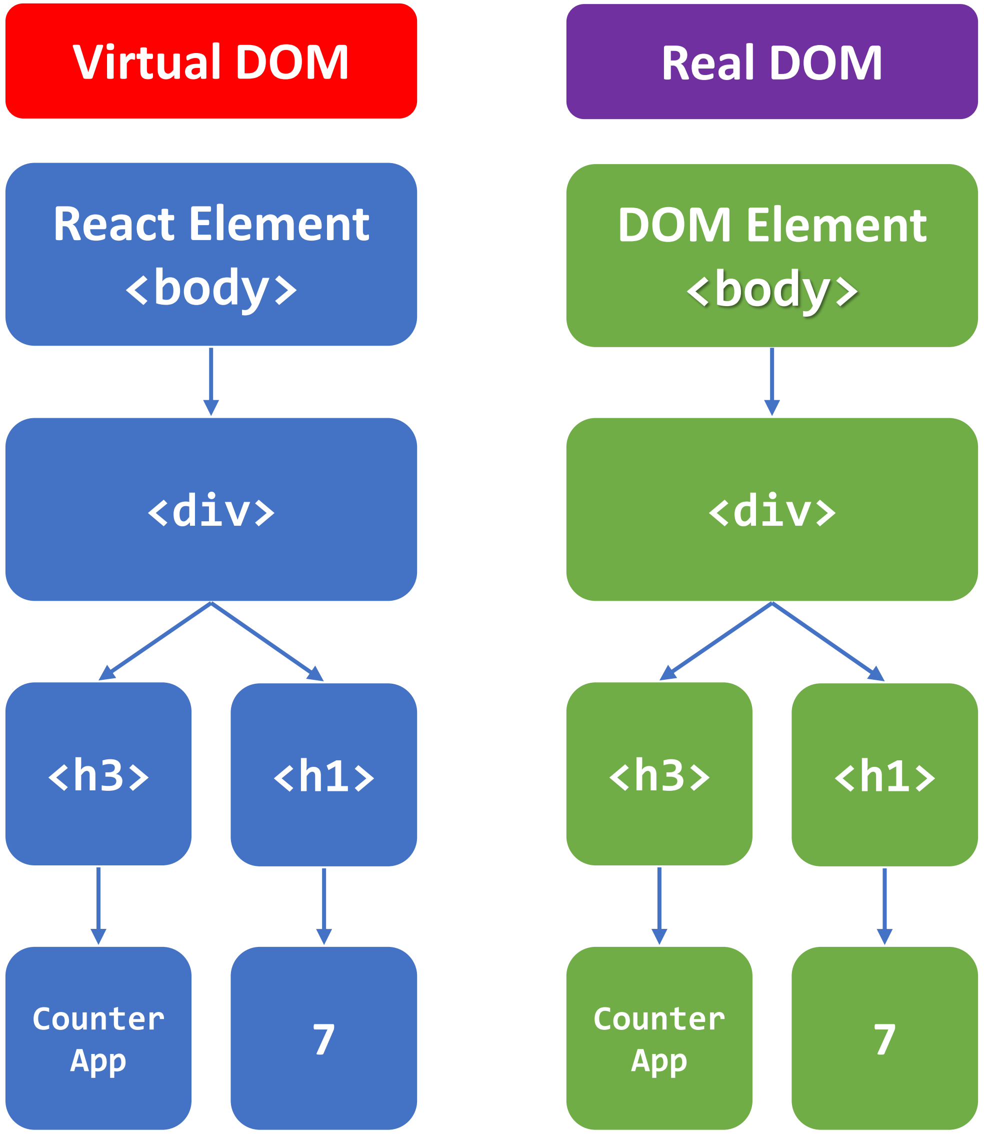
```
// Render method in App Component
render () {
  return (
    <div>
      <h3>Counter App</h3>
      <h1>{this.state.count}</h1>
    </div>
  );
}
```



React Element

- React Element is a plain JavaScript object and keeps a copy of Real DOM
- React keeps a lightweight representation of **Real DOM element in memory → Virtual DOM**

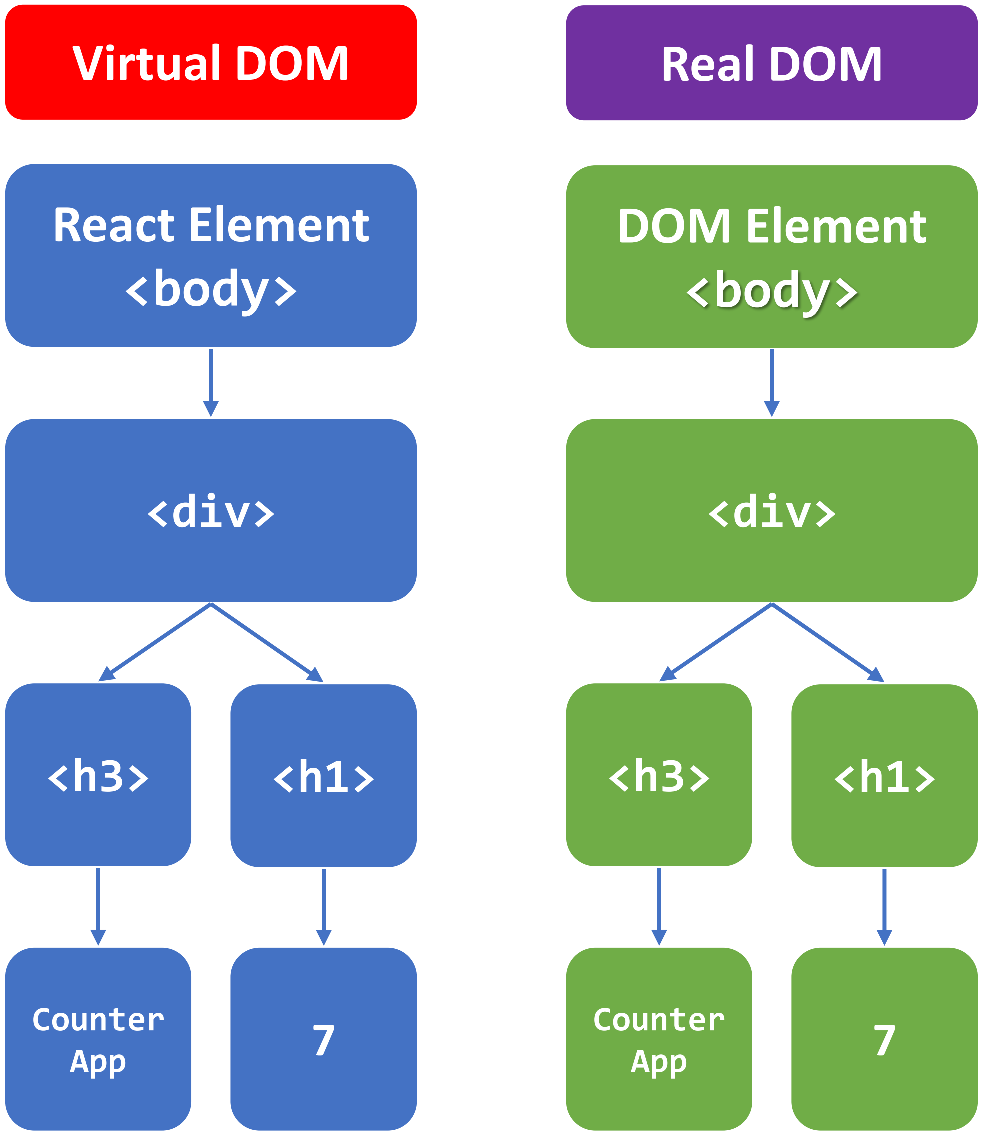
```
// Render method in App Component
render () {
  return (
    <div>
      <h3>Counter App</h3>
      <h1>{this.state.count}</h1>
    </div>
  );
}
```



React Element

- React Element is a plain JavaScript object and keeps a copy of Real DOM
- React keeps a lightweight representation of **Real DOM element in memory → Virtual DOM**

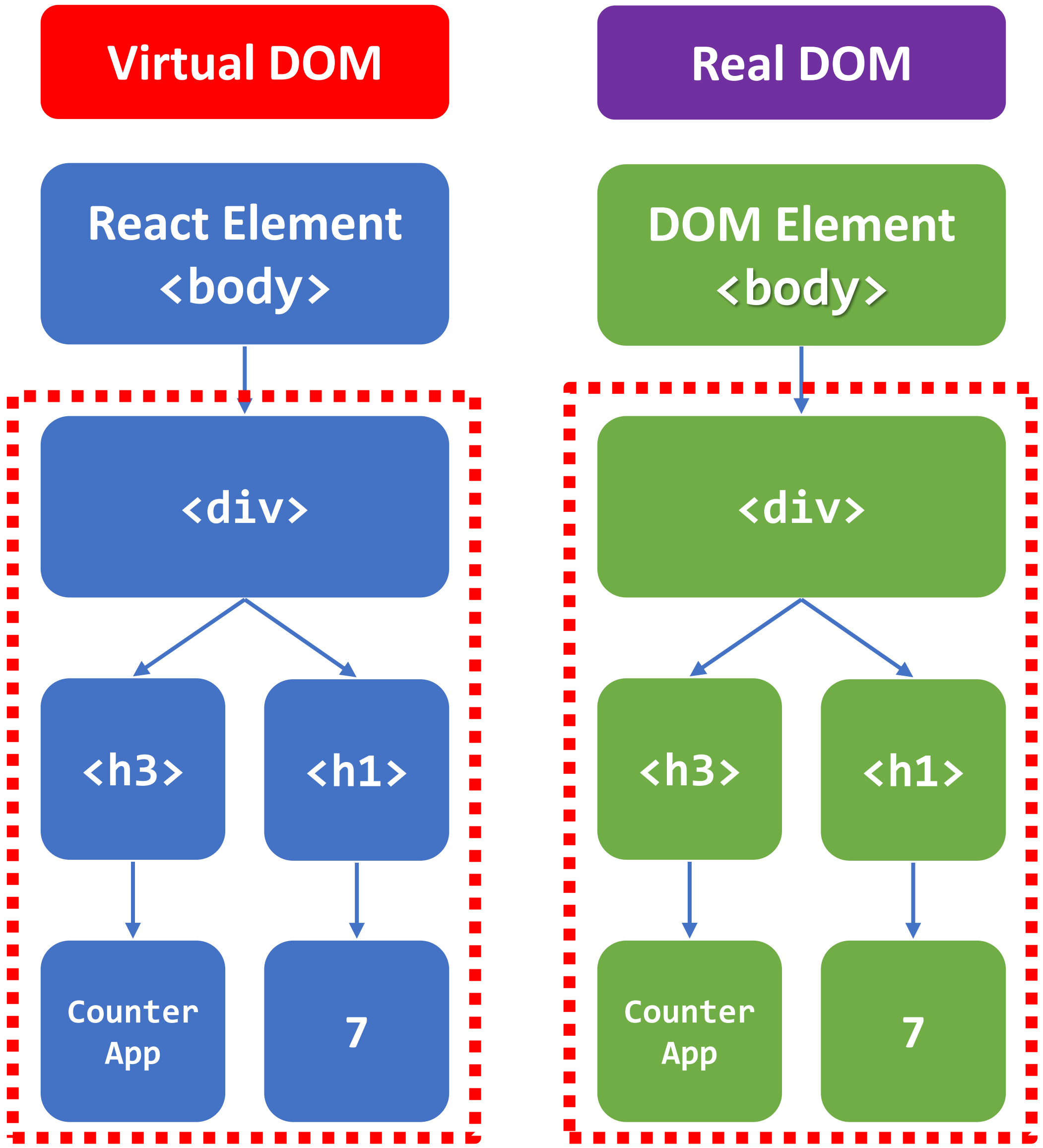
```
// Render method in App Component
render () {
   return (
    <div>
       <h3>Counter App</h3>
       <h1>{this.state.count}</h1>
    </div>
   );
}
```



React Element

- React Element is a plain JavaScript object and keeps a copy of Real DOM
- React keeps a lightweight representation of **Real DOM element in memory → Virtual DOM**

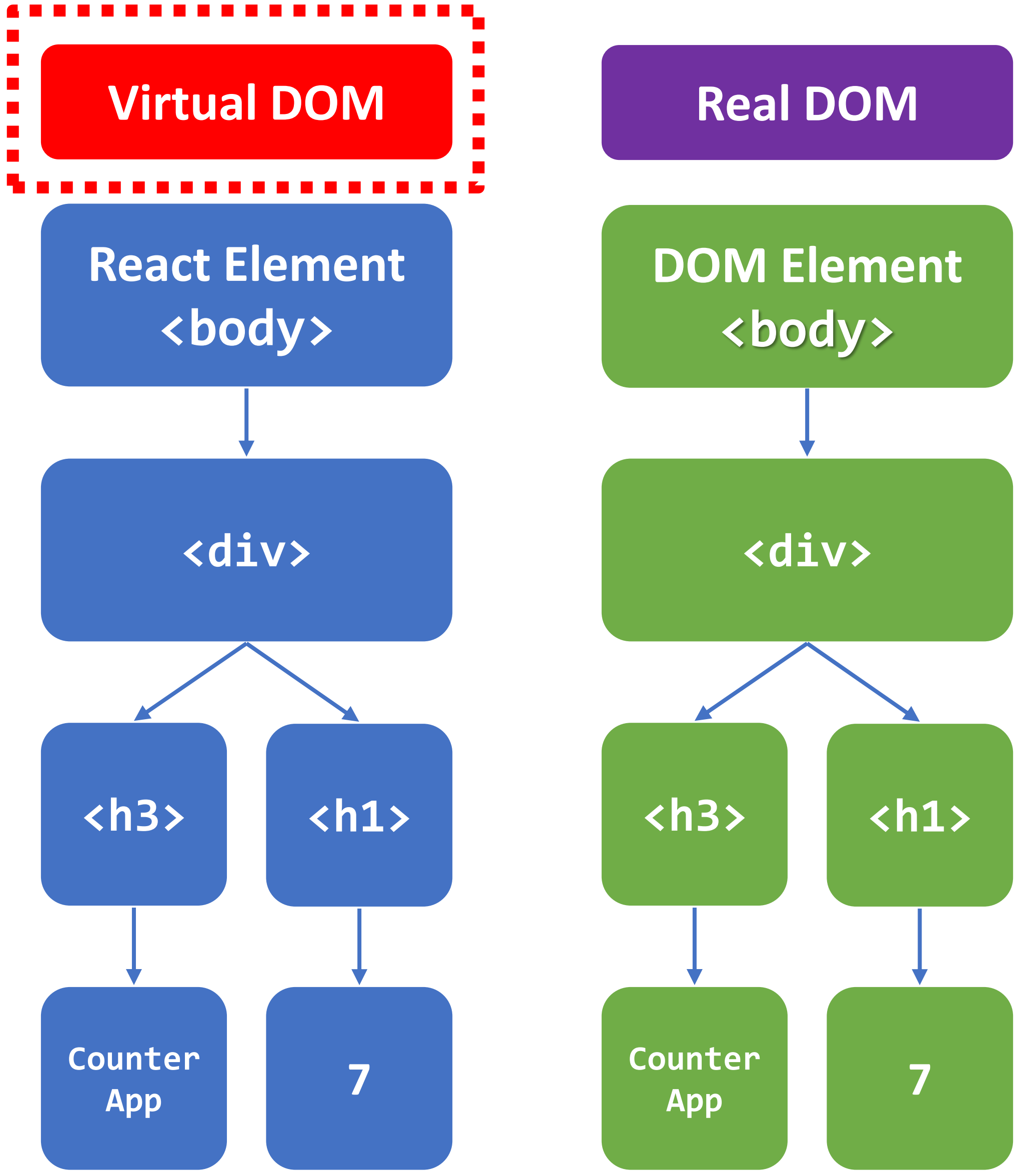
```
// Render method in App Component
render () {
  return (
    <div>
      <h3>Counter App</h3>
      <h1>{this.state.count}</h1>
    </div>
  );
}
```



React Element

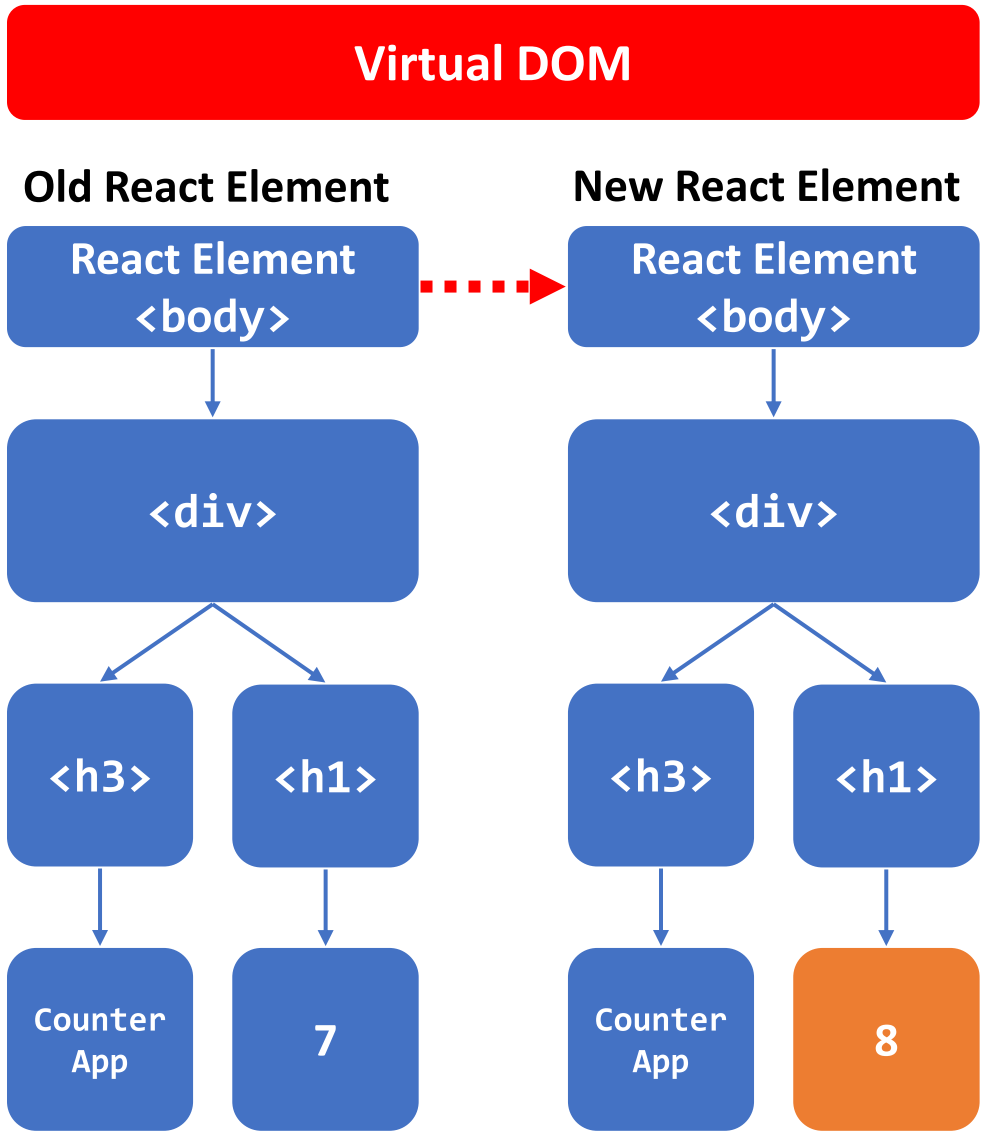
- React Element is a plain JavaScript object and keeps a copy of Real DOM
- React keeps a lightweight representation of **Real DOM element in memory → Virtual DOM**

```
// Render method in App Component
render () {
  return (
    <div>
      <h3>Counter App</h3>
      <h1>{this.state.count}</h1>
    </div>
  );
}
```



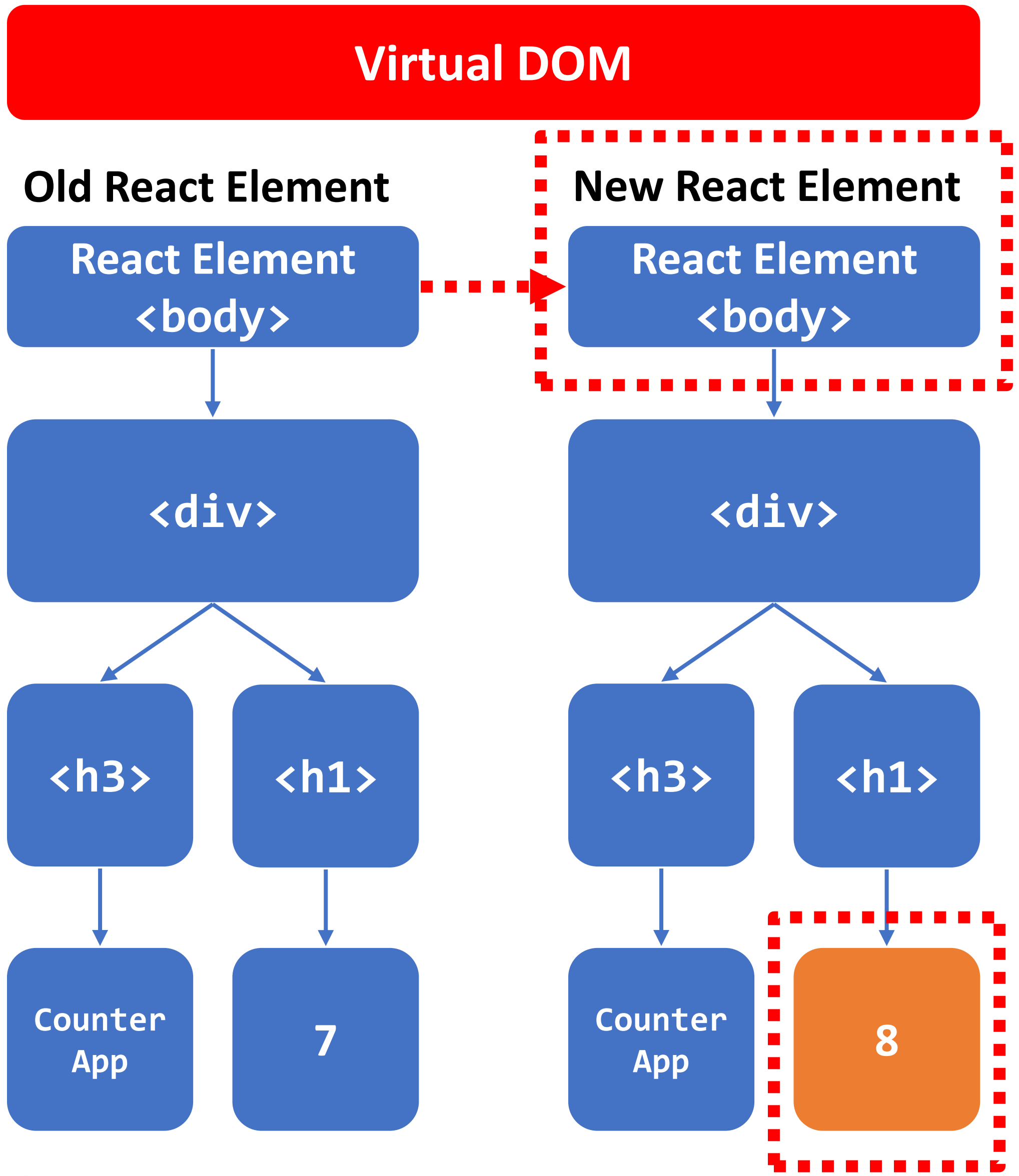
Virtual DOM

- State change creates a new React Element
- Compares the old React Element with the new React Element



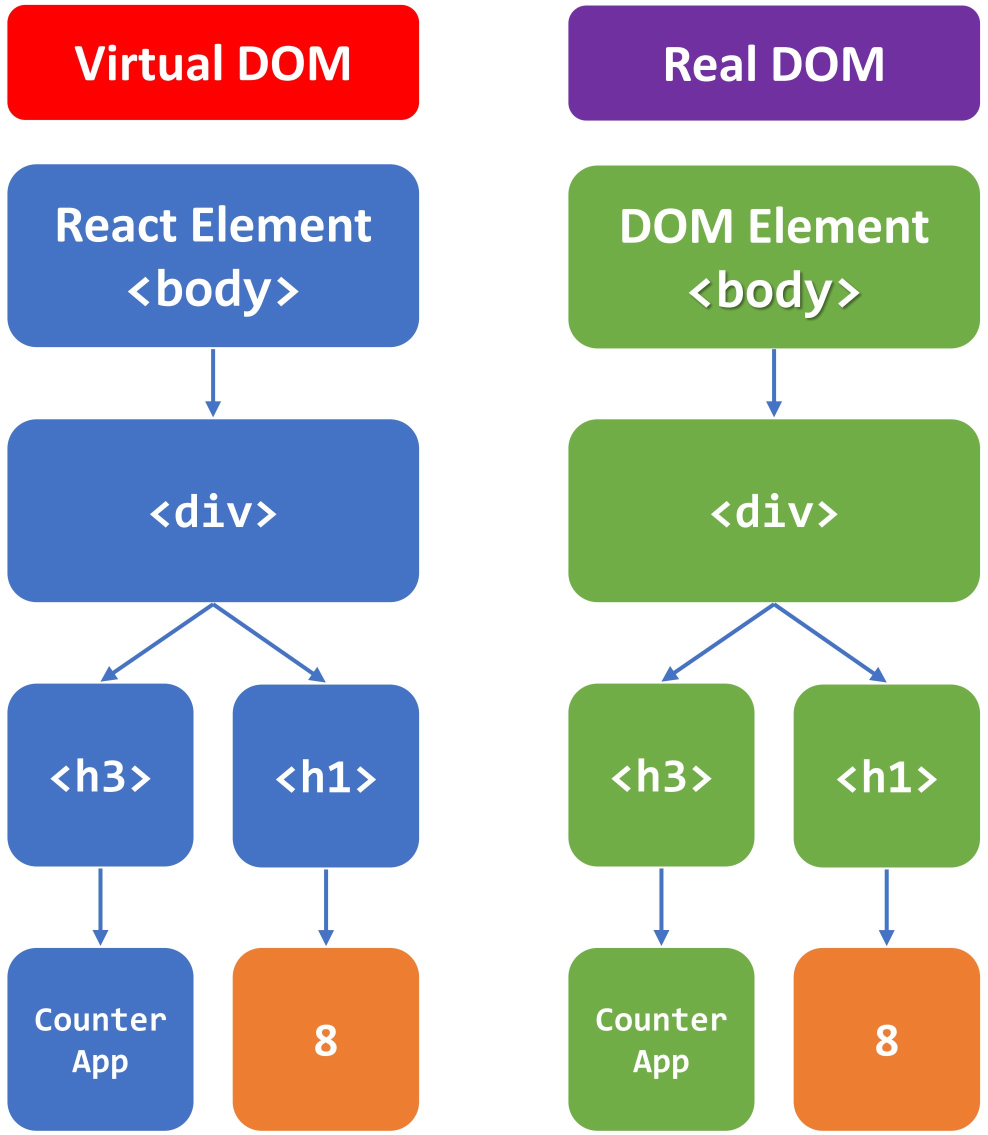
Virtual DOM

- State change creates a new React Element
- Compares the old React Element with the new React Element



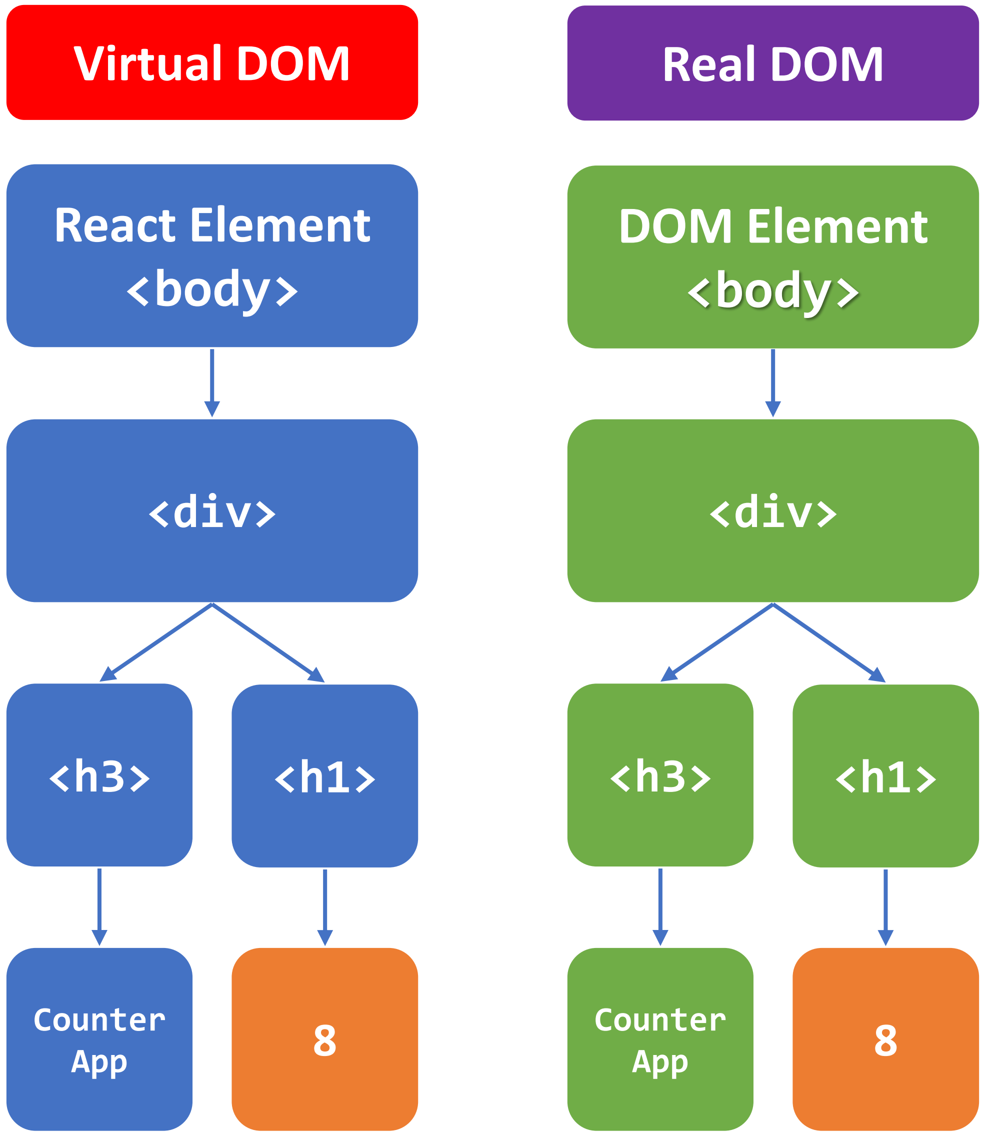
Virtual DOM

- State change creates a new React Element
- Compares the old React Element with the new React Element
- Updates in the real DOM
  - Keeping in sync with Virtual DOM



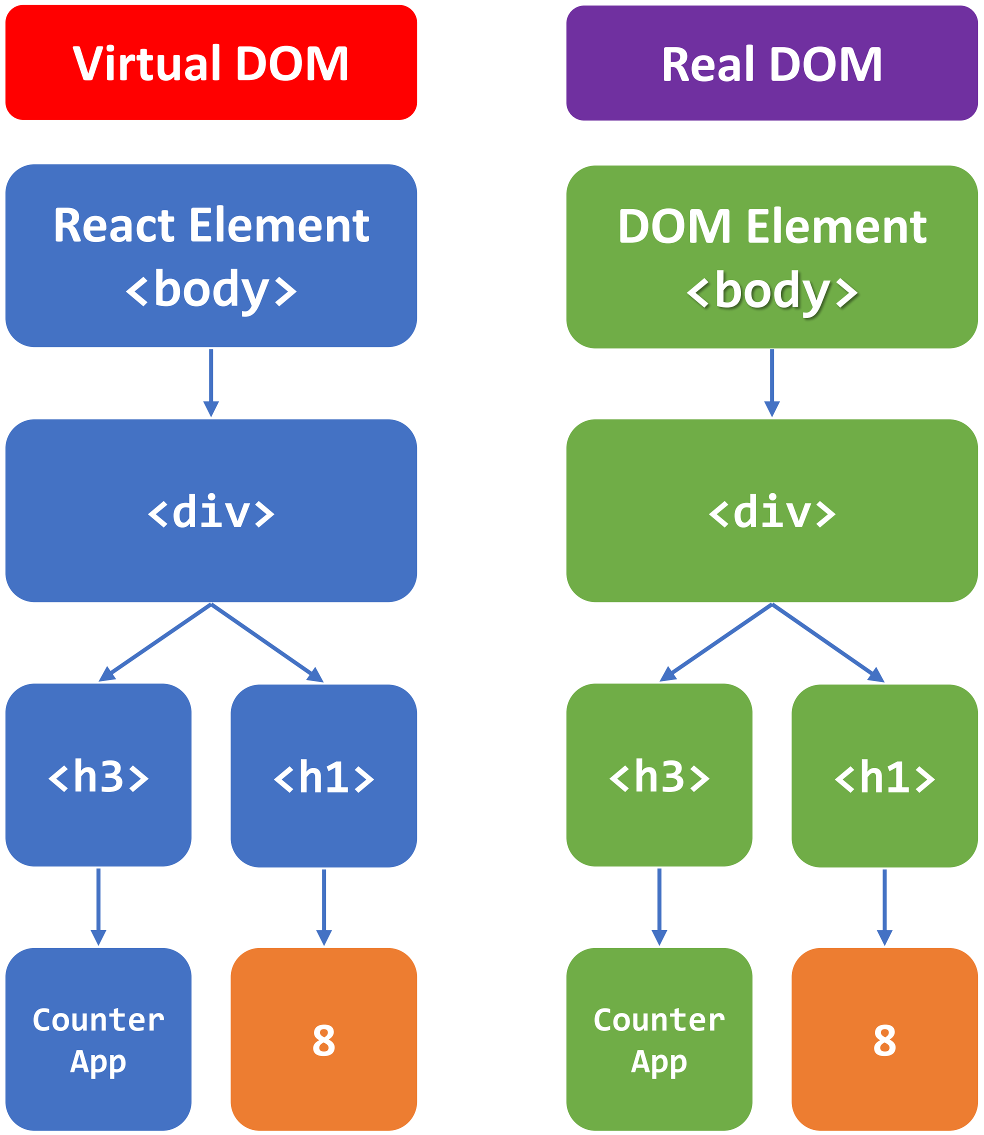
Virtual DOM

- State change creates a new React Element
- Compares the old React Element with the new React Element
- Updates in the real DOM
  - Keeping in sync with Virtual DOM
- Advantages?



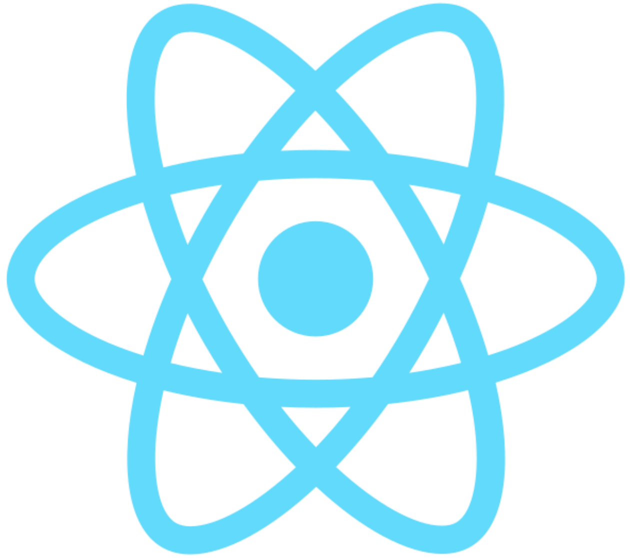
Virtual DOM

- State change creates a new React Element
- Compares the old React Element with the new React Element
- Updates in the real DOM
  - Keeping in sync with Virtual DOM
- Advantages:
  - Re-rendering Real DOM after each update is costly
  - Updating in-memory DOM leads to performance gain



Objectives: React (1/2)

- Component
- States
- React Element and Virtual DOM
- **JavaScript XML (JSX)**
- Event handlers
- Props
- Data binding



JavaScript XML (JSX)

- Output of render method is some Mark-up syntax
- Syntax is neither a string nor html

→ **JavaScript XML (JSX)**

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

JavaScript XML (JSX)

- React uses Babel[1] JavaScript library
  - A JavaScript XML compiler
- Converts JSX to plain JavaScript (JS)
  - Browsers can understand JS

```
React.createElement("div", null,
  React.createElement("h3", null,
                      "Counter\xA0App"),
    React.createElement("h1", null,
                        this.state.count)
);
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

[1] https://babeljs.io/repl

JavaScript XML (JSX)

- React element with three properties
  - Type: `div`
  - Props: `null` or `{}`
  - Children/value: child root elements with h3 and h1 tags

```
React.createElement("div", null,
  React.createElement("h3", null,
                "Counter\xA0App"),
    React.createElement("h1", null,
                this.state.count)
);
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

JavaScript XML (JSX)

- React element with three properties
  - Type: h3
  - Props: `null` or `{}`
  - Children/value: `"Counter App"`

```
React.createElement("div", null,
    React.createElement("h3", null,
                               "Counter\xA0App"),
        React.createElement("h1", null,
                               this.state.count)
);
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
       count: 7,
    };
  }
  render () {
    return (
       <div>
         <h3>Counter App</h3>
         <h1>{this.state.count}</h1>
       </div>
    );
  }
}
```

JavaScript XML (JSX)

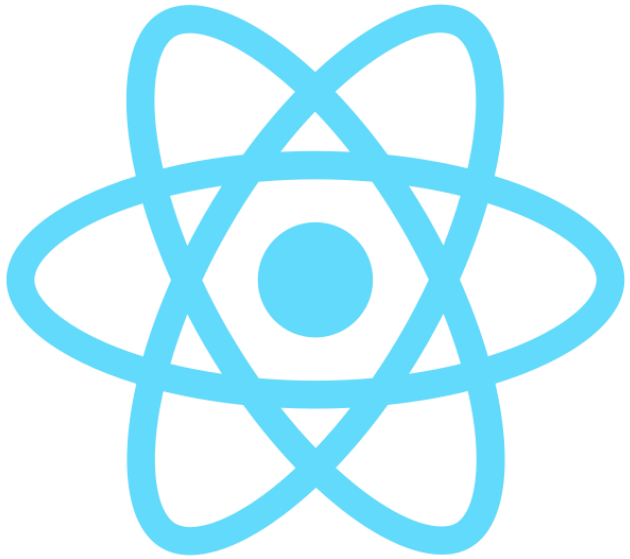
- React element with three properties
  - Type: h1
  - Props: null or {}
  - Children/value: 7

```
React.createElement("div", null,
  React.createElement("h3", null,
                      "Counter\xA0App"),
    React.createElement("h1", null,
                        this.state.count)
);
```

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  render () {
    return (
      <div>
        <h3>Counter App</h3>
        <h1>{this.state.count}</h1>
      </div>
    );
  }
}
```

Objectives: React (1/2)

- Component
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- **Event handlers**
- Props
- Data binding



Event handlers

- Ways to interact with an application e.g.
  - Clicking buttons, moving mouse cursor, typing in input fields, pressing Shift/Control keys etc.
- React's event handling system → **Synthetic Events**
  - Cross-browser wrapper of browser's native events[1]
  - Similar to event handlers in DOM elements

```
// HTML
<button onclick="handleReset()">
  Reset
</button>
```

```
// In React
<Button onClick={this.handleReset}>
  Reset
</Button>
```

[1] https://reactjs.org/docs/events.html

Event handlers

- Ways to interact with an application e.g.
  - Clicking buttons, moving mouse cursor, typing in input fields, pressing Shift/Control keys etc.
- React's event handling system → **Synthetic Events**
  - Cross-browser wrapper of browser's native events[1]
  - Similar to event handlers in DOM elements
- Event handlers in React are in **`camelCase`** form e.g. `onClick` not `onclick`

```
// HTML
<button onclick="handleReset()">
  Reset
</button>
```

```
// In React
<Button onClick={this.handleReset}>
  Reset
</Button>
```

1 https://reactjs.org/docs/events.html

Event handlers

- Ways to interact with an application e.g.
  - Clicking buttons, moving mouse cursor, typing in input fields, pressing Shift/Control keys etc.
- React's event handling system → **Synthetic Events**
  - Cross-browser wrapper of browser's native events[1]
  - Similar to event handlers in DOM elements
- Event handlers in React are in **`camelCase`** form e.g. `onClick` not `onclick`
- Pass a function in JSX unlike strings in HTML

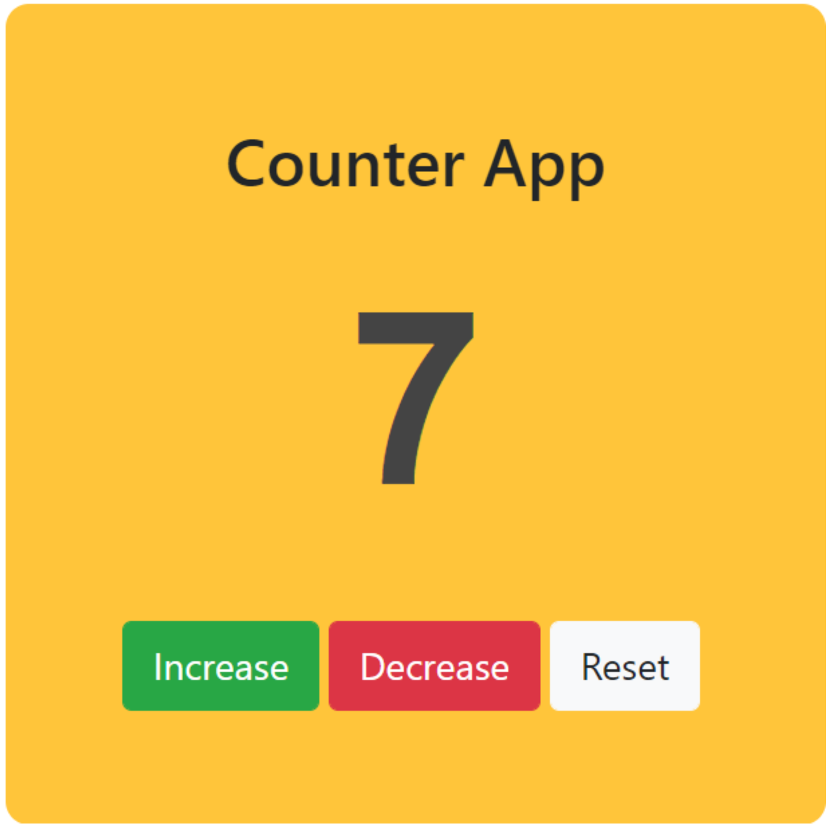
```
// HTML
<button onclick="handleReset()">
  Reset
</button>
```

```
// In React
<Button onClick={this.handleReset}>
  Reset
</Button>
```

[1] https://reactjs.org/docs/events.html

Event handlers

- Example →
  Interactive buttons for the counter application



Event handlers

- **onClick** is one of the Synthetic Events
  - Triggers action due to mouse click
  - Other event handlers in React. onFocus, onChange, onBlur etc.

```
// App Component

handleReset() {
  this.setState({
    count: 0,
  });
}

render () {
  return (
  <>
    ...
    <Button
      className="btn-light"
      onClick={this.handleReset}>
        Reset
    </Button>
    ...
  </>
  );
}
```

https://react-bootstrap.github.io/

Event handlers

- **onClick** is one of the Synthetic Events
  - Triggers action due to mouse click
  - Other event handlers in React. onFocus, onChange, onBlur etc.
- The **this** keyword represents the component that owns the method

```
// App Component

handleReset() {
  this.setState({
    count: 0,
  });
}

render () {
  return (
  <>
    ...
    <Button
      className="btn-light"
      onClick={this.handleReset}>
        Reset
    </Button>
    ...
  </>
  );
}
```

https://react-bootstrap.github.io/

Event handlers

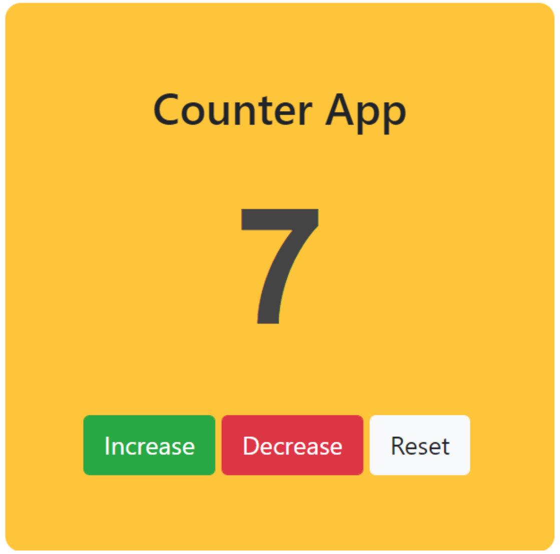
- **`onClick`** is one of the Synthetic Events
  - Triggers action due to mouse click
  - Other event handlers in React. onFocus, onChange, onBlur etc.
- The **`this`** keyword represents the component that owns the method

```
// App Component
handleReset() {
  this.setState({
    count: 0,
  });
}

render () {
  return (
  <>
    ...
   <Button
     className="btn-light"
     onClick={this.handleReset}>
       Reset
   </Button>
    ...
  </>
  );
}
```

https://react-bootstrap.github.io/

Event handlers

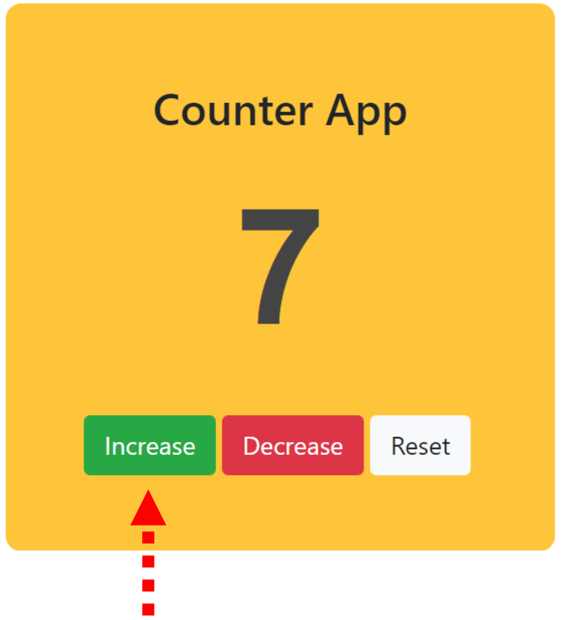
- The functionality of each buttons are as follows:



```
// Render method of App Component
render () {
  return (
  <>
    ...
    <Button className="btn-success"
            onClick={this.handleIncrease} >
      Increase
    </Button>

    <Button className="btn-danger"
            style={gutter}
            onClick={this.handleDecrease} >
      Decrease
    </Button>

    <Button className="btn-light"
            onClick={this.handleReset} >
        Reset
    </Button>
    ...
  </>
  );
}
```

Event handlers

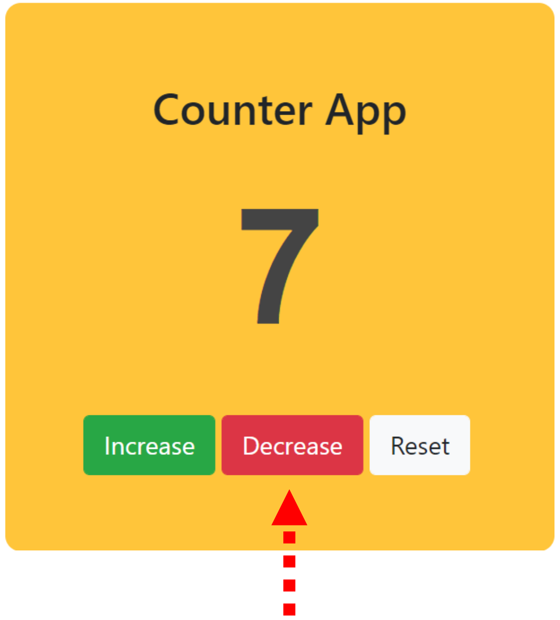
- The functionality of each buttons are as follows:
  - To **increase** the count by 1



```
// Render method of App Component
render () {
  return (
  <>
    ...
    <Button className="btn-success"
            onClick={this.handleIncrease} >
      Increase
    </Button>

    <Button className="btn-danger"
            style={gutter}
            onClick={this.handleDecrease} >
      Decrease
    </Button>

    <Button className="btn-light"
            onClick={this.handleReset} >
        Reset
    </Button>
    ...
  </>
  );
}
```

Event handlers

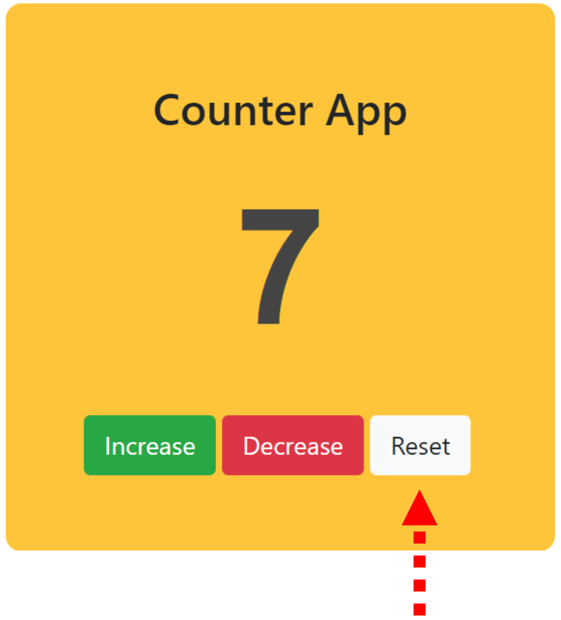
- The functionality of each buttons are as follows:
  - To increase the count by 1
  - To **decrease** the count by 1



```
// Render method of App Component
render () {
  return (
  <>
    ...
    <Button className="btn-success"
            onClick={this.handleIncrease} >
      Increase
    </Button>

    <Button className="btn-danger"
            style={gutter}
            onClick={this.handleDecrease} >
      Decrease
    </Button>

    <Button className="btn-light"
            onClick={this.handleReset} >
        Reset
    </Button>
    ...
  </>
  );
}
```

Event handlers

- The functionality of each buttons are as follows:
  - To increase the count by 1
  - To decrease the count by 1
  - To **reset** the count to 0



```
// Render method of App Component
render () {
  return (
  <>
    ...
    <Button className="btn-success"
            onClick={this.handleIncrease} >
      Increase
    </Button>

    <Button className="btn-danger"
            style={gutter}
            onClick={this.handleDecrease} >
      Decrease
    </Button>

    <Button className="btn-light"
            onClick={this.handleReset} >
        Reset
    </Button>
    ...
  </>
  );
}
```

Event handlers

- Three methods are assigned to each buttons
  - `handleIncrease()`
    - To increase the count by 1
  - `handleDecrease()`
    - To decrease the count by 1
  - `handleReset()`
    - To set the count to 0

```
// App Component

// Event handler functions
handleIncrease() {
  this.setState({
    count: this.state.count + 1
  })
}

handleDecrease() {
  this.setState({
    count: this.state.count - 1
  })
}

handleReset() {
  this.setState({
    count: 0,
  });
}
```

Event handlers

- Three methods are assigned to each buttons
  - `handleIncrease()`
    - To increase the count by 1
  - `handleDecrease()`
    - To decrease the count by 1
  - `handleReset()`
    - To set the count to 0

```
// App Component

// Event handler functions
handleIncrease() {
  this.setState({
    count: this.state.count + 1
  })
}

handleDecrease() {
  this.setState({
    count: this.state.count - 1
  })
}

handleReset() {
  this.setState({
    count: 0,
  });
}
```

Event handlers

- The **this** keyword represents the component that owns the method
- Problem →
  - setState returns undefined
  - **this** keyword is not explicitly defined in the method
- Few techniques available to manually attach `this` keyword to the method
  - Refers to the bound object (state) when function is called

```
// App Component
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
    };
  }
  ...
  handleReset() {
    this.setState({
      count: 0,
    });
  }
  ...
  render () {
    return (
    <>
      ...
       <Button className="btn-light"
               onClick={this.handleReset}>
            Reset
       </Button>
      ...
    </>
  );
}
```

Read more

Event handlers

- There are two possible solutions

- First Solution →
  bind **`this`** keyword to the component instance using `.bind()` method

```
// App Component
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 7,
      name: "Counter App",
    };

    // Binding methods
    this.handleIncrease =
      this.handleIncrease.bind(this);

    this.handleDecrease =
      this.handleDecrease.bind(this);

    this.handleReset =
      this.handleReset.bind(this);

  }
...
```

Event handlers

- There are two possible solutions
- First Solution →
  bind **this** keyword to the component instance using `.bind()` method
- Second Solution →
  use the arrow function syntax provided in ES6 standard

  "methodName = `() => { }`"

  → manual binding not required

```
// App Component
// Event handler functions
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}

handleDecrease = () => {
  this.setState({
    count: this.state.count - 1
  })
}

handleReset = () => {
  this.setState({
    count: 0,
  });
}
```

Parameters in methods

- handleReset method has no input parameter

```
// App Component
// Method with no input parameter
handleReset = () => {
  this.setState({
    count: 0,
  });
}
```

Parameters in methods

- handleReset method has no input parameter
- Pass parameters to methods by using the following syntax →

```
methodName = (param1, param2, ...) => {
   ...
}
```

```
// App Component
// Method with no input parameter
handleReset = () => {
  this.setState({
    count: 0,
  });
}
```

```
// App Component
// Method with input parameter
handleReset = (value) => {
  this.setState({
    count: value,
  });
}
```

Pass arguments

- Pass arguments using anonymous arrow function

```
// App Component
handleReset = (value) => {
  this.setState({
    count: value,
  });
}
...
render () {
  return (
  <>
    ...
    <Button
      className="btn-light"
      onClick={() => this.handleReset(0)}
    >
        Reset
    </Button>
    ...
  </>
  );
}
```

Pass arguments

- Pass arguments using anonymous arrow function

```
// App Component
handleReset = (value) => {
  this.setState({
    count: value,
  });
}
...
render () {
  return (
  <>
    ...
    <Button
      className="btn-light"
      onClick={() => this.handleReset(0)}
    >
        Reset
    </Button>
    ...
  </>
  );
}
```

Pass arguments

```
// App Component
handleReset = (value) => () => {
    this.setState({
        count: value,
    });
}
...
render () {
  return (
  <>
    ...
    <Button
     className="btn-light"
     onClick={this.handleReset(0)}
    >
       Reset
    </Button>
    ...
  </>
  );
}
```

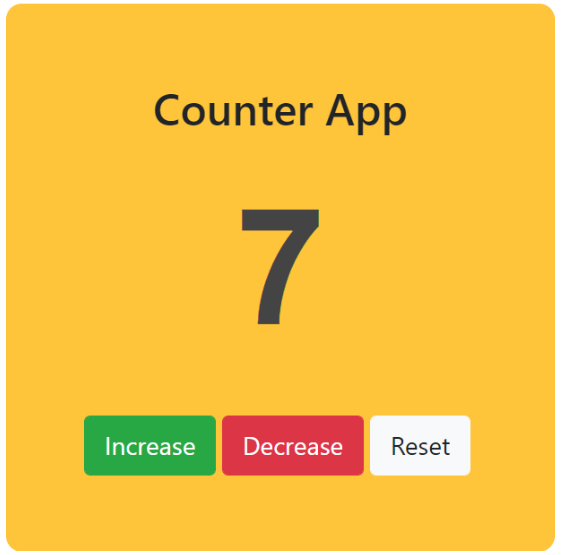
```
// App Component
handleReset = (value) => {
  this.setState({
    count: value,
  });
}
...
render () {
  return (
  <>
    ...
    <Button
     className="btn-light"
     onClick={() => this.handleReset(0)}
    >
       Reset
    </Button>
    ...
  </>
  );
}
```

Pass arguments

```
// App Component
handleReset = (value) => () => {
    this.setState({
        count: value,
    });
}
...
render () {
  return (
  <>
    ...
   <Button
     className="btn-light"
     onClick={this.handleReset(0)}
   >
       Reset
   </Button>
    ...
  </>
  );
}
```

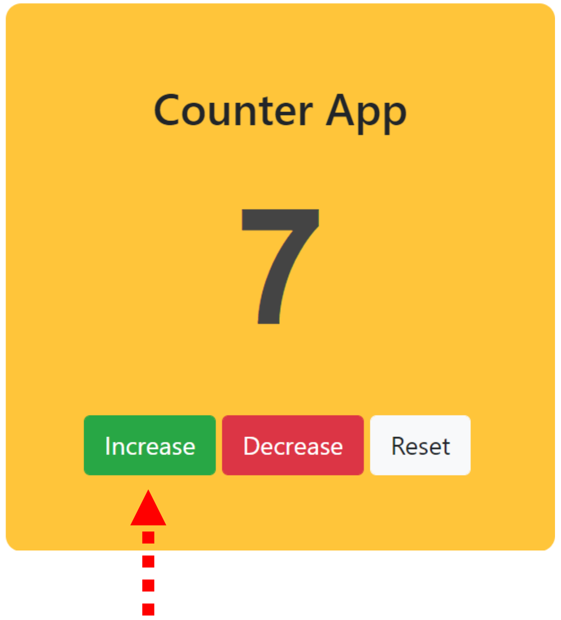
```
// App Component
handleReset = (value) => {
  this.setState({
    count: value,
  });
}
...
render () {
  return (
  <>
    ...
   <Button
     className="btn-light"
     onClick={() => this.handleReset(0)}
   >
       Reset
   </Button>
    ...
  </>
  );
}
```

Enhanced handleIncrease method



```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

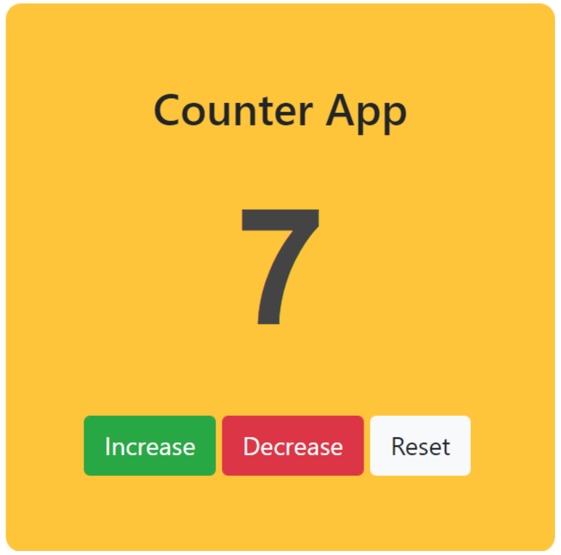
Enhanced handleIncrease method



```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

Enhanced handleIncrease method

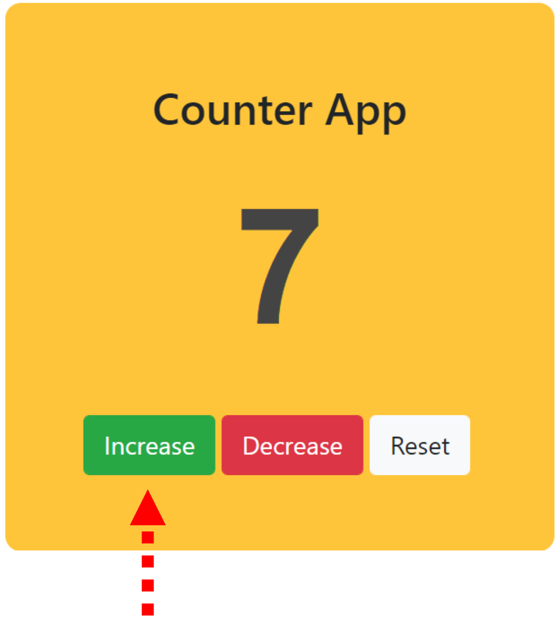
- Create an enhanced `handleIncrease` method



```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

Enhanced handleIncrease method

- Create an enhanced `handleIncrease` method
  - Pass a **number** as an **input parameter**

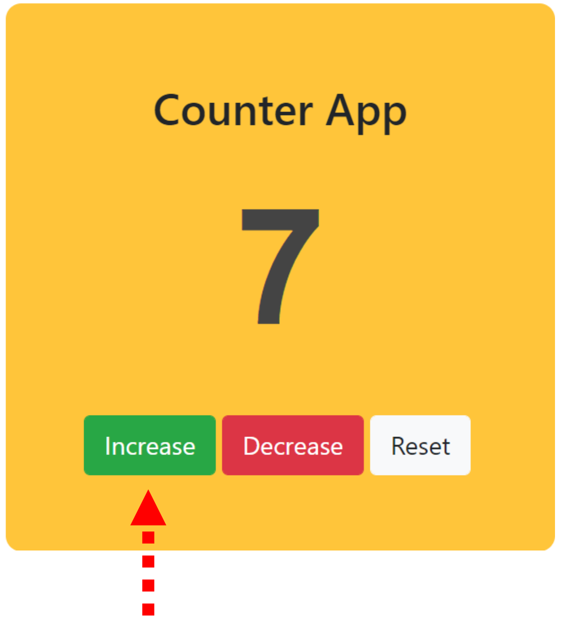


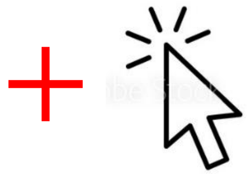
```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

Enhanced handleIncrease method

- Create an enhanced `handleIncrease` method
  - Pass a **number** as an **input parameter**
  - Press **shift key + mouse click** together to increase the count by 10

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```





Click Increase button

Enhanced handleIncrease method

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Enhanced handleIncrease method

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Enhanced handleIncrease method

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Enhanced handleIncrease method

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Enhanced handleIncrease method

```
// Method inside App Component
// Old increase count method
handleIncrease = () => {
  this.setState({
    count: this.state.count + 1
  })
}
```

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Demo

Event parameter

- Event parameter (e or event) provided by a Synthetic Event
- It is an object that provides useful metadata information
  - Different for different event handler types
  - onClick event handler provides information about shiftKey
    - Value of shiftKey is either `true` or `false`

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

Demo

Calling handleByShiftKey in render method

```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

```
// App Component
render () {
  return (
    <>
      ...
      <Button
        className="btn-success"
        onClick={this.handleByShiftKey(1)}
      >
          Increase
      </Button>
      ...
    </>
  );
}
```

Modular handleByShiftKey

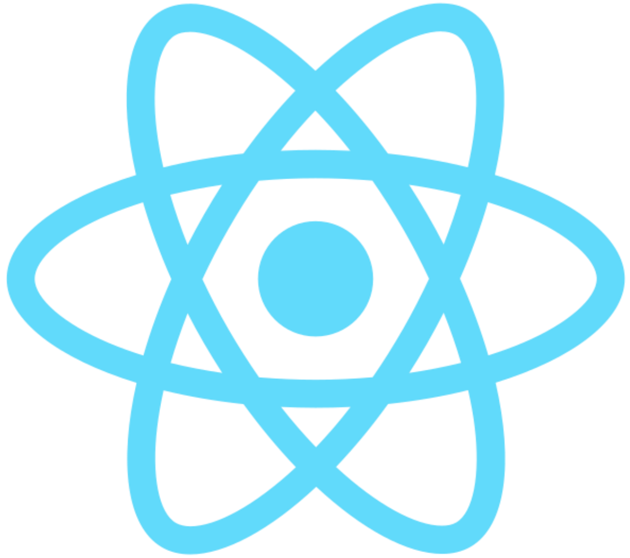
```
// Method inside App Component
// Enhanced increase count method
handleByShiftKey = (number) => (e) => {
  let current = this.state.count;
  if (e.shiftKey) {
    this.setState({
      count: current + (number*10),
    });
  } else {
    this.setState({
      count: current + number,
    });
  }
};
```

```
// Render method in App Component
render () {
  return (
  <>
    ...
    <Button
      className="btn-success"
      onClick={this.handleByShiftKey(1)}>
        Increase
    </Button>

    <Button
      className="btn-danger"
      onClick={this.handleByShiftKey(-1)}>
        Decrease
    </Button>
    ...
  </>
  );
}
```

Demo

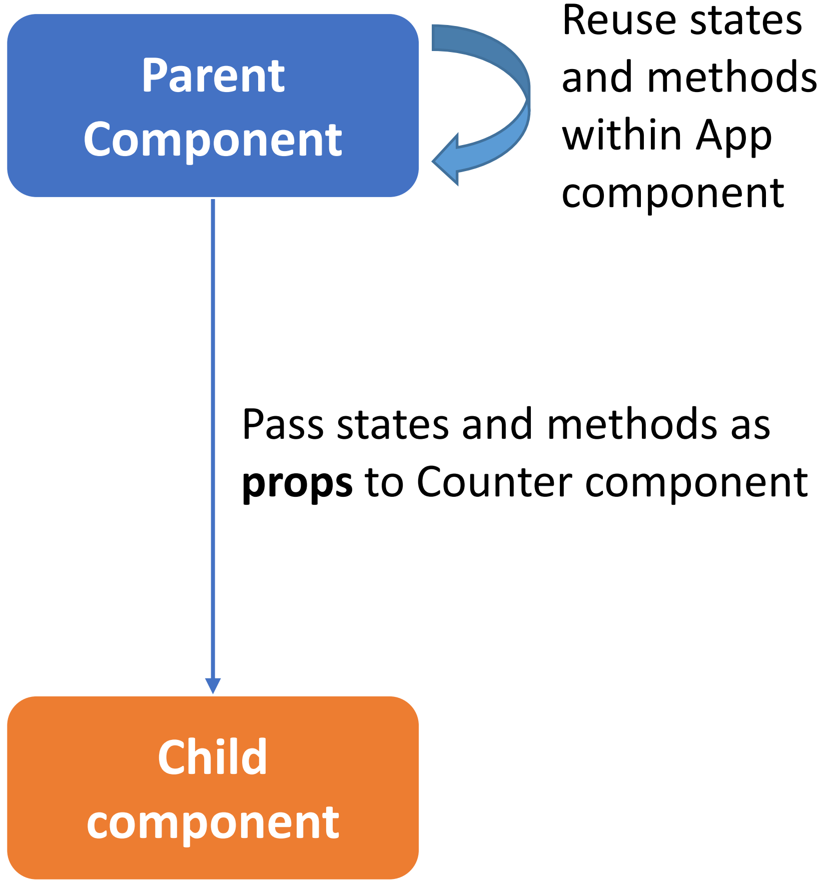
Objectives: React (1/2)

- Component
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event handlers
- **Props**
- Data binding

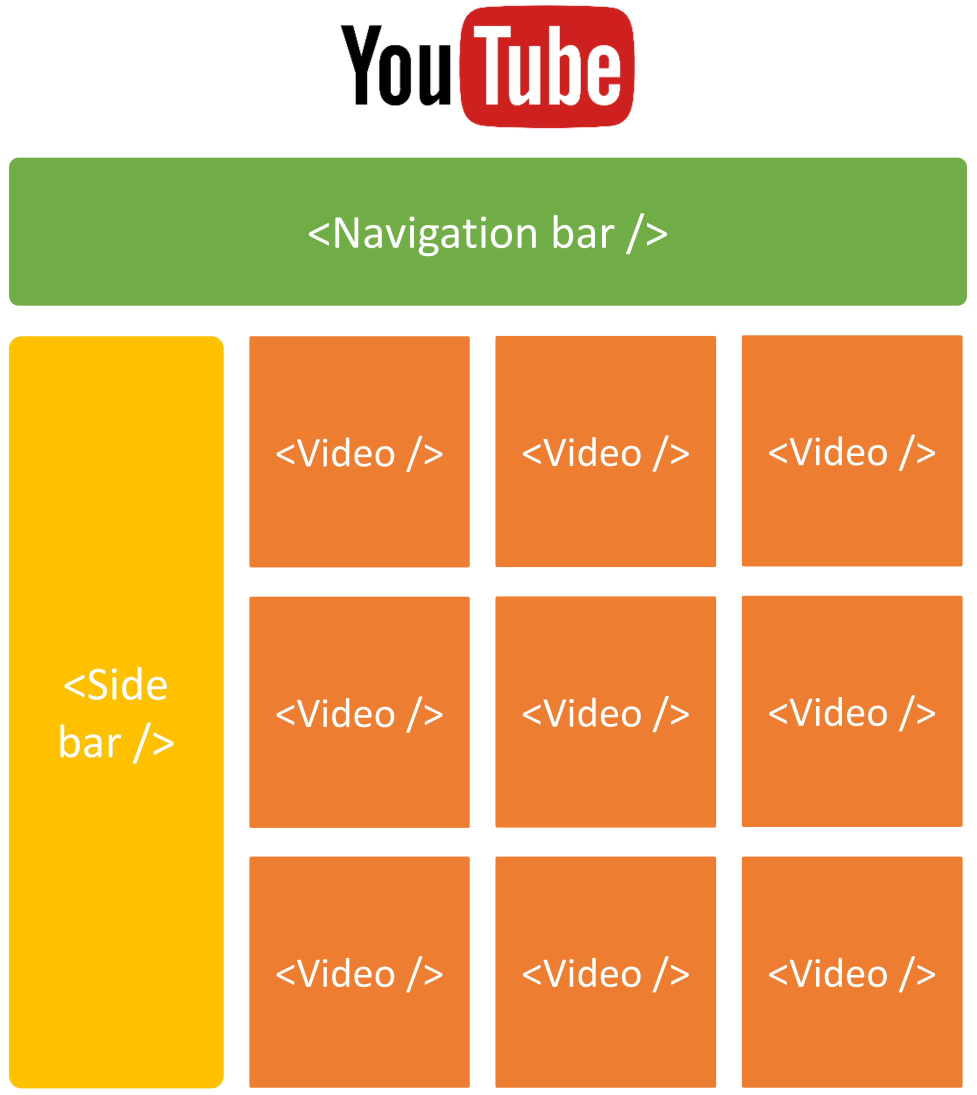


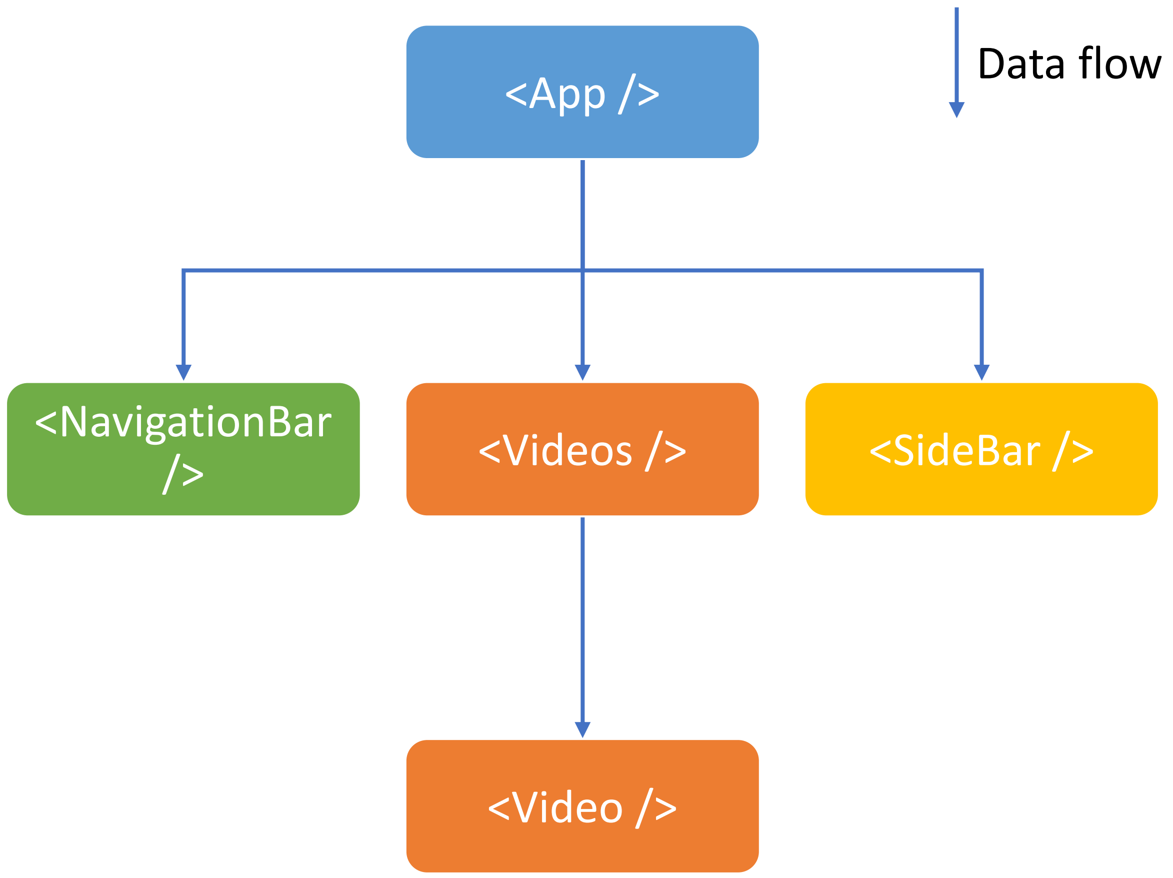
Props

- Props are short for properties
- States and methods of **parent component** are **passed as props** to **child components**
- Props are **immutable**
  - Whereas, states are **mutable**


Parent Component
Reuse states and methods within App component
Pass states and methods as props to Counter component
Child component

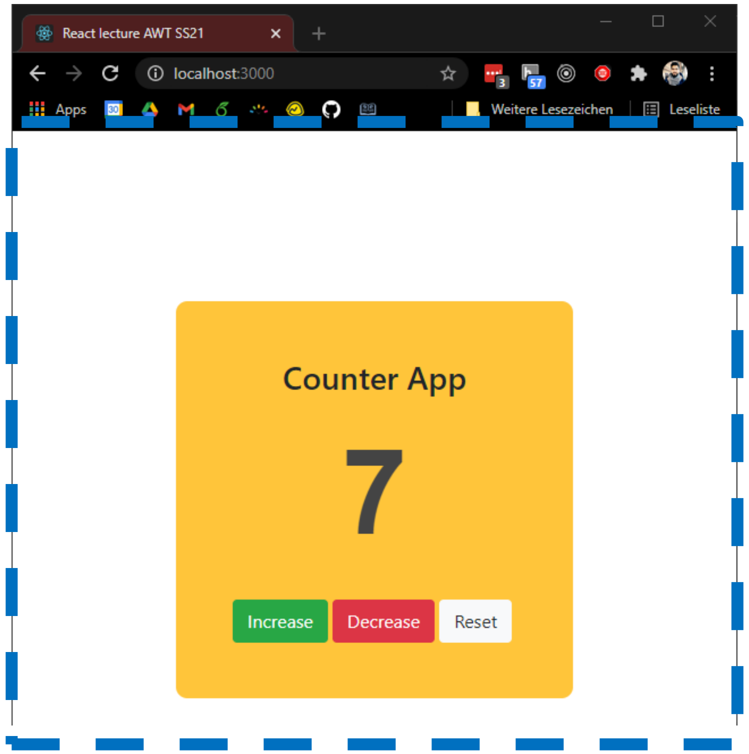
Recap: Component

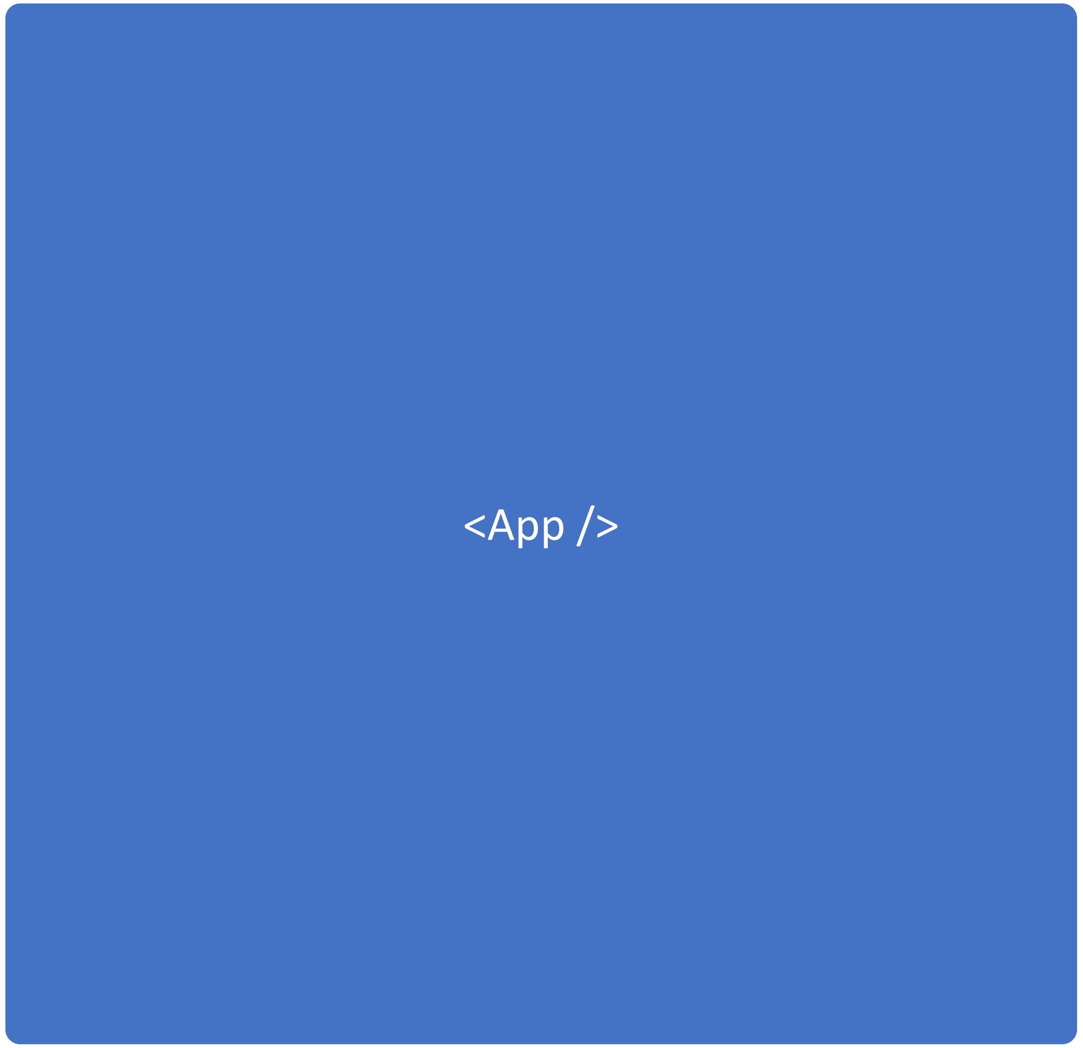




Props example: App Component

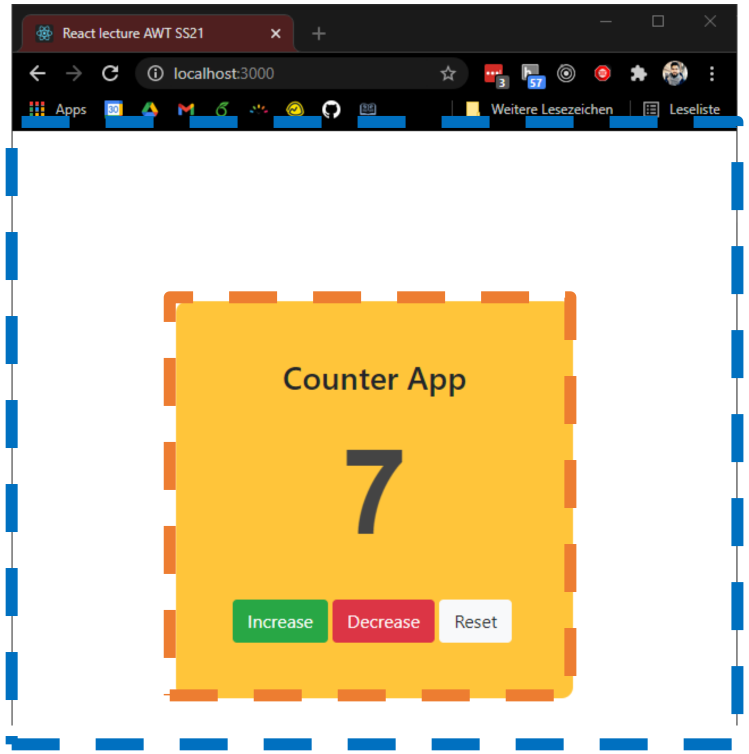
- Example →
  Create a child component inside our App component

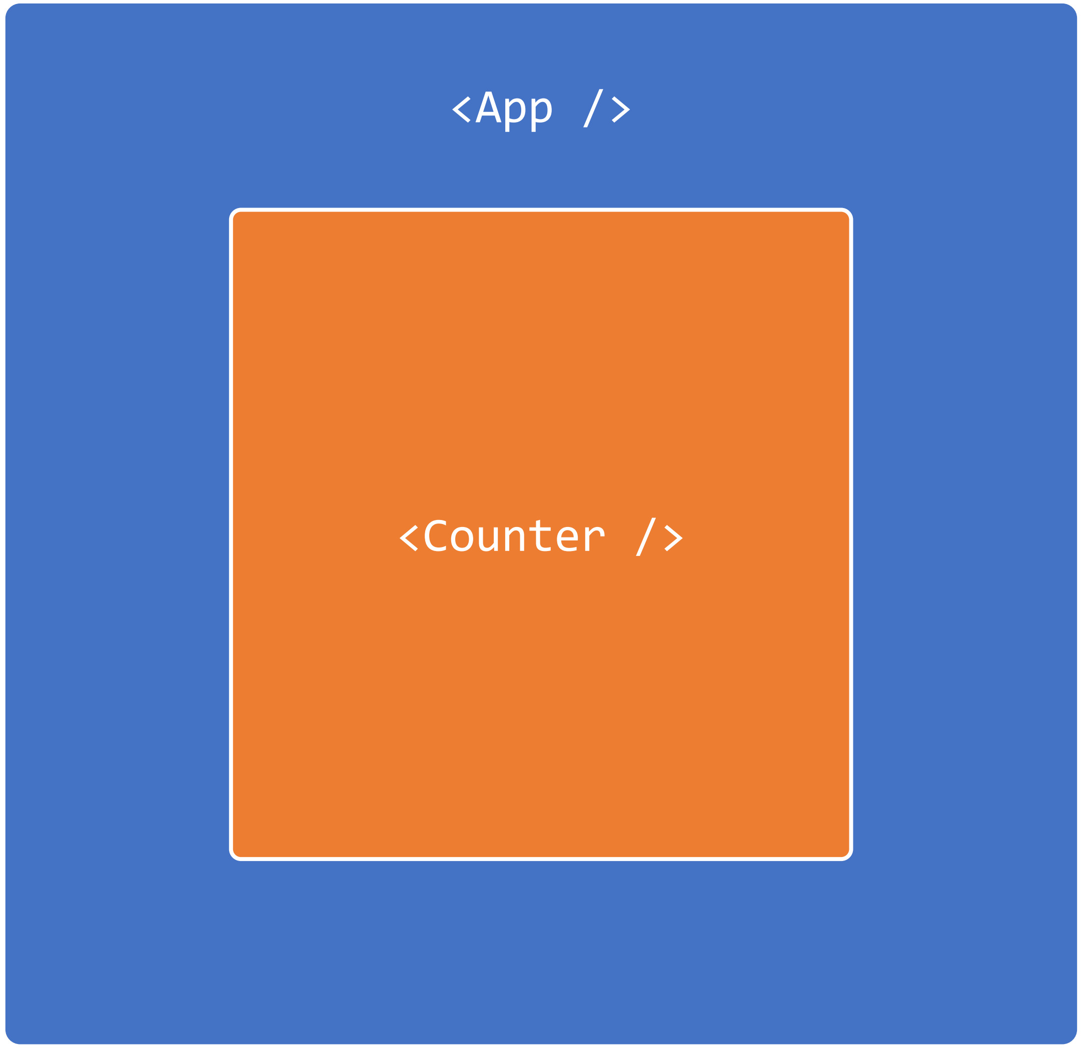




Props example: Create Counter Component

- Example →
  Create a child component inside our App component





Render method of App component

Render method in App component

- Name of the counter

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Render method of App component

Render method in App component

- Name of the counter
- Count value

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Render method of App component

Render method in App component

- Name of the counter
- Count value
- Increase button

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Render method of App component

Render method in App component

- Name of the counter
- Count value
- Increase button
- Decrease button

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Render method of App component

Render method in App component

- Name of the counter
- Count value
- Increase button
- Decrease button
- Reset button

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Pass states & methods to Counter component

```
// Inside render method of the App Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.state.name}</h3>
  </Col>
</Row>
<Row style={textStyle} className="justify-content-md-center">
  <Col>{this.state.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button
      className="btn-success" onClick={this.handleByShiftKey(1)}
    >
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.handleByShiftKey(-1)}
    >
      Decrease
    </Button>
    <Button className="btn-light" onClick={this.handleReset(0)}>
      Reset
    </Button>
  </Row>
</Container>
```

Create a child component and name it Counter

```
// App Component
render () {
  return (
    <>
      ...
       <Counter />
      ...
    </>
  );
}
```

Props

- Provide a prop name for each state and method to be sent as props
  - `name`
  - `count`
  - `increase`
  - `decrease`
  - `reset`

```
// Render method of App Component
render () {
  return (
  <>
     ...
      <Counter
        name={this.state.name}
        count={this.state.count}
        increase={this.handleByShiftKey(1)}
        decrease={this.handleByShiftKey(-1)}
        reset={this.handleReset(0)}
      />
     ...
  </>
  );
}
```

Access Props in Counter Component

- Access the props using `this.props`, therefore:
  - `this.props.name`

```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
    className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
           onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
           onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Access Props in Counter Component

- Access the props using `this.props`, therefore:
  - `this.props.name`
  - `this.props.count`

```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
     className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
            onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
            onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Access Props in Counter Component

- Access the props using `this.props`, therefore:
  - `this.props.name`
  - `this.props.count`
  - `this.props.increase`

```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
    className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
            onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
            onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Access Props in Counter Component

- Access the props using `this.props`, therefore:
  - `this.props.name`
  - `this.props.count`
  - `this.props.increase`
  - `this.props.decrease`

```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
     className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
           onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
           onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Access Props in Counter Component

- Access the props using `this.props`, therefore:
  - `this.props.name`
  - `this.props.count`
  - `this.props.increase`
  - `this.props.decrease`
  - `this.props.reset`

```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
     className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
           onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
           onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Counter component

- Counter component is now maintainable and reusable

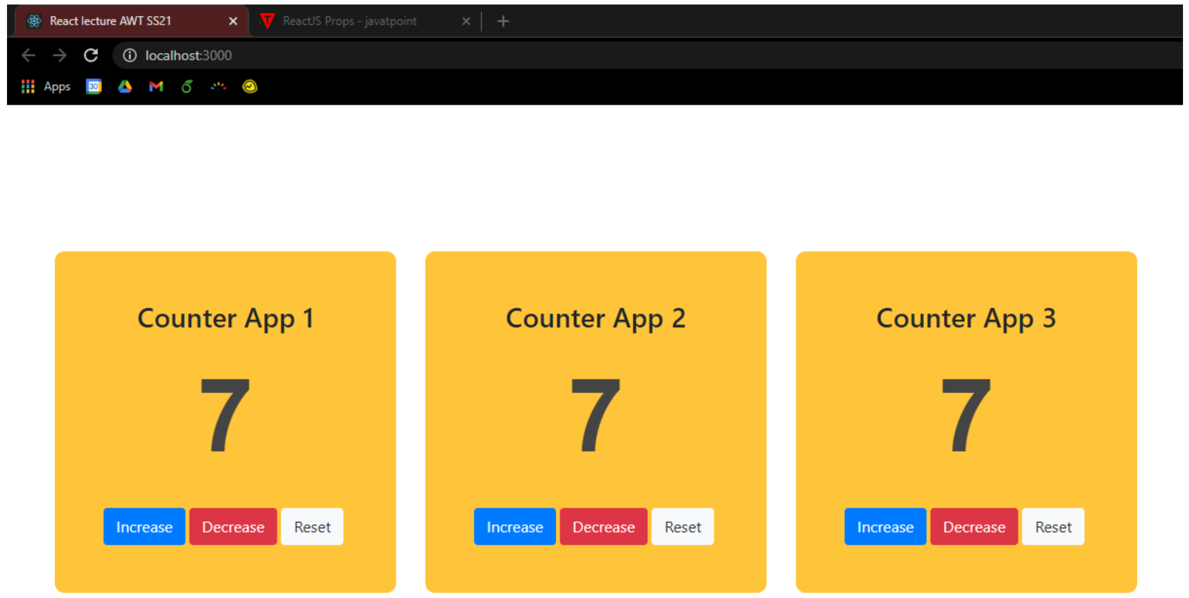
```
// Inside the render method of
App Component
<Counter
  name={this.state.name}
  count={this.state.count}
  increase={this.handleByShiftKey(1)}
  decrease={this.handleByShiftKey(-1)}
  reset={this.handleReset(0)}
/>
```

```
// Inside the render method of Counter Component
<Row className="justify-content-md-center">
  <Col>
    <h3>{this.props.name}</h3>
  </Col>
</Row>
<Row style={textStyle}
     className="justify-content-md-center">
  <Col>{this.props.count}</Col>
</Row>
<Container>
  <Row className="align-item-md-center">
    <Button className="btn-success"
            onClick={this.props.increase}>
      Increase
    </Button>
    <Button
      className="btn-danger" style={gutter}
      onClick={this.props.decrease}>
      Decrease
    </Button>
    <Button className="btn-light"
            onClick={this.props.reset}>
      Reset
    </Button>
  </Row>
</Container>
```

Demo

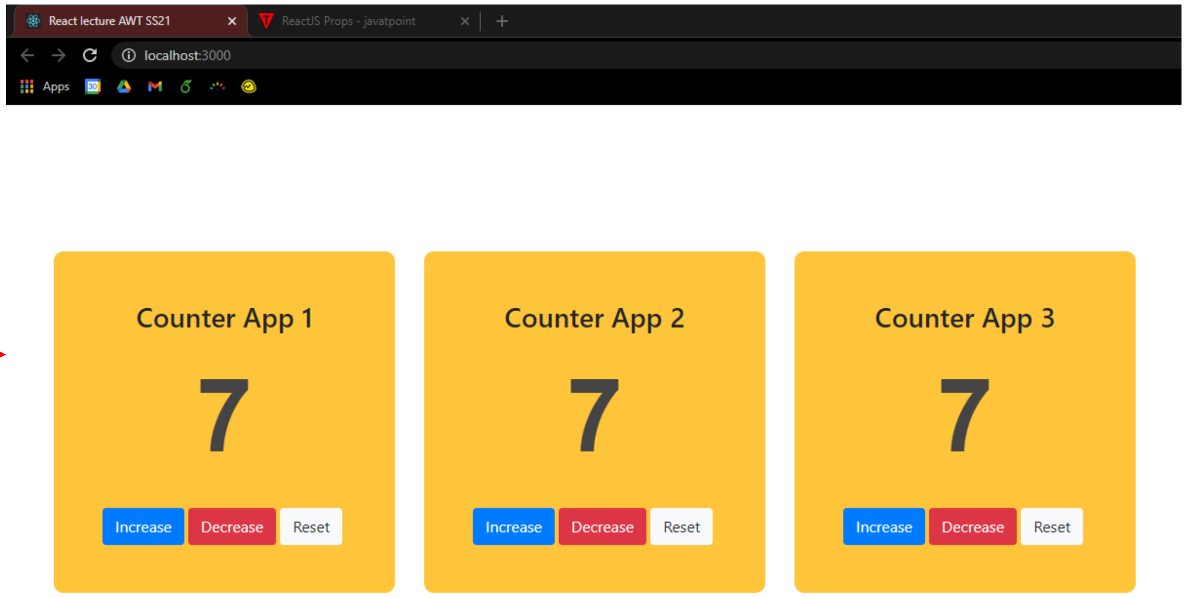
Multiple Counter components inside App

```
// Inside the render method of App Component
<Row>
  <Col>
    {/* Counter Component 1*/}
    <Counter
      name={"Counter App 1"}
      count={this.state.count}
      increase={this.handleByShiftKey(1)}
      decrease={this.handleByShiftKey(-1)}
      reset={this.handleReset(0)}
    />
  </Col>

  <Col>
    {/* Counter Component 2*/}
    <Counter
      name={"Counter App 2"}
      count={this.state.count}
      increase={this.handleByShiftKey(1)}
      decrease={this.handleByShiftKey(-1)}
      reset={this.handleReset(0)}
    />
  </Col>

  <Col>
    {/* Counter Component 3*/}
    <Counter
      name={"Counter App 3"}
      count={this.state.count}
      increase={this.handleByShiftKey(1)}
      decrease={this.handleByShiftKey(-1)}
      reset={this.handleReset(0)}
    />
  </Col>
</Row>
```



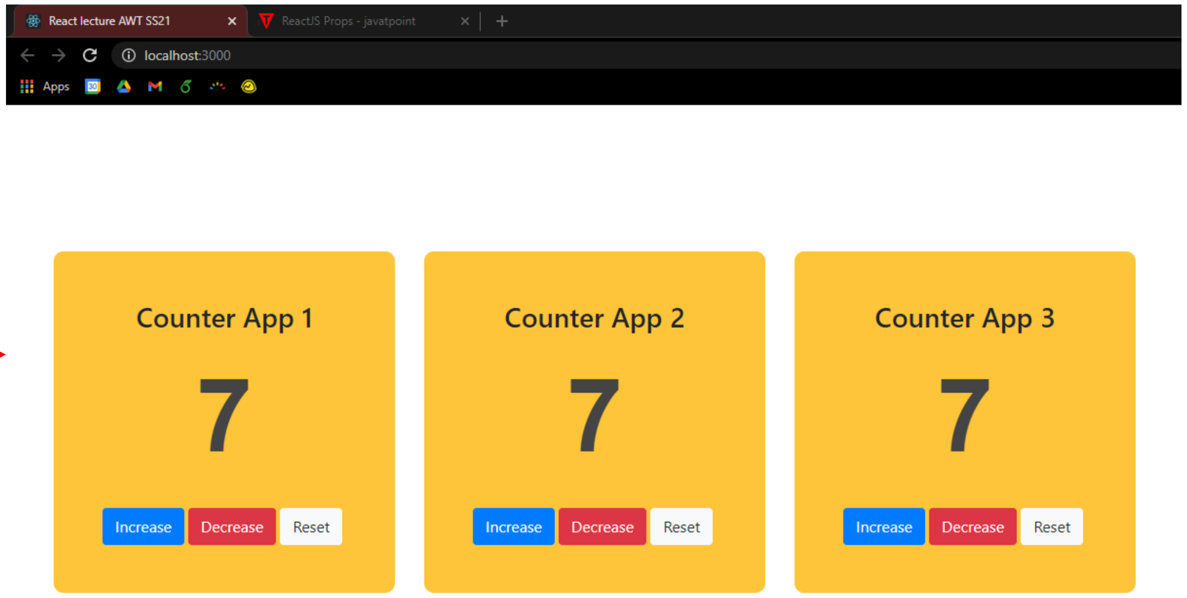
Use map function

```
// Render method of App Component
render() {
  // Styling
  let containerStyle = {marginTop: 150};

  let counterArr = ["Counter App 1", "Counter App 2",
                    "Counter App 3"]

  return (
    <>
      <Container style={containerStyle}>
        <Row>
          {counterArr.map((counter, index) => {
            return (
              <Col key={index}>
                <Counter
                  name={counter}
                  count={this.state.count}
                  increase={this.handleByShiftKey(1)}
                  decrease={this.handleByShiftKey(-1)}
                  reset={this.handleReset(0)}
                />
              </Col>
            )
          })}
        </Row>
      </Container>
    </>
  );
}
```



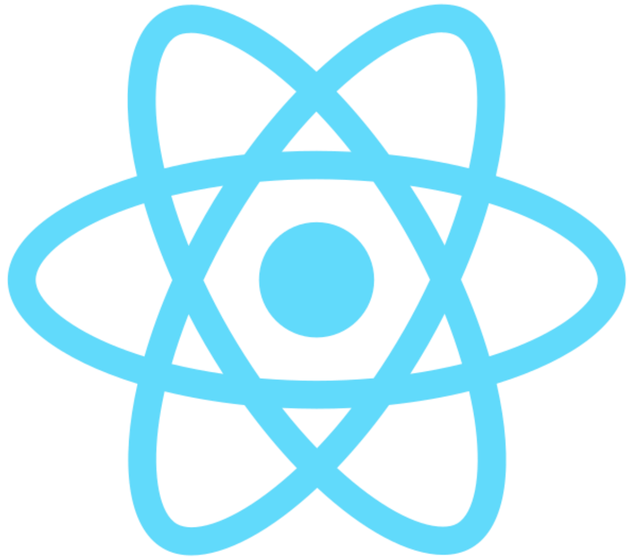
Use map function

```
// Render method of App Component
render() {
  // Styling
  let containerStyle = {marginTop: 150};

  let counterArr = ["Counter App 1", "Counter App 2",
                    "Counter App 3"]

  return (
    <>
      <Container style={containerStyle}>
        <Row>
          {counterArr.map((counter, index) => {
            return (
              <Col key={index}>
                <Counter
                  name={counter}
                  count={this.state.count}
                  increase={this.handleByShiftKey(1)}
                  decrease={this.handleByShiftKey(-1)}
                  reset={this.handleReset(0)}
                />
              </Col>
            )
          })}
        </Row>
      </Container>
    </>
  );
}
```



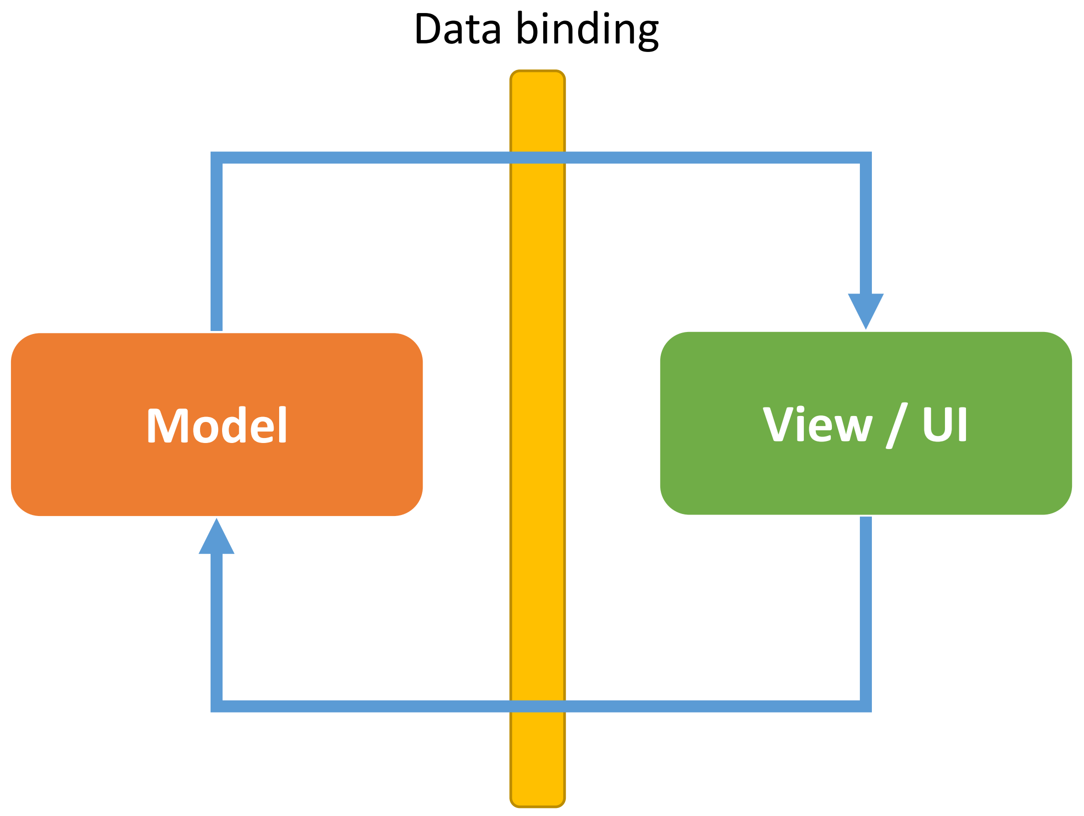
Objectives: React (1/2)

- Component
- States
- React Element and Virtual DOM
- JavaScript XML (JSX)
- Event handlers
- Props
- **Data binding**



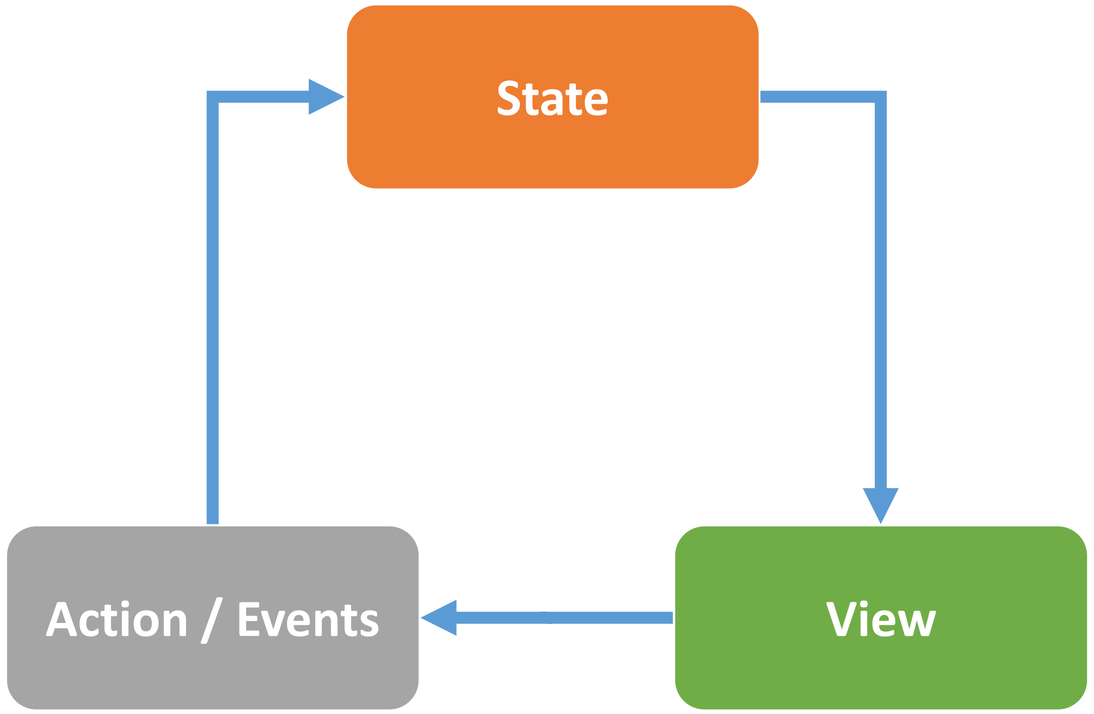
Data binding

- The connection between the Model and the View → Data binding
- Any change in the model (State) gets reflected inside the view (UI)
- Any change in the view (UI) gets reflected inside the component's logic (State)
- There are two types:
  - One-way data binding
  - Two-way data binding



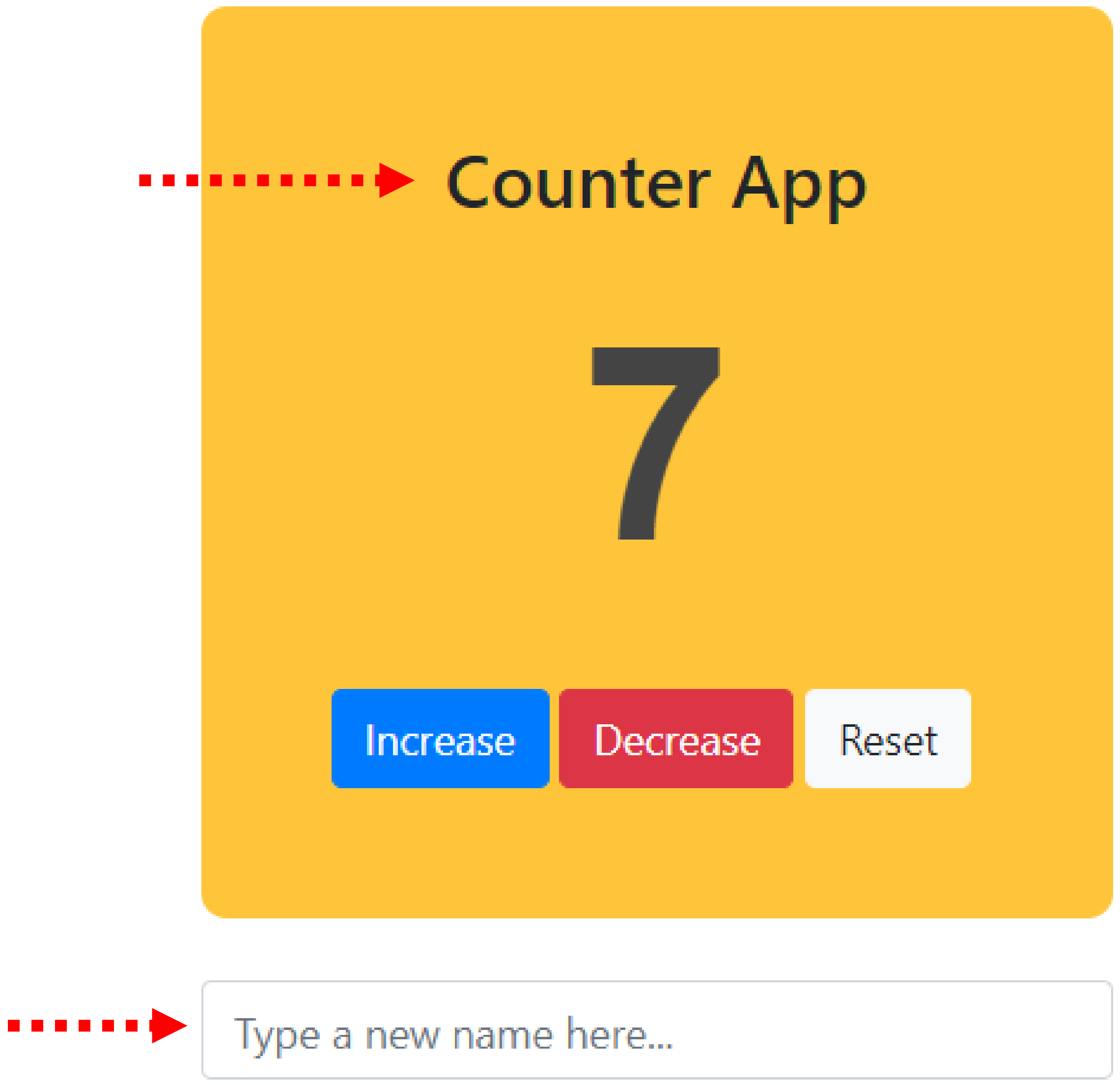
One-way data binding

- React support only one-way data binding
- Bound to only one-way data connection
- View cannot manipulate State directly in React
  - Only way is through an Event (Action)



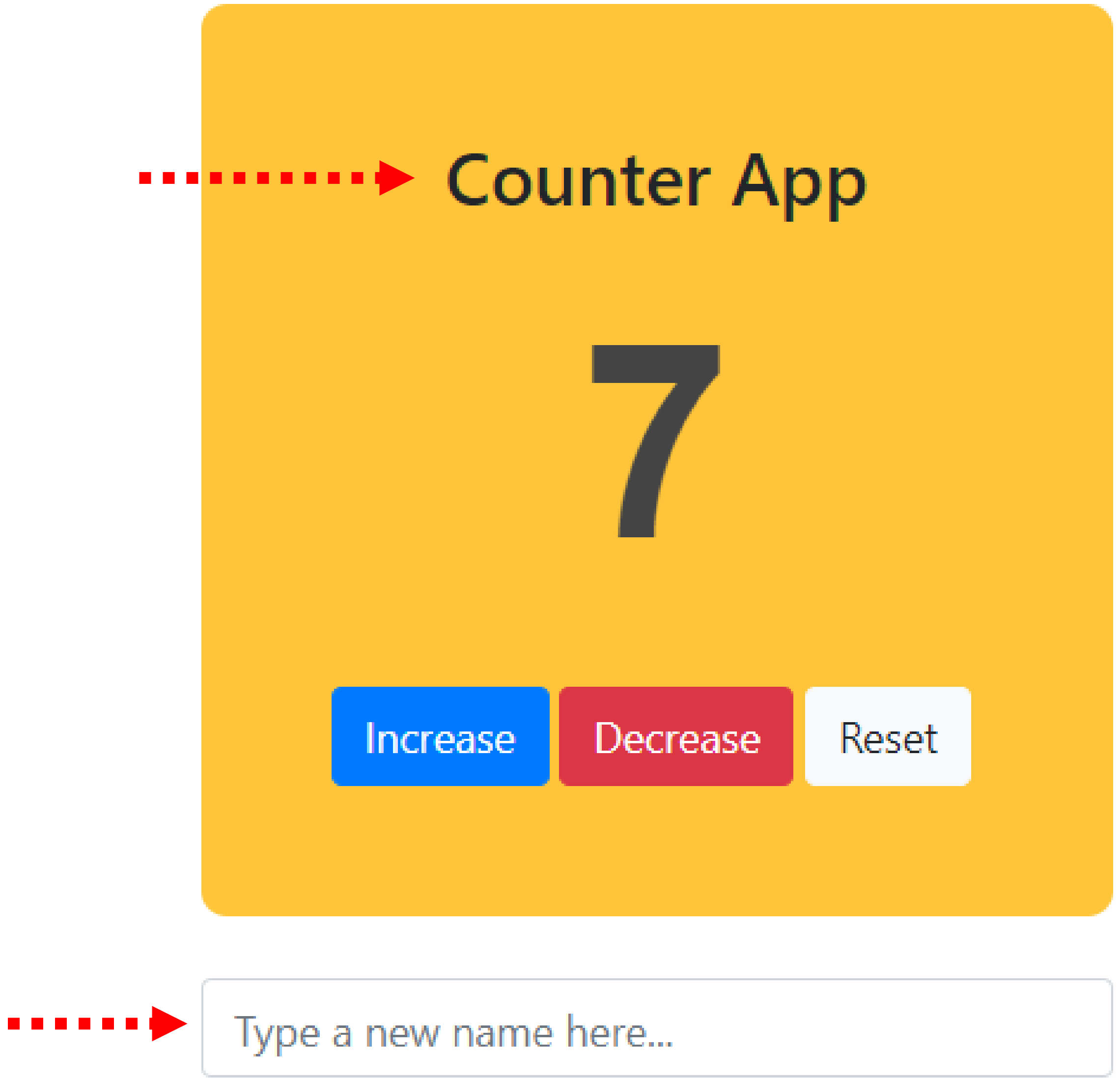
Example

- In one way data binding → **An event is required to change the state**
- In two way data binding → **Directly change the model from the view**



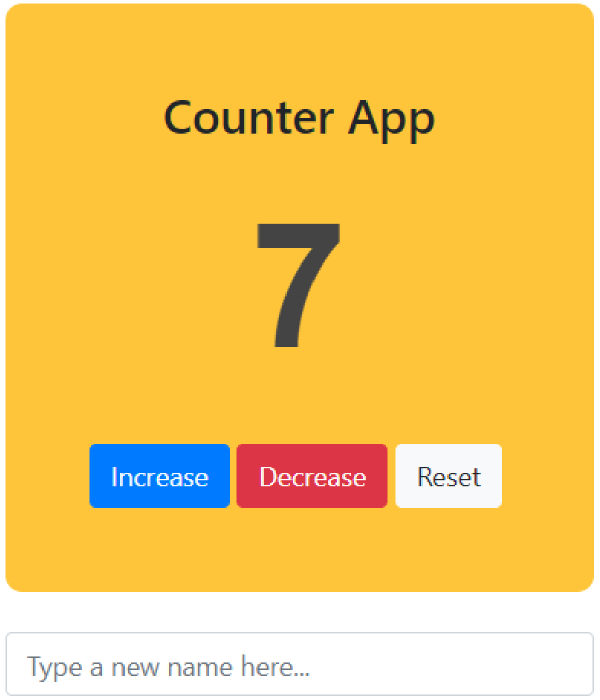
One-way data binding

- Example →
  Use an input field to change the name of the application



One-way data binding

- `<input />` (`FormControl`[1]) is controlled by the render function

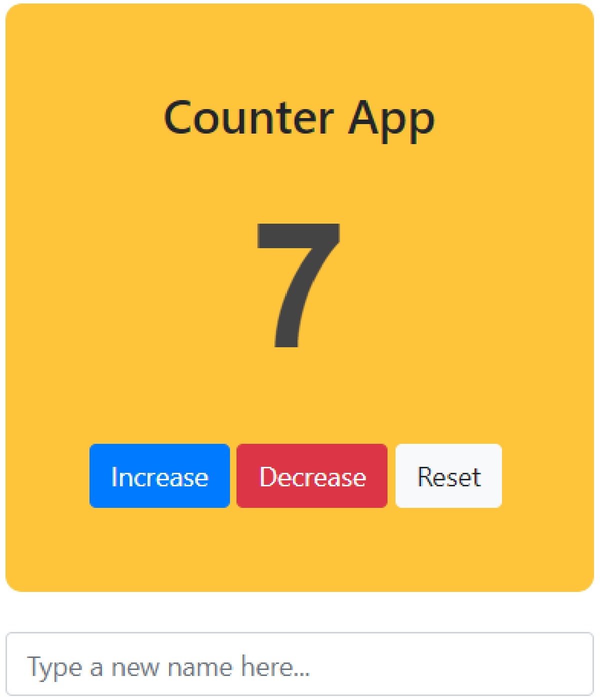


```
// Render method of Counter Component
render () {
 return (
 <>
   ...
   <Row style={inputText}>
    <FormControl
      placeholder="Type a new name here..."
      onChange={this.handleChangeName}
    />
   </Row>
   ...
 </>
 );
}
```

[1] https://react-bootstrap.github.io/components/forms/

One-way data binding

- `<input />` (FormControl[1]) is controlled by the render function



```
// Render method of Counter Component
render () {
 return (
 <>
  ...
  <Row style={inputText}>
   <FormControl
     placeholder="Type a new name here..."
     onChange={this.handleChangeName}
   />
  </Row>
  ...
 </>
 );
}
```

[1] https://react-bootstrap.github.io/components/forms/

One-way data binding

- `<input />` (FormControl[1]) is controlled by the render function
- The only way to change the State from View is through some Events
- Attach an onChange event to the `<FormControl />`

```
// Render method of Counter Component
render () {
 return (
 <>
  ...
  <Row style={inputText}>
   <FormControl
    placeholder="Type a new name here..."
    onChange={this.handleChangeName}
   />
  </Row>
  ...
 </>
 );
}
```

[1] https://react-bootstrap.github.io/components/forms/

One-way data binding

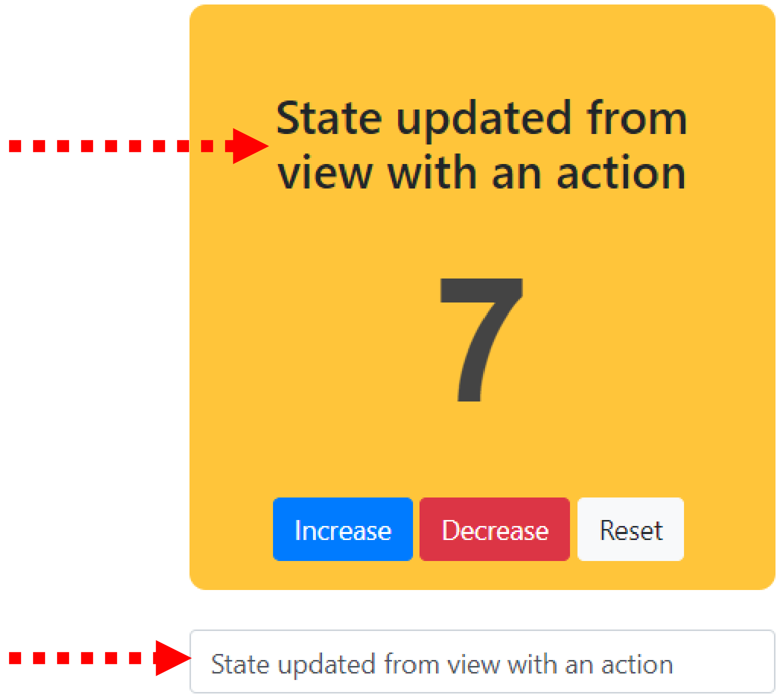
- handleChangeName method calls the setState method to update name
- Event parameter "e" from the Synthetic event onChange provides useful metadata information such as id, value, name etc.

```
class Counter extends Component {
 constructor(props) {
  super(props);
  this.state = {
   count: 7,
   name: "Counter App",
 };
}

...
handleChangeName = (e) => {
  this.setState({
    name: e.target.value,
  });
};
...
```

Demo

One-way data binding

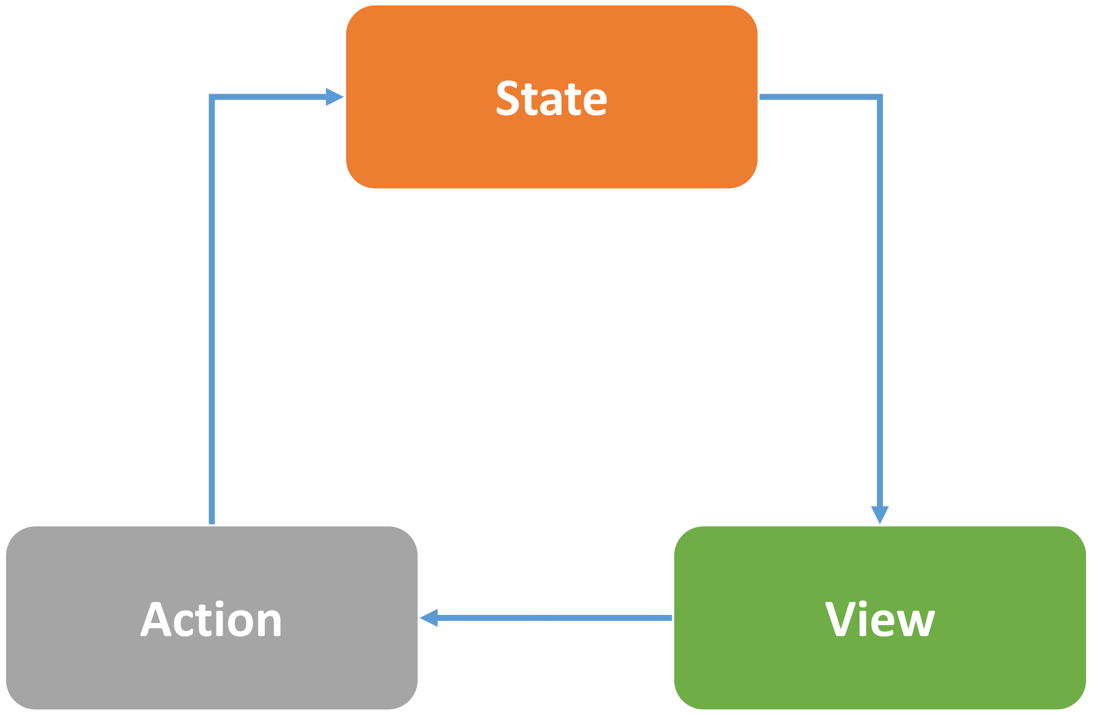
- Direct manipulation of state not possible from view
- The state changes due to an event caused from the view



State updated from view with an action

7

State updated from view with an action



References

State

- https://daveceddia.com/why-not-modify-react-state-directly/
- https://stackoverflow.com/questions/37755997/why-cant-i-directly-modify-a-components-state-really
- https://www.javatpoint.com/react-state

React Element & JavaScript XML (JSX)

- https://www.javatpoint.com/react-fragments
- https://stackoverflow.com/questions/47761894/why-are-fragments-in-react-16-better-than-container-divs
- https://babeljs.io/docs/en/
- https://babeljs.io/repl

Event handlers

- https://www.freecodecamp.org/news/javascript-events-explained-in-simple-english/
- https://gist.github.com/fongandrew/f28245920a41788e084d77877e65f22f
- https://reactjs.org/docs/events.html
- https://reactjs.org/docs/handling-events.html
- https://www.w3schools.com/react/react_events.asp

References

- https://dev.to/nagwan/react-synthetic-events-34e5
- https://stackoverflow.com/questions/42597602/react-onclick-pass-event-with-parameter
- https://stackoverflow.com/questions/32782922/what-do-multiple-arrow-functions-mean-in-javascript
- https://medium.com/byte-sized-react/what-is-this-in-react-25c62c31480#:~:text=The%20'this'%20keyword%20typically%20references,or%20context%20of%20its%20use.
- https://stackoverflow.com/questions/38046970/react-component-this-is-not-defined-when-handlers-are-called
- https://gist.github.com/dfoverdx/2582340cab70cff83634c8d56b4417cd

Props

- https://www.javatpoint.com/react-props
- https://reactjs.org/docs/components-and-props.html
- https://www.w3schools.com/react/react_props.asp
- https://ui.dev/react-router-v4-pass-props-to-components/

Data binding

- https://stackoverflow.com/questions/34519889/can-anyone-explain-the-difference-between-reacts-one-way-data-binding-and-angula

Additional resources

Example source codes used in lectures

- https://github.com/ude-soco/AdvWebTech-React-Lecture