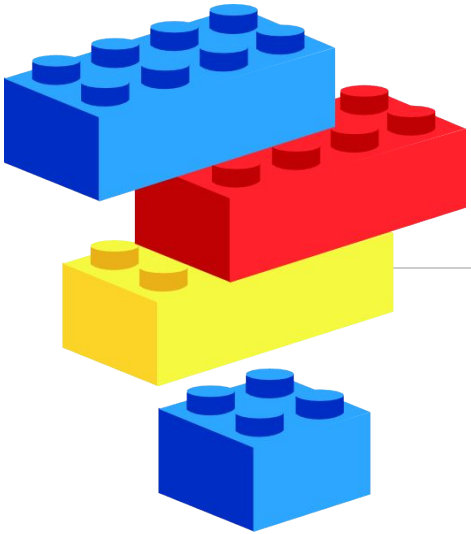
SENG 365 Week 8

React: JSX and Components

This week

- JSX
- Class and Function components

JSX and Components



Creating a React project

- Several toolchains available
  - Create React App easiest way to start with a project from scratch
  - See also: https://reactjs.org/docs/create-a-new-react-app.html
  - You can also modify an existing project (w/o toolchain) by adding React JS using script tags and an appropriate JSX preprocessor

```
npx create-react-app my-app
cd my-app
npm start
```

What's up?...

Notifications Messages

User Status

15 minutes ago

26 Likes 6 Comments 3 Shares

16 3

Title

Bacon ipsum dolor amet pork chop pig commodo, biltong ham hock adipisicing venison rump spare ribs ut. Incididunt alcatra lorem sint tail strip steak

Username commented on your photo:

Bacon ipsum dolor amet pork chop pig commodo, biltong ham hock.

just now

Username added a new gallery.

2h

Username is now following you.

4h

Username is now following you.

5h

Username

Username

Username

Username

Username

Username

Username

Many ways to group files

```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```

```
api/
  APIUtils.js
  APIUtils.test.js
  ProfileAPI.js
  UserAPI.js
components/
  Avatar.js
  Avatar.css
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
```

No one right way

Think about `import` statements

Consistency

Anatomy of a React app

```
import ReactDOM from "react-dom"
import App from "./App"

ReactDOM.render(<App />, document.getElementById("root"))
```

Anatomy of a React app

```
import React from 'react'
import ReactDOM from 'react-dom/client'

import App from './App'

ReactDOM.createRoot(document.getElementById('root')).render(<App />)
```

This has changed slightly in React 18

Anatomy of a React app

```
import ReactDOM from "react-dom"
import App from "./App"

ReactDOM.render(<App />, document.getElementById("root"))
```

ReactDOM is a JavaScript library that renders JSX to elements in the document object model (DOM)

Anatomy of a React app

```
import ReactDOM from "react-dom"
import App from "./App"

ReactDOM.render(<App />, document.getElementById("root"))
```

Application components (e.g. `App`) are written in JSX

Imported components can have `.js` or `.jsx` extension and it does not need to be indicated in the `import` statement

Anatomy of a React app

```
import ReactDOM from "react-dom"
import App from "./App"

ReactDOM.render(<App />, document.getElementById("root"))
```

The second parameter is the HTML element that the compiled JSX should be attached to (in this case the element with id `root`)

JSX basics

- JSX is a syntax extension of JavaScript
- Looks a bit like a mix of JS and HTML
- Compiled to HTML before running in the browser
- Your project needs a JSX preprocessor (this is already installed by Create React App)

```
npm install babel-cli@6 babel-preset-react-app@3
```

JSX examples

Single element

```
const title = <h1>Welcome all!</h1>
```

JSX examples

Single element with attributes (like HTML)

```
const example = <h1 id="example">JSX Attributes</h1>;
```

JSX examples

Multiline and nested expressions

- Requires surrounding brackets: ( )
- Must be **only one** outermost tag (e.g. `<ul> </ul>`)

```
const myList = (
  <ul>
    <li>item 1</li>
    <li>item 2</li>
    <li>item 3</li>
  </ul>
);
```

JSX examples

Can contain evaluated JavaScript

- Denoted by curly brackets: { }

```
let expr = <h1>{10 * 10}</h1>;
// above will be rendered as <h1>100</h1>
```

JSX conditionals

Can be tricky and there is more than one approach

1. JavaScript Boolean short circuit evaluation

```
// All of the list items will display if
// baby is false and age is above 25
const tasty = (
  <ul>
    <li>Applesauce</li>
    { !baby && <li>Pizza</li> }
    { age > 15 && <li>Brussels Sprouts</li> }
    { age > 20 && <li>Oysters</li> }
    { age > 25 && <li>Grappa</li> }
  </ul>
);
```

JSX conditionals

Can be tricky and there is more than one approach

2. Ternary operator <expr> ? <expr> : <expr>

```
// Using ternary operator
const headline = (
  <h1>
    { age >= drinkingAge ? 'Buy Drink' : 'Do Teen Stuff' }
  </h1>
);
```

Arrays and JSX collections

Use `map` function to generate a collection from an array

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

<ul>{listItems}</ul>
```

But this generates a warning message that the list items should have a key

List item keys in JSX

- Keys are necessary because they tell React when a render needs to happen because a list element has changed or is added/removed.

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

Key as id field in list object

```
function Car(props) {
  return <li>I am a { props.brand }</li>;
}

function Garage() {
  const cars = [
    {id: 1, brand: 'Ford'},
    {id: 2, brand: 'BMW'},
    {id: 3, brand: 'Audi'}
  ];
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <ul>
        {cars.map((car) => <Car key={car.id} brand={car.brand} />)}
      </ul>
    </>
  );
}
```

JSX compilation

JSX is syntactic sugar for JavaScript that calls `React.createElement`

```
const App = () => {
  const now = new Date()
  const a = 10
  const b = 20

  return (
    <div>
      <p>Hello world, it is {now.toString()}</p>
      <p>
        {a} plus {b} is {a + b}
      </p>
    </div>
  )
}
```

```
const App = () => {
  const now = new Date()
  const a = 10
  const b = 20
  return React.createElement(
    'div',
    null,
    React.createElement(
      'p', null, 'Hello world, it is ', now.toString()
    ),
    React.createElement(
      'p', null, a, ' plus ', b, ' is ', a + b
    )
  )
}
```

React.createElement

Takes three parameters:

type, props, children

It returns a JavaScript object

```
> React.createElement('div', { }, 'Hello')
<· ▼{$$typeof: Symbol(react.element), type:
    $$typeof: Symbol(react.element)
    key: null
  ▶ props: {children: "Hello"}
    ref: null
    type: "div"
    _owner: null
  ▶ __proto__: Object
```

React.createElement

React elements can be nested in children

`ReactDOM.render` is passed one of these nested objects

```
React.createElement('div', { }, React.createElement('p', {}, 'A p inside a div'))
▼{$$typeof: Symbol(react.element), type: "div", key: null, ref: null, props: {…}, …}
  $$typeof: Symbol(react.element)
  key: null
  ▼props:
    ▼children:
      $$typeof: Symbol(react.element)
      key: null
      ▼props:
        children: "A p inside a div"
        ▶__proto__: Object
      ref: null
      type: "p"
      _owner: null
      ▶__proto__: Object
    ▶__proto__: Object
  ref: null
  type: "div"
  _owner: null
  ▶__proto__: Object
```

Defining your own Components

Class component

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

Defining your own Components

Class component

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

Function component

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

Equivalent

Composing Components

`props` stands for properties and is a read-only object that is passed to the component

Components in JSX have to start with a capital letter to differentiate them from HTML tags

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

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

What does this render?

```
const Hello = (props) => {
  return (
    <div>
      <p>
        Hello {props.name}, you are {props.age} years old
      </p>
    </div>
  )
}

const App = () => {
  const name = 'Peter'
  const age = 10

  return (
    <div>
      <h1>Greetings</h1>
      <Hello name="Maya" age={26 + 10} />
      <Hello name={name} age={age} />
    </div>
  )
}
```

What does this render?

```
const Hello = (props) => {
  return (
    <div>
      <p>
        Hello {props.name}, you are {props.age} years old
      </p>
    </div>
  )
}

const App = () => {
  const name = 'Peter'
  const age = 10

  return (
    <div>
      <h1>Greetings</h1>
      <Hello name="Maya" age={26 + 10} />
      <Hello name={name} age={age} />
    </div>
  )
}
```

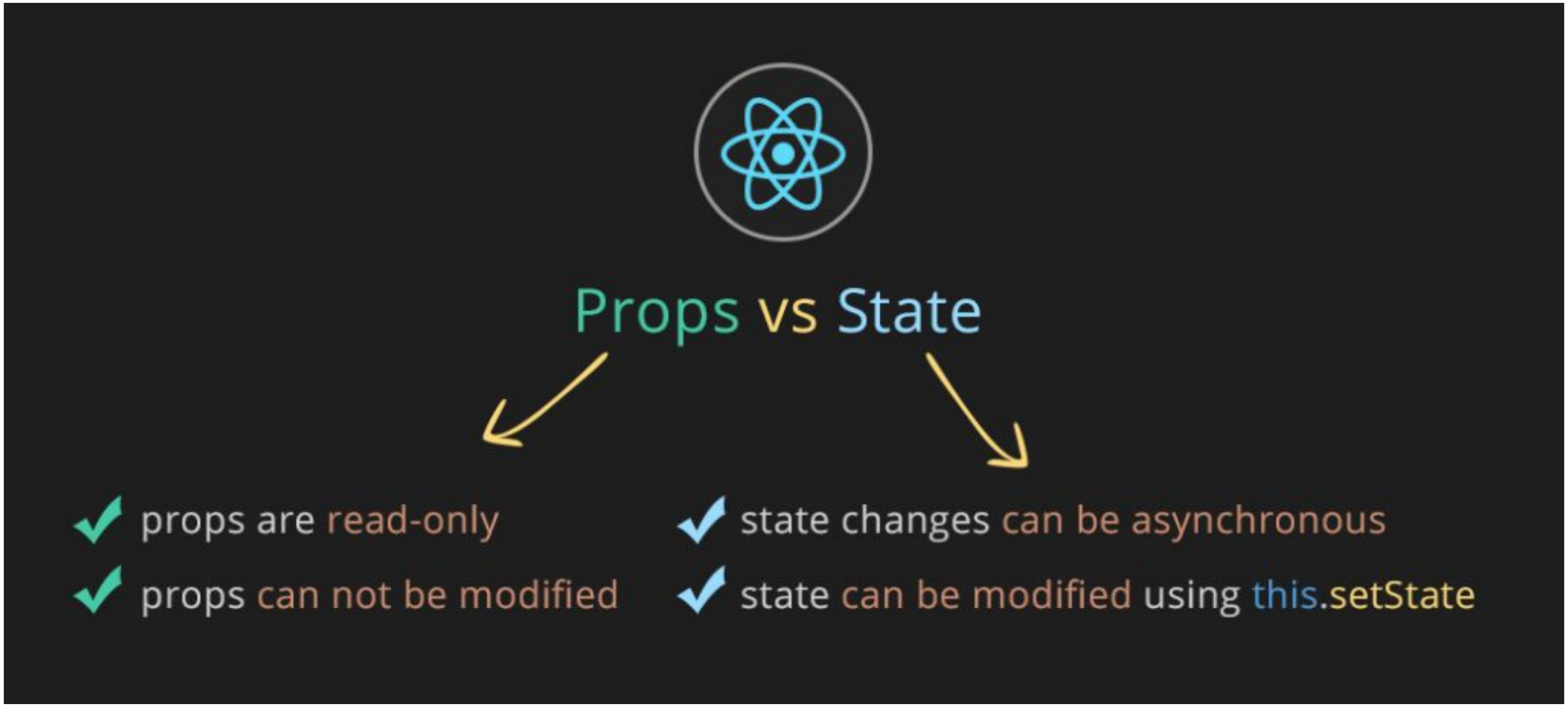
Greetings

Hello Maya, you are 36 years old

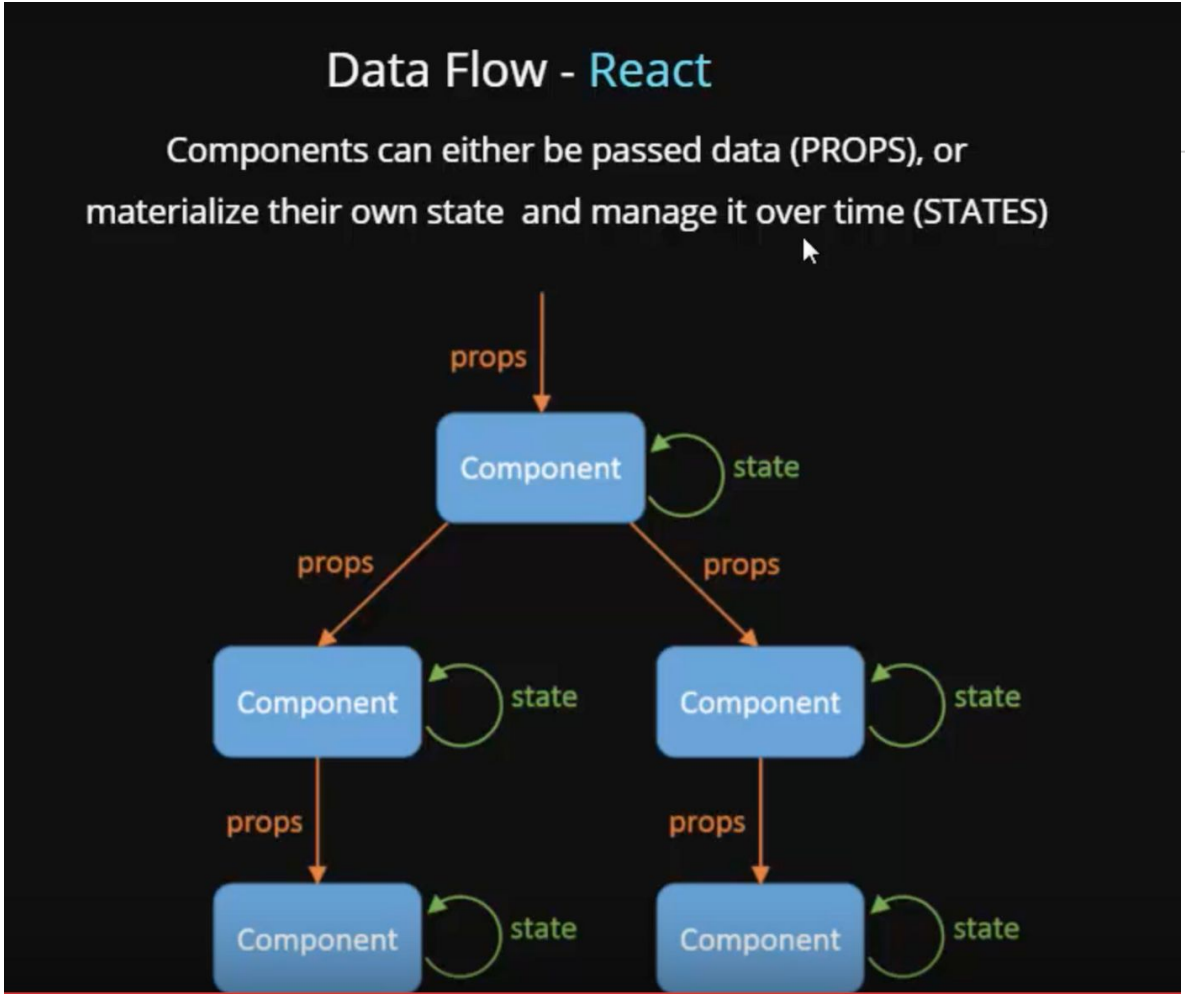
Hello Peter, you are 10 years old

Component lifecycle and managing state

Making components stateful



Data flow


Data Flow - React
Components can either be passed data (PROPS), or
materialize their own state and manage it over time (STATES)
props
Component
state
props
props
Component
state
Component
state
props
props
Component
state
Component
state

State and Lifecycle in Class Components

Class components extend `React.Component`.

`this.state` is the component's state object

It is initialized in the component's constructor, which takes `props` as a parameter. It should always call `super(props);` at the beginning.

The state object is how we change the view (made by the `render` method) based on events.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
```

State and Lifecycle in Class Components

Every React has a lifecycle, accessed through component methods, e.g.:

- `componentDidMount`
- `componentDidUpdate`
- `componentWillMount`

`this.state` should not be set directly, instead use `this.setState()`

`this.setState()` is a *request* to change the state, not updated immediately

The object sent to `setState` is *merged* with the existing state in a batch operation

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }
```

State and Lifecycle in Class Components

The state can be referenced with `this.state` in the `render()` method

You should **not** update the state in `render()`

Why?

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
```

State and Lifecycle in Class Components

The state can be referenced with `this.state` in the `render()` method

You should **not** update the state in `render()`

Why?

The state update will trigger a new render of the view.

In this example the state is updated every second using `setInterval`. How about events triggered by user actions?

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
```