REACT – The complete guide - Maximilian

React is a library for creating highly reactive and superfast js based web applications.

JS runs in the browser.

React is a JS library for building user interfaces

React is writing custom HTML elements

React is all about writing components

Codepen.io – to create a workspace for writing html/css/js code

ReactDOM is for rendering the components to real DOM

In its basic form, a React component is just a function.

The syntax React uses is called jsx.

Html --

<div id="p1"> </div>

<div class="person">

<h1> Hello </h1>

<p> age: 29 </o>

</div>

<div id='p2'> <h3> Romeo </h3> </div>

--

Css –

.person {

color: red

}

---

Js –

function Person(){

return (

<div class=”person”>

<h1> Max </h1>

<p> Your age: 28 </p>

</div>

);

}

ReactDOM.render(<Person />, document.querySelector('#p1'));

Output:

# Max

Your age: 28

# Hello

age: 29

### Romeo

Html –

<div id="p1"> </div>

<div class="person">

<h1> Hello </h1>

<p> age: 29 </o>

</div>

<div id='p2'> <h3> Romeo </h3> </div>

--

Css --

.person {

color: red

}

.person1 {

color: green

}

--

function Person(props){

return (

<div class="person1" >

<h1> {props.name} </h1>

<p> {props.age} </p>

</div>

);

}

ReactDOM.render(<Person name="Jaison" age="45" />, document.querySelector('#p1'));

---

# Jaison

45

# Hello

age: 29

### Romeo

Html –

<div id='app'</div>

--

Css –

.person {

color: red

}

.person1 {

color: orange

}

---

function Person(props){

return (

<div class="person1" >

<h1> {props.name} </h1>

<p> {props.age} </p>

</div>

);

}

var app = (<div> <Person name="Jaison" age="45" /><Person name="Sonia" age="39" /> </div>)

ReactDOM.render(app, document.querySelector('#app'));

---

Output:

# Jaison

45

# Sonia

39

In 3 ways, the component is written. 1) by function and render. 2) by function with parameter and render with argument. 3) create function with parameter and define the property, declare variable with function property definition, render with variable. With this approach, can create single page applications.

Angular an Vue are alternatives to React. JQuery is more about traversing the DOM and targeting the DOM elements.

Single page and multi page applications.

In a single page application, the root component manages the entire page.

Redux is a third party library

Jsbin.com is a web editor

When ‘this’ keyword is used with arrow functions, it always keeps its context.

Classes are blueprints for js objects. Properties are like variables attached to objects, methods are like functions attached to objects.

‘spread’ operator is used to split up array elements or object properties.

‘rest’ is used to merge a list of function arguments into an array

‘destructuring’ is extract array elements or object properties and store them in variables.

‘jsx’ is html in js syntax

Dependency management tool: npm or yarn. Bundler is Webpack. Native js compiler is Babel

Npm install –g create-react-app

Create-react-app my-app –scripts-version 1.1.5

Cd my-app // npm start

The nodes-module holds all the dependencies and sub-dependences of the project

The ‘public’ folder is surfed by the web server

Manifest.json is the metadata about the application

React.createElement(‘h1’,{ object} / null,’Hello’)

--

import React, { Component } from 'react';

import './App.css';

class App extends Component {

render() {

return (

// <div className="App">

// <h1> Hi, I am a React App!!</h1>

// </div>

React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

In JSX, use className. Class is reserved in JS

JSX elements are not HTML. React library provides it.

A component is just a function that returns a jsx (html)

Components & JSX Cheat Sheet

Components are the **core building block of React apps**. Actually, React really is just a library for creating components in its core.

A typical React app therefore could be depicted as a **component tree** - having one root component ("App") and then an potentially infinite amount of nested child components.

Each component needs to return/ render some **JSX** code - it defines which HTML code React should render to the real DOM in the end.

**JSX is NOT HTML** but it looks a lot like it. Differences can be seen when looking closely though (for example className in JSX vs class in "normal HTML"). JSX is just syntactic sugar for JavaScript, allowing you to write HTMLish code instead of nested React.createElement(...) calls.

When creating components, you have the choice between**two different ways:**

1. **Functional components** (also referred to as "presentational", "dumb" or "stateless" components - more about this later in the course) => const cmp = () => { return <div>some JSX</div> } (using ES6 arrow functions as shown here is recommended but optional)
2. **class-based components** (also referred to as "containers", "smart" or "stateful" components) => class Cmp extends Component { render () { return <div>some JSX</div> } }

We'll of course dive into the difference throughout this course, you can already note that you should use 1) as often as possible though. It's the best-practice.

Children refers to any element between the opening and closing tags of the component

‘porps’ are object giving access to all the attributes given in the component.

‘state’ is a property defined in a class which extends Component. ‘state’ is managed from inside a component. ‘state’ is a js object.

Props & State

props  and state  are **CORE concepts** of React. Actually, only changes in props  and/ or state  trigger React to re-render your components and potentially update the DOM in the browser (a detailed look at how React checks whether to really touch the real DOM is provided in section 6).

**Props**

props  allow you to pass data from a parent (wrapping) component to a child (embedded) component.

**Example:**

**AllPosts Component:**

1. const posts = () => {
2. return (
3. <div>
4. <Post title="My first Post" />
5. </div>
6. );
7. }

Here, title  is the custom property (prop ) set up on the custom Post  component. We basically replicate the default HTML attribute behavior we already know (e.g. <input type="text">  informs the browser about how to handle that input).

**Post Component:**

1. const post = (props) => {
2. return (
3. <div>
4. <h1>{props.title}</h1>
5. </div>
6. );
7. }

The Post  component receives the props  argument. You can of course name this argument whatever you want - it's your function definition, React doesn't care! But React will pass one argument to your component function => An object, which contains all properties you set up on <Post ... /> .

{props.title}  then dynamically outputs the title  property of the props  object - which is available since we set the title  property inside AllPosts  component (see above).

**State**

Whilst props allow you to pass data down the component tree (and hence trigger an UI update), state is used to change the component, well, state from within. Changes to state also trigger an UI update.

**Example:**

**NewPost Component:**

1. class NewPost extends Component { // state can only be accessed in class-based components!
2. state = {
3. counter: 1
4. };
6. render () { // Needs to be implemented in class-based components! Needs to return some JSX!
7. return (
8. <div>{this.state.counter}</div>
9. );
10. }
11. }

Here, the NewPost  component contains state . Only class-based components can define and use state . You can of course pass the state  down to functional components, but these then can't directly edit it.

state  simply is a property of the component class, you have to call it state  though - the name is not optional. You can then access it via this.state  in your class JSX code (which you return in the required render()  method).

Whenever state  changes (taught over the next lectures), the component will re-render and reflect the new state. The difference to props  is, that this happens within one and the same component - you don't receive new data (props ) from outside!

To Which Events Can You Listen?

In the last lecture, we saw that you can react to the onClick event - but to which other events can you listen? You can find a list of supported events here: <https://reactjs.org/docs/events.html#supported-events>

#### **Clipboard Events**

Event names:

1. onCopy onCut onPaste

Properties:

1. DOMDataTransfer clipboardData

Composition Events

Event names:

1. onCompositionEnd onCompositionStart onCompositionUpdate

Properties:

1. string data

Keyboard Events

Event names:

1. onKeyDown onKeyPress onKeyUp

Properties:

1. boolean altKey
2. number charCode
3. boolean ctrlKey
4. boolean getModifierState(key)
5. string key
6. number keyCode
7. string locale
8. number location
9. boolean metaKey
10. boolean repeat
11. boolean shiftKey
12. number which

Focus Events

Event names:

1. onFocus onBlur

These focus events work on all elements in the React DOM, not just form elements.

Properties:

1. DOMEventTarget relatedTarget

Form Events

Event names:

1. onChange onInput onInvalid onSubmit

For more information about the onChange event, see [Forms](https://reactjs.org/docs/forms.html).

Mouse Events

Event names:

1. onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
2. onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
3. onMouseMove onMouseOut onMouseOver onMouseUp

The onMouseEnter and onMouseLeave events propagate from the element being left to the one being entered instead of ordinary bubbling and do not have a capture phase.

Properties:

1. boolean altKey
2. number button
3. number buttons
4. number clientX
5. number clientY
6. boolean ctrlKey
7. boolean getModifierState(key)
8. boolean metaKey
9. number pageX
10. number pageY
11. DOMEventTarget relatedTarget
12. number screenX
13. number screenY
14. boolean shiftKey

Selection Events

Event names:

1. onSelect

Touch Events

Event names:

1. onTouchCancel onTouchEnd onTouchMove onTouchStart

Properties:

1. boolean altKey
2. DOMTouchList changedTouches
3. boolean ctrlKey
4. boolean getModifierState(key)
5. boolean metaKey
6. boolean shiftKey
7. DOMTouchList targetTouches
8. DOMTouchList touches

UI Events

Event names:

1. onScroll

Properties:

1. number detail
2. DOMAbstractView view

Wheel Events

Event names:

1. onWheel

Properties:

1. number deltaMode
2. number deltaX
3. number deltaY
4. number deltaZ

Media Events

Event names:

1. onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied onEncrypted
2. onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause onPlay
3. onPlaying onProgress onRateChange onSeeked onSeeking onStalled onSuspend
4. onTimeUpdate onVolumeChange onWaiting

Image Events

Event names:

1. onLoad onError

Animation Events

Event names:

1. onAnimationStart onAnimationEnd onAnimationIteration

Properties:

1. string animationName
2. string pseudoElement
3. float elapsedTime

Transition Events

Event names:

1. onTransitionEnd

Properties:

1. string propertyName
2. string pseudoElement
3. float elapsedTime

Other Events

Event names:

1. onToggle

----

This.getState({}) – applicable only in class –state

From React 16.8, react ‘hooks’ can be used to manage ‘state’ from functions

React hooks are a collection of functions which can be used inside functional components.

‘useState’ is the most important React hook.

‘stateful’ components are called smart or container components. Others are stateless presentational or dumb components.

2 ways of styling: .css file, inline styling.

Ek.js

import React from 'react'

const ek = () => {

return <p> Hello EK</p>

}

export default ek

--

App.js

Import Ek from ‘./Person/Ek’

----

Person.js

//props

import React from 'react'

const person = (props) =>{

return <p> Hello I am {props.name} {props.age} years old {props.children} </p>

}

export default person

---

App.js

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name= "Jaison" age = "45"/>

<Person name="Sonia" age= "39"> Facebook </Person>

<Ek />

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

//state App.js

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{name:'Max',age:28},

{name:'Manu', age:29},

{name:'Stephanie',age:27}

]

}

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name= {this.state.persons[0].name} age = {this.state.persons[0].age}/>

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

App.js

//eventHandler

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{name:'Max',age:28},

{name:'Manu', age:29},

{name:'Stephanie',age:27}

]

}

clickHandler = () => console.log("I am clicked")

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name= {this.state.persons[0].name} age = {this.state.persons[0].age}/>

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={this.clickHandler}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

App.js //setState

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{name:'Max',age:28},

{name:'Manu', age:29},

{name:'Stephanie',age:27}

]

}

clickHandler = () => {

this.setState({persons: [

{ name: 'Maxmilian', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 22 }

]})

}

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name= {this.state.persons[0].name} age = {this.state.persons[0].age}/>

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={this.clickHandler}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

‘useState’ always returns 2 elements, exactly 2 elements!. The first element which returns always is the ‘current state’. 2nd return element is the function that updates the current state.

App.js

//useState

import React, { useState } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

const app = () => {

const [currentState, updateState] = useState({

persons: [

{name:'Max',age:28},

{name:'Manu', age:29},

{name:'Stephanie',age:27}

]})

const clickHandler = () => {

updateState({persons: [

{ name: 'Maxmilian', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie Max', age: 22 }

]})

}

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name={currentState.persons[0].name} age={currentState.persons[0].age}/>

<Person name={currentState.persons[1].name} age={currentState.persons[1].age}> Facebook </Person>

<Person name={currentState.persons[2].name} age={currentState.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={clickHandler}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

export default app;

---

eventHandler.bind(this,’Name’) – to pass arguments.

---

App.js

// passing method reference to other objects

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{ name: 'Max', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 27 }

]

}

clickHandler = () => {

this.setState({

persons: [

{ name: 'Maxmilian', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 22 }

]

})

}

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person click={this.clickHandler} name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={this.clickHandler}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

Person.js

//props

import React from 'react'

const person = props =>{

return <p onClick={props.click}> Hello I am {props.name} {props.age} years old {props.children} </p>

}

export default person

---

App.js

// passing as argument using bind and method call

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{ name: 'Max', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 27 }

]

}

clickHandler = (newName) => {

this.setState({

persons: [

{ name: newName, age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 22 }

]

})

}

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

<Person name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person click= {() => this.clickHandler('Sonia')} name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={this.clickHandler.bind(this,'Jaison')}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

App.js

// onChange with an event object passed

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Ek from './Person/Ek'

class App extends Component {

state = {

persons: [

{ name: 'Max', age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 27 }

]

}

clickHandler = (newName) => {

this.setState({

persons: [

{ name: newName, age: 28 },

{ name: 'Manu', age: 29 },

{ name: 'Stephanie', age: 22 }

]

})

}

changeName = (event) => {

this.setState({

persons: [

{ name: 'Maximilan', age: 28 },

{ name: event.target.value, age: 29 },

{ name: 'Stephanie', age: 22 }

]

})

}

render() {

return (

<div className="App">

<h1> Hi, I am a React App!!</h1>

<Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

<Person changed={this.changeName} name={this.state.persons[1].name} age={this.state.persons[1].age}> Facebook </Person>

<Person click= {() => this.clickHandler('Sonia')} name={this.state.persons[2].name} age={this.state.persons[2].age}> Whatsapp </Person>

<Ek />

<button onClick={this.clickHandler.bind(this,'Jaison')}>Click</button>

</div>

// React.createElement('div',{className:'App'},React.createElement('h1',null,"From React"))

);

}

}

export default App;

---

Person.js

import React from 'react'

const person = props =>{

return (

<div>

<p onClick={props.click}> Hello I am {props.name} {props.age} years old {props.children} </p>

<input type="text" onChange={props.changed} value={props.name} />

</div>

)

}

export default person

---

‘inline’ style is defined inside the ‘render()’ method in the class, which can be used in the element declared under ‘return’

Single curly braces {} used to input dynamic content.

Useful Resources & Links

* create-react-app: <https://github.com/facebookincubator/create-react-app>
* Introducing JSX: <https://reactjs.org/docs/introducing-jsx.html>
* Rendering Elements: <https://reactjs.org/docs/rendering-elements.html>
* Components & Props: <https://reactjs.org/docs/components-and-props.html>
* Listenable Events: <https://reactjs.org/docs/events.html>

Javascript expressions can be used inside jsx using single curly braces { expressions }.

Jsx code start with the return statement in the ‘render()’ method in the component. Inside jsx, if condition cannot be used. Instead, ternary operator can be used.

App.js

// Using map function to return array elements in jsx

render() {

let p = null

if(this.state.personOnoff) {

p = (

<div>

{

this.state.persons.map(person => {

return (

<Person name={person.name}

age={person.age} />

)

})

---

App.js

//removing an element from array by click on the text

removeHandler = (personIndex) => {

const list = this.state.persons

list.splice(personIndex,1)

this.setState({persons:list})

}

render() {

let p = null

if(this.state.personOnoff) {

p = (

<div>

{

this.state.persons.map((person,index) => {

return (

<Person click={() => this.removeHandler(index)} name={person.name}

age={person.age} />

)

})

---

newArray = Array.slice() – without arguments simply copies the array and returns a new array.

newArray = […oldArray] – also can be used to copy an array (using spread operator)

using a ‘key’ as a property inside the component is sometimes advisable

Useful Resources & Links

* Conditional Rendering: <https://reactjs.org/docs/conditional-rendering.html>
* Lists & Keys: <https://reactjs.org/docs/lists-and-keys.html>

Dynamic styles: first define multiple styles inside a .css file as classes, then define a variable and assign classes as array values then join them. Apply the variable conditionally. Inside the component, assign the variable against the component ‘style className’.

Radium – package to install style selector. Npm install –save radium. (save us used to save package into project .json folder). This allows to use pseudo selectors in ‘in-style’ selectors.

Import Radium from ‘radium’

Export default Radium(app)

App.js

render () {

const style = {

backgroundColor: 'green',

color: 'white',

font: 'inherit',

border: '1px solid blue',

padding: '8px',

cursor: 'pointer',

':hover': { //hover

backgroundColor: 'lightgreen',

color: 'black'

}

};

let persons = null;

if ( this.state.showPersons ) {

persons = (

<div>

{this.state.persons.map( ( person, index ) => {

return <Person

click={() => this.deletePersonHandler( index )}

name={person.name}

age={person.age}

key={person.id}

changed={( event ) => this.nameChangedHandler( event, person.id )} />

} )}

</div>

);

style.backgroundColor = 'red';

style[':hover'] = { //using hover

backgroundColor: 'salmon',

color: 'black'

};

}

const classes = [];

if ( this.state.persons.length <= 2 ) {

classes.push( 'red' ); // classes = ['red']

}

if ( this.state.persons.length <= 1 ) {

classes.push( 'bold' ); // classes = ['red', 'bold']

}

return (

<StyleRoot>

<div className="App">

<h1>Hi, I'm a React App</h1>

<p className={classes.join( ' ' )}>This is really working!</p>

<button

style={style}

onClick={this.togglePersonsHandler}>Toggle Persons</button>

{persons}

</div>

</StyleRoot>

);

// return React.createElement('div', {className: 'App'}, React.createElement('h1', null, 'Does this work now?'));

}

}

export default Radium( App );

Userinput.js

const inputText = (props) => {

const inputStyle = {

border: '8px solid red',

padding: '15px',

text\_align: 'center',

'@media (min-width:500px)': {

width:'450px'

}

}

---

App.js

import Radium, { StyleRoot } from 'radium'

render() {

const styl = {

backgroundColor:'green',

color: 'red'

}

return (

<StyleRoot>

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<h1 className="App-title">Welcome to React</h1>

</header>

<p className="App-intro">

To get started, edit <code>src/App.js</code> and save to reload.

</p>

<p style={styl}> Hellooo</p>

<UserInput changed = {this.changeHandler} currentName={this.state.username}/>

<UserOutput userName = {this.state.username}/>

<UserOutput userName={this.state.username}/>

<UserOutput userName = "Sonia"/>

</div>

</StyleRoot>

);

---

Styled-components.com

Tagged templates use `` (tow backticks) at the end

Npm install –save styled-components

Import styled from ‘styled-components’

App.js

import React, { Component } from 'react';

import styled from 'styled-components';

import './App.css';

import Person from './Person/Person';

const StyledButton = styled.button`

background-color: ${props => props.alt ? 'red' : 'green'};

color: white;

font: inherit;

border: 1px solid blue;

padding: 8px;

cursor: pointer;

&:hover {

background-color: ${props => props.alt ? 'salmon' : 'lightgreen'};

color: black;

}

`;

class App extends Component {

state = {

persons: [

{ id: 'asfa1', name: 'Max', age: 28 },

{ id: 'vasdf1', name: 'Manu', age: 29 },

{ id: 'asdf11', name: 'Stephanie', age: 26 }

],

otherState: 'some other value',

showPersons: false

};

nameChangedHandler = (event, id) => {

const personIndex = this.state.persons.findIndex(p => {

return p.id === id;

});

const person = {

...this.state.persons[personIndex]

};

// const person = Object.assign({}, this.state.persons[personIndex]);

person.name = event.target.value;

const persons = [...this.state.persons];

persons[personIndex] = person;

this.setState({ persons: persons });

};

deletePersonHandler = personIndex => {

// const persons = this.state.persons.slice();

const persons = [...this.state.persons];

persons.splice(personIndex, 1);

this.setState({ persons: persons });

};

togglePersonsHandler = () => {

const doesShow = this.state.showPersons;

this.setState({ showPersons: !doesShow });

};

render() {

const style = {

backgroundColor: 'green',

color: 'white',

font: 'inherit',

border: '1px solid blue',

padding: '8px',

cursor: 'pointer',

':hover': {

backgroundColor: 'lightgreen',

color: 'black'

}

};

let persons = null;

if (this.state.showPersons) {

persons = (

<div>

{this.state.persons.map((person, index) => {

return (

<Person

click={() => this.deletePersonHandler(index)}

name={person.name}

age={person.age}

key={person.id}

changed={event => this.nameChangedHandler(event, person.id)}

/>

);

})}

</div>

);

// style.backgroundColor = 'red';

// style[':hover'] = {

// backgroundColor: 'salmon',

// color: 'black'

// };

}

const classes = [];

if (this.state.persons.length <= 2) {

classes.push('red'); // classes = ['red']

}

if (this.state.persons.length <= 1) {

classes.push('bold'); // classes = ['red', 'bold']

}

return (

<div className="App">

<h1>Hi, I'm a React App</h1>

<p className={classes.join(' ')}>This is really working!</p>

<StyledButton alt={this.state.showPersons} onClick={this.togglePersonsHandler}>

Toggle Persons

</StyledButton>

{persons}

</div>

);

// return React.createElement('div', {className: 'App'}, React.createElement('h1', null, 'Does this work now?'));

}

}

export default App;

---

More on CSS Modules

**CSS Modules** are a relatively new concept (you can dive super-deep into them here: <https://github.com/css-modules/css-modules>). With CSS modules, you can write normal CSS code and make sure, that it only applies to a given component.

It's not using magic for that, instead it'll simply **automatically generate unique CSS class names** for you. And by importing a JS object and assigning classes from there, you use these dynamically generated, unique names. So the imported JS object simply exposes some properties which hold the generated CSS class names as values.

**Example:**

**In Post.css File**

1. .Post {
2. color: red;
3. }

**In Post Component File**

1. import classes from './Post.css';
3. const post = () => (
4. <div className={classes.Post}>...</div>
5. );

Here, classes.Post  refers to an automatically generated Post  property on the imported classes  object. That property will in the end simply hold a value like Post\_\_Post\_\_ah5\_1 .

So your .Post  class was automatically transformed to a different class (Post\_\_Post\_\_ah5\_1 ) which is unique across the application. You also can't use it accidentally in other components because you don't know the generated string! You can only access it through the classes  object. And if you import the CSS file (in the same way) in another component, the classes  object there will hold a Post  property which yields a **different** (!) CSS class name. Hence it's scoped to a given component.

By the way, if you somehow also want to define a global (i.e. un-transformed) CSS class in such a .css  file, you can prefix the selector with :global .

**Example:**

:global .Post { ... }

Now you can use className="Post"  anywhere in your app and receive that styling.

Useful Resources & Links

* Using CSS Modules in create-react-app Projects: <https://medium.com/nulogy/how-to-use-css-modules-with-create-react-app-9e44bec2b5c2>
* More information about CSS Modules: <https://github.com/css-modules/css-modules>

---

Add react developer tools to the chrome webstore

Useful Resources & Links

* Error Boundaries: <https://reactjs.org/docs/error-boundaries.html>
* Chrome Devtool Debugging: <https://developers.google.com/web/tools/chrome-devtools/javascript/>

Restructure the src folder:

* 1. Components (for individual components)
  2. Assets (for images …)
  3. Container (for App.js, .css)
  4. Cockpit (Cockpit.js)
  5. App.js

A presentational component is a functional component that does not manage state.

Component life-cycle is only available in class-based components

setState causes re-rendering.

‘usEffect’ is not a life-cycle hook. It’s a react hook. It is written in the function body, which takes a function which will run for every render() cycle.

useEffect runs for every update on the component where it is defined. Its component mount and component update in one place.

usEffect has an ‘array’ as its 2nd argument.

### Ract component life cycle: The Component Lifecycle

Each component has several “lifecycle methods” that you can override to run code at particular times in the process. **You can use**[**this lifecycle diagram**](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)**as a cheat sheet.** In the list below, commonly used lifecycle methods are marked as **bold**. The rest of them exist for relatively rare use cases.

#### **Mounting**

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

* [**constructor()**](https://reactjs.org/docs/react-component.html#constructor)
* [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [**render()**](https://reactjs.org/docs/react-component.html#render)
* [**componentDidMount()**](https://reactjs.org/docs/react-component.html#componentdidmount)

**Note:**

These methods are considered legacy and you should [avoid them](https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html) in new code:

* [UNSAFE\_componentWillMount()](https://reactjs.org/docs/react-component.html#unsafe_componentwillmount)

#### **Updating**

An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:

* [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [shouldComponentUpdate()](https://reactjs.org/docs/react-component.html#shouldcomponentupdate)
* [**render()**](https://reactjs.org/docs/react-component.html#render)
* [getSnapshotBeforeUpdate()](https://reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
* [**componentDidUpdate()**](https://reactjs.org/docs/react-component.html#componentdidupdate)

**Note:**

These methods are considered legacy and you should [avoid them](https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html) in new code:

* [UNSAFE\_componentWillUpdate()](https://reactjs.org/docs/react-component.html#unsafe_componentwillupdate)
* [UNSAFE\_componentWillReceiveProps()](https://reactjs.org/docs/react-component.html#unsafe_componentwillreceiveprops)

#### **Unmounting**

This method is called when a component is being removed from the DOM:

* [**componentWillUnmount()**](https://reactjs.org/docs/react-component.html#componentwillunmount)

#### **Error Handling**

These methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

* [static getDerivedStateFromError()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromerror)
* [componentDidCatch()](https://reactjs.org/docs/react-component.html#componentdidcatch)

### Other APIs

Each component also provides some other APIs:

* [setState()](https://reactjs.org/docs/react-component.html#setstate)
* [forceUpdate()](https://reactjs.org/docs/react-component.html#forceupdate)

### Class Properties

* [defaultProps](https://reactjs.org/docs/react-component.html#defaultprops)
* [displayName](https://reactjs.org/docs/react-component.html#displayname)

### Instance Properties

* [props](https://reactjs.org/docs/react-component.html#props)
* [state](https://reactjs.org/docs/react-component.html#state)

Export default React.memo(person) // react does memoization of the component.

React maintains old virtual dom and the rerendered virtual dom

An array [] of components with ‘key’ as property can be used to return an array of different components in a render() method.

React.Fragment – can also be used in place for returning multiple elements. Can create another element as children and wrap the element to return them elements inside the render.(HOC higher order components)

Npm install prop-types

React.createRef()

To use in functional components, import useRef, then useRef()

‘ref’ is a property like ‘key’ understood by ‘react’ components.

‘useEffect’ runs after every render cycle.

React.createContext() // AuthContext.Provider()

Import useContext allows to get access to contexts from functional components

Static contextType = AuthContext – is for class components

Useful Resources & Links

Useful Resources:

* More on useEffect(): <https://reactjs.org/docs/hooks-effect.html>
* State & Lifecycle: <https://reactjs.org/docs/state-and-lifecycle.html>
* PropTypes: <https://reactjs.org/docs/typechecking-with-proptypes.html>
* Higher Order Components: <https://reactjs.org/docs/higher-order-components.html>
* Refs: <https://reactjs.org/docs/refs-and-the-dom.html>

Both React and ReactDOM are available over a CDN.

<script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

The versions above are only meant for development, and are not suitable for production. Minified and optimized production versions of React are available at:

<script crossorigin src="https://unpkg.com/react@17/umd/react.production.min.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js"></script>

---

JSX produces React “elements”.

You can put any valid [JavaScript expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#Expressions) inside the curly braces in JSX.

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

Don’t put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expressions), but not both in the same attribute.

Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names.

By default, React DOM [escapes](https://stackoverflow.com/questions/7381974/which-characters-need-to-be-escaped-on-html) any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent [XSS (cross-site-scripting)](https://en.wikipedia.org/wiki/Cross-site_scripting) attacks.

Babel compiles JSX down to React.createElement() calls.

These two examples are identical:

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

--

Elements are what components are “made of”, and we encourage you to read this section before jumping ahead.

React elements are [immutable](https://en.wikipedia.org/wiki/Immutable_object). Once you create an element, you can’t change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

**All React components must act like pure functions with respect to their props.**

State allows React components to change their output over time in response to user actions, network responses, and anything else, without violating this rule.

Class components should always call the base constructor with props.

componentDidMount() // componentWillUnmount()

setInterval(), clearInterval()

The only place where you can assign this.state is the constructor.

‘state’ is not accessible to any component other than the one that owns and sets it.

* With JSX you pass a function as the event handler, rather than a string.

Generally, if you refer to a method without () after it, such as onClick={this.handleClick}, you should bind that method.

e.preventDefault();

#### Jiji