

REACT

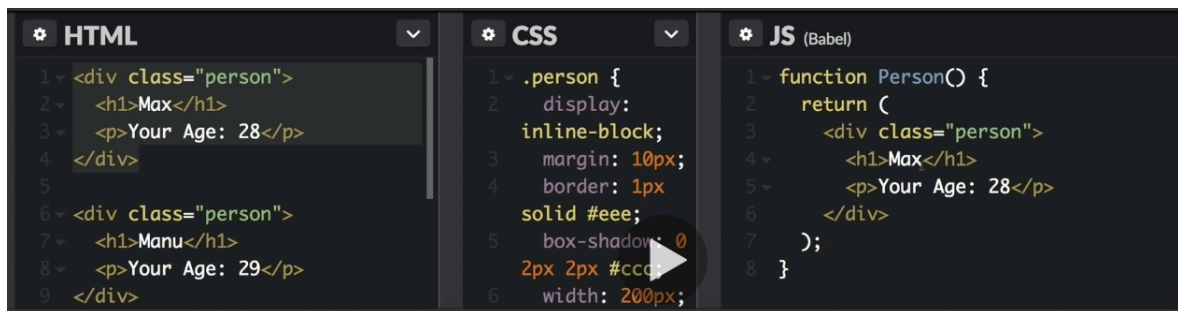
1) React package (package.json)

Have code to understand the components we have written

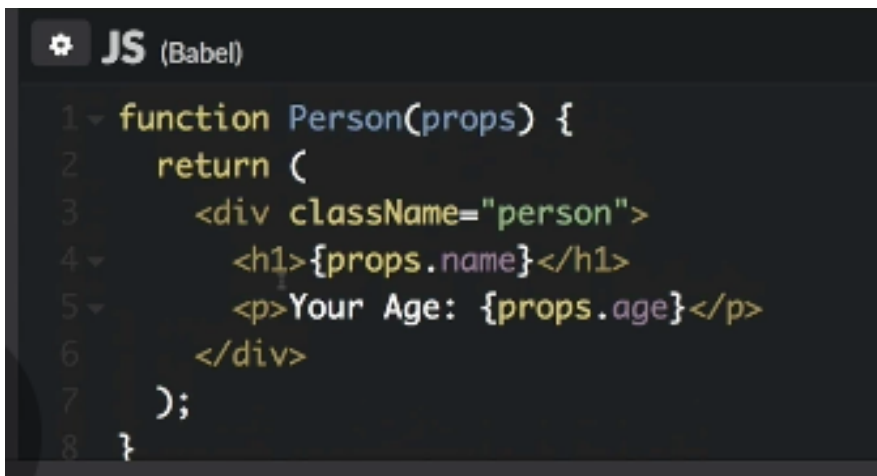
React DOM package(package.json)

Have code to render the react code in DOM

2)How the code from js is returned to html to make it reusable



3)Function component which has props argument always which will have the arguments we pass.





```
graph LR; A([Getting Started]) --> B([The Basics]); B --> C([Debugging]); C --> D([Styling Components]); D --> E([Components Deep Dive]); E --> F([HTTP Requests]); F --> G([Routing]); G --> H([Forms & Validation]); H --> I([Redux]); I --> J([Authentication]); J --> K([Testing Introduction]); K --> L([Deployment]); L --> M([Bonus  
(Animations,  
Next Steps,  
Webpack)]);
```

The diagram illustrates a structured learning path for web development. It begins with 'Getting Started' and 'The Basics', moving through 'Debugging' and 'Styling Components' to 'Components Deep Dive'. From there, it covers 'HTTP Requests' and 'Routing', then 'Forms & Validation' and 'Redux', followed by 'Authentication' and 'Testing Introduction'. The path continues to 'Deployment' and finally to a 'Bonus' section covering 'Animations', 'Next Steps', and 'Webpack'. A man is shown in the bottom right corner, pointing towards the 'Bonus' topic.

ES basics

Arrow Functions

```
function myFnc() {  
  ...  
}
```

```
const myFnc = () => {  
  ...  
}
```

JavaScript ▾

```
const printMyName = (name) => {  
  console.log(name);  
}  
printMyName('Max');
```

Works exactly for 1 variable below syntax

```
const printMyName = name => {  
  console.log(name);  
}  
printMyName('Max');
```

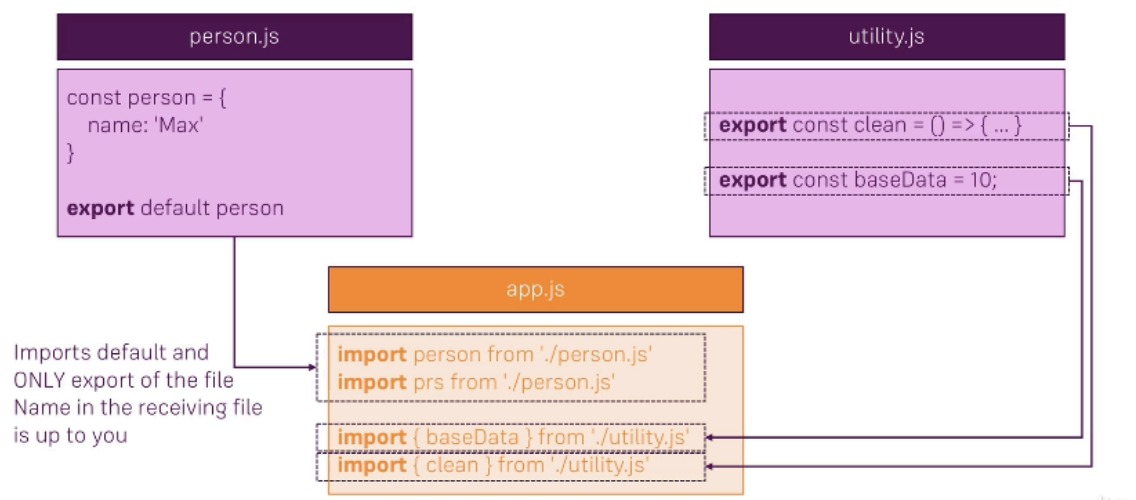
```
const printMyName = (name, age) => {  
  console.log(name, age);  
}  
printMyName('Max', 28);
```

Single line arrow function

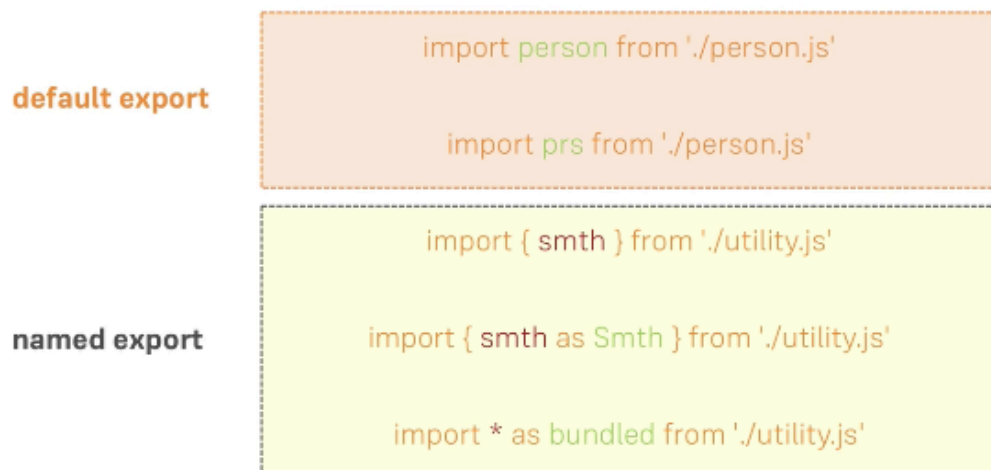
```
const multiply = number => number * 2;
console.log(multiply(2));
```

Imports and exports concept

Exports & Imports (Modules)



Exports & Imports (Modules)



You choose the name

Name is defined by export

Classes

```

class Human {
  constructor() {
    this.gender = 'male';
  }

  printGender() {
    console.log(this.gender);
  }
}

class Person extends Human {
  constructor() {
    super();
    this.name = 'Max';
  }

  printMyName() {
    console.log(this.name);
  }
}

const person = new Person();
person.printMyName();
person.printGender();

```

```

"Max"
"male"

```

Classes, Properties & Methods

Properties are like "variables
attached to classes/ objects"

ES6

```

constructor() {
  this.myProperty = 'value'
}

```

ES7

```

myProperty = 'value'

```

Methods are like "functions
attached to classes/ objects"

ES6

```

myMethod() { ... }

```

ES7

```

myMethod = () => { ... }

```

ES6 / Babel ▾

```
class Human {  
  gender = 'male';  
  
  printGender = () => {  
    console.log(this.gender);  
  }  
}  
  
class Person extends Human {  
  name = 'Max';  
  gender = 'female';  
  
  printMyName = () => {  
    console.log(this.name);  
  }  
}  
  
const person = new Person();  
person.printMyName();  
person.printGender();
```

Console

"Max"

"female"

>



Spread vs rest

Spread & Rest Operators

...

Spread

Used to split up array elements OR object properties

```
const newArray = [...oldArray, 1, 2]  
const newObject = { ...oldObject, newProp: 5 }
```

Rest

Used to merge a list of function arguments into an array

```
function sortArgs(...args) {  
  return args.sort()  
}
```

```
const filter = (...args) => {  
  return args.filter(el => el === 1);  
}  
  
console.log(filter(1, 2, 3));
```

Destructuring

Easily extract array elements or object properties and store them in variables

Array Destructuring

```
[a, b] = ['Hello', 'Max']  
console.log(a) // Hello  
console.log(b) // Max
```

Object Destructuring

```
{name} = {name: 'Max', age: 28}  
console.log(name) // Max  
console.log(age) // undefined
```

ES6 / Babel ▾

```
const numbers = [1, 2, 3];  
[num1, , num3] = numbers;  
console.log(num1, num3);
```

Primitive vs reference

Primitive values will be copied for eg number, string

Reference value will be point to the same object

So if we assign an object to another object and change values in one object it will change the value of other object

So how to avoid this since this will affect the other object

In the below example we are using spread operator to avoid pointing to same object

```
const person = {  
  name: 'Max'  
};  
  
const secondPerson = {  
  ...person  
};  
  
person.name = 'Manu';  
  
console.log(secondPerson);
```

Array Function

ES6 / Babel ▾

```
const numbers = [1, 2, 3];  
  
const doubleNumArray = numbers.map((num) => {  
  return num * 2;  
});  
  
console.log(numbers);  
console.log(doubleNumArray);
```

Console

[1, 2, 3]

[2, 4, 6]



JS Array Functions

Not really next-gen JavaScript, but also important: JavaScript array functions like `map()`, `filter()`, `reduce()` etc.

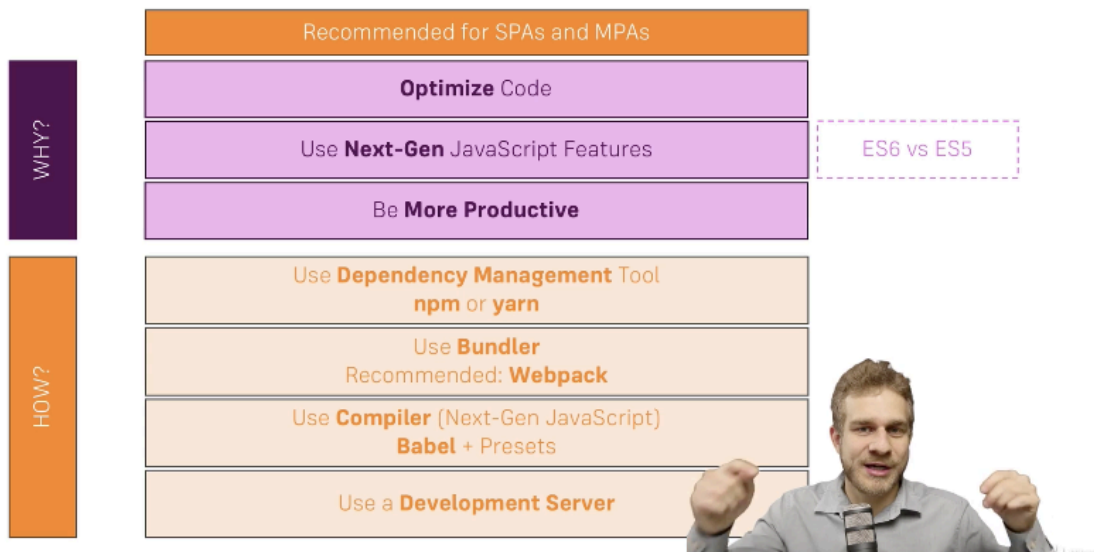
You'll see me use them quite a bit since a lot of React concepts rely on working with arrays (in immutable ways).

The following page gives a good overview over the various methods you can use on the array prototype - feel free to click through them and refresh your knowledge as required: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Particularly important in this course are:

- `map()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- `find()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find
- `findIndex()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex
- `filter()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
- `reduce()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce?v=b
- `concat()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/concat?v=b
- `slice()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice
- `splice()` => https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

Using a Build Workflow

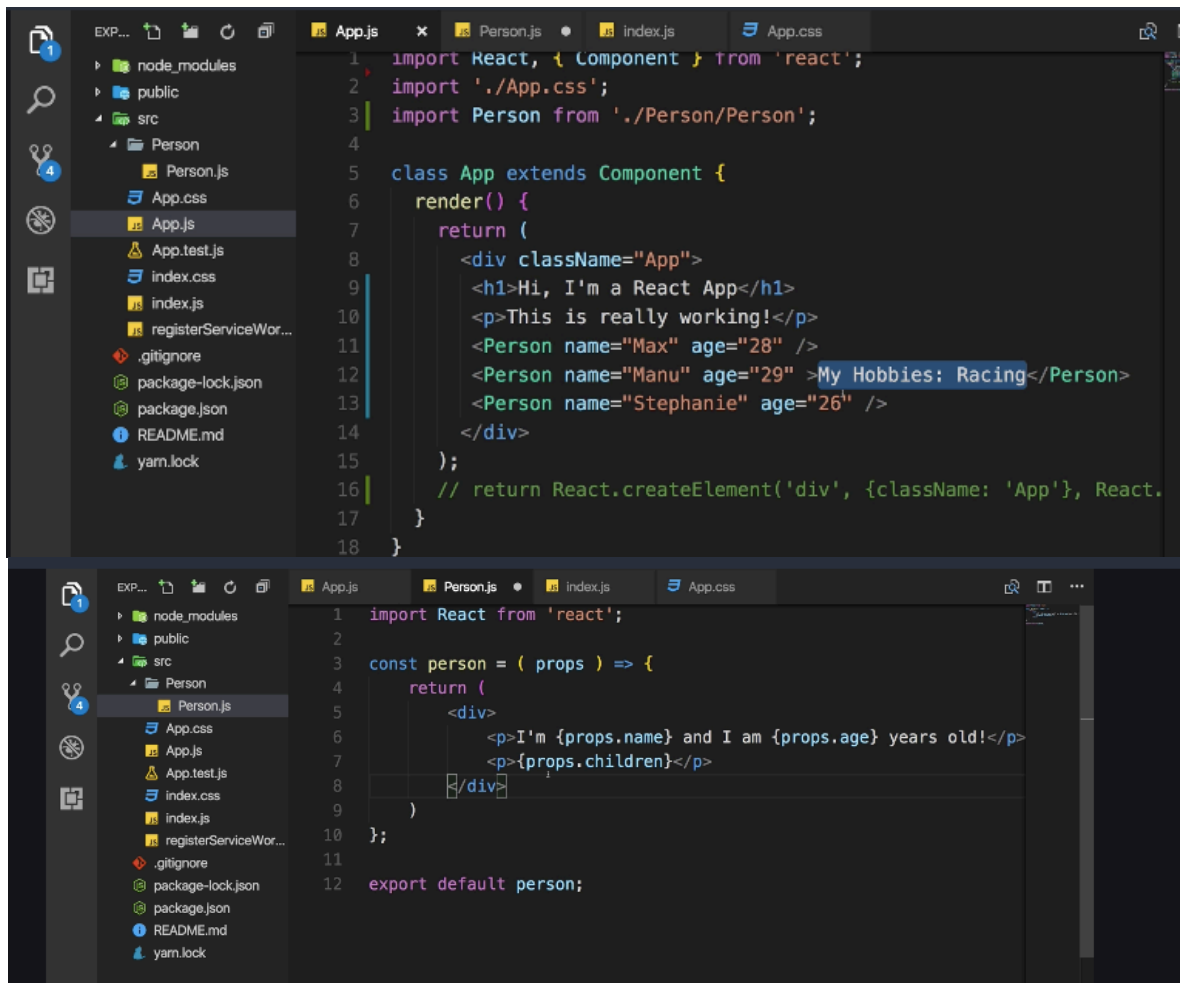


Class component

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like App.css, App.js, App.test.js, index.css, index.js, logo.svg, registerServiceWorker.js, .gitignore, package-lock.json, package.json, README.md, and yarn.lock. The code editor shows the following code:

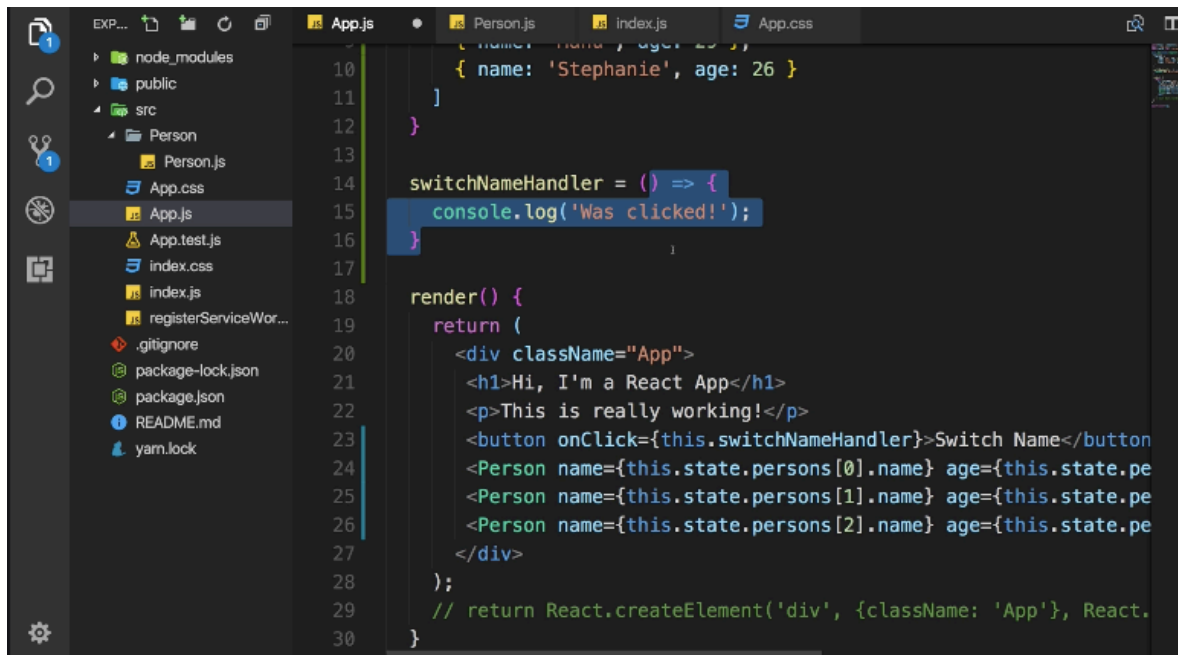
```
1 import React, { Component } from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 class App extends Component {
6   render() {
7     return (
8       <div className="App">
9         <h1>Hi, I'm a React App</h1>
10      </div>
11    );
12  }
13 }
14
15 export default App;
```

Accessing the children within the tag in react check below



Class component which extends component has the state variable in the class(as props in functional component)

Event handling

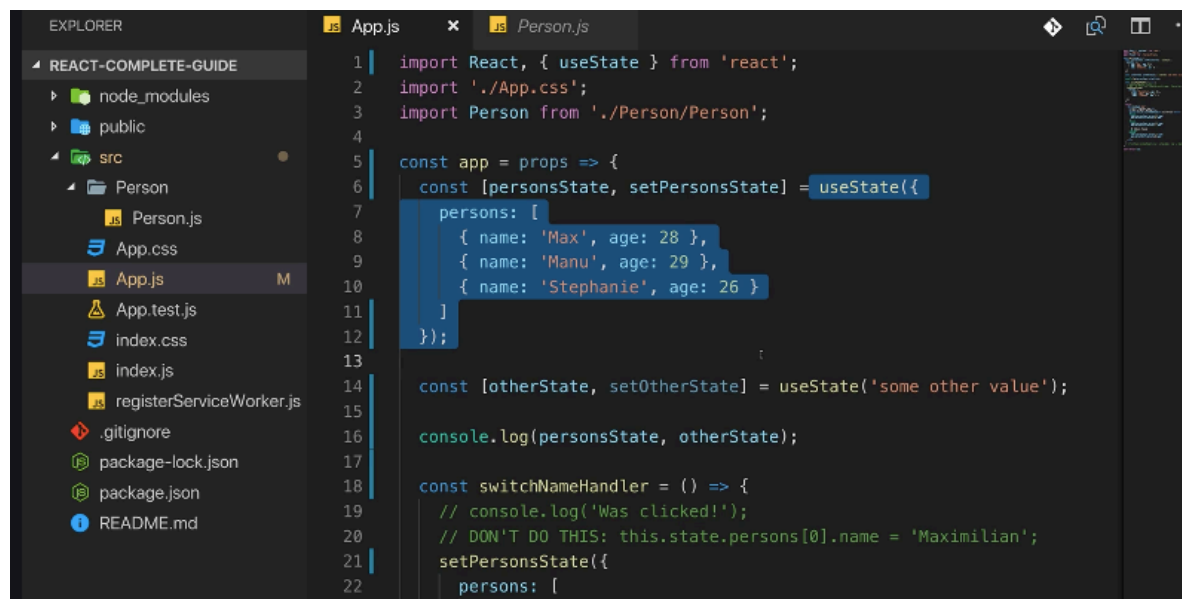


```
10 { name: 'Manu', age: 29 },
11 { name: 'Stephanie', age: 26 }
12 }
13
14 switchNameHandler = () => {
15   console.log('Was clicked!');
16 }
17
18 render() {
19   return (
20     <div className="App">
21       <h1>Hi, I'm a React App</h1>
22       <p>This is really working!</p>
23       <button onClick={this.switchNameHandler}>Switch Name</button>
24       <Person name={this.state.persons[0].name} age={this.state.pe
25       <Person name={this.state.persons[1].name} age={this.state.pe
26       <Person name={this.state.persons[2].name} age={this.state.pe
27     </div>
28   );
29   // return React.createElement('div', {className: 'App'}, React.
30 }
```

Difference between setState and useState

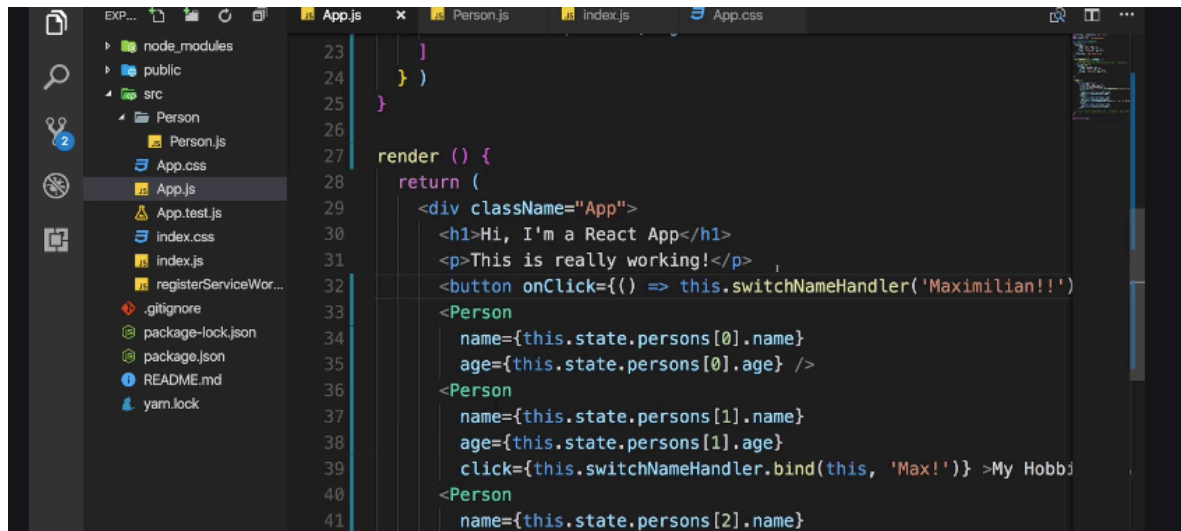
setState merges the data when set an object when using setState

useState replaces the data when set an object when using setState



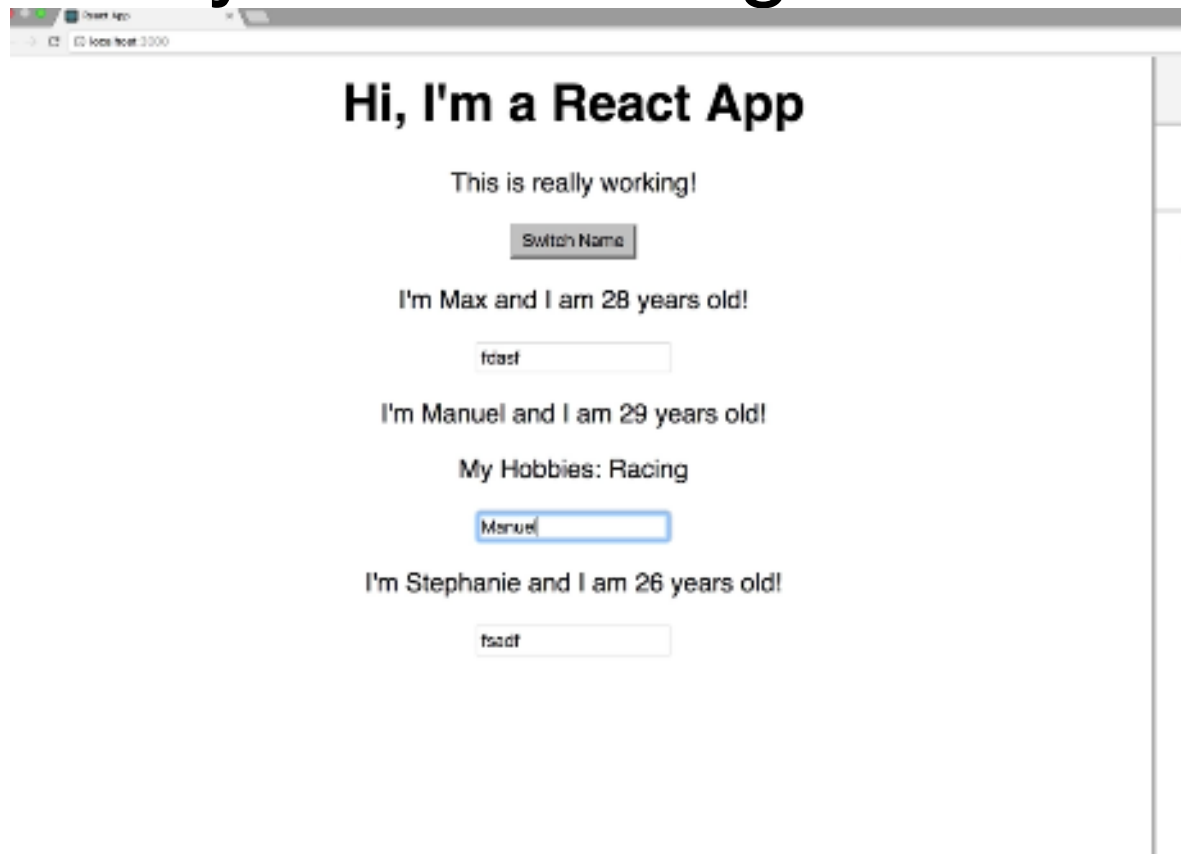
```
1 import React, { useState } from 'react';
2 import './App.css';
3 import Person from './Person/Person';
4
5 const app = props => {
6   const [personsState, setPersonsState] = useState({
7     persons: [
8       { name: 'Max', age: 28 },
9       { name: 'Manu', age: 29 },
10      { name: 'Stephanie', age: 26 }
11    ]
12  });
13
14   const [otherState, setOtherState] = useState('some other value');
15
16   console.log(personsState, otherState);
17
18   const switchNameHandler = () => {
19     // console.log('Was clicked!');
20     // DON'T DO THIS: this.state.persons[0].name = 'Maximilian';
21     setPersonsState({
22       persons: [
```

Passing data on passing click event to child component

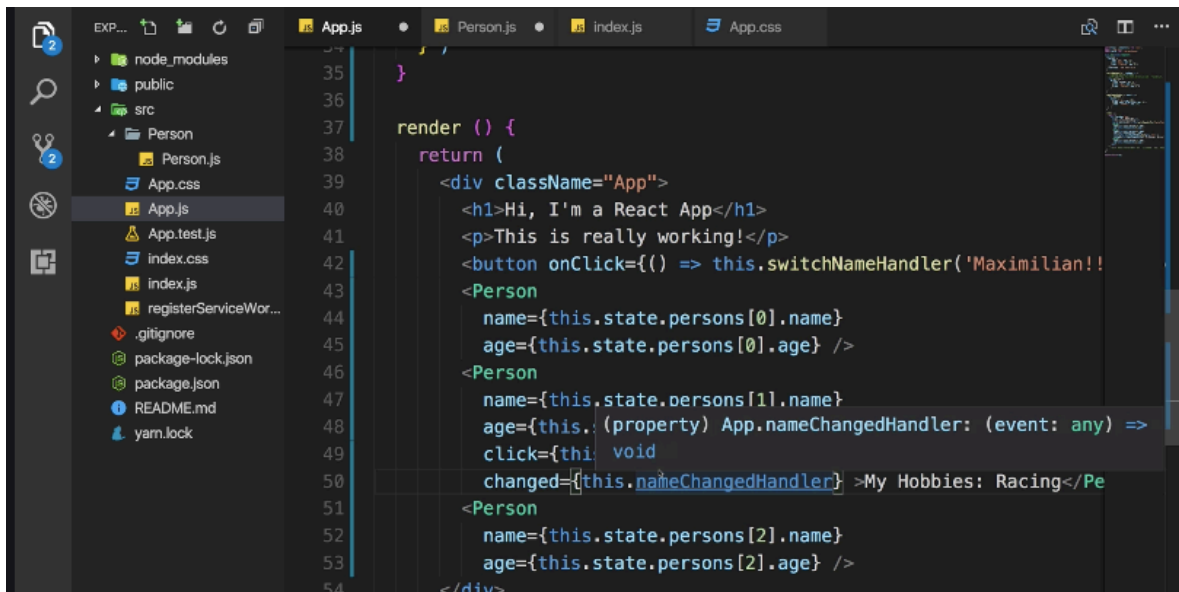


```
23     }  
24   } )  
25 }  
26  
27 render () {  
28   return (  
29     <div className="App">  
30       <h1>Hi, I'm a React App</h1>  
31       <p>This is really working!</p>  
32       <button onClick={() => this.switchNameHandler('Maximilian!')}>  
33         Switch Name  
34       </button>  
35       <Person  
36         name={this.state.persons[0].name}  
37         age={this.state.persons[0].age} />  
38       <Person  
39         name={this.state.persons[1].name}  
40         age={this.state.persons[1].age}  
41         click={this.switchNameHandler.bind(this, 'Max!')} /> My Hobbies:  
42       <Person  
43         name={this.state.persons[2].name}  
44         age={this.state.persons[2].age} />  
45     </div>  
46   )  
47 }
```

2 way data binding

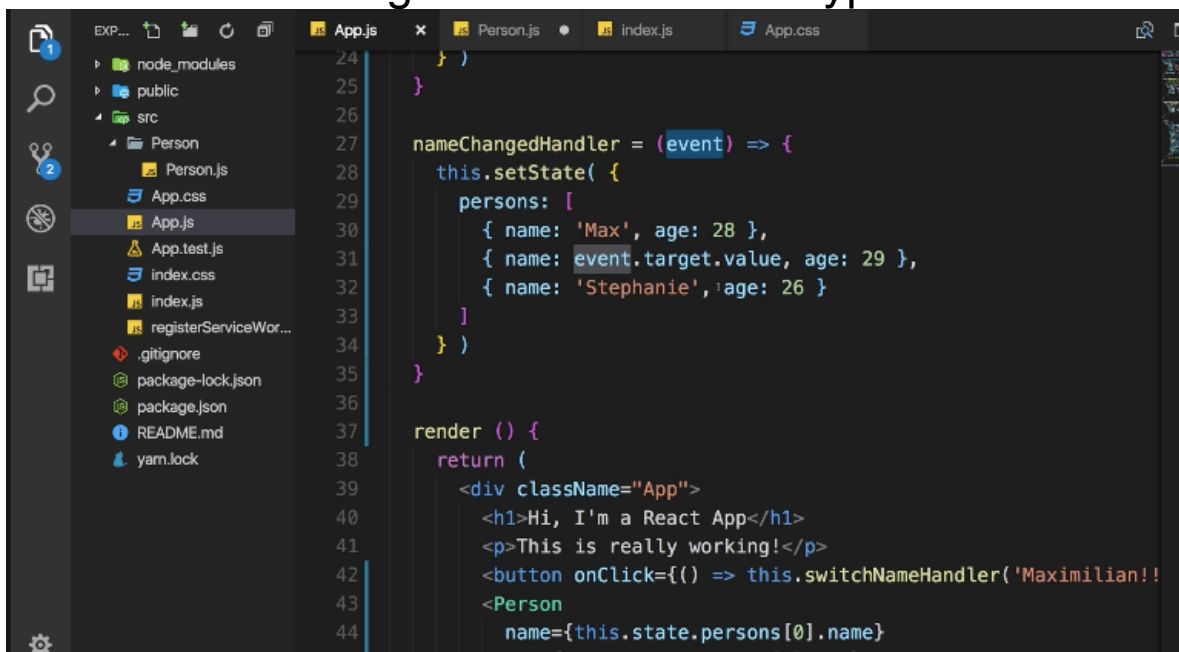


Changed method is triggered when edited in html input tag



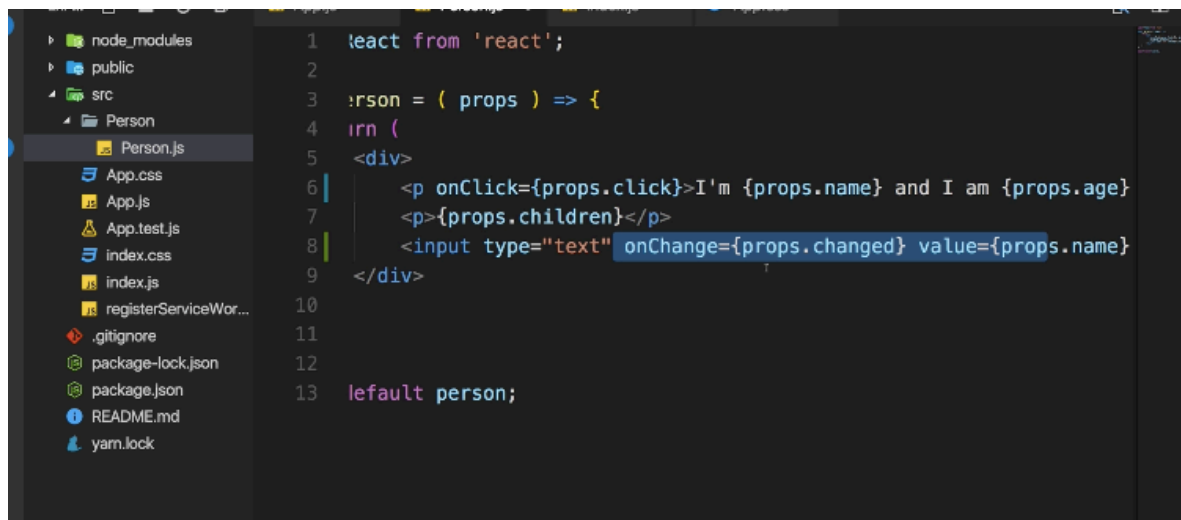
```
35 }
36
37 render () {
38   return (
39     <div className="App">
40       <h1>Hi, I'm a React App</h1>
41       <p>This is really working!</p>
42       <button onClick={() => this.switchNameHandler('Maximilian!!')}>
43         <Person
44           name={this.state.persons[0].name}
45           age={this.state.persons[0].age} />
46         <Person
47           name={this.state.persons[1].name}
48           age={this.state.persons[1].age} />
49         <button onClick={() => this.nameChangedHandler(event.target.value)}>
50           <Person
51             name={this.state.persons[2].name}
52             age={this.state.persons[2].age} />
53         </div>
```

Here we are setting the value which is typed from html



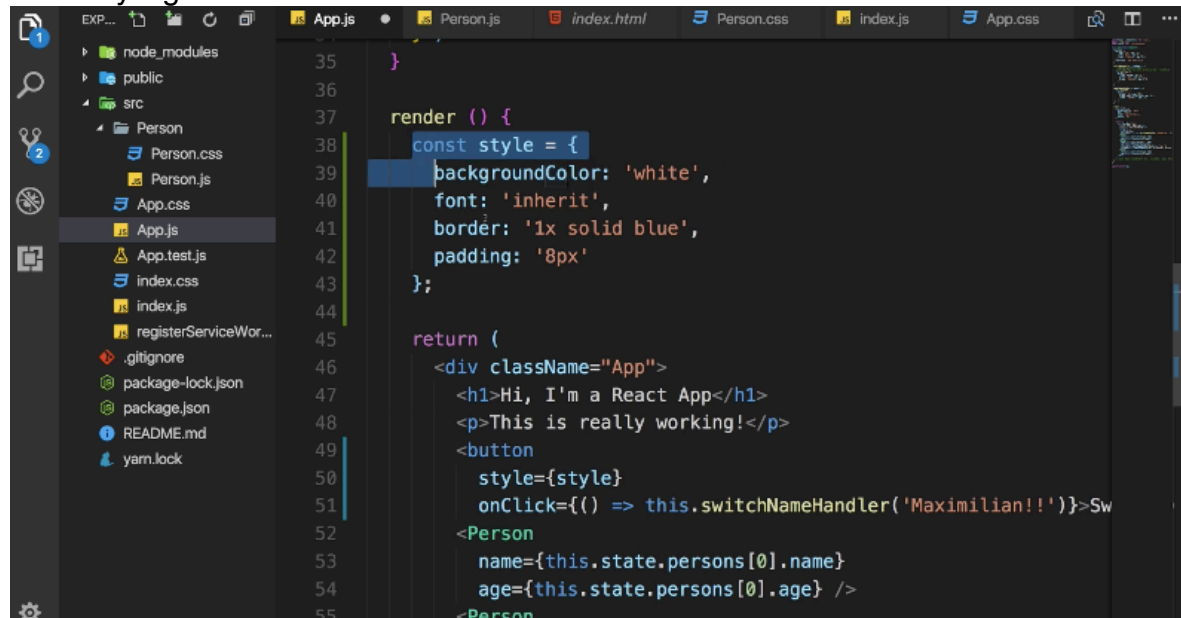
```
24 }
25
26 nameChangedHandler = (event) => {
27   this.setState( {
28     persons: [
29       { name: 'Max', age: 28 },
30       { name: event.target.value, age: 29 },
31       { name: 'Stephanie', age: 26 }
32     ]
33   } )
34 }
35
36 render () {
37   return (
38     <div className="App">
39       <h1>Hi, I'm a React App</h1>
40       <p>This is really working!</p>
41       <button onClick={() => this.switchNameHandler('Maximilian!!')}>
42         <Person
43           name={this.state.persons[0].name}
```

Bringing data from component to html



```
1  import React from 'react';
2
3  const Person = ( props ) => {
4    return (
5      <div>
6        <p onClick={props.click}>I'm {props.name} and I am {props.age}</p>
7        <p>{props.children}</p>
8        <input type="text" onChange={props.changed} value={props.name}>
9      </div>
10    );
11  }
12
13  export default Person;
```

Inline styling



```
35  }
36
37  render () {
38    const style = {
39      backgroundColor: 'white',
40      font: 'inherit',
41      border: '1px solid blue',
42      padding: '8px'
43    };
44
45    return (
46      <div className="App">
47        <h1>Hi, I'm a React App</h1>
48        <p>This is really working!</p>
49        <button
50          style={style}
51          onClick={() => this.switchNameHandler('Maximilian!!')}>Switch Name
52        </button>
53        <Person
54          name={this.state.persons[0].name}
55          age={this.state.persons[0].age} />
56      </div>
57    );
58  }
59
60  export default App;
```