the Master Course

{CODENATION}

An introduction to React.

{CODENATION}

{CODENATION}

Learning Objectives

To interpret what React is and why we would use it.

To be able to create your own components and understand what props are.

React.js

What is React?

A Javascript library for creating **interactive, complex user interfaces**.

{CODENATION}

React.js

How does React work?

Encapsulated components, composed together to create the UI.

{CODENATION}

React.js

By building the user interface with independent, reusable, isolated components our code is much easier to manage and easily updated.

{CODENATION}

React.js

Let's have a look at some webpages which use react, and how they split the UI into components.

{CODENATION}

Why use react?

We could just hard code everything using HTML and JS, but think how much we would be repeating ourselves!

React.js

Why use react?

Working with the actual DOM directly can become difficult with complex UI's or larger applications.

React.js

Why use react?

React is efficient, fast and makes dynamically updating elements much easier.

{CODENATION}

React.js

React is all about components!

So this is where we're going to start.

{CN}®

{CN}®

React.js

What is a component?

In simple terms, it's either a javascript function or class which returns a piece of the user interface.

React.js

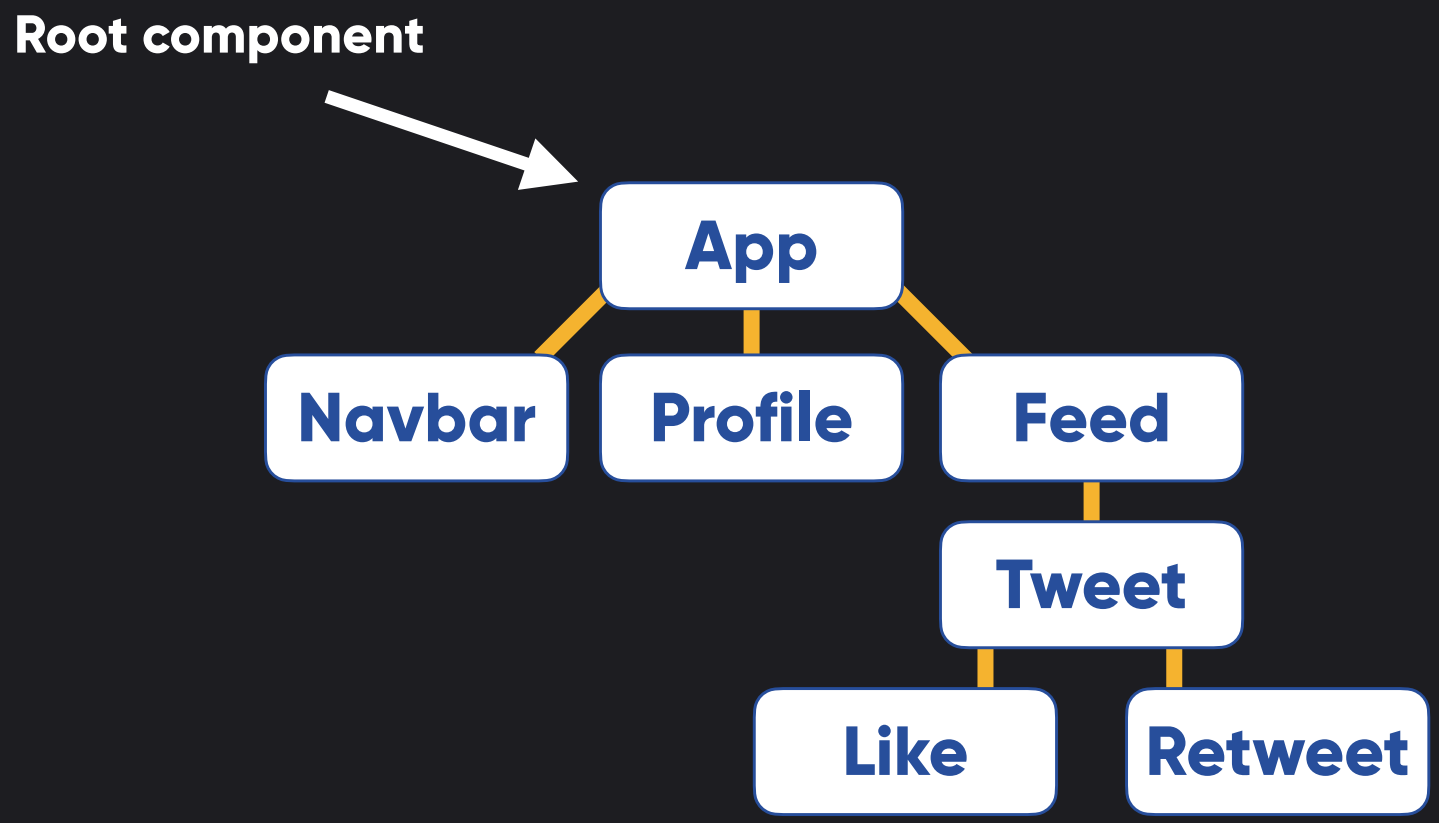
Our components are then rendered by React to represent HTML elements

React.js

We can build our components in isolation but when we use them they have to form a tree structure or hierarchy with one main (root) component.

{CN}®

Component tree



React.js

React creates a virtual DOM, which is a lightweight representation of the actual DOM, stored in memory.

{CN}®

So when the state of our app changes, **React compares the virtual DOM to the actual DOM**. If there's a difference, the actual DOM is updated to keep it in sync.

{ CN }®

React.js

So what does this actually mean? Why is this so exciting?

React.js

We no longer have to work with the DOM API in browsers.

{CODENATION}

React.js

So no more

document.getElementBy.....

{CODENATION}

React.js

If we make a change to our UI, react re-renders the necessary component which updates the real DOM. It reacts.

{CODENATION}

As mentioned earlier, a component is either a pure Javascript function, or a javascript class. Let's have a look.

```
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

This component is a function which returns some JSX. It looks like HTML, but it's not. It is converted to Javascript.

React.js

With React we can use a special syntax called JSX

{CODENATION}

```
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

Note that the return statement is wrapped in normal brackets. This is standard in JS when our return statement is written over multiple lines.

```
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

It is best practice to use capital letters when naming our functional components.

```
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

When returning JSX, there must be ONE parent element. In this case it's a div element.

React.js

Every time we see a custom HTML tag in React, it's just a React method in disguise.

React.createElement( )

{CODENATION}

React.js

This method takes three arguments.

```
React.createElement(arg1, arg2, arg3)
```

{CODENATION}

React.js

```
React.createElement(
type of element or Component name,
{an object of properties},
any children )
```

JS

```
<Component property = “value”>
      <p>Hi I’m a child element</p>
</Component>
```

JSX

{CODENATION}

React.js

```
React.createElement(
Hello,
{name: “Dan”, age: “33”},
React.createElement(‘p’, null, “Hi I’m a child Element”)
)
```

JS

```
<Hello name = “Dan” age = “33”>
        <p>Hi I’m a child element</p>
</Hello>
```

JSX

{CODENATION}

```
React.createElement(
Hello,
{name: “Dan”, age: “33”},
React.createElement(‘div’, null, React.createElement(‘p’, null, “Hi I’m a child Element”))
)
```

JS

```
<Hello name = “Dan” age = “33”>
    <div>
        <p>Hi I’m a child element</p>
    </div>
</Hello>
```

JSX

{CODENATION}

This is all possible with a compiler called Babel, which turns our JSX back into vanilla Javascript for us.

The React build workflow

What do we need to make a react app?

{CODENATION}

React.js

What we need in our project to create a react app:

NPM
Webpack
A compiler like Babel
React & React-Dom
A development server to use locally

{CODENATION}

Luckily for us, there is a package called create-react-app which packages everything we need!!

Using Create-React-App (cra)

Whenever we want to create a new React application we can use the "starter app" provided for us by the React team. This will create a new folder with all the files + folders we need to get started.

In the terminal, navigate to a folder where you want to create a new react app, and run the command:

```
npx create-react-app nameOfYourApp
```

{CODENATION}

npx	create-react-app	appName

npx: executes a node package, removing the need to install it. Not to be confused with npm. We will use npm most of the time.

create-react-app: the name of the package we want to execute. This will give us all the files, folders and dependencies we need to make a create app.

appName: name your app whatever you like.

{CODENATION}

This command will create a new folder when it is run.

Make sure you are in the correct folder when you run this command, so you are putting the react application folder which is auto generated, in the place you want.

Let's try this together.

{CODENATION}

Create a folder called: learning-react

Open this folder in vscode

Open the terminal and run
npx create-react-app my-first-app

...and let's explore what we get.

React.js

You will have noticed some ES6 JS features such as export and import.

We can use these in our react app because babel takes care of the compiling for us.

So even though you are used to using module.exports = { } and the require( ) method, in react we will be using modern features instead.

There are a few ways we can export and a few ways we can import.

Use export default componentName at the end of you component file.
Then import someName from './somefile'
SomeName can be anything you want.

You can also use something called named exports. Here you just use the export keyword before any component. But you must import using the correct name, and put the name in curly brackets.

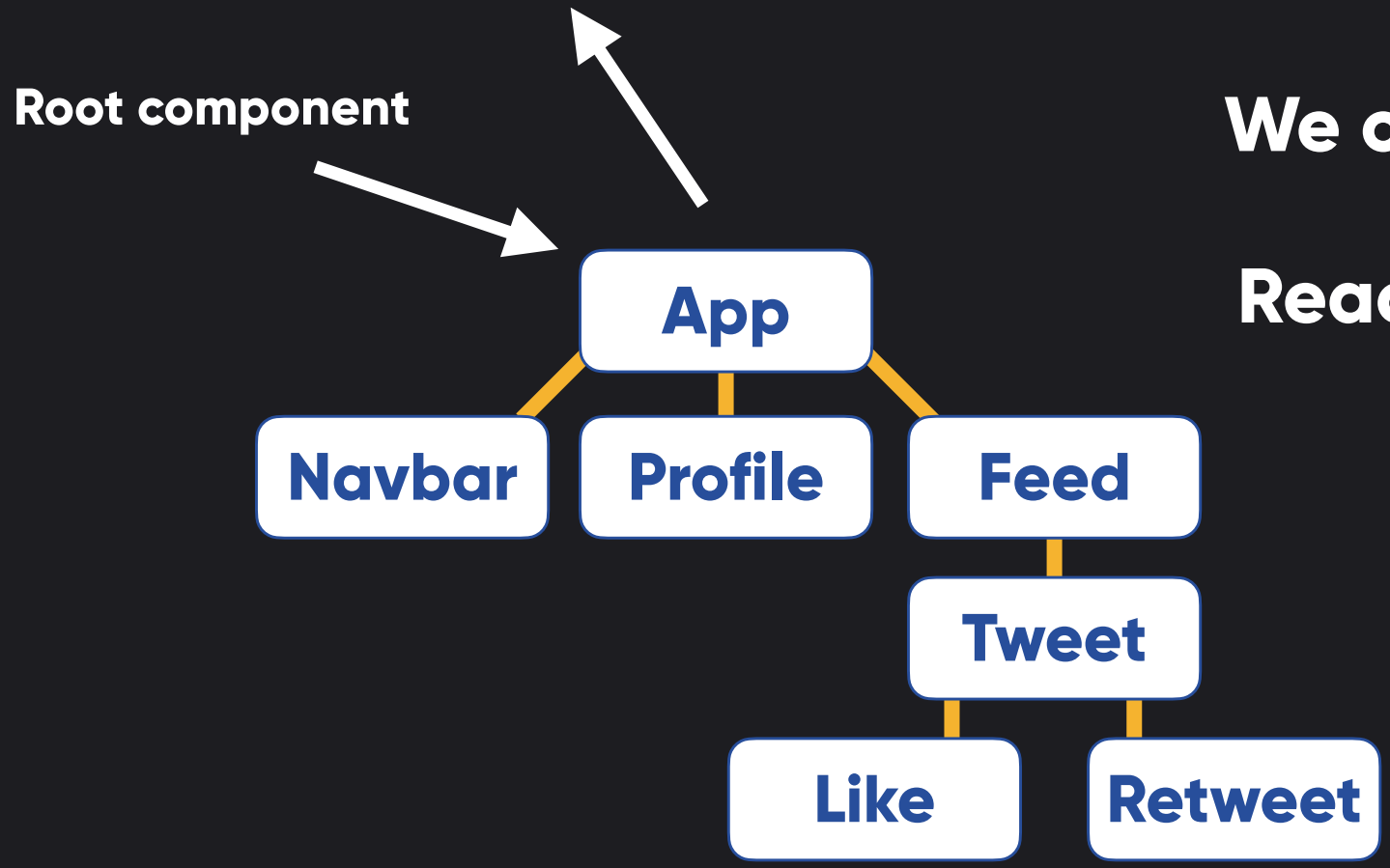
```
export const MyComponent = ( ) => { }
```

In a different file at the top:

```
import {MyComponent} from './somefile'
```

Inside index.js

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



We only need to call the ReactDOM.render( ) method once.

Create-react-app comes with it's own live server built in to help us with development. To start it up, run npm start in the terminal. This should automatically open a browser page.

To stop the server press control+c when you're in the terminal.

React.js

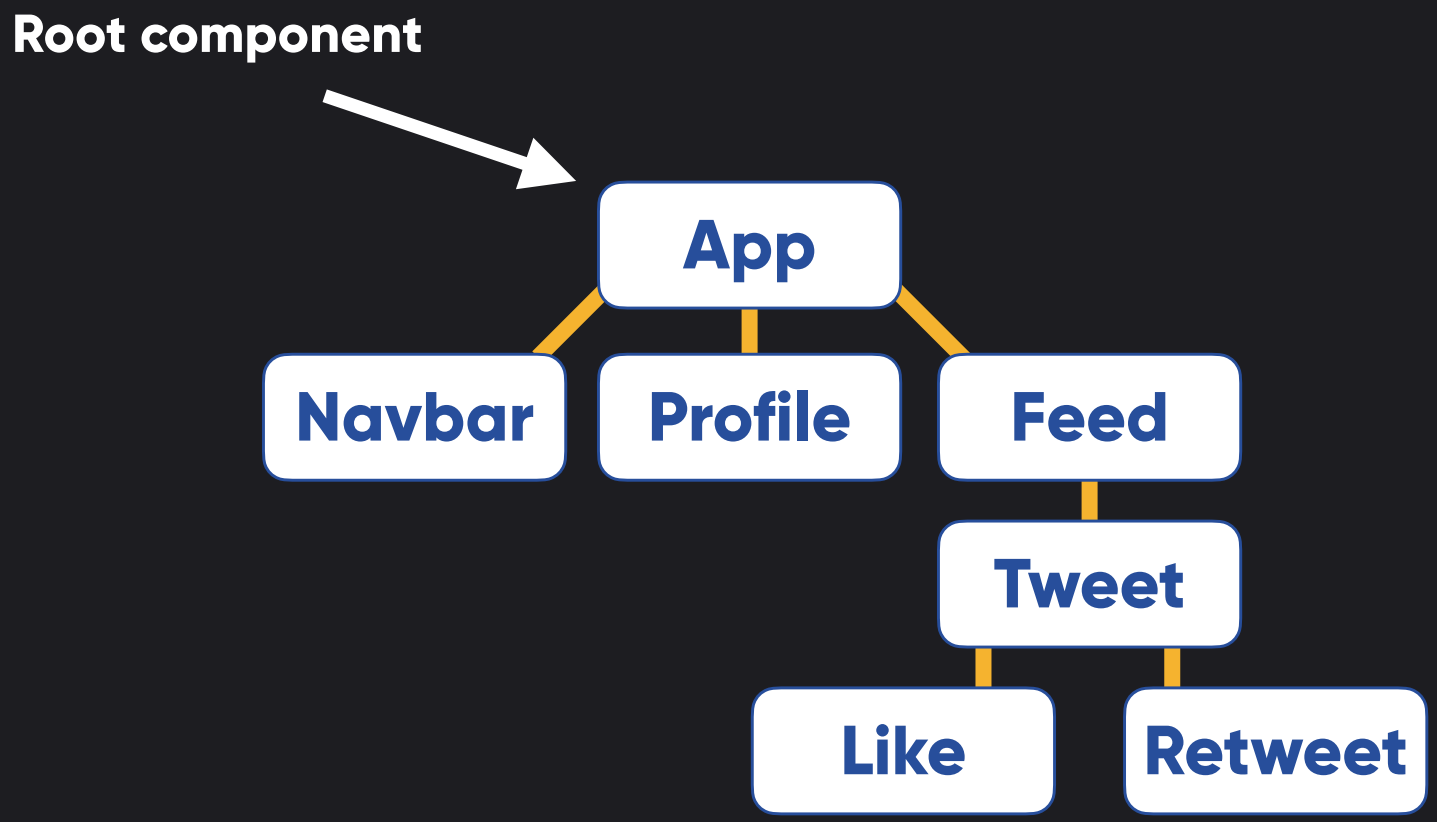
But how do we render our own components?

```jsx
const App = () => {
  return (
    <div>
      <h1>I'm the root component</h1>
      <Person />
    </div>
  );
}

const Person = () => {
  return <p>I'm a functional component</p>;
};

export default App;
```

Custom HTML elements

{CN}®

Remember earlier, when we mentioned React apps have a single root component. Now you know how to make one. Everything else can be rendered inside it.



React.js

Task: Render a functional component 3 times inside a root App component.

{CODENATION}

React.js

Custom HTML elements can be self-closing or not.

1. <Person />
2. <Person> </Person>

{CODENATION}

```
const App = () => {
  return (
    <div>
      <h1>I'm the root component</h1>
      <Person />
      <Person />
      <Person />
    </div>
  );
}

const Person = () => {
  return <p>I'm a functional component</p>;
};

export default App;
```

You should have ended up with something like this. The Person component is being rendered 3 times inside the App component.

{CODENATION}

React.js

Remember the websites we looked at which use react. They had the same components being repeated, but they had different text, or images.

Although the core component was the same, the data being passed to them was different.

React.js

So we use the same core component but pass different data to each one.

Let's have a look at how we might do that.

React.js

What do you remember about HTML attributes?

{CODENATION}

```
const App = () => {
    return (
      <div>
        <h1>I'm the root component</h1>
        <Person name="Dan" age="34" />
        <Person name="Stuart" age="30-something" />
        <Person name="Ben" age="21" />
      </div>
    );
}

const Person = (props) => {
  return (
    <p> Hi I'm {props.name} and I'm {props.age} </p>
  );
};

export default App;
```

{CODENATION}

React.js

In JSX, these HTML-like elements have attributes, but they behave a little differently.

React.js

When react renders the JSX and turns it into standard JS, it turns the attributes on our custom HTML elements into a JS object.

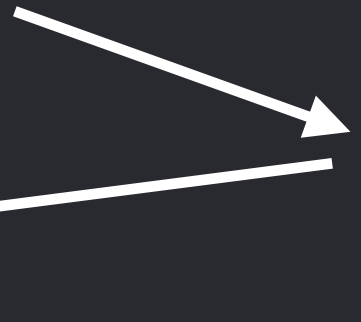
React.js

We refer to this object as props.

And the props object is passed to our components as a function argument.

{CODENATION}

```jsx
const App = () => {
    return (
      <div>
        <h1>I'm the root component</h1>
        <Person name="Dan" age="34" />
        <Person name="Stuart" age="30-something" />
        <Person name="Ben" age="21" />
      </div>
    );
}

const Person = (props) => {
  return (
    <p> Hi I'm {props.name} and I'm {props.age} </p>
  );
};

export default App;
```



```
props = {
    name: "Dan",
    age: "34"
    }
```

{CODENATION}

```jsx
const App = () => {
    return (
      <div>
        <h1>I'm the root component</h1>
        <Person name="Dan" age="34" />
        <Person name="Stuart" age="30-something" />
        <Person name="Ben" age="21" />
      </div>
    );
}

const Person = (props) => {
  return (
    <p> Hi I'm {props.name} and I'm {props.age} </p>
  );
};

export default App;
```

Note: we must pass props in as a parameter to our Component.

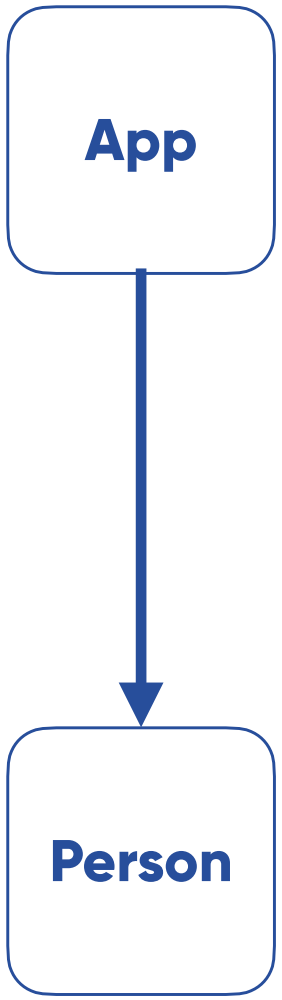
Also note: Whenever we use JS inside JSX, we wrap the JS in curly brackets.

React.js

Passing props is one of the ways we pass data down the hierarchy of components.

{CODENATION}

React.js



Data flows down the component tree

{CODENATION}

```
const App = () => {
  render() {
    return (
      <div>
        <h1>I'm the root component</h1>
        <Person name="Dan" age="34" />
        <Person name="Stuart" age="30-something" />
        <Person name="Ben" age="21" />
      </div>
    );
  }
}

const Job = (props) => {
  return (
    <p> Hi I'm {props.title} </p>
  );
};

export default App;
```

Copy out this example in your app.js file

{CODENATION}

Task:

Create a new functional component called Job - this component should return some text (a p tag for example) "I'm a chef"

Render your Job component in the App component. Save and check on the browser that it has rendered correctly.

Then pass some props to the Job component: job. Give it the value "junior developer".

Inside the Job component, replace the word "chef" with the props you have given to that component.

Pet's name property

React.js

Data flows down the component hierarchy, and we can keep passing props along.

{CODENATION}

Task:

Create a new functional component called Pet - this component should return some text (a p tag for example) "Hi my name is Ben"

Render your Pet component in the Person component. Save and check on the browser that it has rendered correctly.

Then pass some props from a Person in the App component: petsName and type. Give them any values you want.

Inside the Person component create a new props for your Pet component. Then finally inside the Pet component, change the sentence so it uses the values that started

```
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>I'm a class component</h1>
        <Person name="Dan" age="34" />
        <Person name="Stuart" age="30-something" />
        <Person name="Ben" age="21" petsName="john" type="dog" />
      </div>
    );
  }
}

const Person = props => {
  return (
    <div>
      <p>Hi I'm {props.name} and I'm {props.age}</p>
      <Pet pet={props.petsName} />
    </div>
  );
};

const Pet = props => {
  return <p>Hi my pets name is {props.pet}</p>;
};
```

Props: {
name: Ben,
Age: 21,
petsName: John,
Type: dog
}

{CODENATION}

Basic styling in React

We can use CSS in react like we would in standard HTML (exactly the same)

However, as we are working in JS we can't use the class attribute. Can you think why that might be?

{CODENATION}

Basic styling in React

We can style our react components by className, id, or tag name – and the css is the same.

However, we must remember to import the css file at the top of our JS file.

{CODENATION}

JS/React

```
import "./App.css";

const App = () => {
   return (
      <div className="container">
        <h1 id="title">I'm a class component</h1>
      </div>
   );
}
```

CSS

```
.container {
  height: 600px;
  width: 600px;
  background-color: lightpink;
  display: flex;
  justify-content: center;
  align-items: center;
}

#title {
  color: darkred;
}
```

{CODENATION}

I'm a class component

Importing images

Note: You import images in React in the same way that you import and other file. You must have the image in the src directory (create a folder called images to put them in)

Let's look at this together.

{CODENATION}

React.js

Recreate components

Task: I am going to send you a jpg on slack. You need to decide how you might break the image into components, then put your components together so they match the image.

{CN}®

Revisiting Learning Objectives

To interpret what React is and why we would use it.

To be able to create your own components and understand what props are.

{CODENATION}