React Router & Review

Sept. 2022 CT | Week 10, Day 2 | **Building a Single Page Application**

Day 3.

Learning Objectives

1. Review
    a. Class & Functional Components
    b. **.setState( )** & Component Lifecycle
    c. Event-Handling & Fetching asynchronously

Tomorrow: Trivia Game with Leaderboards & Prizes

2. React Router
    a. <BrowserRouter> & <Route(s)>
    b. <Link>, <NavLink> & nested links: <Outlet/>
    c. Reading through a codebase

    Github & CodeSandbox ⟶ **More on Git tomorrow**

3. Q&A:
    This afternoon with
    Warren Longmire

slido

Join at slido.com #056355

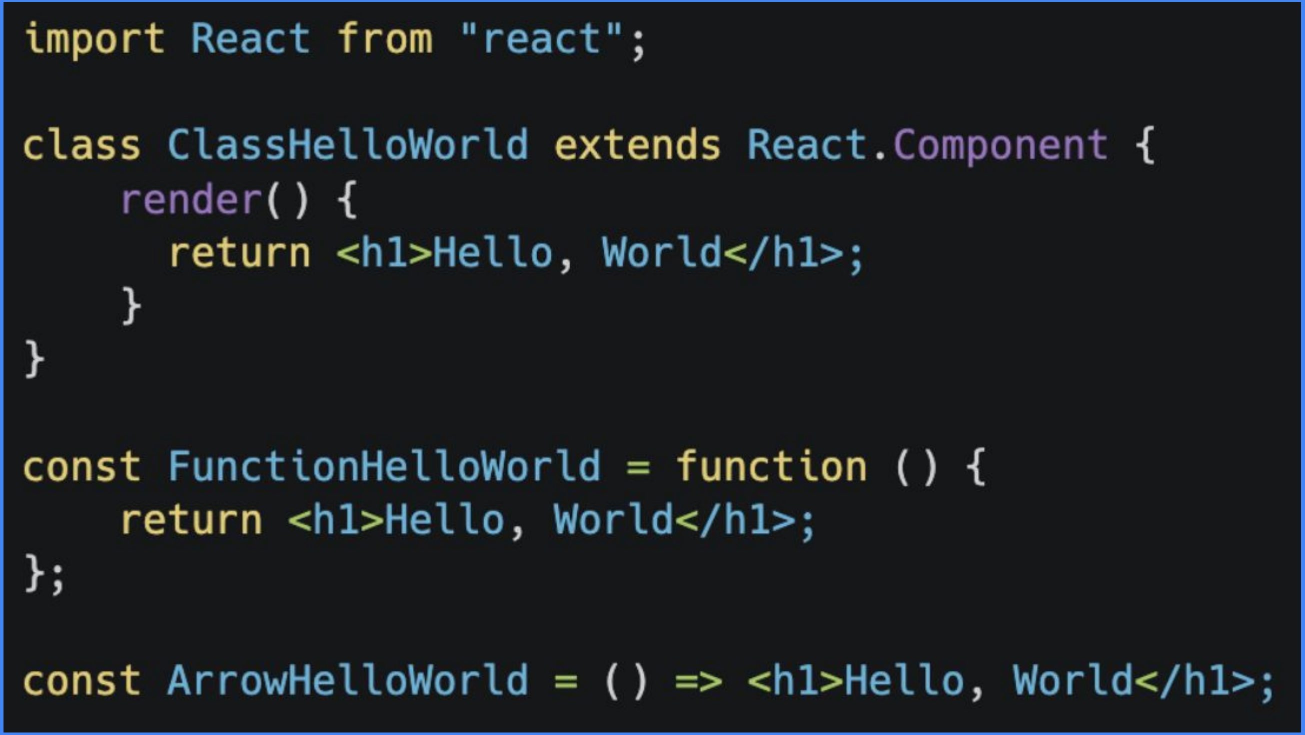
ⓘ Start presenting to display the joining instructions on this slide.

slido

HACK REACTOR

What are some key differences between functional and class components?

Class v. Functional Components



Class Component

```
import React from "react";

class ClassHelloWorld extends React.Component {
    render() {
      return <h1>Hello, World</h1>;
    }
}
```

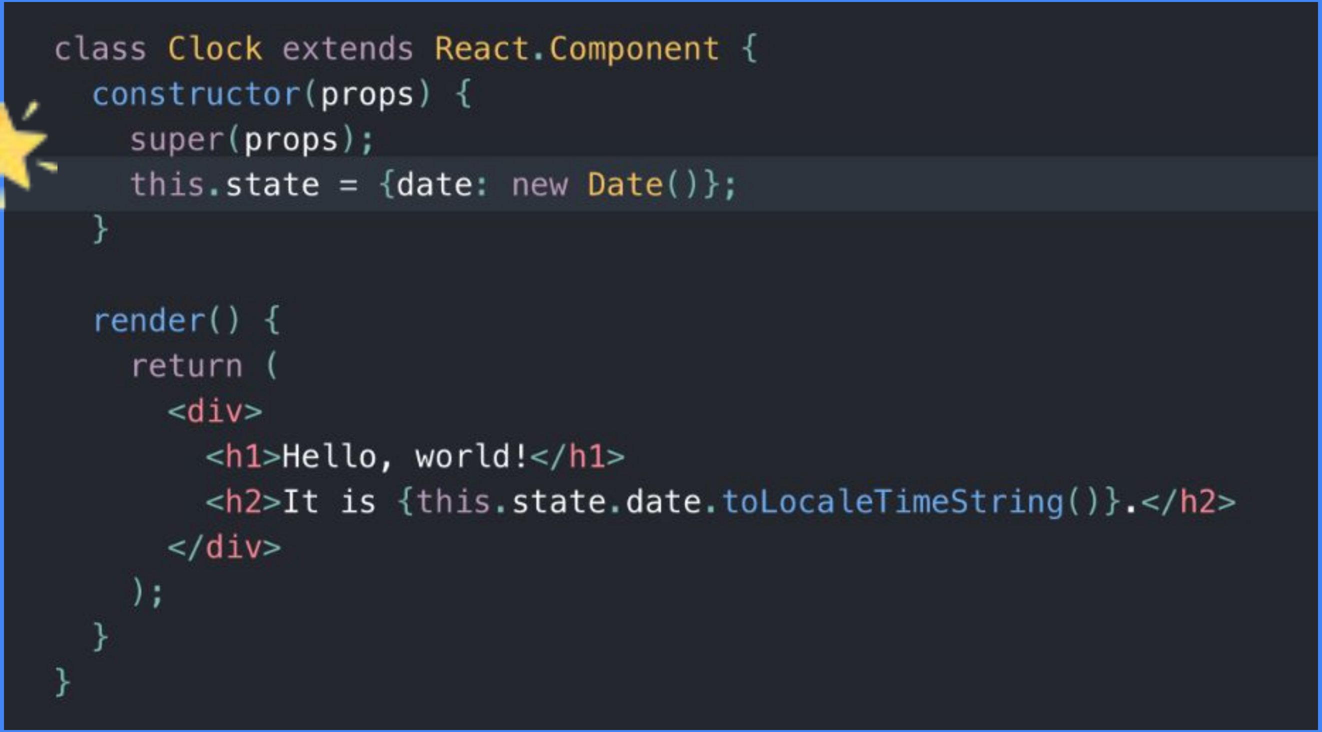
Functional Component

```
const FunctionHelloWorld = function () {
    return <h1>Hello, World</h1>;
};
```

Arrow Function

```
const ArrowHelloWorld = () => <h1>Hello, World</h1>;
```

QUICK REVIEW



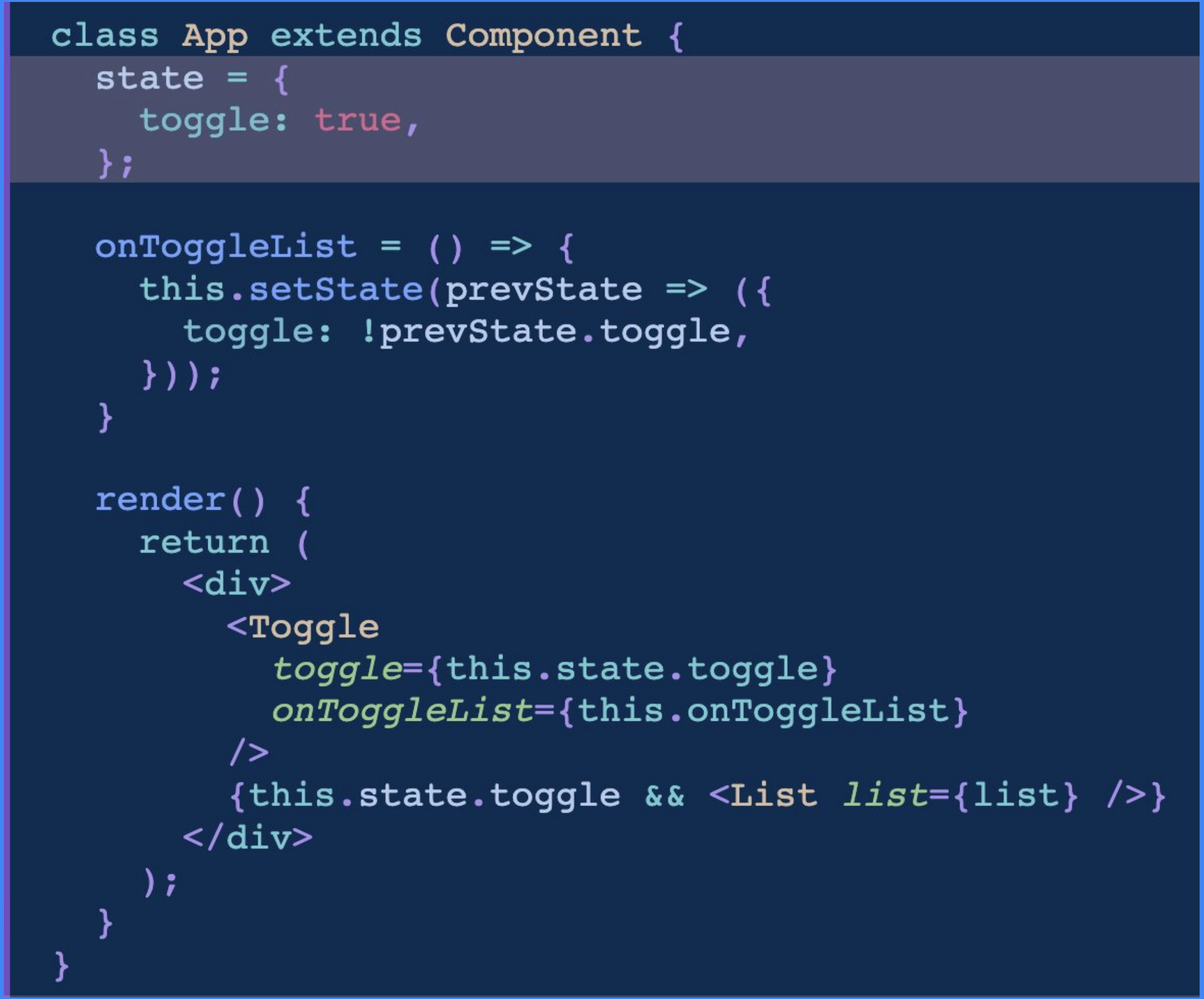
```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

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

Class
Components

Classic object-oriented syntax:

* constructor( )
* super( )
* this
* .bind( )

**** Your labs in Learn uses this syntax.**



```
class App extends Component {
  state = {
    toggle: true,
  };

  onToggleList = () => {
    this.setState(prevState => ({
      toggle: !prevState.toggle,
    }));
  }

  render() {
    return (
      <div>
        <Toggle
          toggle={this.state.toggle}
          onToggleList={this.onToggleList}
        />
        {this.state.toggle && <List list={list} />}
      </div>
    );
  }
}
```

New Class Components Syntax

- **state** object
- No binding methods with arrow functions.

slido

HACK REACTOR

What is the difference between state and props?

State & Props

- Data in a React component is stored locally in the component in an javascript object called **STATE**.

- Parent components can pass data down to their child components via HTML attributes used in React components as arguments called **PROPS**.

State in Class Components

- The **state** object is a mutable store of data inside the component.

- To change a value in the state object, you have to use the **this.setState()** method.

- When a value in the state object changes, the component will re-render. This process takes time, and setState( ) is **asynchronous**.

Yesterday

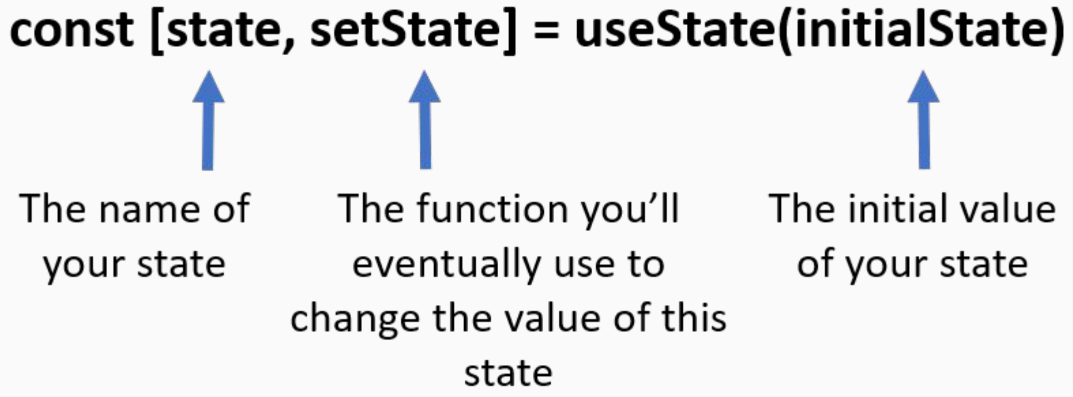
```
class LocationForm extends React.Component {
  constructor(props) {
    super(props)
    this.state = {states: []};
    this.handleNameChange = this.handleNameChange.bind(this);
  }


  handleNameChange(event) {
    const value = event.target.value;
    this.setState({name: value})
  }
```

```
<input onChange={this.handleNameChange} placeholder="Name" required
       type="text" name="name" id="name"
       className="form-control" />
```

State in Functional Components

with a Hook: **useState( )**

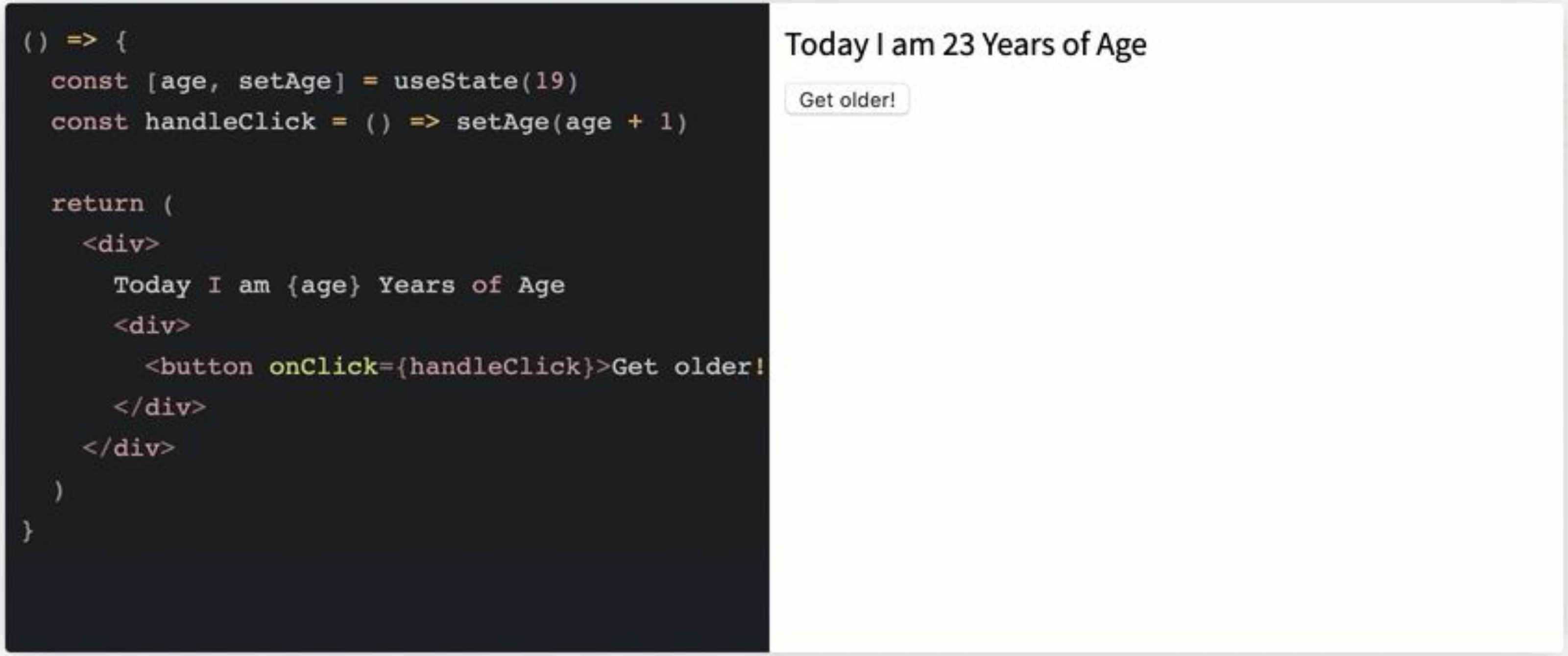
useState( ) lets you access, set and update local state in a functional component.



```
const [state, setState] = useState(initialState)
```

The name of your state
The function you'll eventually use to change the value of this state
The initial value of your state

useState Hook Example



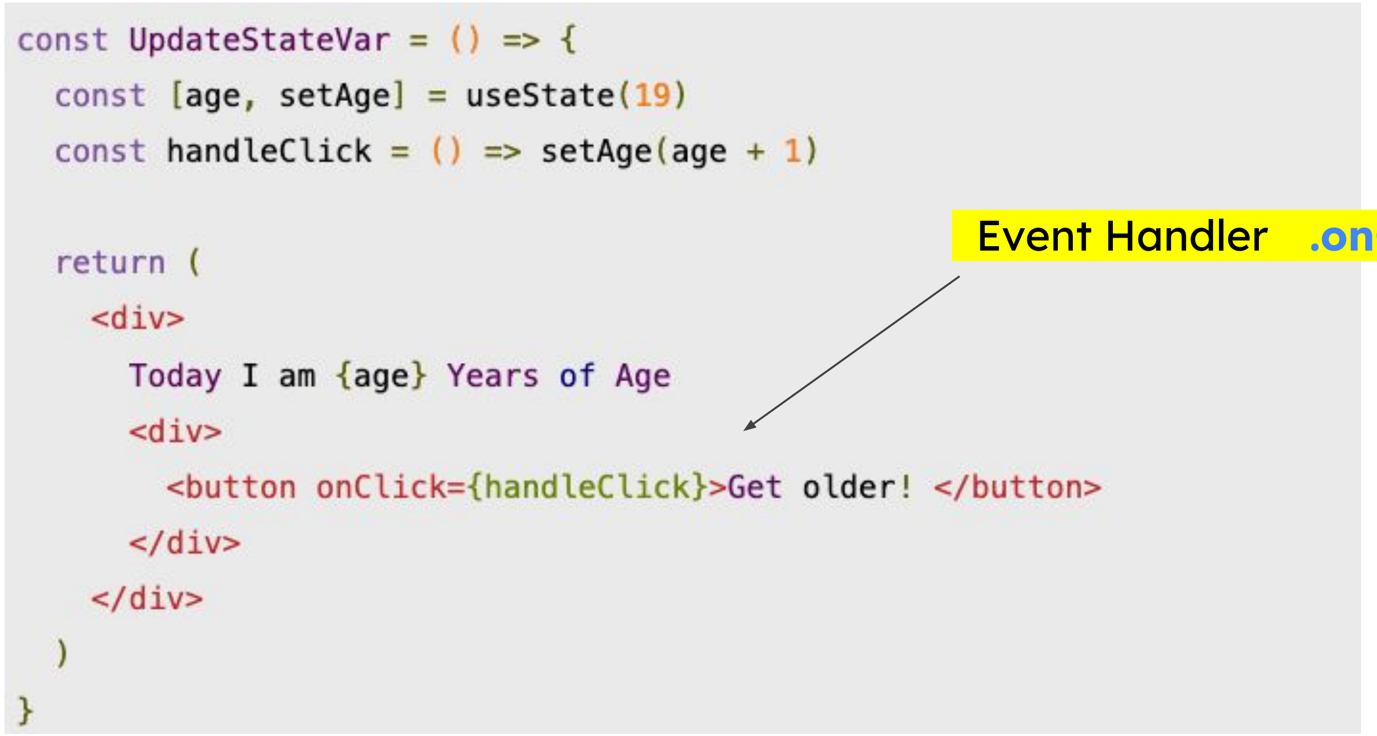
```
() => {
  const [age, setAge] = useState(19)
  const handleClick = () => setAge(age + 1)

  return (
    <div>
      Today I am {age} Years of Age
      <div>
        <button onClick={handleClick}>Get older!
      </div>
    </div>
  )
}
```

Today I am 23 Years of Age

u

Source: LogRocket blog post

useState Hook Code



```
const UpdateStateVar = () => {
  const [age, setAge] = useState(19)
  const handleClick = () => setAge(age + 1)

  return (
    <div>
      Today I am {age} Years of Age
      <div>
        <button onClick={handleClick}>Get older! </button>
      </div>
    </div>
  )
}
```

Event Handler .onClick

Source: LogRocket blog post

slido

HACK REACTOR

What is the difference between `ComponentDidMount( )` and `ComponentDidUpdate( )`?

Component Lifecycle

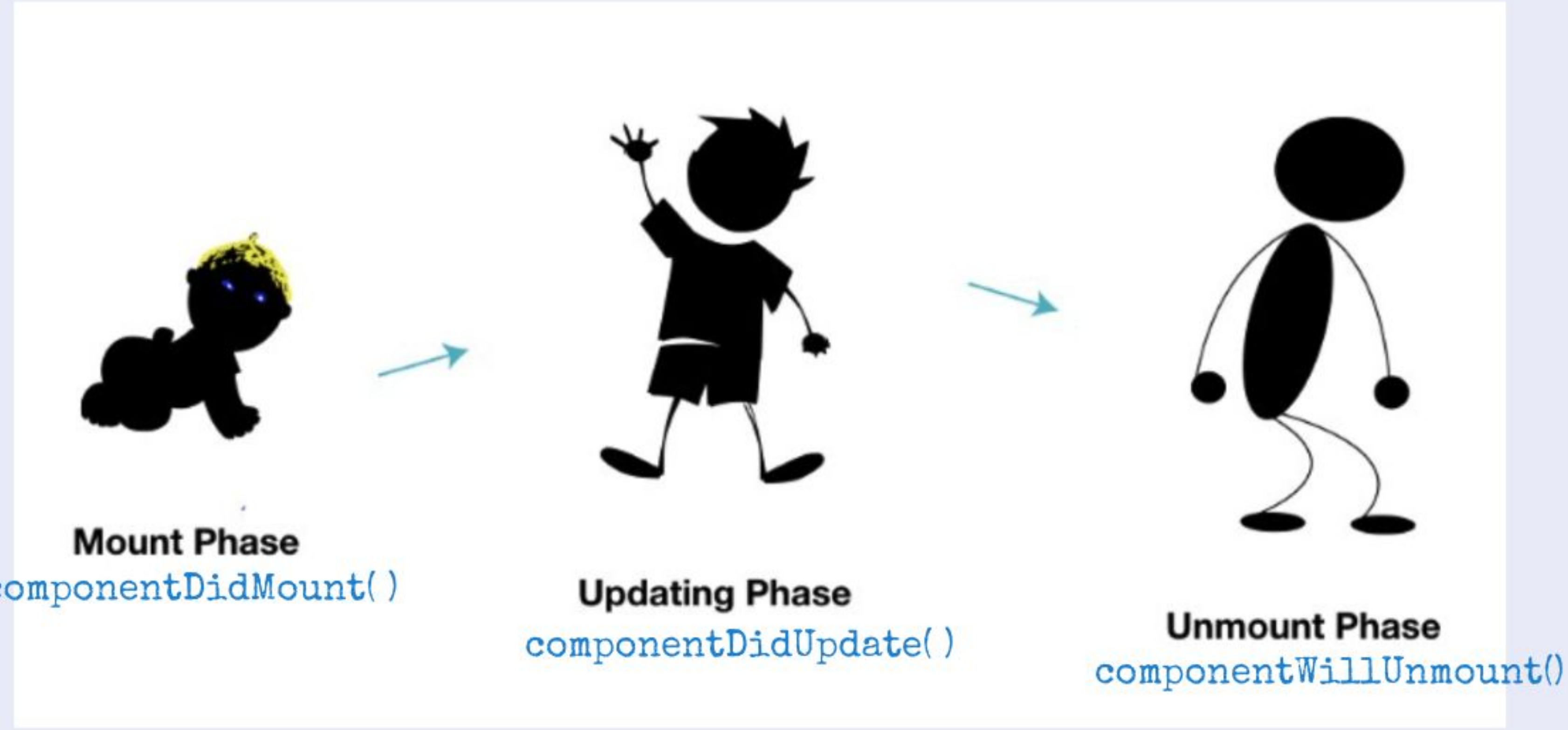
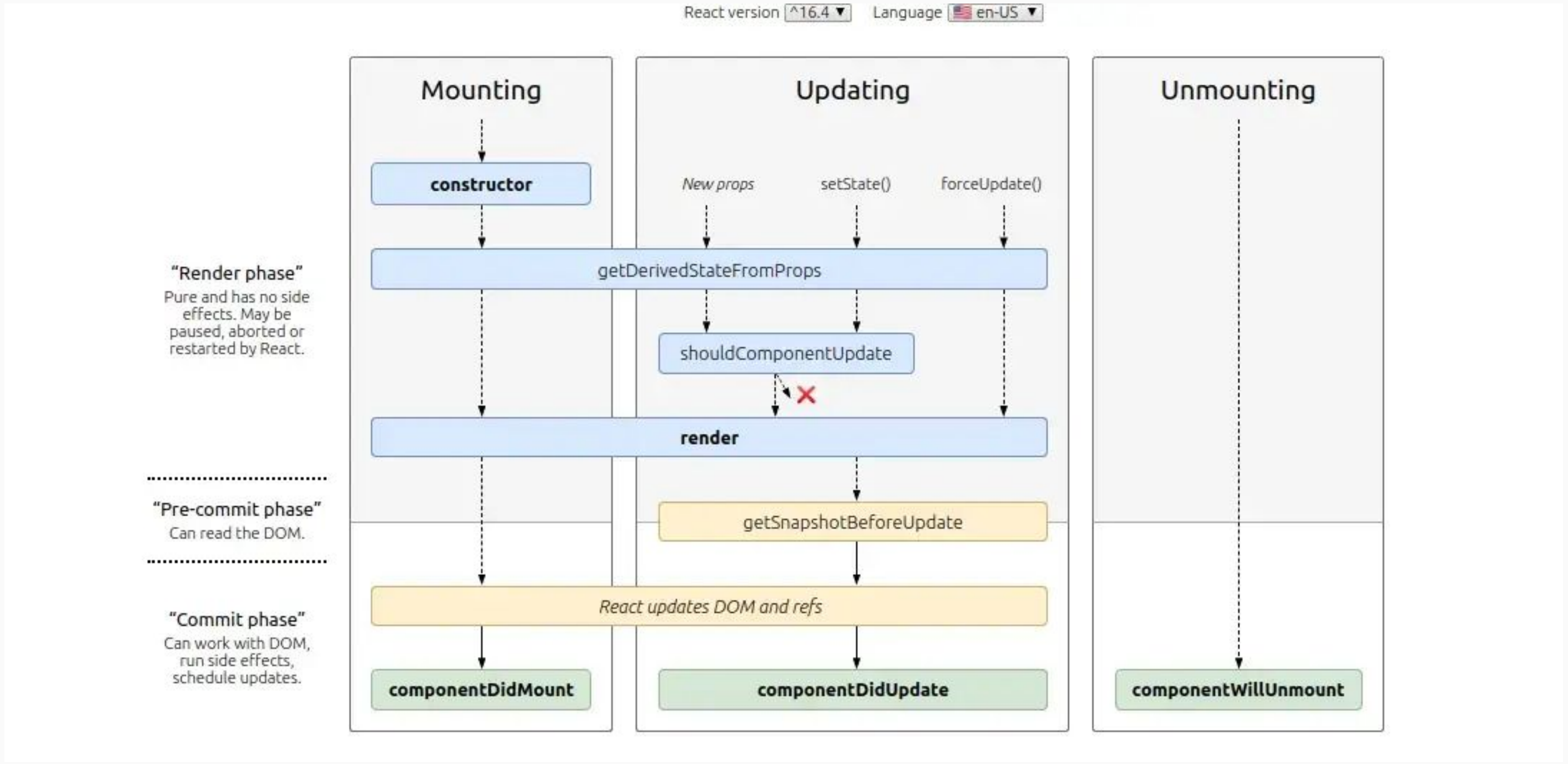
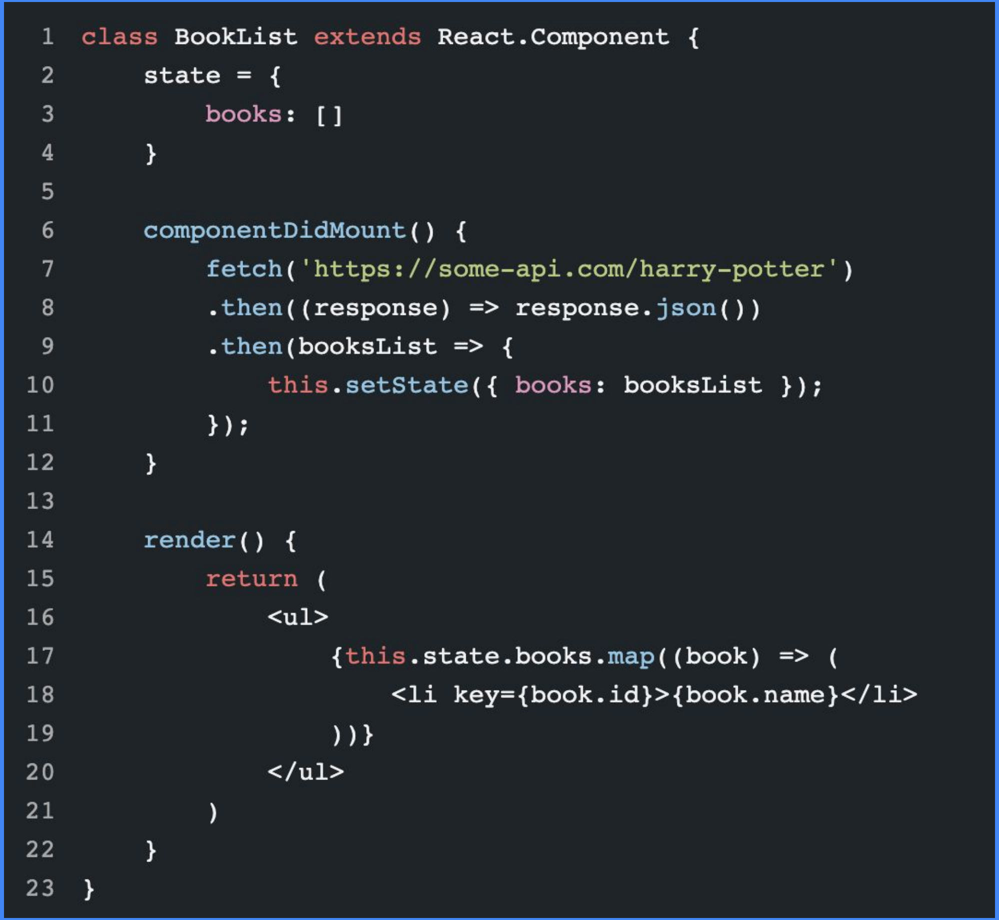


Image Source: https://leanpub.com/reintroducing-react

React Component Lifecycle





```
class BookList extends React.Component {
    state = {
        books: []
    }

    componentDidMount() {
        fetch('https://some-api.com/harry-potter')
        .then((response) => response.json())
        .then(booksList => {
            this.setState({ books: booksList });
        });
    }

    render() {
        return (
            <ul>
                {this.state.books.map((book) => (
                    <li key={book.id}>{book.name}</li>
                ))}
            </ul>
        )
    }
}
```

Full
BookList Component

componentDidUpdate( )

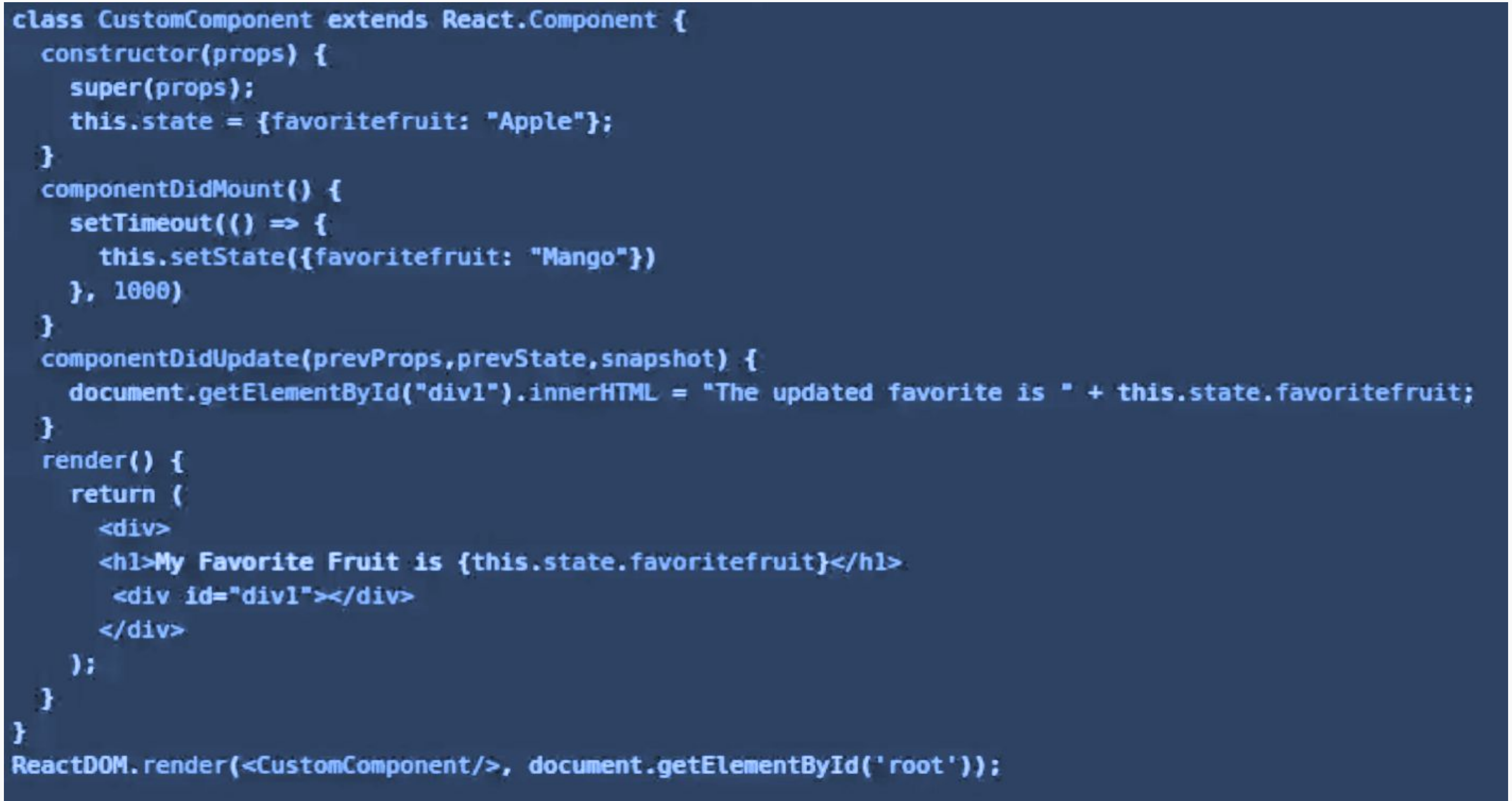
Every change in state or props.

```
componentDidUpdate(prevProps, prevState, snapshot)
```

```
componentDidUpdate(prevProps) {
  // Typical usage (don't forget to compare props):
  if (this.props.userID !== prevProps.userID) {
    this.fetchData(this.props.userID);
  }
}
```

You can get an infinite loop if you .setState( ) inside a componentDidUpdate( ) unless you wrap it in a stop condition.

DISCUSSION: What’s Happening Here?



```
class CustomComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritefruit: "Apple"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritefruit: "Mango"})
    }, 1000)
  }
  componentDidUpdate(prevProps,prevState,snapshot) {
    document.getElementById("div1").innerHTML = "The updated favorite is " + this.state.favoritefruit;
  }
  render() {
    return (
      <div>
      <h1>My Favorite Fruit is {this.state.favoritefruit}</h1>
       <div id="div1"></div>
      </div>
    );
  }
}
ReactDOM.render(<CustomComponent/>, document.getElementById('root'));
```

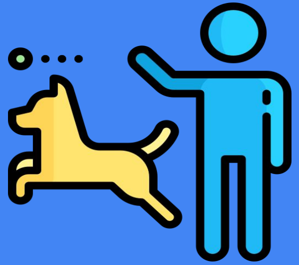
slido

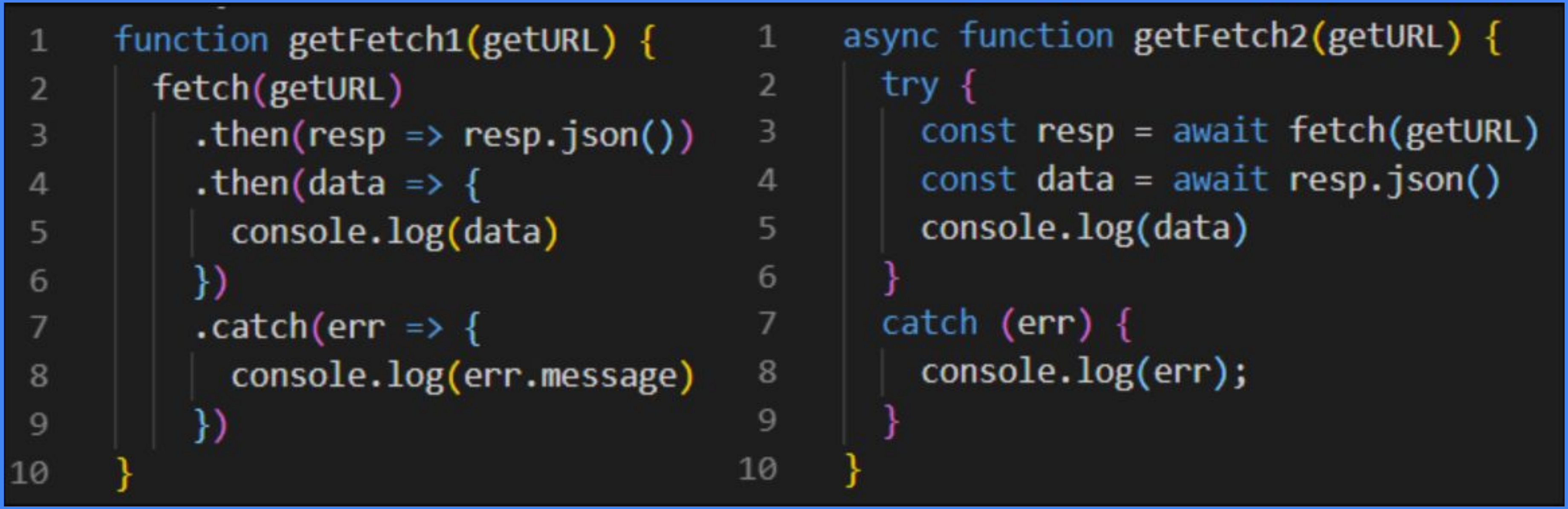
HACK REACTOR

Put these steps of lifecycle methods in order:

Asynchronous Fetch API

Two ways: **.then( )** & **async/await**

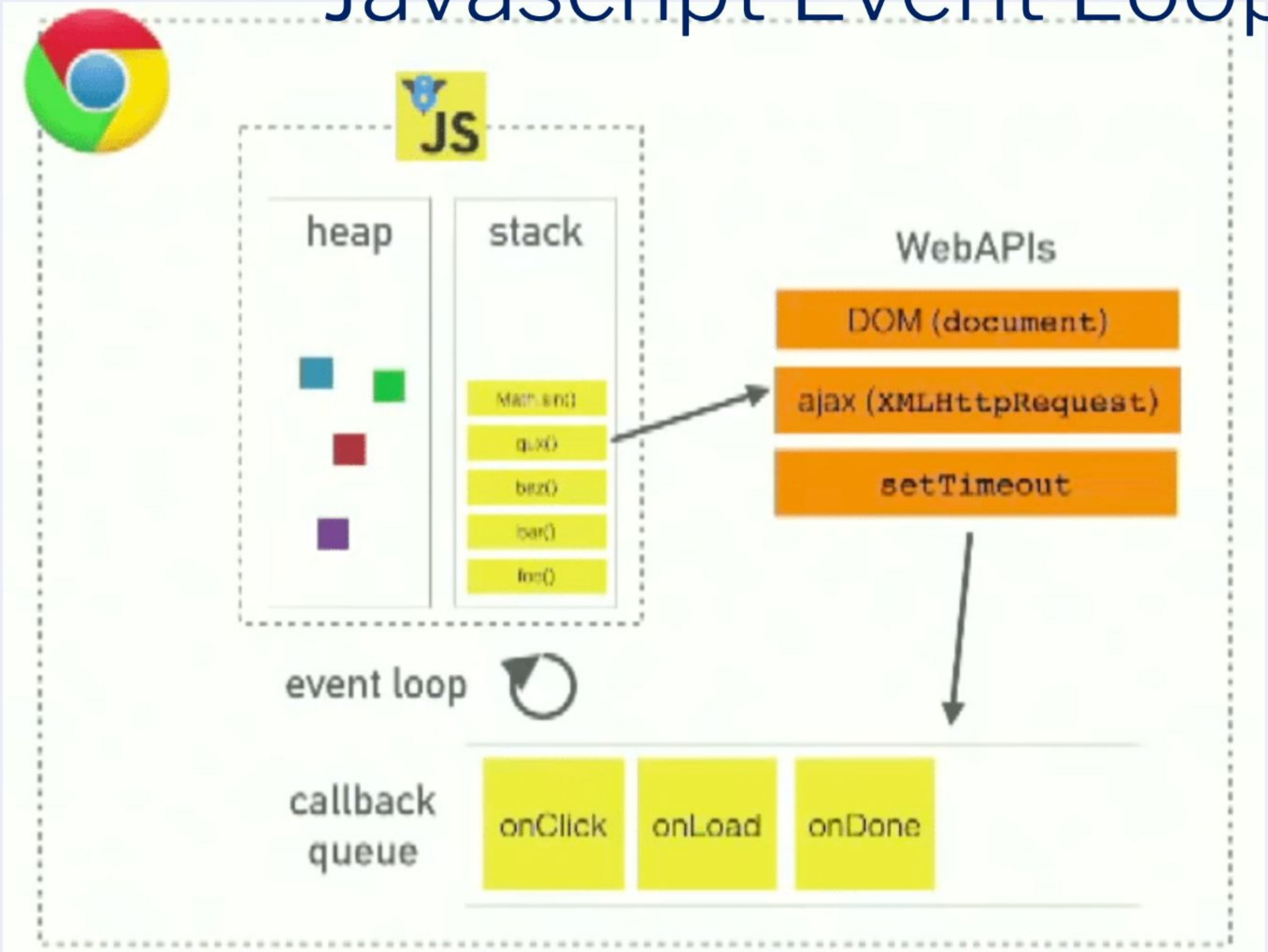




```
function getFetch1(getURL) {
  fetch(getURL)
    .then(resp => resp.json())
    .then(data => {
      console.log(data)
    })
    .catch(err => {
      console.log(err.message)
    })
}
```

```
async function getFetch2(getURL) {
  try {
    const resp = await fetch(getURL)
    const data = await resp.json()
    console.log(data)
  }
  catch (err) {
    console.log(err);
  }
}
```

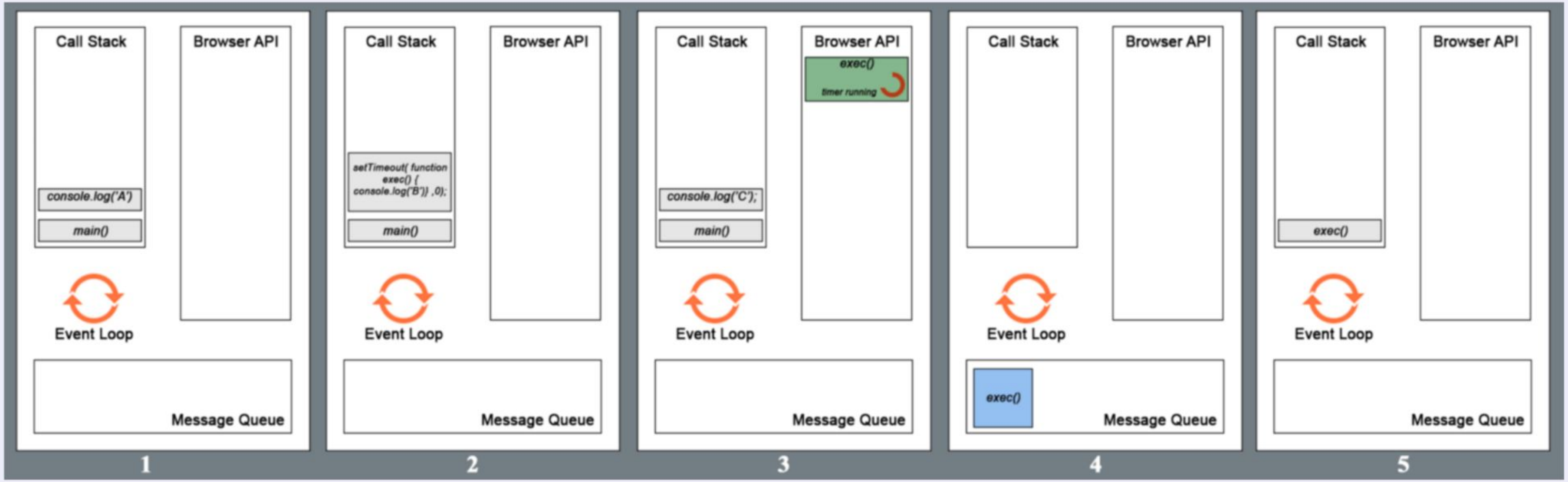
Javascript Event Loop



Javascript is single-threaded

Your Browser is multi-threaded

Javascript Execution Stack



.setState( ) is <u>asynchronous</u> like .setTimeout( )

Pop Quiz!

```
class Counter extends Component {
  constructor() {
    this.state = {
      counter: 0
    }
  }

  render() { … }
}
```

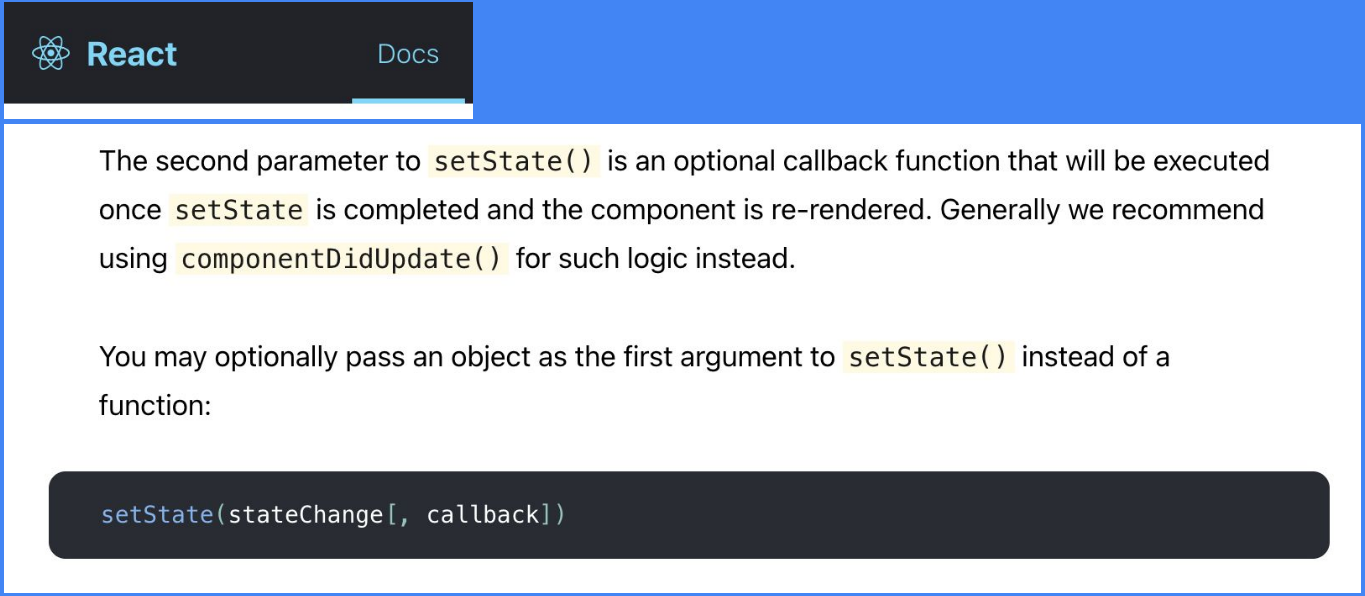
What will we see in the console?

```
this.setState({ count: this.state.count + 1 });
this.setState({ count: this.state.count + 1 });
this.setState({ count: this.state.count + 1 });

console.log(this.state.count);
```

slido

What will we see in the console?

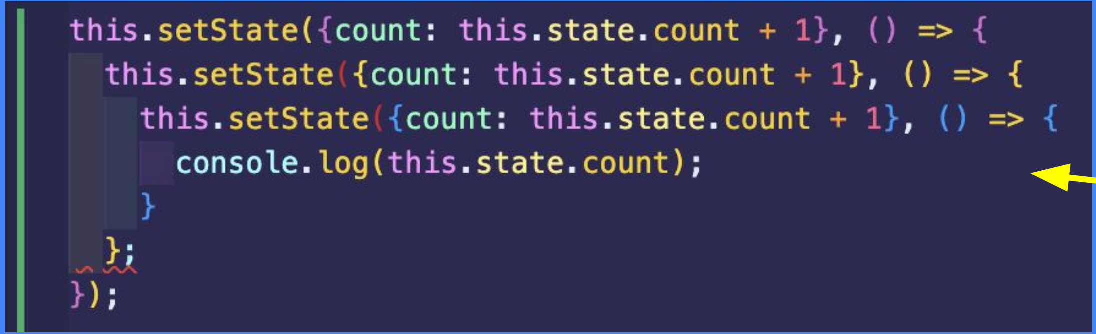
.setState( ) **callback**


React Docs

The second parameter to `setState()` is an optional callback function that will be executed once `setState` is completed and the component is re-rendered. Generally we recommend using `componentDidUpdate()` for such logic instead.

You may optionally pass an object as the first argument to `setState()` instead of a function:

```
setState(stateChange[, callback])
```

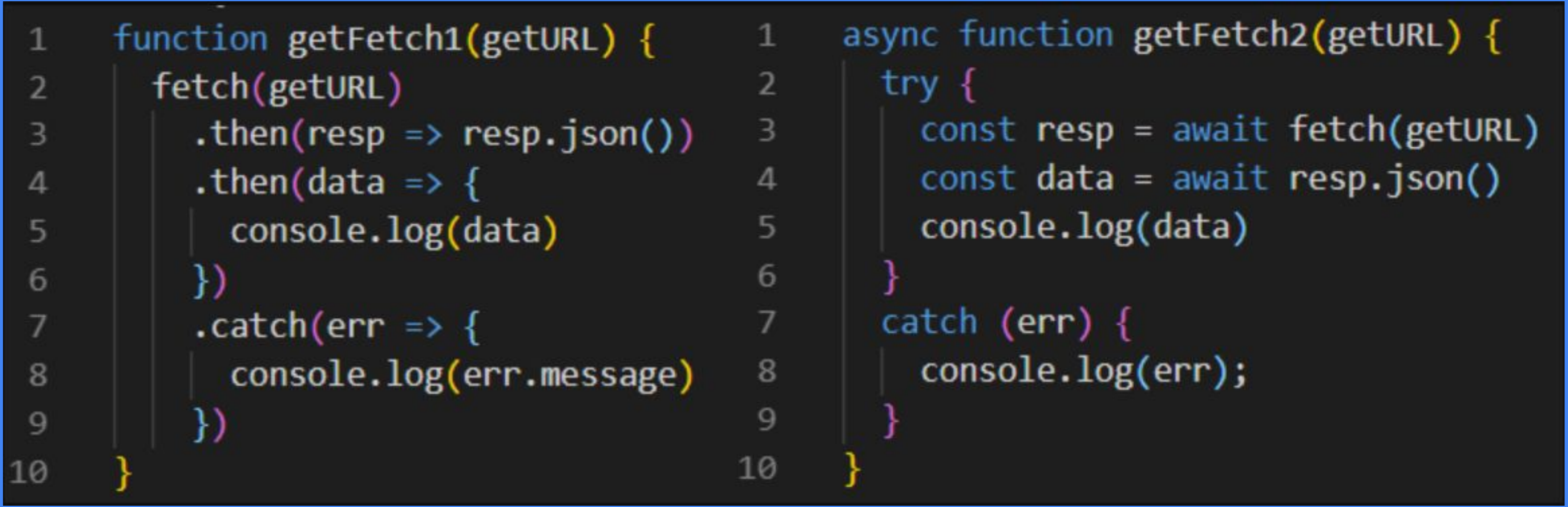


```
this.setState({count: this.state.count + 1}, () => {
  this.setState({count: this.state.count + 1}, () => {
    this.setState({count: this.state.count + 1}, () => {
      console.log(this.state.count);
    }
  };
});
```

AKA “Callback Hell”

Fetch with Async/Await

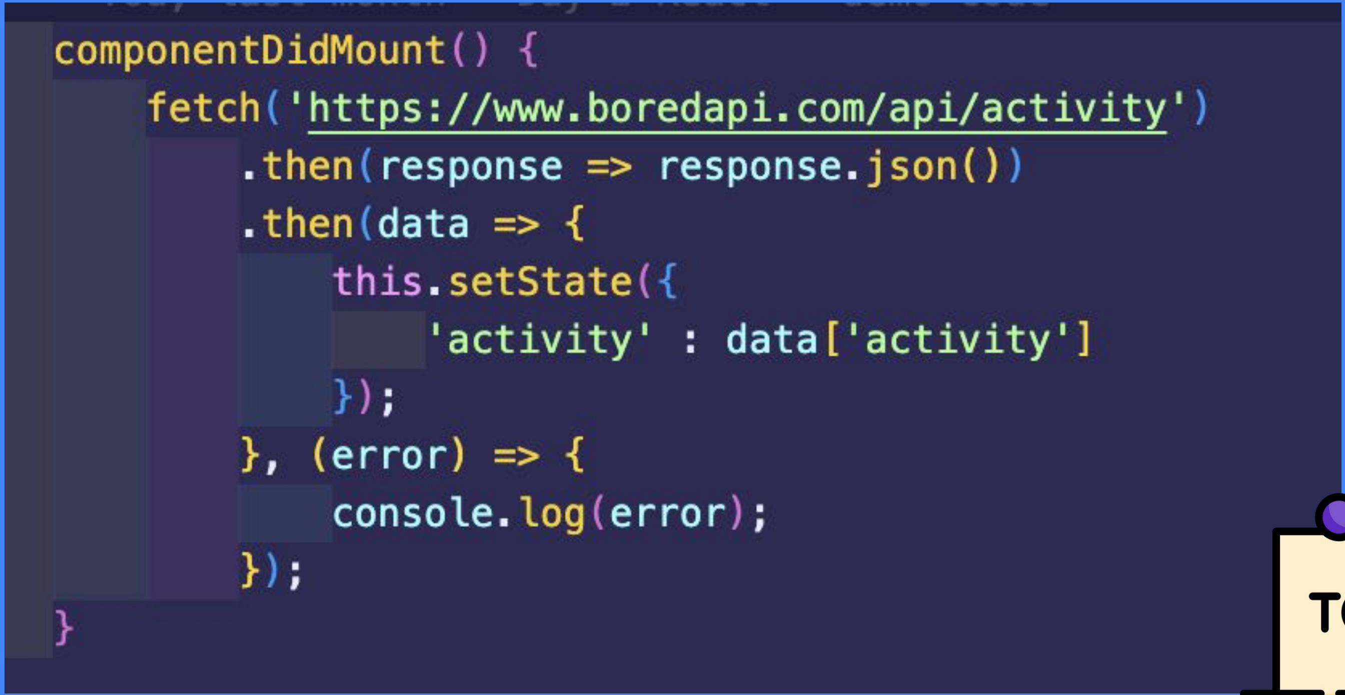
and **Try / Catch**



```
function getFetch1(getURL) {
  fetch(getURL)
    .then(resp => resp.json())
    .then(data => {
      console.log(data)
    })
    .catch(err => {
      console.log(err.message)
    })
}
```

```
async function getFetch2(getURL) {
  try {
    const resp = await fetch(getURL)
    const data = await resp.json()
    console.log(data)
  }
  catch (err) {
    console.log(err);
  }
}
```

Fetch in componentDidMount()



```
componentDidMount() {
    fetch('https://www.boredapi.com/api/activity')
        .then(response => response.json())
        .then(data => {
            this.setState({
                'activity' : data['activity']
            });
        }, (error) => {
            console.log(error);
        });
}
```

useEffect( ) Hook

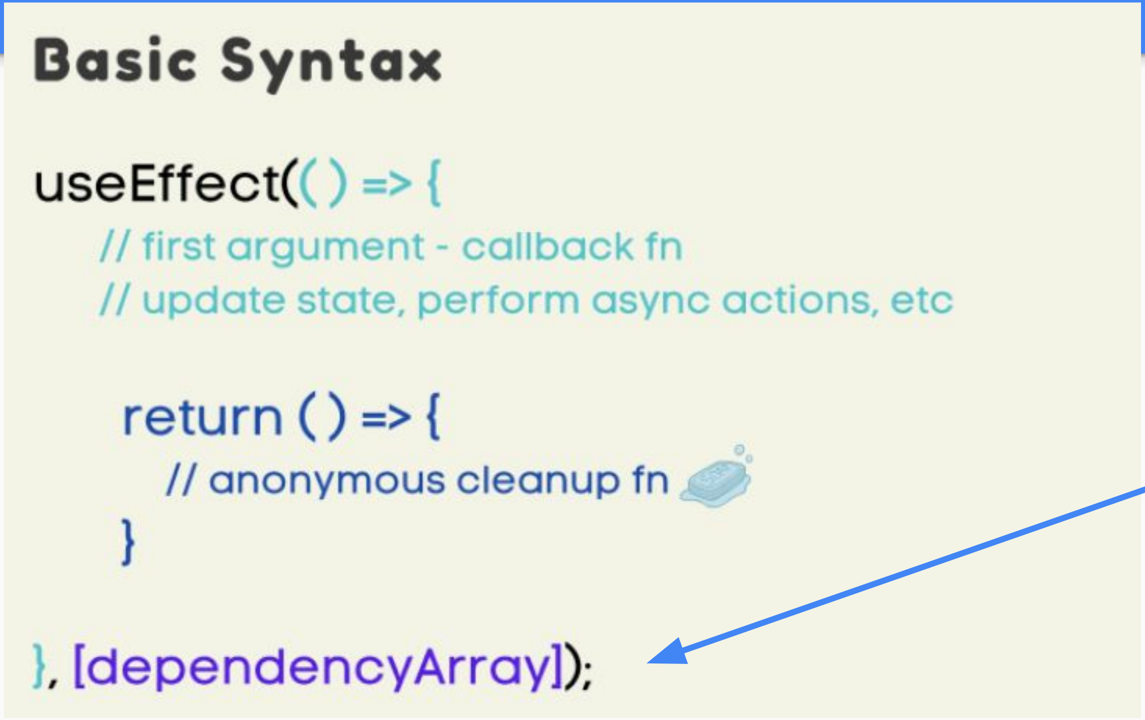
```
import {useEffect} from 'react'
```

useEffect( ) lets you perform “side effects” that automatically trigger when a component renders, or when a certain state changes.

```
useEffect(effectFunction, dependenciesArray);
```

It expects a function as its first parameter and an optional dependency array as its second.

useEffect( ) Hook

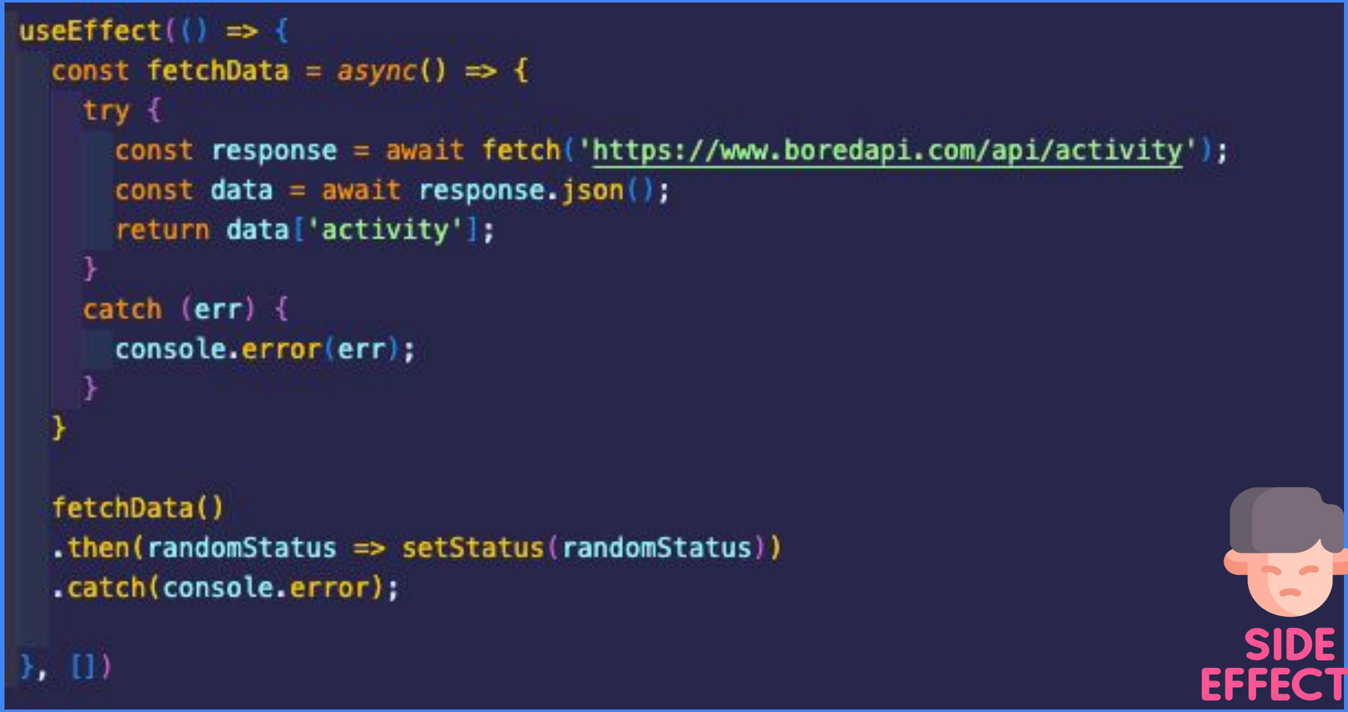


Basic Syntax

```
useEffect(( ) => {
    // first argument - callback fn
    // update state, perform async actions, etc

     return ( ) => {
       // anonymous cleanup fn
     }

}, [dependencyArray]);
```

If an empty array is passed, the function will only be run on the first render of the component.

Fetch in useEffect( )



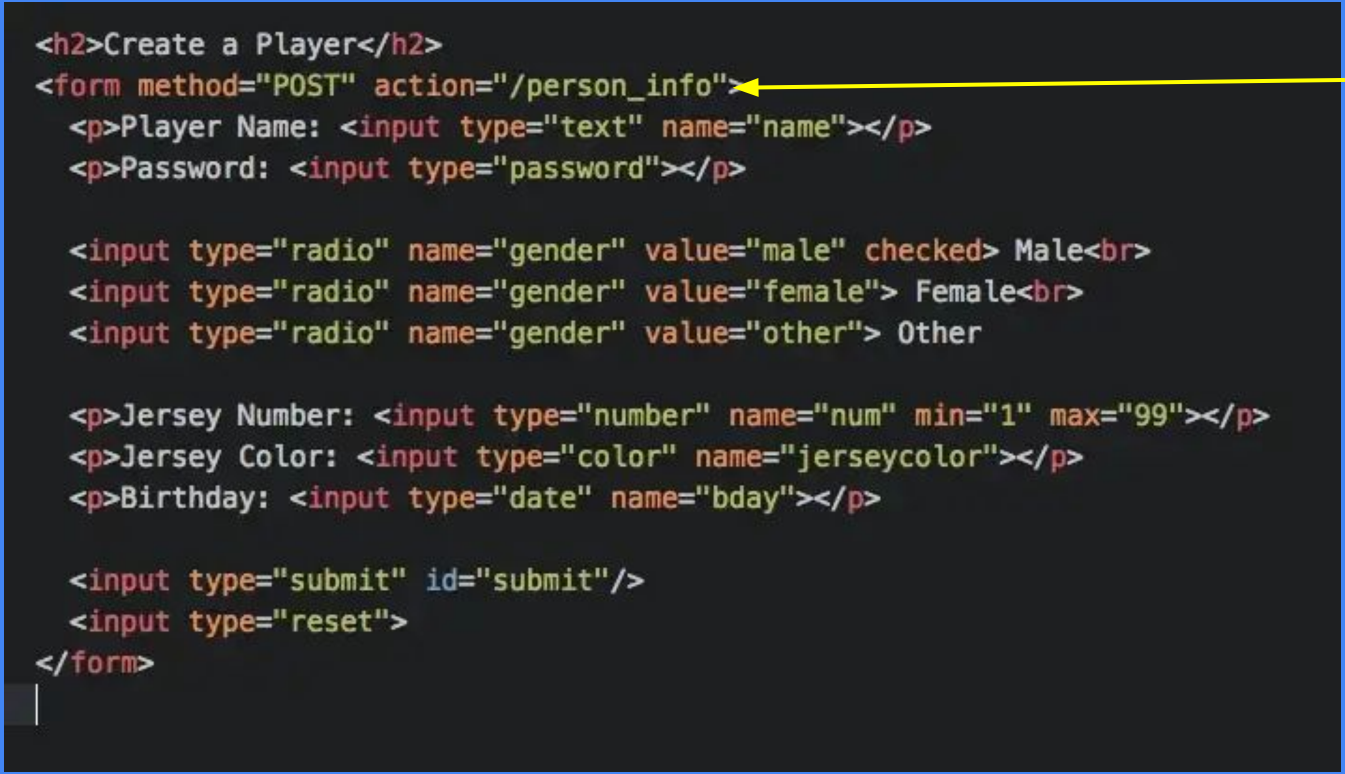
```
useEffect(() => {
  const fetchData = async() => {
    try {
      const response = await fetch('https://www.boredapi.com/api/activity');
      const data = await response.json();
      return data['activity'];
    }
    catch (err) {
      console.error(err);
    }
  }

  fetchData()
  .then(randomStatus => setStatus(randomStatus))
  .catch(console.error);

}, [])
```

slido

HACK REACTOR

Put these steps of a Fetch request in order:

REVIEW: HTML FORMS



```
<h2>Create a Player</h2>
<form method="POST" action="/person_info">
  <p>Player Name: <input type="text" name="name"></p>
  <p>Password: <input type="password"></p>

  <input type="radio" name="gender" value="male" checked> Male<br>
  <input type="radio" name="gender" value="female"> Female<br>
  <input type="radio" name="gender" value="other"> Other

  <p>Jersey Number: <input type="number" name="num" min="1" max="99"></p>
  <p>Jersey Color: <input type="color" name="jerseycolor"></p>
  <p>Birthday: <input type="date" name="bday"></p>

  <input type="submit" id="submit"/>
  <input type="reset">
</form>
```

onSubmit={ } goes in the opening **<form>** tag.

onChange={ } can go in any input tag.

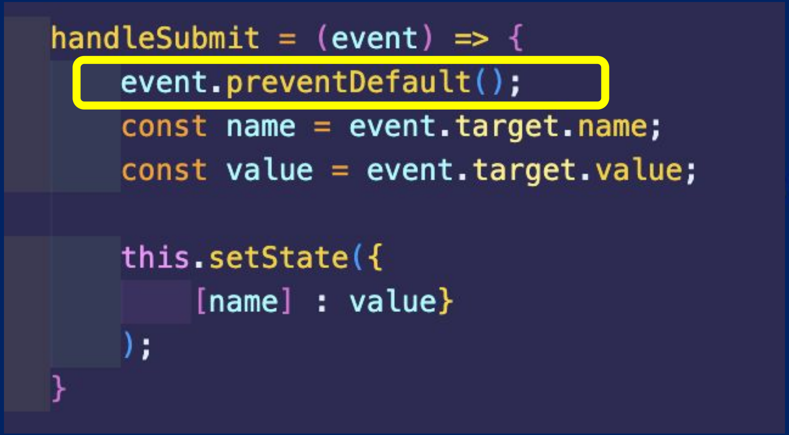
MDN Docs

Event Handling

```
.onClick()
.onSubmit()
```

```
<div>
  <button onClick={this.showAlert}>Click Me</button>
</div>
```

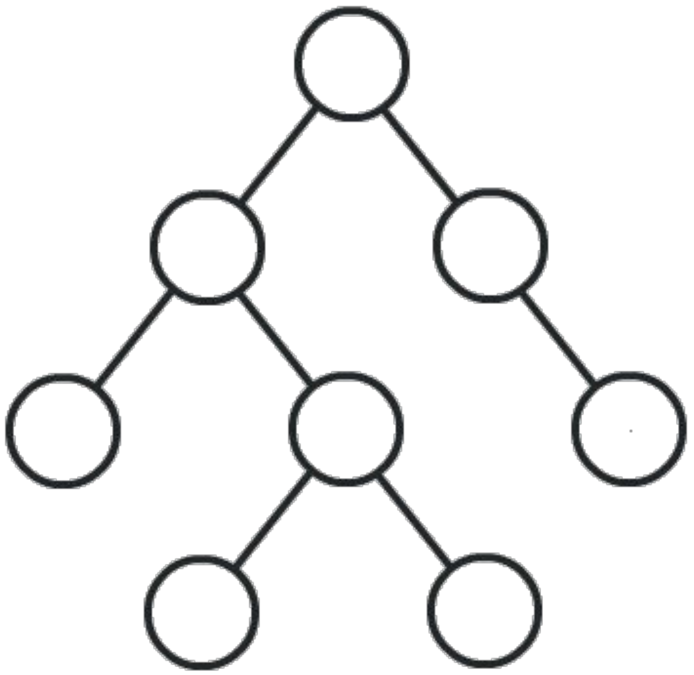
Event listener names are camelCase instead of lowercase, and we pass a function in as the event handler, rather than a string.



```
handleSubmit = (event) => {
    event.preventDefault();
    const name = event.target.name;
    const value = event.target.value;

    this.setState({
        [name] : value}
    );
}
```

Data Flow

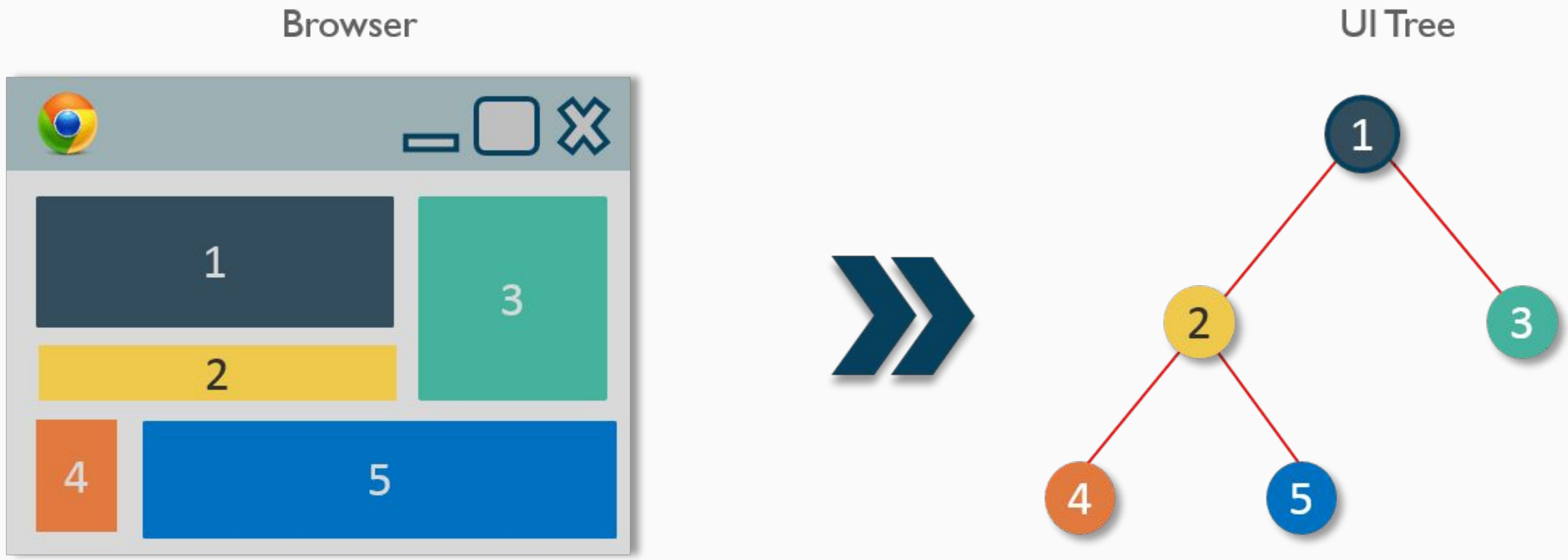
State is passed down through props.



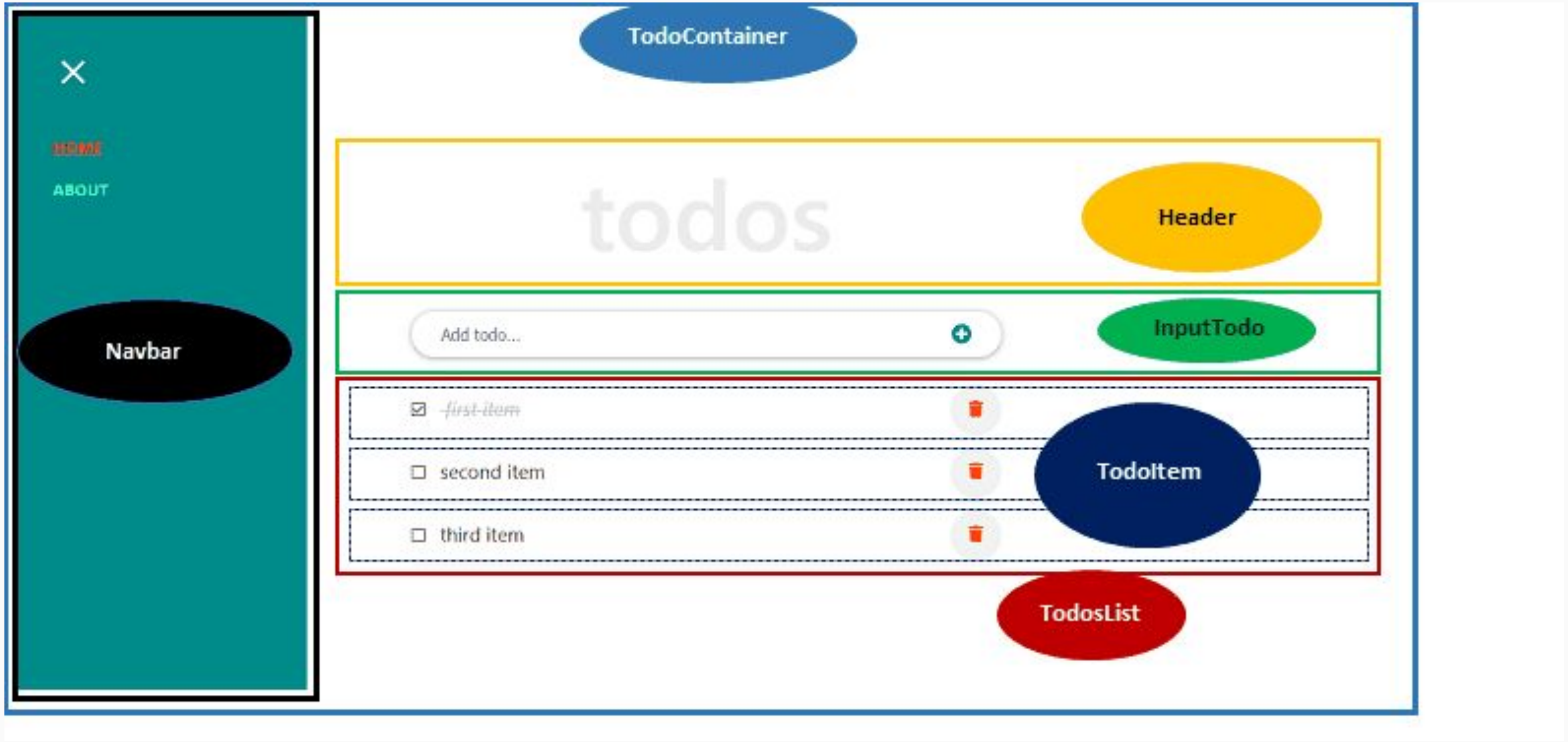
● State change initited

● State change

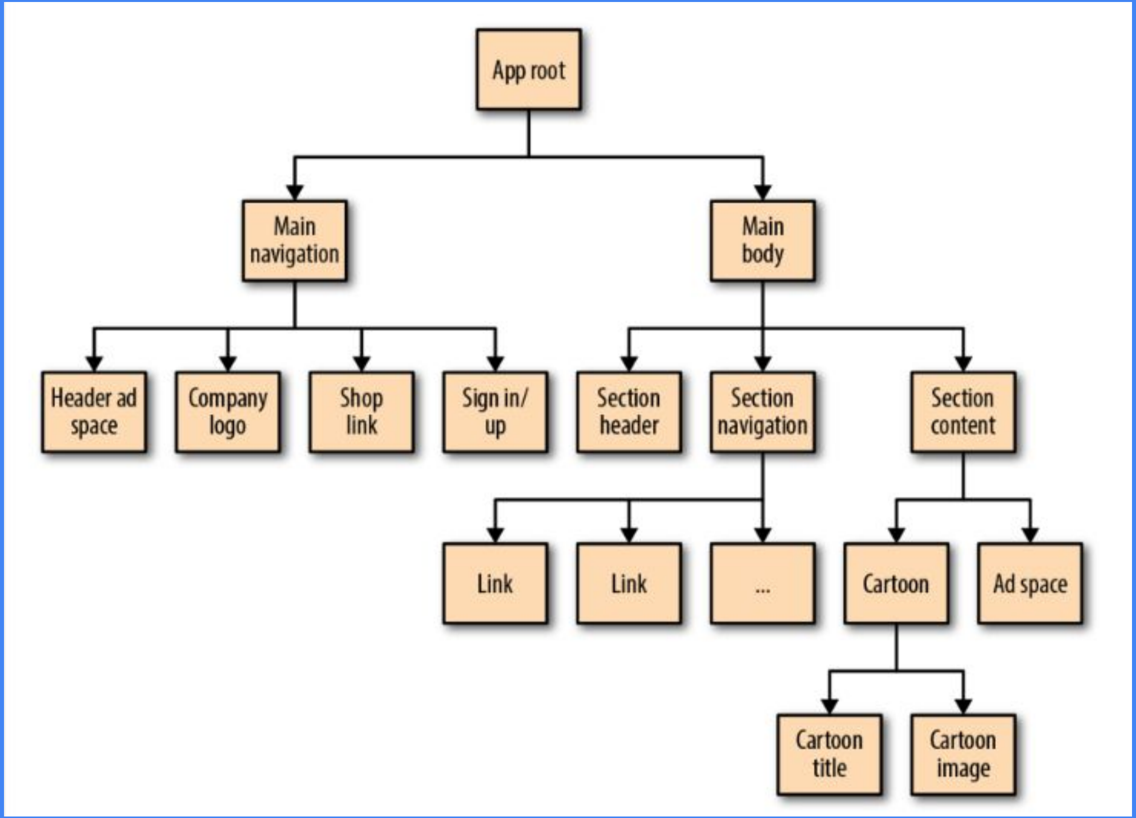
Document Object Model **(DOM)**



React Component Diagramming



Component Hierarchy



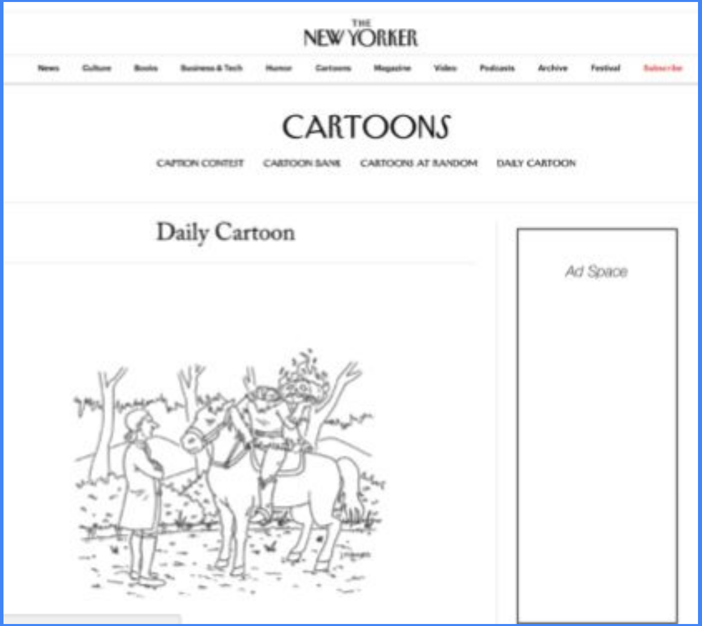
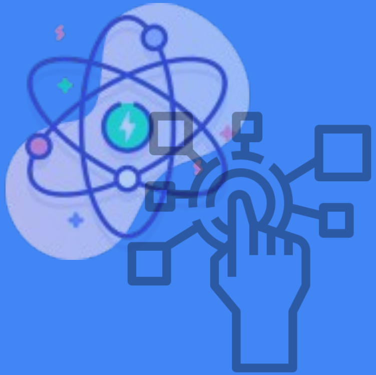
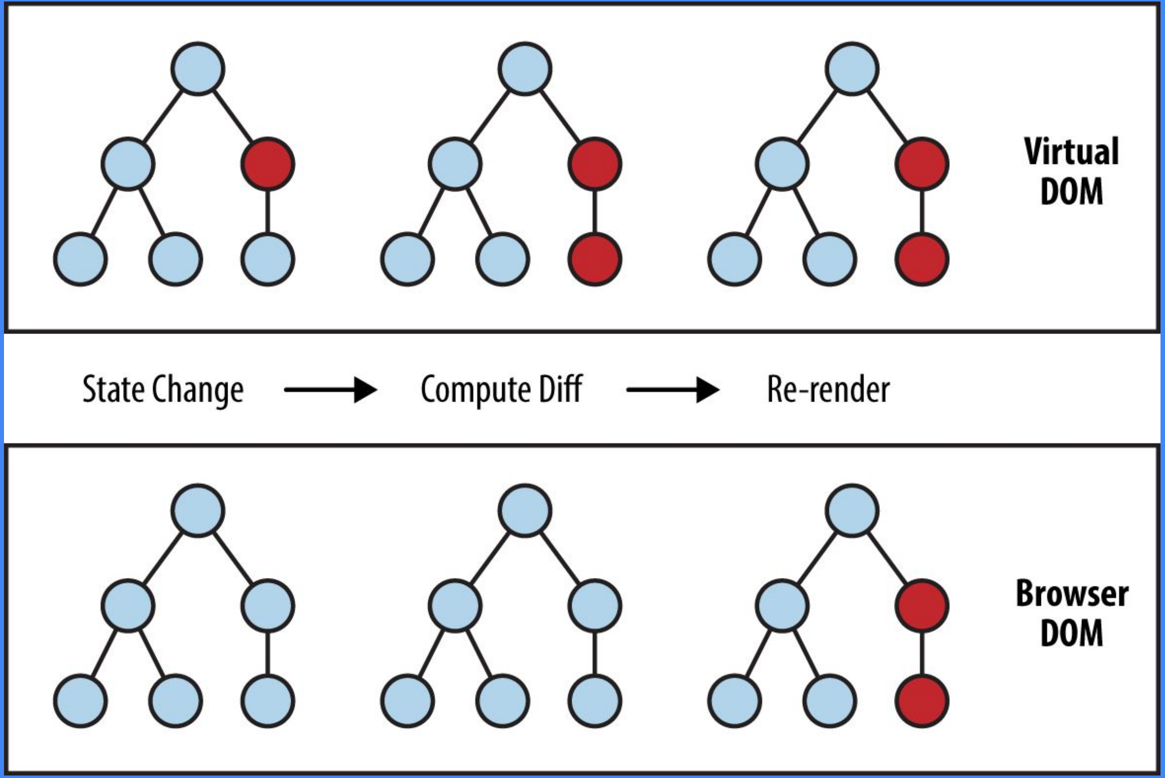


Image Source: **O'Reilly**
https://www.oreilly.com/library/view/what-react-is/9781491996744/ch01.html

Virtual DOM

Efficient re-render
Interactivity

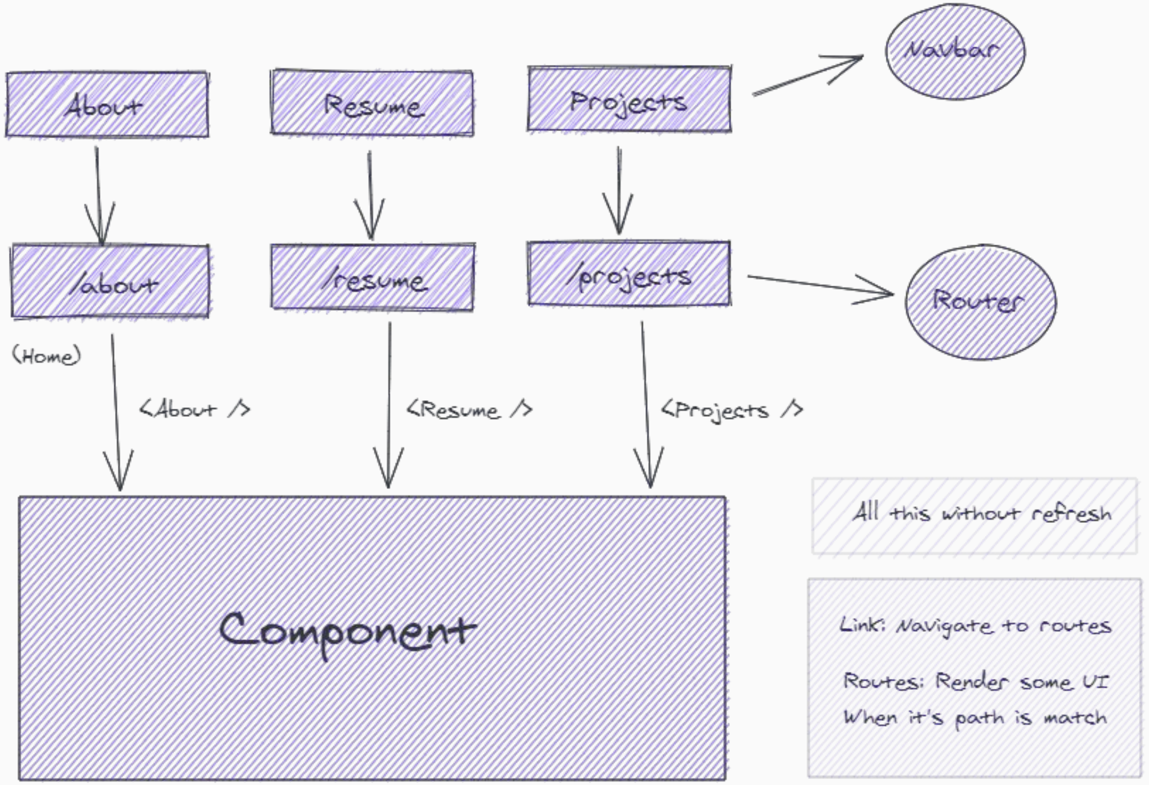




Single Page Application

React Router

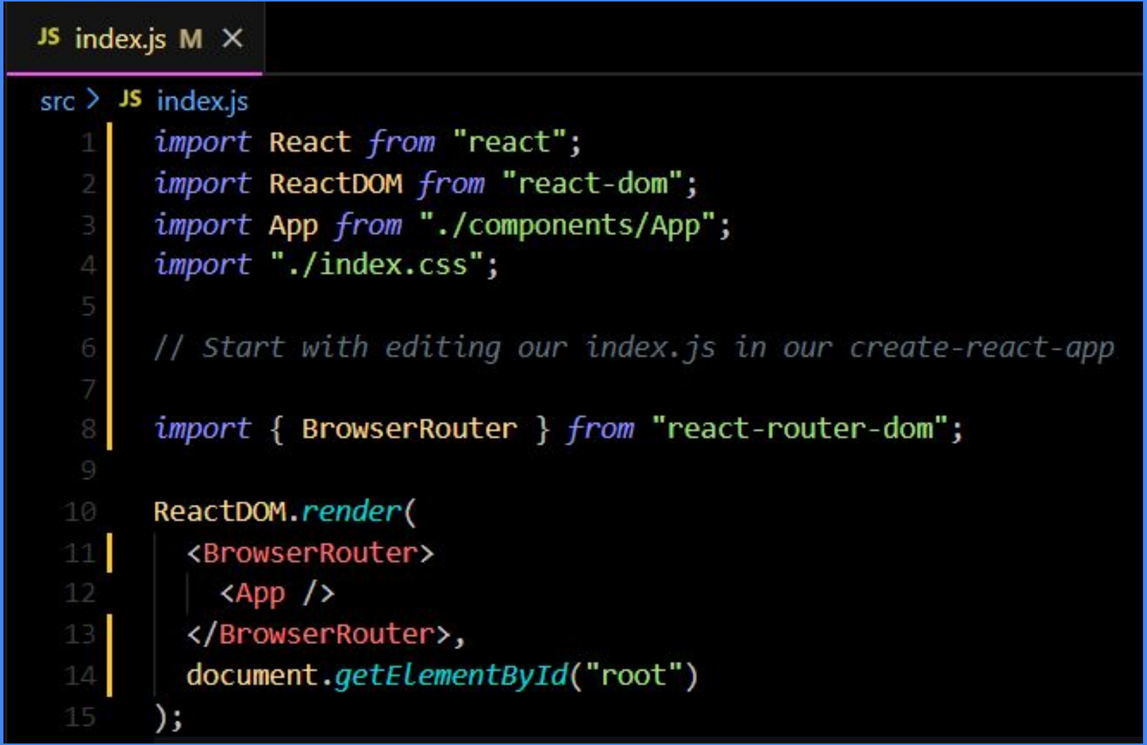
Building a Single Page Application



REACT ROUTER

React Router

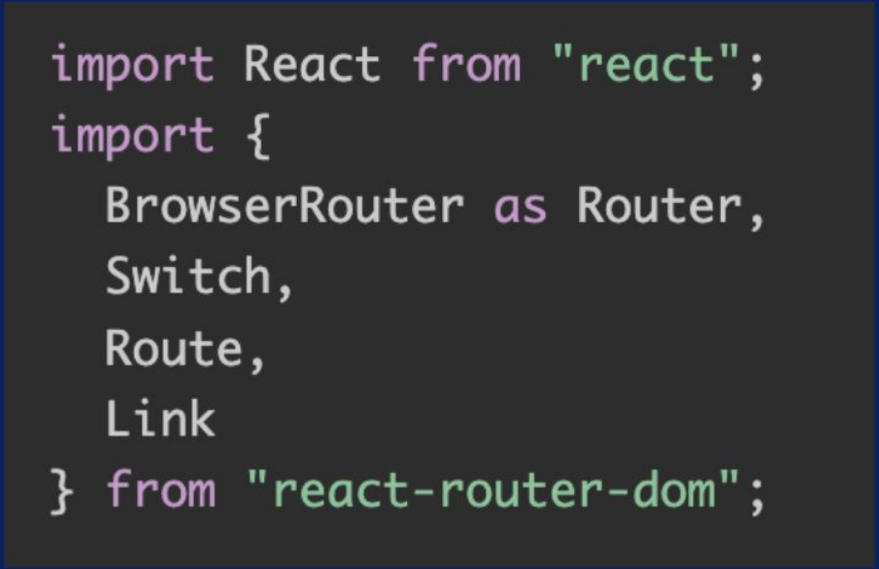
Wrap everything in
<BrowserRouter>



```
import React from "react";
import ReactDOM from "react-dom";
import App from "./components/App";
import "./index.css";

// Start with editing our index.js in our create-react-app

import { BrowserRouter } from "react-router-dom";

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

npm i react-router-dom

React Router

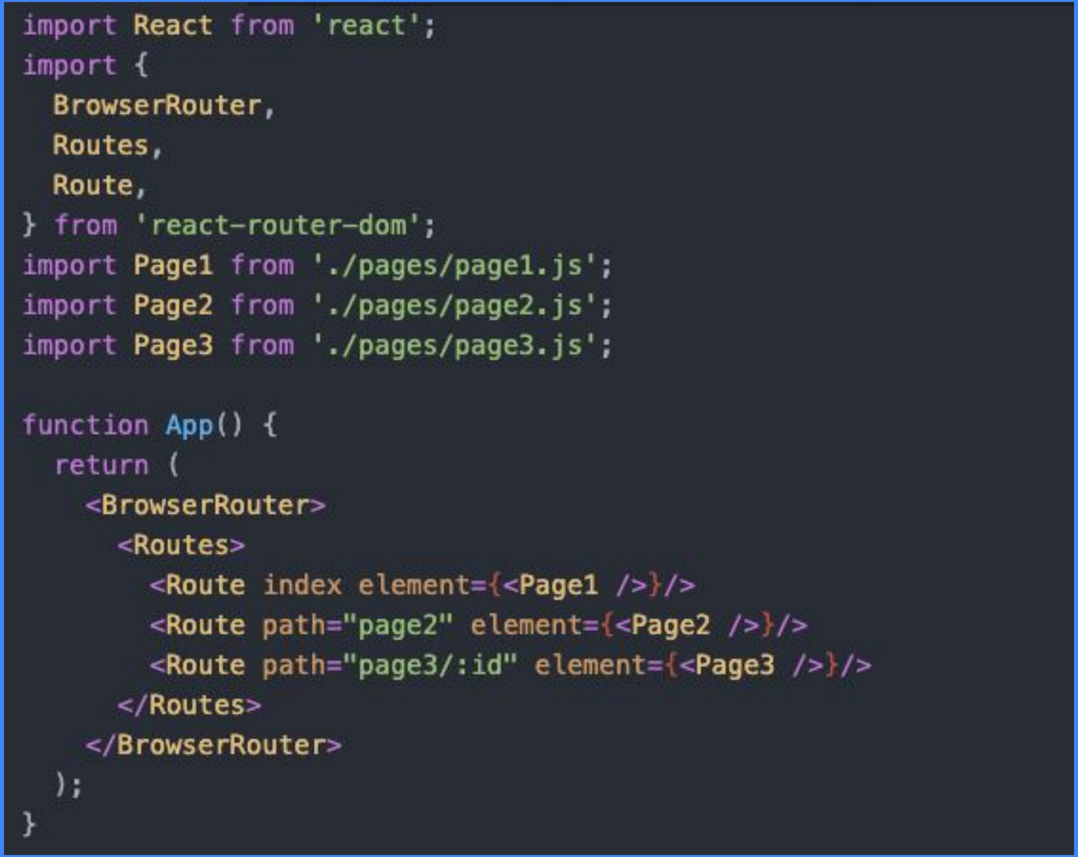


```
import React from "react";
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link
} from "react-router-dom";
```

Import libraries from
react-router-dom
v. 6.3.0

<Routes>

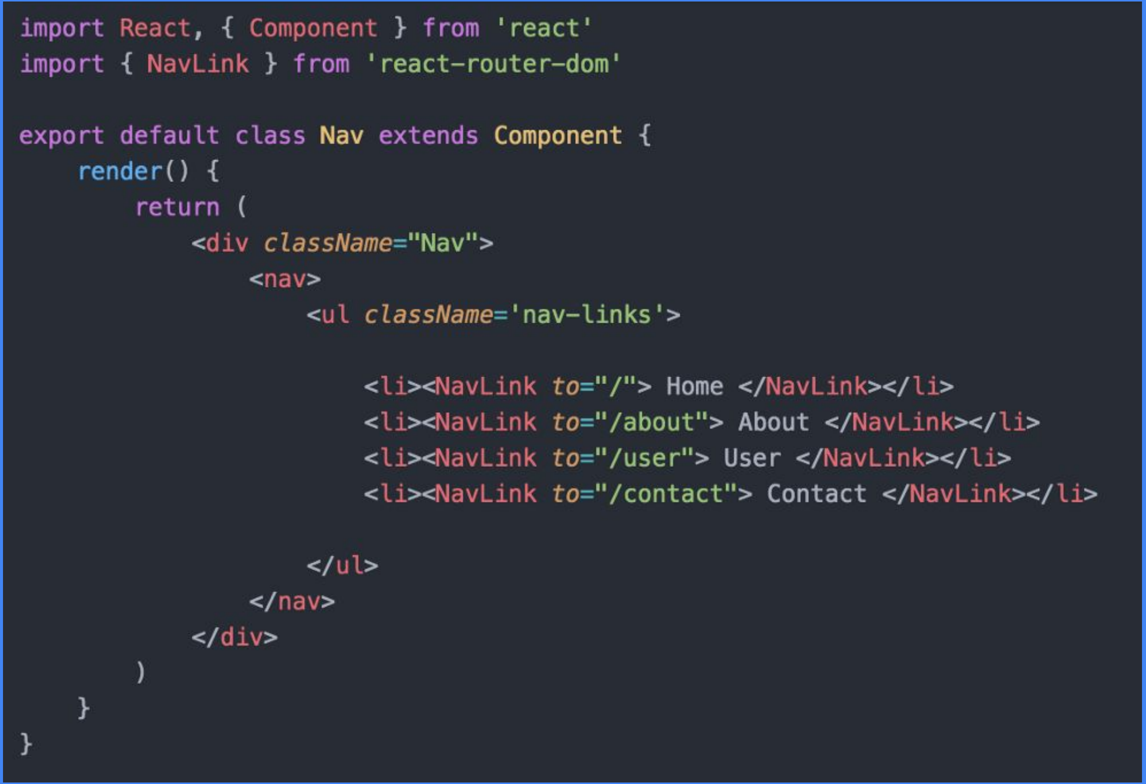
Provide each **<Route>** with a URL path that renders a different component



```
import React from 'react';
import {
  BrowserRouter,
  Routes,
  Route,
} from 'react-router-dom';
import Page1 from './pages/page1.js';
import Page2 from './pages/page2.js';
import Page3 from './pages/page3.js';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Page1 />}/>
        <Route path="page2" element={<Page2 />}/>
        <Route path="page3/:id" element={<Page3 />}/>
      </Routes>
    </BrowserRouter>
  );
}
```

<Link>

Create navigation links that direct to the routes you've specified.

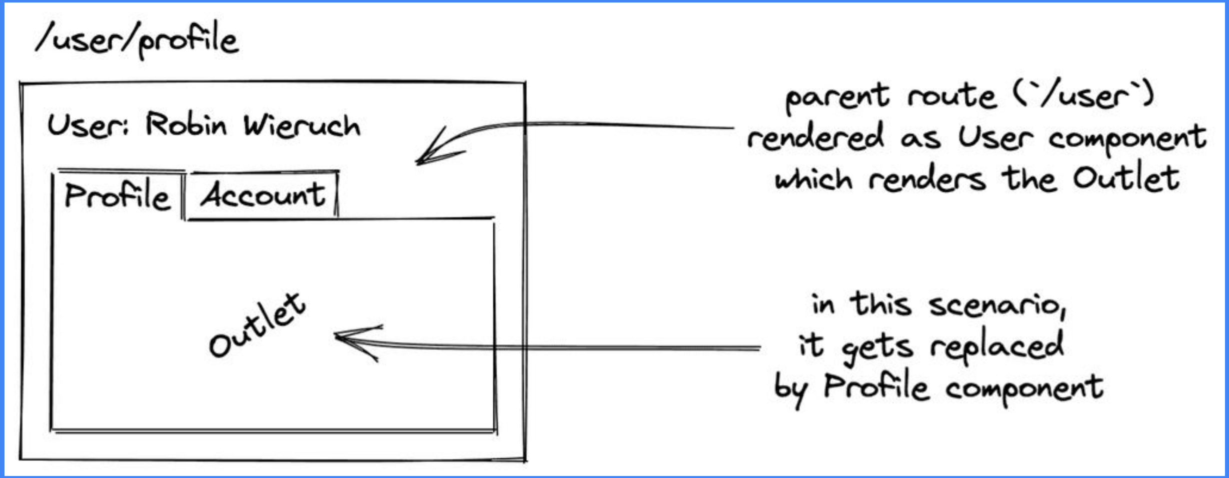


```
import React, { Component } from 'react'
import { NavLink } from 'react-router-dom'

export default class Nav extends Component {
    render() {
        return (
            <div className="Nav">
                <nav>
                    <ul className='nav-links'>

                        <li><NavLink to="/"> Home </NavLink></li>
                        <li><NavLink to="/about"> About </NavLink></li>
                        <li><NavLink to="/user"> User </NavLink></li>
                        <li><NavLink to="/contact"> Contact </NavLink></li>

                    </ul>
                </nav>
            </div>
        )
    }
}
```

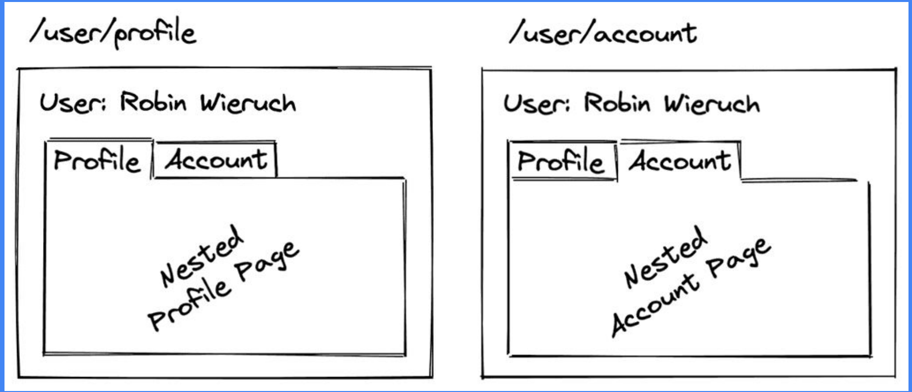
<NavLink> allows for CSS styling.

<Outlet>

Display **nested routes**


/user/profile
User: Robin Wieruch
Profile
Account
Outlet
parent route (`/user`) rendered as User component which renders the Outlet
in this scenario, it gets replaced by Profile component

See Documentation



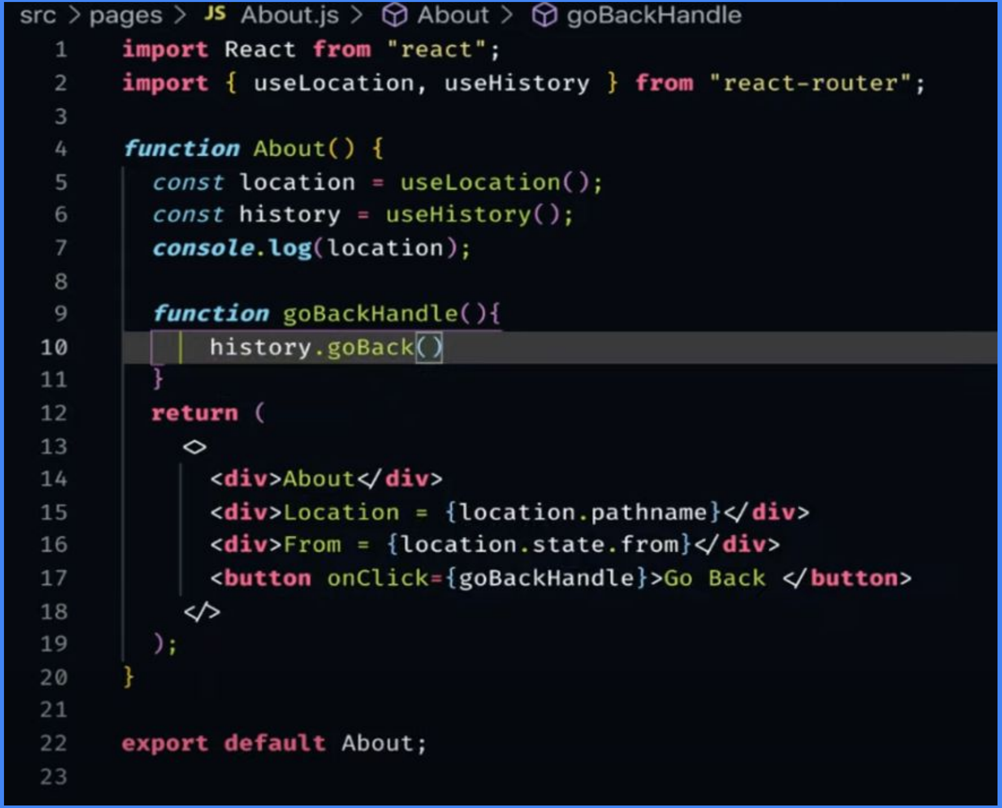
REACT ROUTER

Extra:

React Router with Hooks

useHistory( )
useParams( )
useLocation( )
useRouterMatch( )

Read Docs



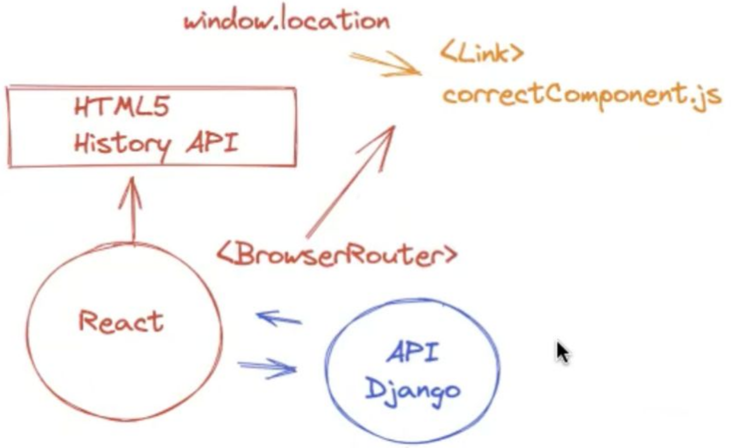
src > pages > JS About.js > About > goBackHandle

```
import React from "react";
import { useLocation, useHistory } from "react-router";

function About() {
  const location = useLocation();
  const history = useHistory();
  console.log(location);

  function goBackHandle(){
    history.goBack()
  }
  return (
    <>
      <div>About</div>
      <div>Location = {location.pathname}</div>
      <div>From = {location.state.from}</div>
      <button onClick={goBackHandle}>Go Back </button>
    </>
  );
}

export default About;
```

Routing!

SPA's - Single Page Applications(React.js)

Client Side Rendering!



<Outlet> - needed
to render children
routes

```
<Routes>
  <Route path=/home>
</Routes>
```

Server Side Rendering

Jinja templates
render(<h1>Hello World</h1>)

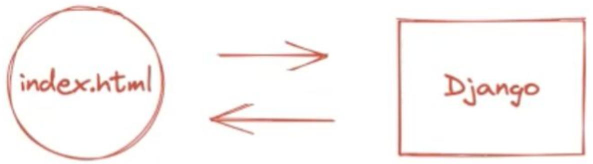


Image: Paul Nnaoji

CodeSandbox for OG Facebook

Github Repo for React Router

Let's dive in!

React Router Tutorial v.6.4.4

Day 3. Recap
What You've Learned Today

1. Review
    a. Class & Functional Components
    b. **.setState( )** & Component Lifecycle
    c. Event-Handling & Fetching asynchronously

Tomorrow: Trivia Game with Leaderboards & Prizes

2. React Router
    a. <BrowserRouter> & <Route(s)>
    b. <Link>, <NavLink> & nested links: <Outlet/>
    c. Reading through a codebase

Github & CodeSandbox ⟶ **More on Git tomorrow**

3. Q&A: ⭐
    This afternoon with
    Warren Longmire