Web Dev 1

CS571: Building User Interfaces

Cole Nelson

Before Lecture

- Install Node 20 and NPM 10
- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` folder.

Disclaimer

As with JS, this is not a comprehensive introduction to React, so be sure to check out the React Docs.

Avoid legacy.reactjs.org

Learning Objectives

1. Understand the history and motivation for React
2. Be able to set up and build a basic React project
3. Differentiate between traditional web programming and React, including the use of JSX
4. Deconstruct websites into their basic components
5. Use the `useState` and `useEffect` hooks

What is React?

Definition: Also called ReactJS, React is a JS library for building user interfaces.

- Developed by Facebook, dating back to 2010.
- Started as an internal development tool, then open-sourced in 2013.

More on the history of React

Why should we use React?

Among many reasons, it **abstracts away interactions with the DOM** and enables a more **declarative** way of constructing the user interface via **components**.

Similar Alternatives: Angular, Vue, Svelte

Traditional Component Construction

`getElement`, `createElement`, and `appendChild`

```
// Set the instructions
const instructionsHTML = document.getElementById("instructions");
for(let step of data.recipe) {
    const node = document.createElement("li");
    node.innerText = step;
    instructionsHTML.appendChild(node)
}
```

This gets quite long with many children! It also focuses on the **how** rather than the **what**.

Traditional Component Construction

`innerHTML` focuses on the **what**, *but*...

```
let html = `<div>`;
html += `<h2>${stud.name.first} ${stud.name.last}</h2>`;
html += `<p><strong>${stud.major}</strong></p>`;
html += `</div>`
document.getElementById('students').innerHTML = studentHtml;
```

...it is dangerous with untrusted input!

XSS

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

Remember: You supplied the data in HW0!

OWASP Definition

The Virtual DOM

Abstracting away direct changes to the DOM.

The Virtual DOM

Rather than updating the `document` ourselves, we describe to React **what** we want to display, and then React does the updating for us!

```
function Person(props) {
  return <div>
    <p>Hi! My name is {props.name} and I am {props.age} years old.</p>
  </div>
}
```

As `name` and `age` change, React will "react" and re-render the component with the new values.

Reconciliation

This is done through periodic *reconciliation*.

Reconciliation is the process of *diffing* and syncing the virtual and real DOM to render changes for the user.

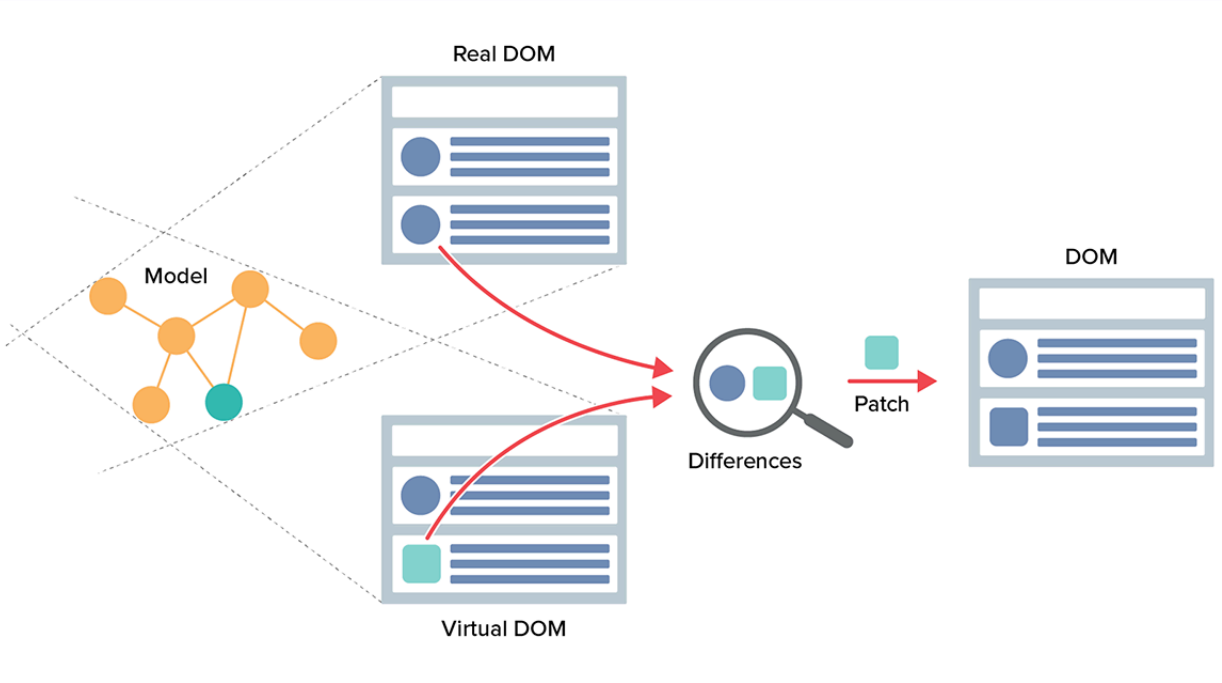


Image Source

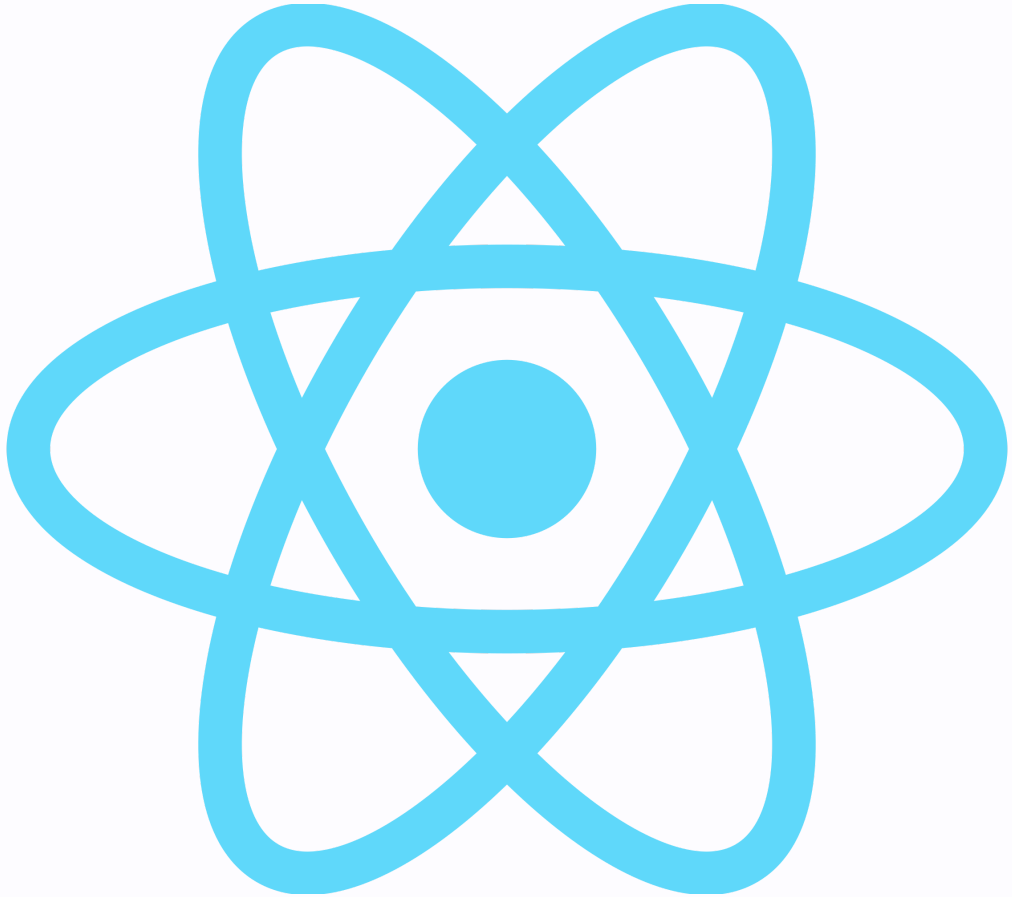
React DOM

`react` itself is platform-agnostic! It is simply an efficient tree reconciler

`react-dom` does the `document` transcription

React

by Meta



React in 100 Seconds

Getting Started

For an existing React project, you simply need to...

```
npm install
npm run dev
```

Clone, install, and start the starter code. You can install and start the completed code later.

This has react-bootstrap installed.

You may need to run this with `sudo`. Do *not* save in OneDrive.

What did I just run?

`npm install` downloads the necessary `dependencies` from npmjs.com specified in your `package.json` to a folder called `node_modules`

`npm run dev` starts a local webserver. We don't open `index.html` anymore -- we go to `localhost:5173`

React 101

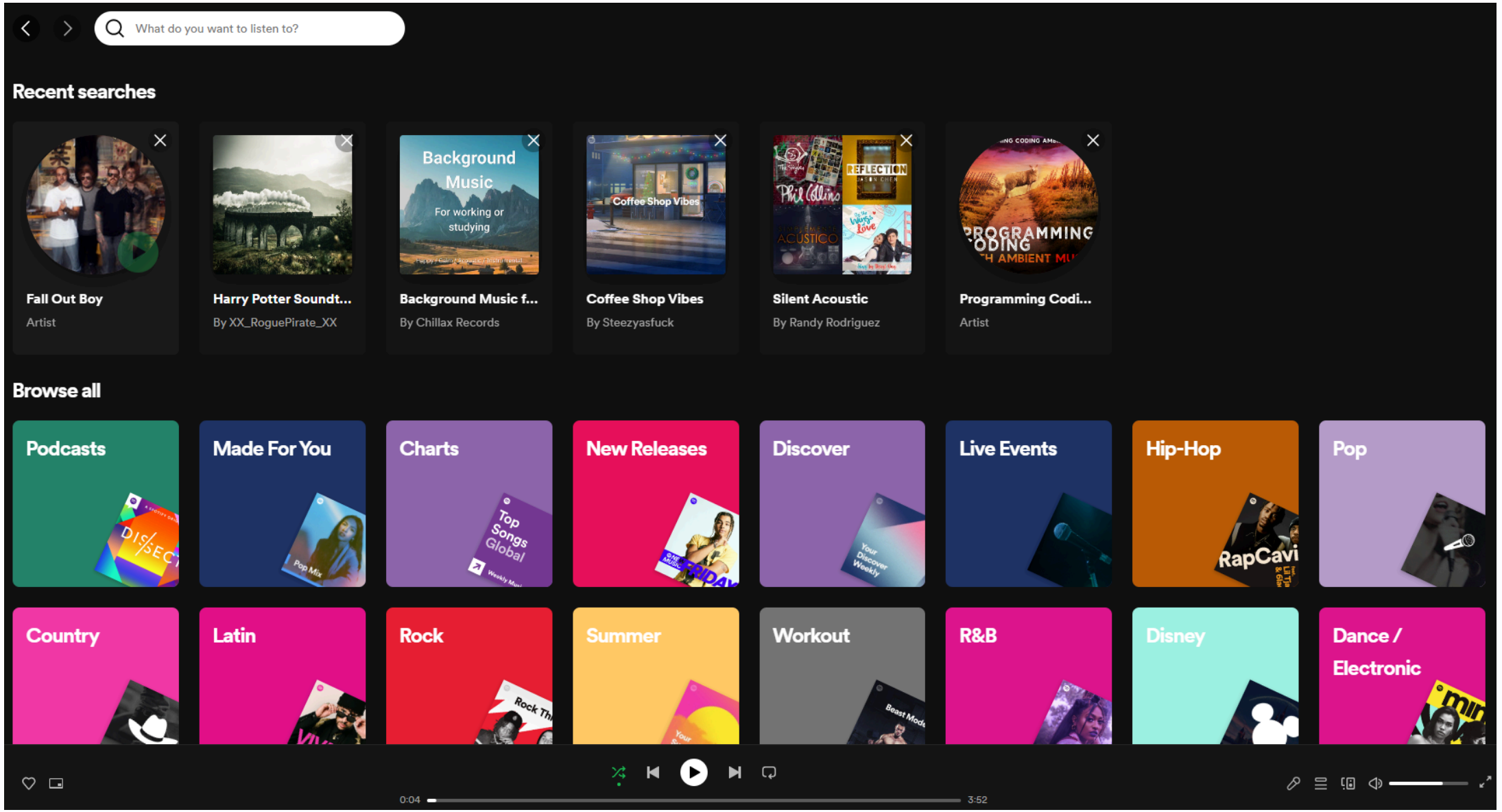
Thinking in a "reactive" way.

React Component Basics

In React, every *thing* is a component. What is a *thing*?

- Similar question: *what defines a class in Java?*
- Some re-usable piece of the interface.
- May have many children, but only one parent.

Identify components in the next slide...


What do you want to listen to?
Recent searches
Fall Out Boy
Artist
Harry Potter Soundt...
By XX_RoguePirate_XX
Background Music
For working or studying
Background Music f...
By Chillax Records
Coffee Shop Vibes
Coffee Shop Vibes
By Steezyasfuck
Silent Acoustic
By Randy Rodriguez
Programming Codi...
Artist
Browse all
Podcasts
Made For You
Charts
New Releases
Discover
Live Events
Hip-Hop
Pop
Country
Latin
Rock
Summer
Workout
R&B
Disney
Dance / Electronic
0:04
3:52

Spotify Deconstruction

- Spotify
  - HeaderBar
  - MainContent
    - RecentSearches
      - Card[]
    - BrowseAll
      - Card[]
  - BottomBar

Your answers may vary! That's okay!

Note: Class vs Functional Components

A class component looks like this...

```
class Welcome extends React.Component {
  render() {
    return <div>
      <h1>Hello World!</h1>
      <p>This is my website.</p>
    </div>
  }
}
```

Functional components were introduced in React 16.8.

We will not cover class components in this course.

React Component Basics

A functional component looks like this...

```
function Welcome() {
  return <div>
    <h1>Hello World!</h1>
    <p>This is my website.</p>
  </div>
}
```

It returns a *single* element of JSX. If you have more than one element, you can return a div `<div></div>` or fragment `<></>` .

React Component Basics

JSX is a syntax that combines HTML, CSS, and JSX.

```
function Welcome(props) {
  return <div>
    <h1>Hello World!</h1>
    <p>{props.msg}</p>
  </div>
}
```

`{}` interpolates any JS *expression* and *safely* inserts it into the DOM. JSX is transpiled into HTML, CSS, and JS at runtime.

React Component Basics

For example, you can specify styling by interpolating a JavaScript object into your JSX.

```
function Welcome(props) {
  return <div style={{border: "dotted", padding: "1rem"}}>
    <h1>Hello World!</h1>
    <p>{props.msg}</p>
  </div>
}
```

Side Note: Specify a CSS class with `className` rather than `class`

React Component Basics

```
function Person() {
  return <div style={{border: "dotted", padding: "1rem"}}>
    <h2>Cole Nelson</h2>
    <p>26 years old</p>
    <p>Works as a(n) Instructor</p>
  </div>
}
```

This is *good*, but...

StackBlitz

React Component Basics

```
function Person(props) {
  return <div style={{border: "dotted", padding: "1rem"}}>
    <h2>{props.name}</h2>
    <p>{props.age} years old</p>
    <p>Works as a(n) {props.role}</p>
  </div>
}
```

... this is even better!

StackBlitz

React Component Basics

Why? It's re-usable!

```
function App() {
  return <div>
    <h1>People</h1>
    <Person name={"Cole Nelson"} age={26} role={"Instructor"}/>
    <Person name={"Zach Potter"} age={24} role={"Teaching Assistant"}/>
  </div>
}
```

Notice: `props` are passed down as key-value pairs.

StackBlitz

Your Turn!

Construct a `Recipe` to display a...

- `name`
- `author`
- `keywords` (as a single string, e.g. "rich comforting")

Use any recipe!

Syntactic Sugar 🍬

Use the `...` spread operator to pass down many `props` all at once.

```
const TA = {name: "Zach Potter", age: 24}

function YellowPages() {
  return <div>
    <Person {...TA}></Person>
  </div>
}
```

Syntactic Sugar 🍬

If the component does not take any children, you can even write it shorthand!

```
const TA = {name: "Zach Potter", age: 24}

function YellowPages() {
  return <div>
    <Person {...TA} />
  </div>
}
```

Syntactic Sugar 🍬

A component receiving `props` can explicitly deconstruct them before use.

```
function Person(props) {
  // Access via props object, e.g. `props.name`, `props.age`
  return ...
}
```

```
function Person({name, age}) {
  // Access directly, e.g. `name`, `age`
  return ...
}
```

Design Sugar 🍭

React-Bootstrap is Bootstrap for React! You can import existing components such as `Card`, `Button`, and more! Beware of your imports!

```
import { Card, Button } from 'react-bootstrap'

export default function Person({name, age}) {
  return <Card>
    <p>Hi! My name is {name} and I am {age} years old.</p>
    <Button onClick={() => alert('hello!')}>Say hello!</Button>
  </Card>
}
```

Imports and Exports

React-Bootstrap exports many components, we import them with curly braces, e.g.

```
import { Card, Button } from 'react-bootstrap'
```

We typically `export` one component by `default` (see last slide) and `import` directly as...

```
import Person from './person' // Use relative pathing for local files
```

Callback Handlers

In React, event listeners take *callback functions* as a parameter -- not function calls!

Correct!

```
<Button onClick={() => alert('hello!')}>Say hello!</Button>
```

Incorrect!

```
<Button onClick={alert('hello!')}>Say hello!</Button>
```

Reactive State Basics

We often use event listeners to *mutate state*.

Whereas *props* is given by the parent component, *state* is managed internally.

```
const [name, setName] = useState("James");
```

We get back a read-only variable and a mutator function; **do not mutate the variable directly!** Use the mutator function.

Reactive State Basics

State variables declared with `useState` are **reactive**, meaning that a change to its state (via its mutator function) will trigger a re-render!

```
setName("Jim") // Will trigger a re-render!
```

This re-render will happen in the *near future*, causing the function to be ran again. **Initial state is only set on component mount.**

Reactive State Basics

```
function Person(props) {
  const [name, setName] = useState();

  function giveName() {
    const allNames = ["Amy", "Ben", "Eli", "Ian", "Leo", "Mia"];
    const randI = Math.floor(Math.random() * allNames.length);
    setName(allNames[randI]);
  }

  return <div>
    <h1>My name is {name}.</h1>
    <button onClick={giveName}>Give New Name</button>
  </div>
}
```

StackBlitz

Detour: Conditional Rendering

```
<h1>My name is {name}.</h1>
```

The initial render isn't ideal... How can we handle this?

```
{name ? <h1>My name is {name}.</h1> : <p>I don't have a name yet...</p>}
```

You control what is returned; **do a conditional render!**
Remember that `{}` will interpolate any JS expression.

Reactive State Basics

```
function Person(props) {
  const [name, setName] = useState();

  function giveName() {
    const allNames = ["Amy", "Ben", "Eli", "Ian", "Leo", "Mia"];
    const randI = Math.floor(Math.random() * allNames.length);
    setName(allNames[randI]);
  }

  return <div>
    {name ? <h1>My name is {name}.</h1> : <p>I don't have a name yet...</p>}
    <button onClick={giveName}>Give New Name</button>
  </div>
}
```

Reactive State Basics

Finally, add a `console.log('My name is', name)` immediately after calling `setName` and click the button again... What do you notice?

```
My name is undefined
```

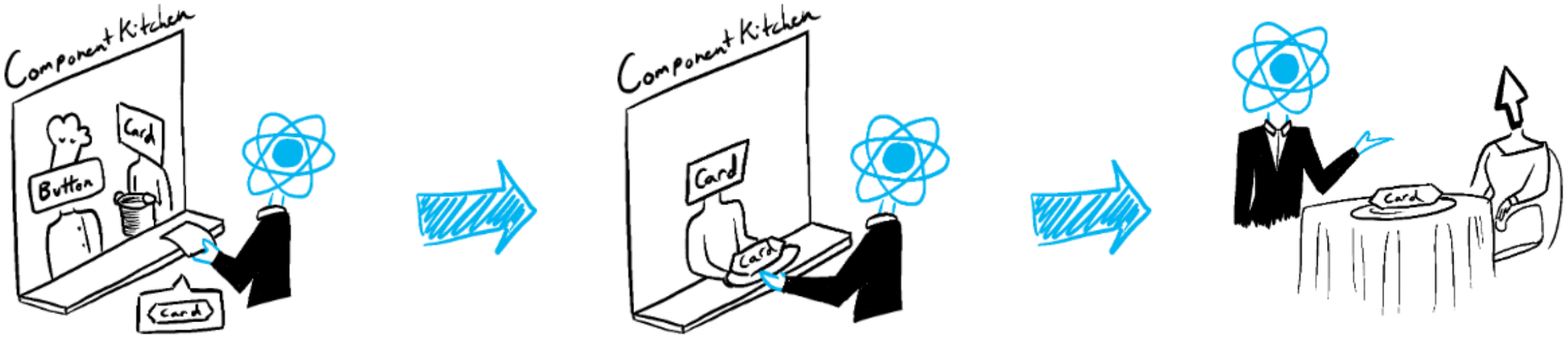
It is its previous value, why do you think this is?

StackBlitz

Reactive State Basics

Setting state happens *asynchronously*!

Just because the food is cooked doesn't mean the server has already brought it to the table!



Trigger Render Commit

Illustration by Rachel Lee Narbors

Derived Reactive State

A *derived* variable does not necessarily need to be declared with `useState`.

```
const [name, setName] = useState("James");
const derivedInitial = name[0];
```

`derivedInitial` will be calculated each render, and changes to `name` are already tracked via `setName`, so another `useState` would be redundant!

StackBlitz

Derived Reactive State

In fact, can you tell what's wrong with this snippet?

```
function Person() {
  const [name, setName] = useState("James");
  const [initial, setInitial] = useState();
  setInitial(name[0])
  // ...
```

Derived Reactive State

We get an infinite loop!

```
function Person() {
  const [name, setName] = useState("James");
  const [initial, setInitial] = useState();
  setInitial(name[0]); // <--- oops!
  // ...
```

The function runs. `setInitial` changes the state. This triggers a re-render, running the function. ↲

Takeaway: Reactive State

Reactive state is a powerful way to write declarative components, but we should be careful!

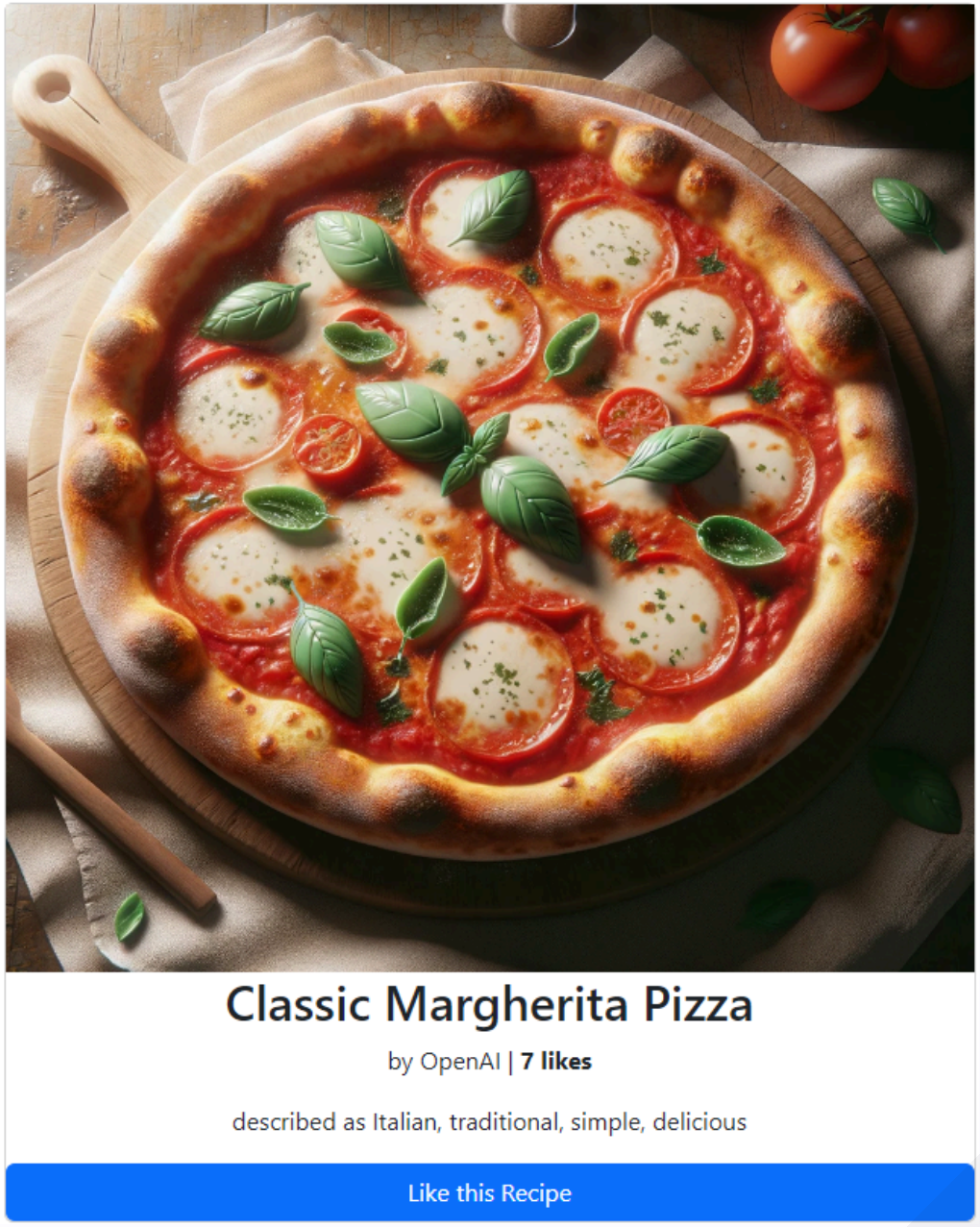
`useState` should be used for UI elements that change, and the state should only be updated conditionally.

All changes to state (and props) will trigger a re-render, although this happens *asynchronously*.

Your Turn!

Display a `Recipe` for *each* recipe as well as a like button that increases the number of likes for *that* recipe.

Optional: Make use of Bootstrap components like `Card` and `Button`



2 Ways to Set State

- `setLikes(0)` overwrites the existing value
- `setLikes(old => old + 1)` updates existing value

The second notation (using a *callback function*) is never a wrong answer, it can just be verbose. What you `return` is what the state will be set as.

`fetch` 'ing data with `useEffect`

`useEffect` allows us to run a function at a particular stage of the update cycle.

```
useEffect(() => {

  // This code will ONLY be run on component mount.

}, []); // <-- Don't forget the []!
```

We will be covering some other ways to use `useEffect` next lecture!

`fetch` 'ing data with `useEffect`

```
fetch("https://cs571api.cs.wisc.edu/rest/su24/ice/pizza", {
    headers: {
        "X-CS571-ID": CS571.getBadgerId()
    }
})
.then(res => res.json())
.then(data => {
    console.log("Received pizza!", data);
    setPizza(data);
})
```

What would happen if this was not within a `useEffect` that only runs on mount?

```
function PizzaDisplayer() {
  const [pizza, setPizza] = useState();
  fetch("https://cs571api.cs.wisc.edu/rest/su24/ice/pizza", {
      headers: {
          "X-CS571-ID": CS571.getBadgerId()
      }
  })
  .then(res => res.json())
  .then(data => {
      setPizza(data);
  });

  return <p>{JSON.stringify(pizza)}</p>
}
```

The function runs. The `fetch` changes the state. This triggers a re-render, running the function. ↲

```
function PizzaDisplayer() {
  const [pizza, setPizza] = useState();
  useEffect(() => {
    fetch("https://cs571api.cs.wisc.edu/rest/su24/ice/pizza", {
        headers: {
            "X-CS571-ID": CS571.getBadgerId()
        }
    })
    .then(res => res.json())
    .then(data => {
        setPizza(data);
    });
  }, [])

  return <p>{JSON.stringify(pizza)}</p>
}
```

Much better! This will *only* run the `fetch` on mount.

Rendering Basics

A component will re-render shortly after...

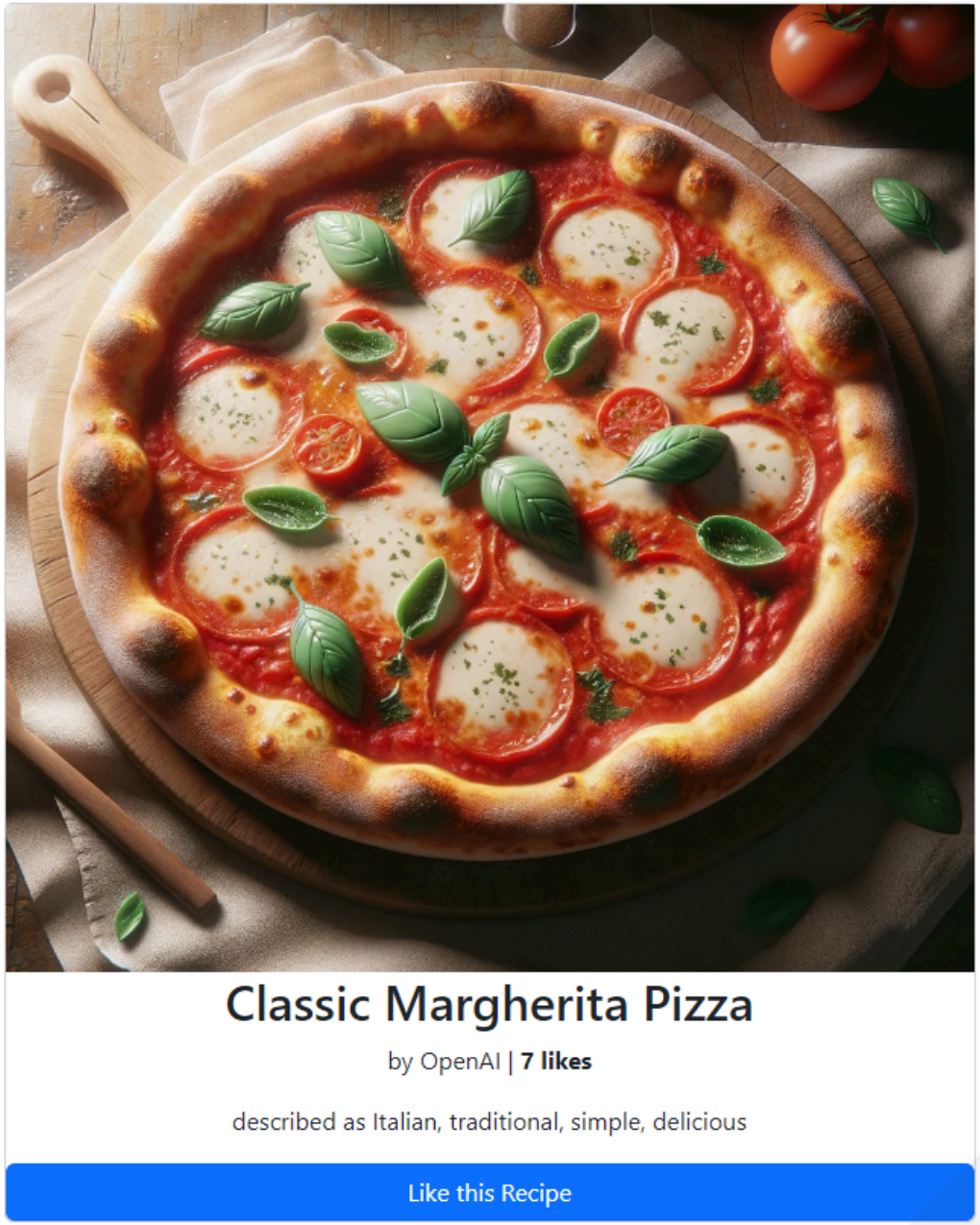
1. Its `props` changes
2. Its state changes

We will expand on this lifecycle next lecture!

Your Turn!

`fetch` the data for the recipes instead of using the hardcoded data!

Hint: You'll need to create 3 state variables, we'll explore a better way in the next lecture!



It's not working!

A laundry list of common problems we experience...

Blank Screen

Use your Developer Tools! (F12)

Check both the "Console" and "Network" tabs.

HTTP 401

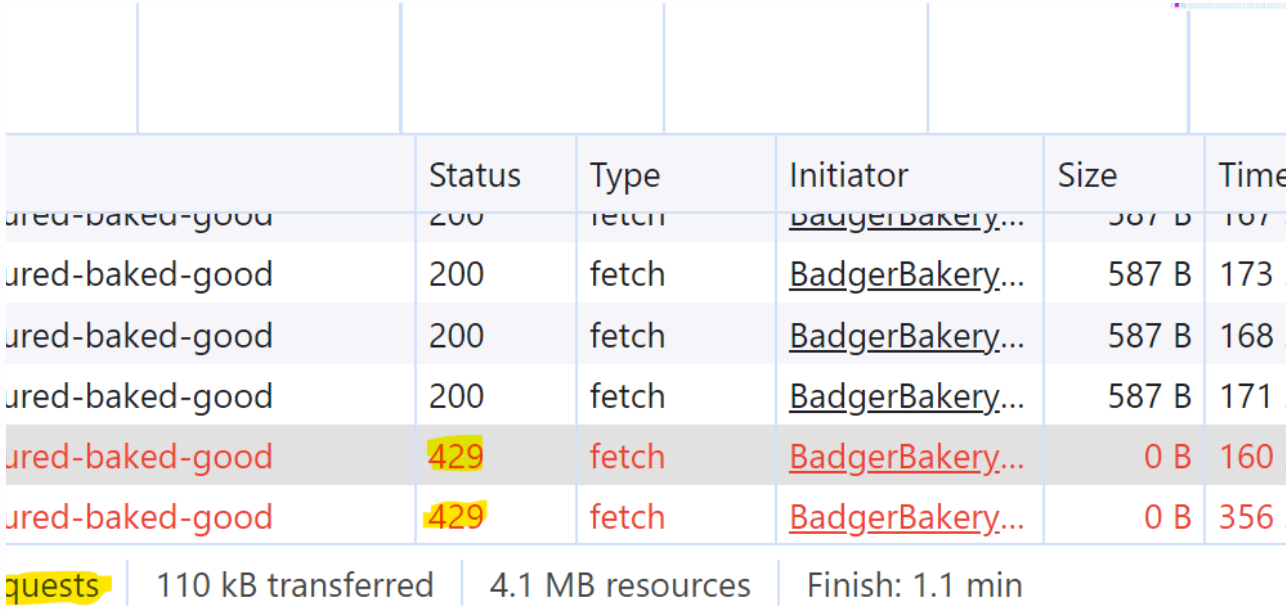
Unauthorized... Are you passing a `X-CS571-ID` header?

HTTP 429

Too many requests...

```
useEffect(() => {
  // your code here!
}, []) // <--- don't forget me!
```

... did you forget the empty dep. array?



X is not defined

Did you import it? e.g.

Common React Imports

```
import { useState, useEffect } from "react";
```

Common Bootstrap Imports

```
import { Button, Card } from "react-bootstrap";
```

```
▶ Uncaught ReferenceError: Button is not defined
    at BakedGood (BakedGood.jsx:20:14)
```

```
▶ Uncaught ReferenceError: Card is not defined
    at BakedGood (BakedGood.jsx:15:13)
```

Empty State

Setting state is asynchronous!

```
const [name, setName] = useState("James");
console.log(name);
setName("Jim");
console.log(name); // still James!
```

We will talk about more ways to `useEffect` next time.

Questions?