React 2

CS571: Building User Interfaces

Cole Nelson & Yuhang Zhao

Before Lecture

- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` and `solution` folders.

React Recap

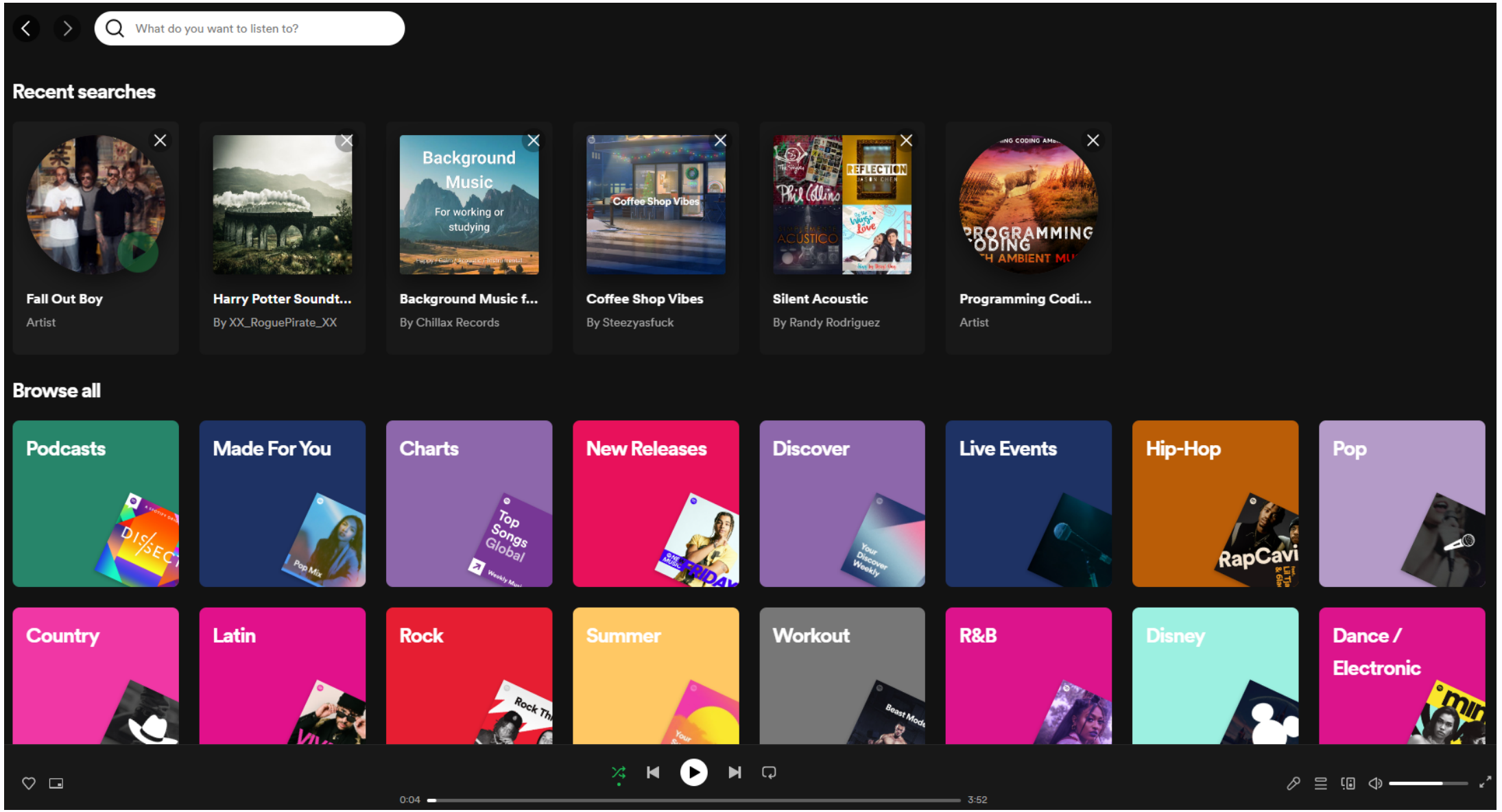
React Essentials

Every "thing" is a component.

Every component is a function, inheriting `props` and maintaining an internal `state`.

What defines a component?

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


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

Example of a React Component

This React component displays Hello World on the webpage using JSX.

```
function Welcome() {
  return <h1>Hello World!</h1>;
}
```

ESBuild transpiles JSX into HTML, CSS, and JS.

StackBlitz

Two Hooks

`useState` store some state.

```
const [name, setName] = useState('Charles')
```

`useEffect` conditionally run logic, e.g. fetch on component load, alert on name change

```
useEffect(() => {
  alert("Your name has been changed!")
}, [name])
```

What will we learn today?

- How do we work with arrays of data in React?
  - How do we `map` out components?
  - How does the `key` property work?
- How do we do responsive design in React?
- How can we break up large sums of data using pagination?
- How can we use controlled `input` components?

Brain Teaser

Two ways to set the state...

`setNum(4)` - overwriting the value

`setNum(n => n + 1)` - update the existing value

Why do we use this syntax?

BAD Example

```
export default function Counter() {
  const [number, setNumber] = useState(0);
  return <div>
    <h1>{number}</h1>
    <button onClick={() => {
      setNumber(number + 1);
      setNumber(number + 1);
      setNumber(number + 1);
    }}>+3</button>
  </div>
}
```

Why do we use this syntax?

Good Example

```
export default function Counter() {
  const [number, setNumber] = useState(0);
  return <div>
    <h1>{number}</h1>
    <button onClick={() => {
      setNumber(n => n + 1);
      setNumber(n => n + 1);
      setNumber(n => n + 1);
    }}>+3</button>
  </div>
}
```

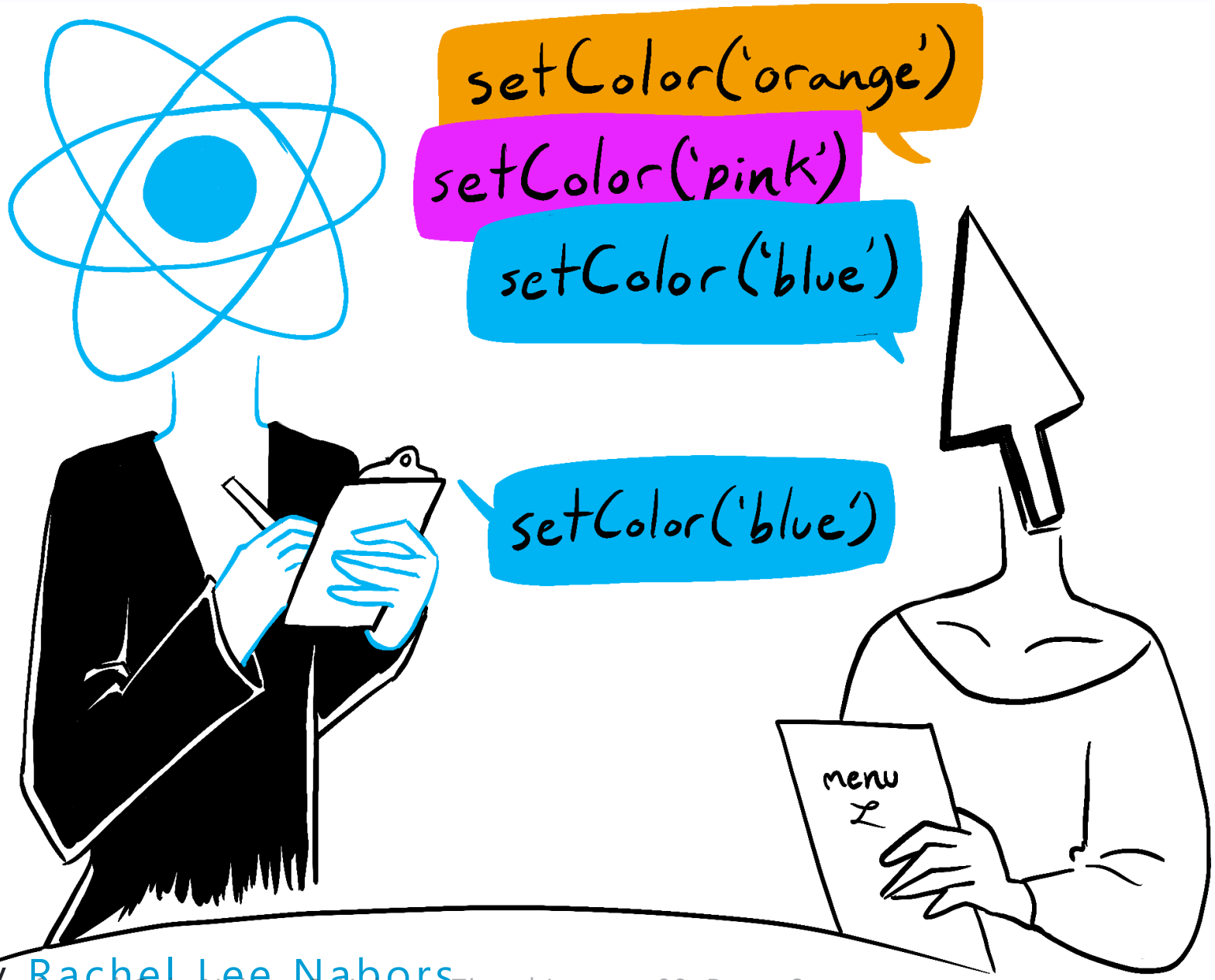


Illustration by Rachel Lee Nabors

Your turn!

Use Postman to explore the hurricane data from...

```
https://cs571.org/api/f23/weekly/week05
```

... how will we add this to a state variable?

Fetching Data

NEVER assign/push directly to a state variable...

```
function AllHurricanes() {
  const [hurricanes, setHurricanes] = useState([]);
  useEffect(() => {
    fetch("https://cs571.org/api/f23/weekly/week05")
      .then(res => res.json())
      .then(data => {
        for (let hurr of data) {
          hurricanes.push(hurr) // !!! VERY BAD !!!
        }
      })
  }, [])
  // ...
}
```

Fetching Data

When changing the array, use the callback syntax...

```
function AllHurricanes() {
  const [hurricanes, setHurricanes] = useState([]);
  useEffect(() => {
    fetch("https://cs571.org/api/f23/weekly/week05")
      .then(res => res.json())
      .then(data => {
        for (let hurr of data) {
          setHurricanes([...hurricanes, hurr]) // !!! BAD !!!
        }
      })
  }, [])
  // ...
}
```

Fetching Data

... like this!

```
function AllHurricanes() {
  const [hurricanes, setHurricanes] = useState([]);
  useEffect(() => {
    fetch("https://cs571.org/api/f23/weekly/week05")
      .then(res => res.json())
      .then(data => {
        for (let hurr of data) {
          setHurricanes(oldHurrs => [...oldHurrs, hurr]) // Better :)
        }
      })
  }, [])
  // ...
}
```

Note on Hot Reloading

React (Vite) will keep your old state when hot reloading. The prior solution will result in duplicates upon saving your solution.

Fetching Data

Best. **Why?** We are *overwriting* the value. No need to worry about duplicates.

```
function AllHurricanes() {
  const [hurricanes, setHurricanes] = useState([]);
  useEffect(() => {
    fetch("https://cs571.org/api/f23/weekly/week05")
      .then(res => res.json())
      .then(data => {
        setHurricanes(data) // Good :)
      })
  }, [])
  // ...
}
```

Your turn!

Do steps 0 and 1 of the starter code.

Displaying Arrays of Data

Goal: Make *components* out of the *data*.

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {/* TODO Show hurricane components here! */}
  </div>
}
```

Displaying Arrays of Data

Solution: `map` each piece of data to JSX!

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {
      hurricanes.map(hurr => <Hurricane></Hurricane>)
    }
  </div>
}
```

Displaying Arrays of Data

You'll often see this written short-hand.

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {
      hurricanes.map(hurr => <Hurricane/>)
    }
  </div>
}
```

Displaying Arrays of Data w/ Props

Don't forget to pass each hurricane its `props`!

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {
      hurricanes.map(hurr => <Hurricane
        name={hurr.name}
        category={hurr.category}
        start_date={hurr.start_date}
      />)
    }
  </div>
}
```

Displaying Arrays of Data w/ Props

You'll often see this written short-hand.

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {
      hurricanes.map(hurr => <Hurricane {...hurr}/>)
    }
  </div>
}
```

Uh oh!

Check your console!

```
▶ Warning: Each child in a list should have a unique "key" prop.      react-jsx-dev-runtime.development.js:87

Check the render method of `AllHurricanes`. See https://reactjs.org/link/warning-keys for more information.
    at http://localhost:5173/node_modules/.vite/deps/react-bootstrap.js?v=f4b00ec1:3635:10
    at AllHurricanes (http://localhost:5173/src/components/AllHurricanes.jsx?t=1696287519629:23:39)
    at App
```

Each component needs a unique `key`.

React `key` Prop

The `key` prop is used by React to speed up rendering.

- Always use a *unique* key for the *parent-most* element rendered in a list.
- This key needs to be *unique among siblings*.
- This key should *usually* not be the index of the item (e.g. what if the order changes?)

Learn More

Displaying Arrays of Data

Must specify a key! Not accessible via `props.key`.

```
function AllHurricanes() {
  // ...
  return <div>
    <h1>Hurricane Finder</h1>
    {
      hurricanes.map(hurr => <Hurricane key={hurr.id} {...hurr}/>)
    }
  </div>
}
```

Your turn!

Do steps 2 and 3 of the starter code.

Responsive Design

We use react-bootstrap.

See the grid docs.

Important takeaways...

- Use `Container`, `Row`, and `Col` components.
- `xs`, `sm`, `md`, `lg`, `xl`, and `xxl` are props.

Responsive Design

This is how we wrote Bootstrap in Vanilla JS...

```
<div class="container">
  <div class="row">
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
  </div>
</div>
```

Responsive Design

...this is how we will in React!

```
<Container>
  <Row>
    <Col xs={12} md={6} lg={3}></Col>
    <Col xs={12} md={6} lg={3}></Col>
    <Col xs={12} md={6} lg={3}></Col>
    <Col xs={12} md={6} lg={3}></Col>
  </Row>
</Container>
```

StackBlitz

Your turn!

Do steps 4 and 5 of the starter code.

Pagination

Also available in react-bootstrap.

Useful for handling large sums of data.

```
{/* Display items here. */}

<Pagination>
  <Pagination.Item active={false}>1</Pagination.Item>
  <Pagination.Item active={false}>2</Pagination.Item>
  <Pagination.Item active={false}>3</Pagination.Item>
  <Pagination.Item active={false}>4</Pagination.Item>
</Pagination>
```

Pagination

Use a state variable to track which page is active.

```
function SomeBigData() {
  const [page, setPage] = useState(1)
  return <div>
    {/* Display some data here! */}
    <Pagination>
      <Pagination.Item active={page === 1} onClick={() => setPage(1)}>1</Pagination.Item>
      <Pagination.Item active={page === 2} onClick={() => setPage(2)}>2</Pagination.Item>
      <Pagination.Item active={page === 3} onClick={() => setPage(3)}>3</Pagination.Item>
      <Pagination.Item active={page === 4} onClick={() => setPage(4)}>4</Pagination.Item>
    </Pagination>
  </div>
}
```

StackBlitz

Pagination

When displaying the data, use `slice` to only show the items on the current page!

```
function SomeBigData() {
  const [page, setPage] = useState(1)
  return <div>
    {
      bigData.slice((page - 1) * 16, page * 16).map(name => <p>{name}</p>)
    }
    {/* Display Pagination Items here! */}
  </div>
}
```

StackBlitz

Your turn!

Complete the hurricane example.

Handling Text Input

We can get user input using the HTML `input` tag or the React-Bootstrap `Form.Control` component.

We can get user input...

- in a *controlled* way using its `value` and tracking `onChange` events
- in an *uncontrolled* manner using `useRef`.
  - we'll cover this next time!

Controlled Components

We can *control* an input component via its `value` and `onChange` properties.

Example of a controlled input component

Example of a controlled input component (Bootstrap)

What did we learn today?

- How do we work with arrays of data in React?
  - How do we `map` out components?
  - How does the `key` property work?
- How do we do responsive design in React?
- How can we break up large sums of data using pagination?
- How can we use controlled `input` components?

Questions?