

CS 4530: Fundamentals of Software Engineering

Module 7: React

Adeel Bhutta and Mitch Wand
Khoury College of Computer Sciences

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
 - Understand how the React framework binds data (and changes to it) to a UI
 - Create simple React components that use state and properties

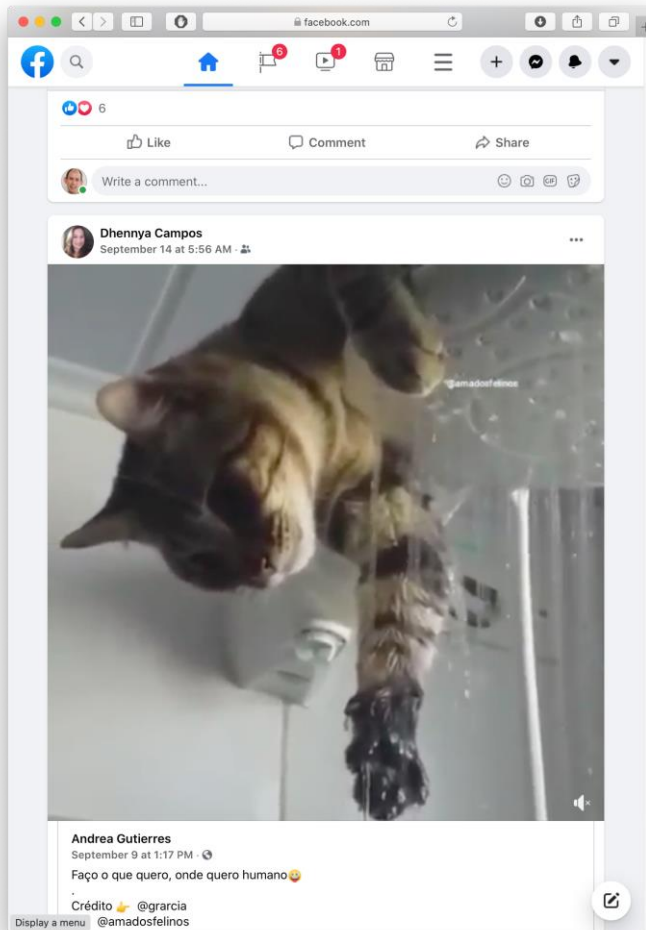
HTML: The Markup Language of the Web

- Language for describing structure of a document
- Denotes hierarchy of elements
- What might be elements in this document?



Rich, interactive web apps

- Infinite scrolling of cats



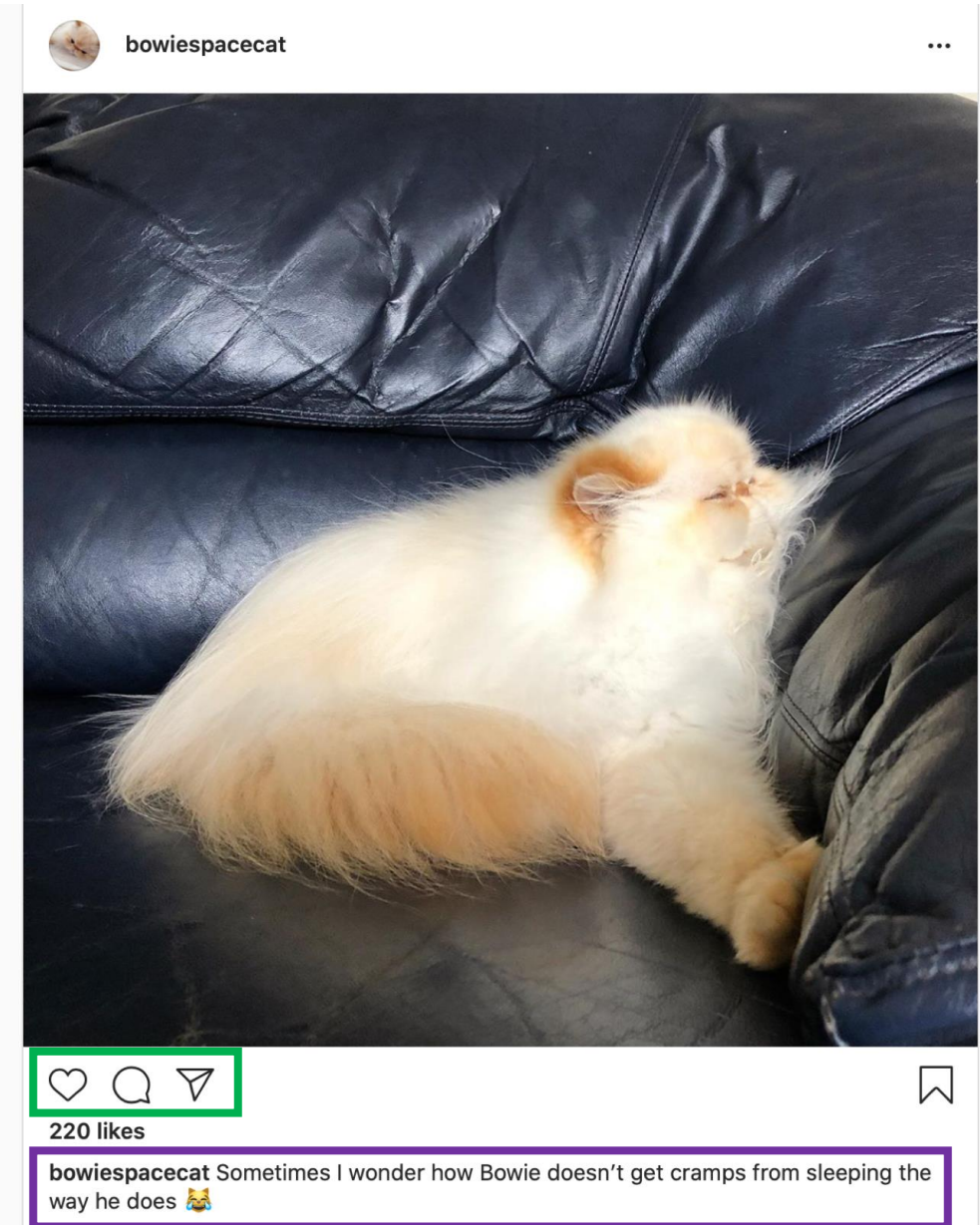
Typical properties of web app UIs

- Each widget has both visual presentation & logic
- Some widgets occur more than once
 - e.g., comment/like widgets
- Changes to data should cause changes to widget
 - e.g., new images, new comments should show up in real time
- Widgets have hierarchical structure
- Action on a widget may affect other widgets
 - e.g., clicking on 'like' button executes some logic related to the widget itself,
 - It may also affect the widget the contains the 'like' button



Components represent widgets in object-like style

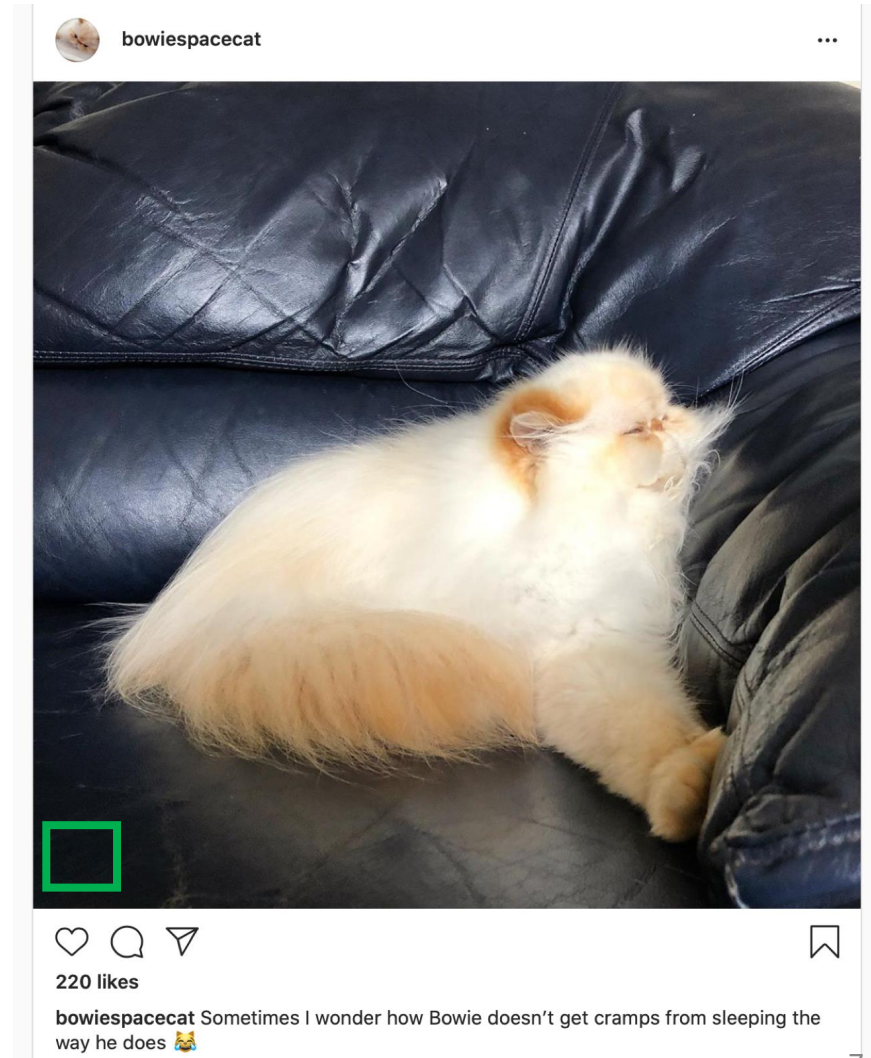
- Organize related logic and presentation into a single unit
 - Includes necessary state and the logic for updating this state
 - Includes presentation for rendering this state into HTML
- Synchronizes state and visual presentation
 - Whenever state changes, HTML should be rendered again



Components

Example: Like button component

- What does the button keep track of?
 - Is it liked or not
 - What post this is associated with
- What logic does the button have?
 - When changing like status, send update to server
- How does the button look?
 - Filled in if liked, hollow if not



Server side vs. client side

- Where should template/component be instantiated?
- Server-side frameworks: Template instantiated on server
 - Examples: JSP, ColdFusion, PHP, ASP.NET
 - Logic executes on server, generating HTML that is served to browser
- Front-end framework: Template runs in web browser
 - Examples: React, Angular, Meteor, Ember, Aurelia, ...
 - Server passes template to browser; browser generates HTML on demand

Expressing Logic

- Templates/components require combining logic with HTML
 - Conditionals - only display presentation if some expression is true
 - Loops - repeat this template once for every item in collection
- How should this be expressed?
 - Embed code in HTML (ColdFusion, JSP, Angular)
 - Embed HTML in code (React)

Embedding Code in HTML

- Template takes the form of an HTML file, with extensions
 - Popular for server-side frameworks
 - Uses another language (e.g., Java, C) or custom language to express logic
 - Found in frameworks such as PHP, Angular, ColdFusion, ASP (NOT React)
 - Can't type check anything

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
    <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
    <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

Embedding HTML in TypeScript

Aka JSX or TSX

- How do you embed HTML in TypeScript and get syntax checking?
- Idea: extend the language: JSX, TSX
 - JavaScript (or TypeScript) language, with additional feature that expressions may be HTML
- It's a new language
 - Browsers do not natively run JSX (or TypeScript)
 - We use build tools that compile everything into JavaScript

```
export function HelloMessage(props: IProps) {  
  return (  
    <div>  
      Hello, {props.name}  
    </div>  
  )  
}  
  
ReactDOM.render(  
  <React.StrictMode>  
    <HelloMessage name='Satya' />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

JSX/TSX Embeds HTML in TypeScript

- Example:

```
return <div>Hello {someVariable}</div>;
```

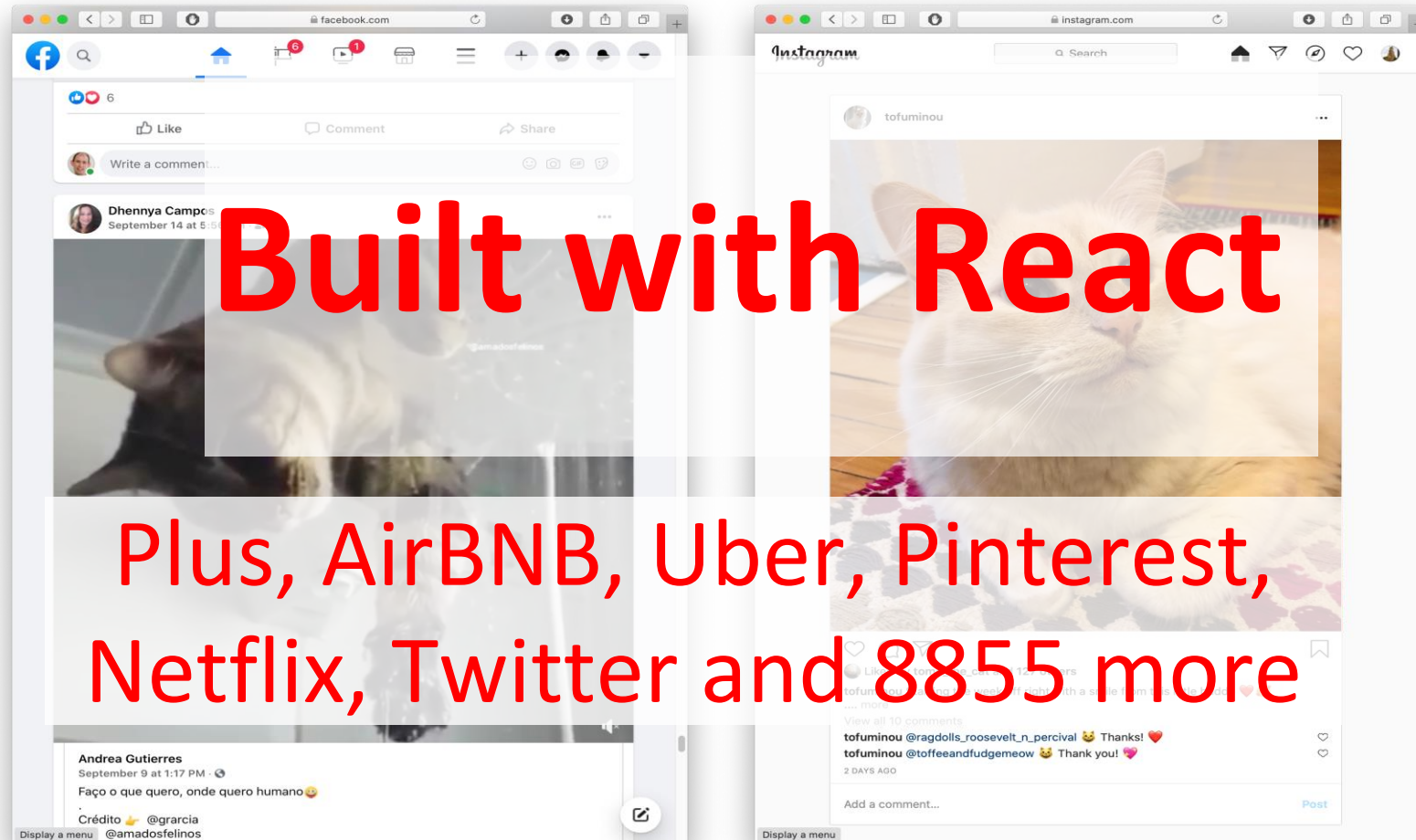
- HTML embedded in TypeScript
 - HTML can be used as an expression
 - HTML is checked for correct syntax
- Can use `{ expr }` to evaluate an expression and return a value
 - e.g., `{ 5 + 2 }`, `{ foo() }`
- To wrap on multiple lines, wrap the TSX/JSX in parentheses (...)
- Value of expression is a piece of HTML

React is a Framework for Components

- Created by Facebook
- Powerful abstractions for describing UI components
- Official documentation & tutorials: <https://reactjs.org/>
- Components are constructed in the browser (“front-end”)
- Key concepts:
 - Embed HTML in TypeScript
 - Track application “state”
 - Automatically and efficiently re-render page in browser based on changes to state
- But: some implementations of React allow components to be pre-constructed in the server.

Rich, interactive web apps

Infinite scrolling of cats



Creating React applications

- A React application is a complicated beast.
- There are several popular frameworks for building such an application
- The one we will use is called **next.js** .
- It is a full-featured framework; we will use only a small fraction of its features.


Creating New React Applications

- React applications must be “transpiled” into a format that browsers can understand
- `create-next-app` is a set of scripts to automate this all
- `npx create-next-app` starts an interactive session that creates a fully-featured TS package
- Probably you will never do this in this course— the “fully-featured TS package” is a big beast.
- Better plan is to modify one of the packages that we supply you.

Here's a sample interaction...

```
$ npx create-next-app
? What is your project named? » sample
✓ What is your project named? ... sample
? Would you like to use TypeScript? » No / Yes
✓ Would you like to use TypeScript? ... No / Yes
? Would you like to use ESLint? » No / Yes
✓ Would you like to use ESLint? ... No / Yes
? Would you like to use Tailwind CSS? » No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
? Would you like to use `src/` directory? » No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
? Would you like to use App Router? (recommended) » No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
? Would you like to customize the default import alias? » No / Yes
✓ Would you like to customize the default import alias? ... No / Yes
```

React Has a Rich Component Library

chakra

Getting Started

Styled System

Components

Hooks

Community

Changelog

Blog

LAYOUT

Aspect Ratio

Box

Center

Container

Flex

Grid

🔍 Search the docs

⌘ K

v2.2.9 ▾

Components

Chakra UI provides prebuild components to help you build your projects faster

Here is an overview of the component categories:

Disclosure

Avatar with badge

Feedback

Accordion

Tabs

Visual

EDITABLE EXAMPLE

COPY

```
<Stack direction='row' spacing={4}>
  <Avatar>
    <AvatarBadge boxSize='1.25em' bg='green.500' />
  </Avatar>

  {/* You can also change the borderColor and bg of the badge */}
  <Avatar>
    <AvatarBadge borderColor='papayawhip' bg='tomato' boxSize='1.25em' />
  </Avatar>
</Stack>
```

18

Installing Chakra for next.js:

- Just say:

```
npm i --save @chakra-ui/react @chakra-ui/next-js \
  @emotion/react @emotion/styled framer-motion
```

Hello World in React

```
import * as React from 'react';
import {
  Heading,
  VStack
} from '@chakra-ui/react';

function HelloWorldComponent() {
  return (
    <VStack>
      <Heading>Hello World</Heading>
    </VStack>
  )
}

export default function App() {
  return (<HelloWorldComponent />)
}
```

“Return the following HTML whenever the component is rendered”

The HTML is dynamically generated by the library.

Next.js renders whatever is in app/page.tsx

```
import App from './Apps/HelloWorld'  
// import App from './Apps/HelloWorldDave'  
// import App from './Apps/App1';
```

```
export default function HomePage() {  
  return (  
    <ChakraProvider>  
      <App />  
    </ChakraProvider>  
  )  
}
```

You may see “Class” components, too – but we won’t write them

```
var HelloMessage = React.createClass({  
  render: function() {  
    return <div>Hello, World!</div>  
  }  
})
```

Hello World, Circa 2016
(Before the “Class” keyword!)

```
class HelloMessage extends React.Component {  
  render() {  
    return <div>Hello, World!</div>  
  }  
}
```

Hello World, Circa 2020
(Defined as a Class)

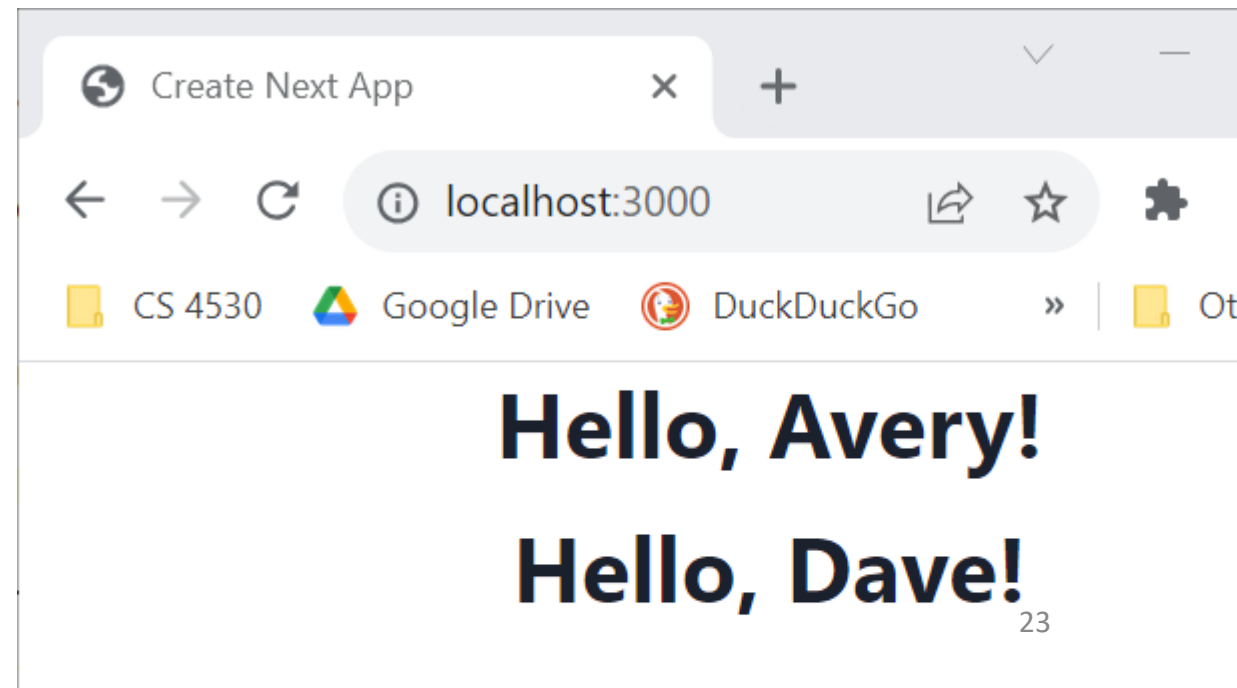
```
export function HelloMessage() {  
  return <div>Hello, World!</div>  
}
```

Hello World, Circa 2022
(Defined as a function)

React Components Can Receive Properties.

- Properties are passed in an argument to the component
- Properties are specified as attributes when the component is instantiated
- Properties can *not* be changed by the component


```
export default function App() {  
  return (<VStack>  
    <HelloWorldWithName name='Avery' />  
    <HelloWorldWithName name='Dave' />  
  </VStack>  
)  
}
```



Component State is Data That Changes

- State is data that, when changed, should trigger UI update
- State is created by `useState`.
- The state is accessed through *state variables* in the component.
- The first variable is the accessor, the second is the setter.
- The only way to change the value of a state variable is with the setter

```
import { useState } from 'react';  
function Foo() {  
  const [count, setCount] = useState(0)  
  ...  
}
```



You could choose any names for the variable and its setter; for this class, please follow the naming convention (`goodVariableName`, `setGoodVariableName`) that we've used here.

Example

```
export default function App() {  
  
  const [count, setCount] = useState(0)  
  
  function handleClick() { setCount(count + 1) }  
  
  return (  
    <VStack>  
      <Box> count = {count} </Box>  
      <Button onClick={handleClick} >  
        Increment Count!  
      </Button>  
    </VStack>  
  )  
}
```

(Some styling has been removed to reduce clutter on this screen.)

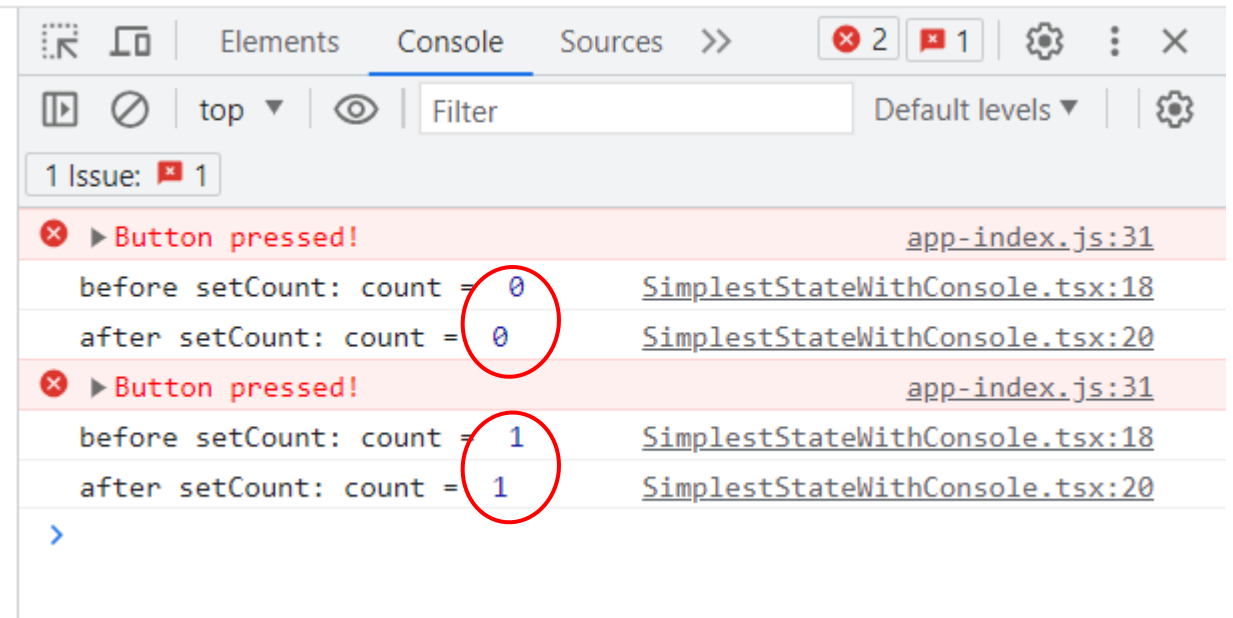
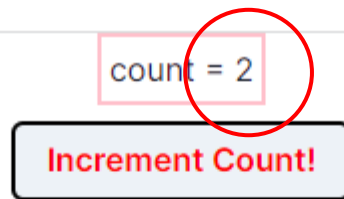
Setters are asynchronous

- A setter doesn't change the state immediately: it tells REACT to update the state when this component is redisplayed.

```
function handleClick() {  
  console.error('Button pressed!');  
  console.log('before setCount: count = ', count)  
  setCount(count + 1)  
  console.log('after setCount: count = ', count)  
}
```

Console methods: <https://developer.mozilla.org/en-US/docs/Web/API/console>

Setters are asynchronous



Nest Components, Passing State as Properties

app/Apps/TwoCountingButtons.tsx

```
import { CountingButton } from './CountingButton';

export default function App() {
  const [globalCount, setGlobalCount] = useState(0)

  function handleClick() {setGlobalCount(globalCount + 1)}

  return (
    <VStack>
      <Box border="1px" padding='1'>Total count = {globalCount}</Box>
      <Box h="20px" />
      <CountingButton name="Button A" onClick={handleClick} />
      <Box h="20px" />
      <CountingButton name="Button B" onClick={handleClick} />
    </VStack>
  )}
}
```

A common pattern in React is to nest components, passing information from parent to child via props.

CountingButton.tsx

app/Apps/CountingButton.tsx

```
export function CountingButton(props: {
  name:string, onClick:() => void }) {

  const name = props.name
  const [localCount, setLocalCount] = useState(0)

  function handleClick() {
    console.error(props.name, 'pressed!');
    setLocalCount(localCount + 1)
    props.onClick() // propagate to parent
  }
}
```

```
return (
  <VStack>
    <Box>
      count for {props.name} = {localCount}
    </Box>

    <Button onClick={handleClick}>
      Increment {name}!
    </Button>
  </VStack>
)
}
```

(Some styling has been removed to reduce clutter on this screen.)

TwoCountingButtons demo

Total count = 6

count for Button A = 2

Increment Button A!

count for Button B = 4

Increment Button B!

A ToDo App

app/Apps/ToDoApp.tsx

```
export default function ToDoApp () {
  const [todoList, setTodolist] = useState<TodoItem[]>([])
  function handleAdd (newItem:TodoItem) {
    if (newItem.title === '') {return} // ignore blank button presses
    setTodolist(todoList.concat(newItem))
  }
  function handleDelete(targetId:string) {
    const newList = todoList.filter(item => item.id !== targetId)
    setTodolist(newList)
  }

  return (
    <VStack>
      <Heading>TODO List</Heading>
      <ToDoItemEntryForm onAdd={handleAdd}/>
      <ToDoListDisplay items={todoList} onDelete={handleDelete}/>
    </VStack>
  )
}
```

Typical Page

TODO List

Add TODO item here:

Add TODO item

TITLE	PRIORITY	DELETE
first item	11	
second item	22	
third item	optional	

Pattern: display a list of items using **map**

```
export function ToDoListDisplay(props: { items: ToDoItem[],  
                                         onDelete:(id:string) => void })  
  
  return (  
    <Table>  
      <Tbody>  
        {  
          props.items.map((eachItem) =>  
            <ToDoItemDisplay item={eachItem}  
              key={eachItem.id}  
              onDelete={props.onDelete} />  
          )  
        }  
      </Tbody>  
    </Table>  
  )  
}
```

But using map comes with a big gotcha.

```
export function ToDoListDisplay(props: { items: ToDoItem[],  
                                         onDelete:(id:string) => void })  
  
  return (  
    <Table>  
      <Tbody>  
        {  
          props.items.map((eachItem) =>  
            <ToDoItemDisplay item={eachItem}  
              key={eachItem.id}  
              onDelete={props.onDelete} />)  
        }  
      </Tbody>  
    </Table>  
  )  
}
```

The key attribute must be unique *and stable*.

- This doesn't work:

```
props.items.map((eachItem, index) =>
  <ToDoItemDisplay item={eachItem} key={index.toString()} onDelete={props.onDelete} />
)
```

Summarizing React Behavior

- React uses default state for the first render of our component.
- When setter is called, React *asynchronously* re-renders our component and updates the state variable.
- Updating the DOM in the browser is slow - it is *vital* that React does efficient diff'ing
 - Example: adding a new comment on a YouTube video shouldn't make the browser re-layout the whole page
- React makes re-rendering faster by updating only the part that changes.
 - This is called "Reconciliation"
 - It uses some magic like keeping track of state of each component (e.g., second component was liked)
 - Keys are necessary for correct re-rendering of lists. These should be unique and stable (don't change with each update)

Review

- Now that you've studied this lesson, you should be able to:
 - Understand how the React framework binds data (and changes to it) to a UI
 - Create simple React components that use state and properties
- The next lesson will do a deep dive on patterns of React, including **useState** and its friend, **useEffect**