CS 4530: Fundamentals of Software Engineering
Module 7: React

Jonathan Bell, Adeel Bhutta, Mitch Wand

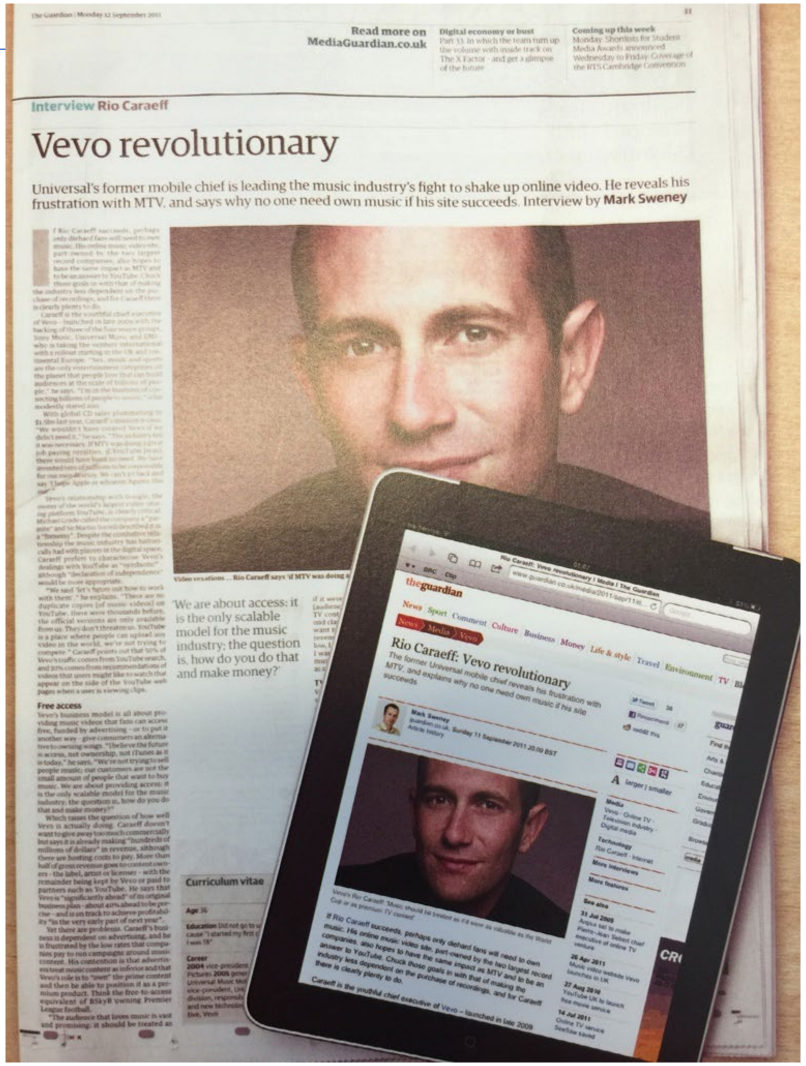
Khoury College of Computer Sciences

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
  - Be able to map the three core steps of a test (construct, act, check) to UI component testing

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?



Read more on MediaGuardian.co.uk

Interview Rio Caraeff

Vevo revolutionary

Universal's former mobile chief is leading the music industry's fight to shake up online video. He reveals his frustration with MTV, and says why no one need own music if his site succeeds. Interview by Mark Sweney

'We are about access: it is the only scalable model for the music industry; the question is, how do you do that and make money?'

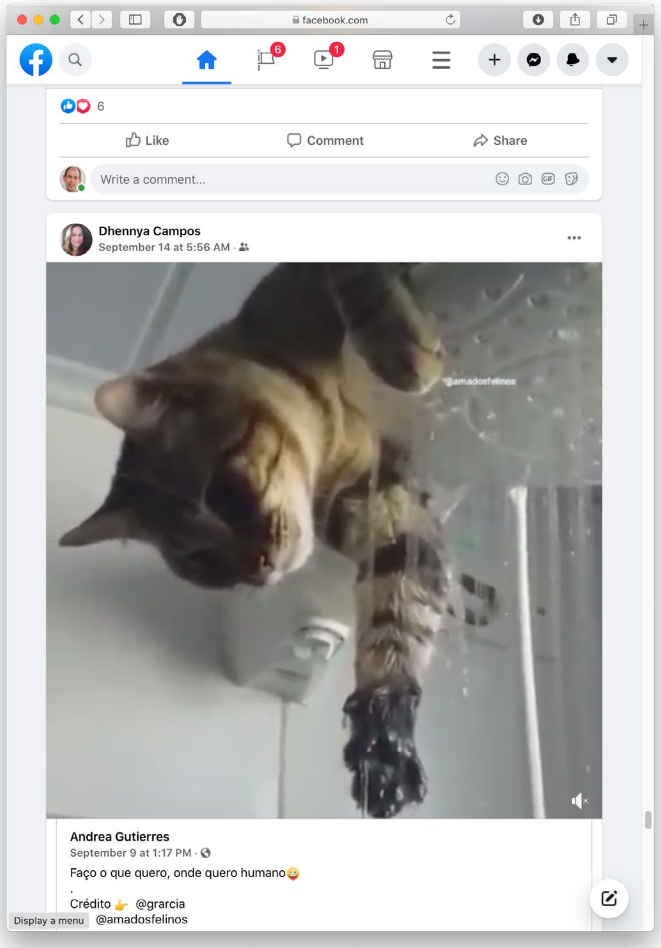
Curriculum vitae

theguardian

Rio Caraeff: Vevo revolutionary

Rich, interactive web apps

- Infinite scrolling of cats

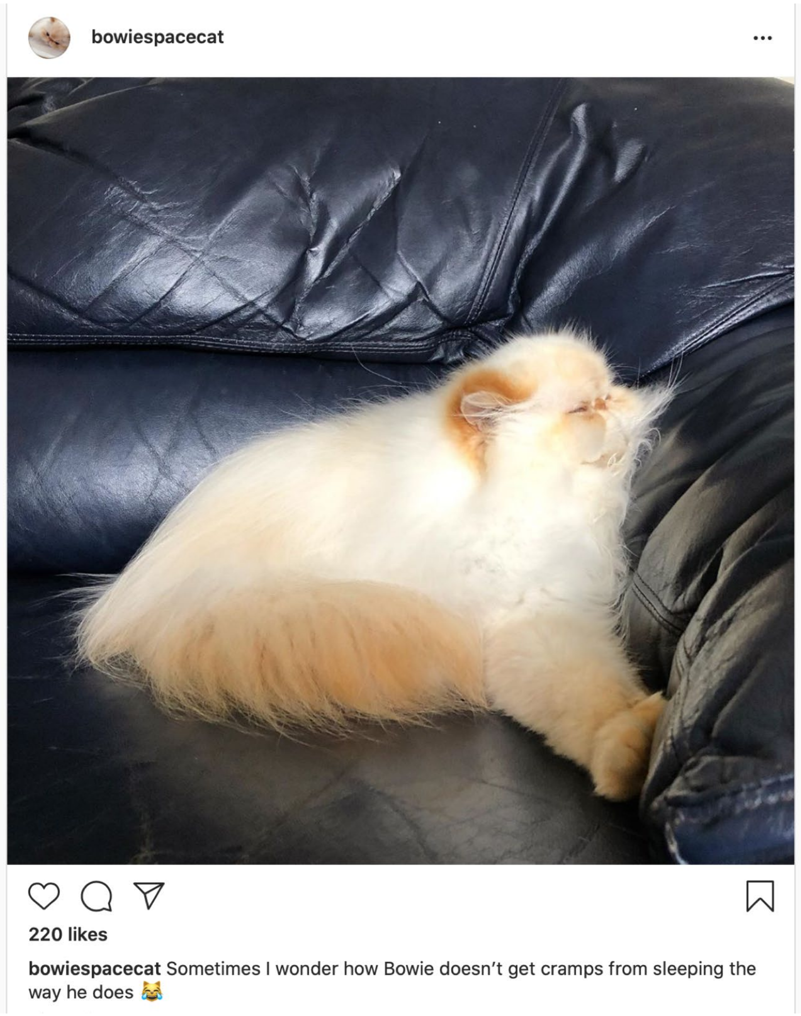


Dhennya Campos
September 14 at 5:56 AM

Andrea Gutierres
September 9 at 1:17 PM
Faço o que quero, onde quero humano😀
.
Crédito 👉 @grarcia
@amadosfelinos

Typical properties of web app Uis
Building abstractions for web app development?

- Each widget has both visual presentation & logic
  - e.g., clicking on like button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets
- 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

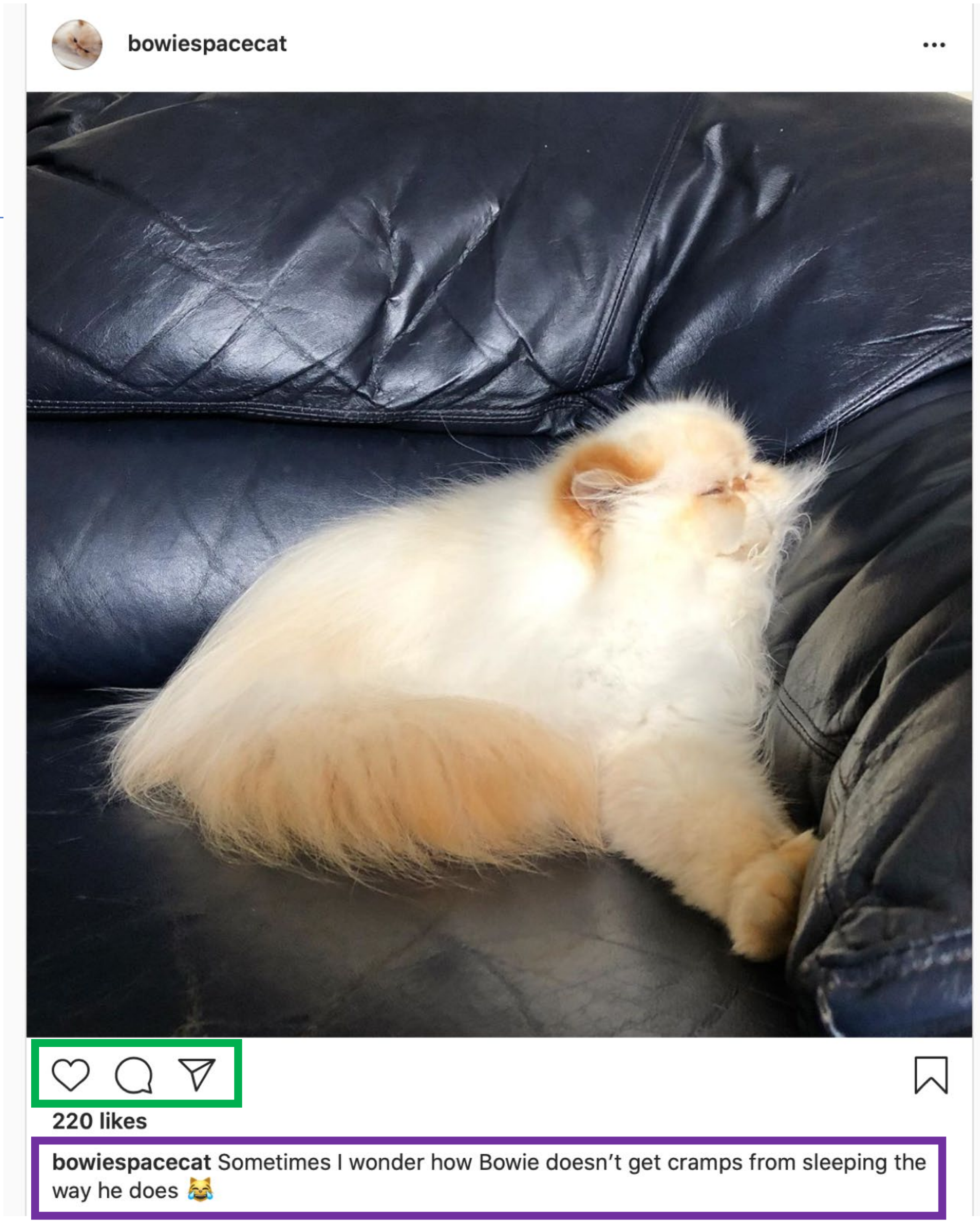


220 likes

bowiespacecat Sometimes I wonder how Bowie doesn't get cramps from sleeping the way he does 😹

Key Idea: Components

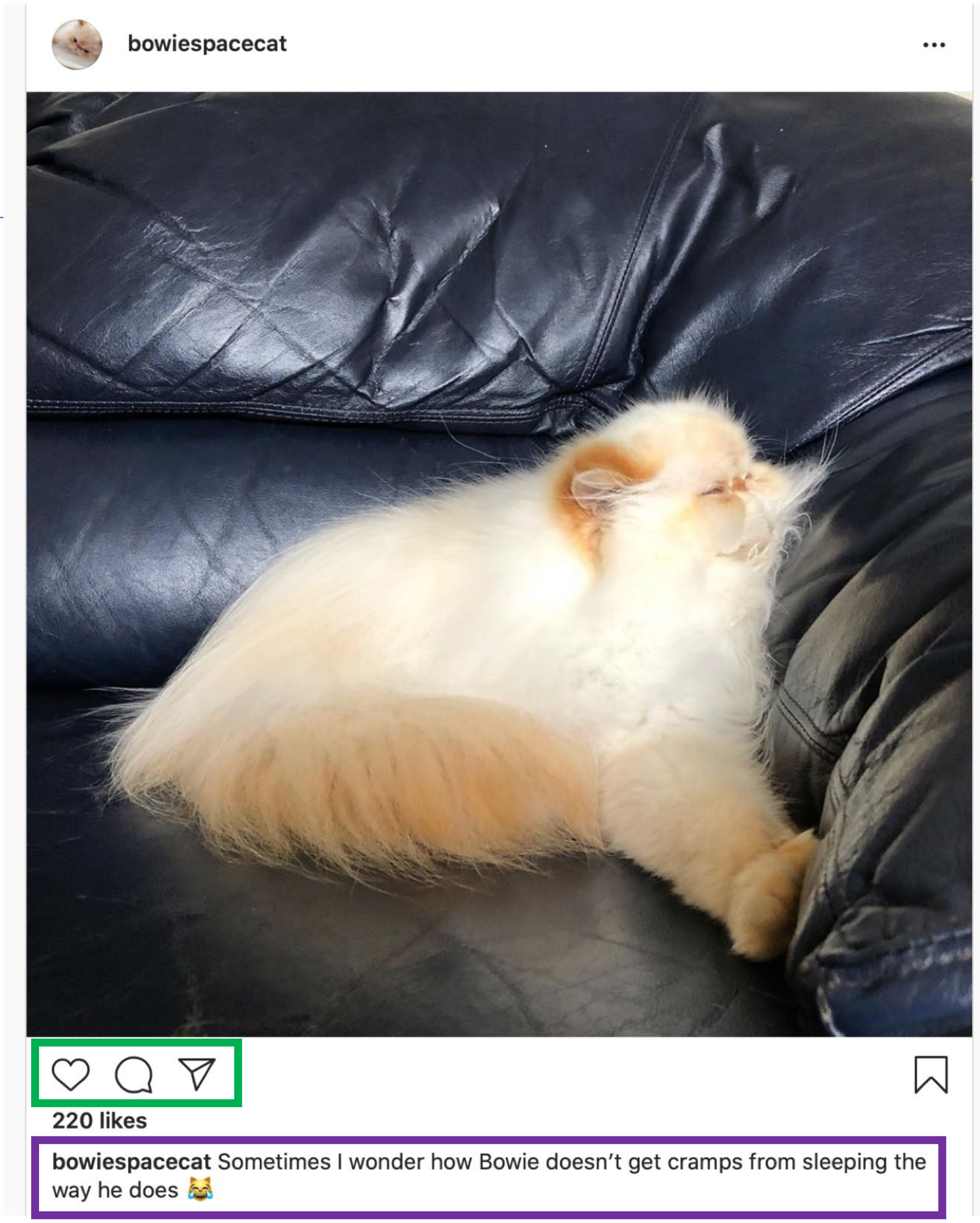
- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components - Easy to repeat, cohesive pieces of code (hopefully with low coupling)



bowiespacecat Sometimes I wonder how Bowie doesn’t get cramps from sleeping the way he does 😹

Components

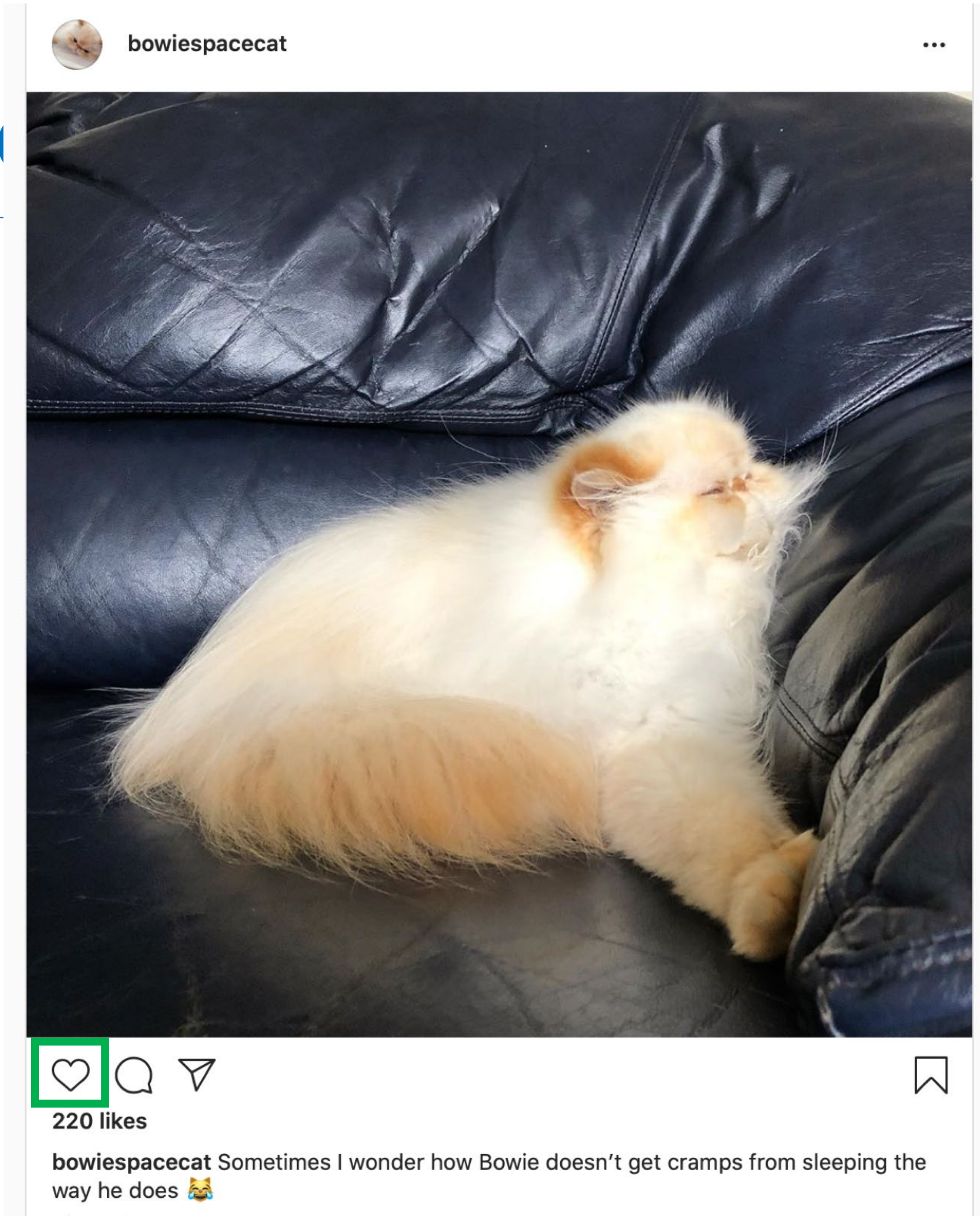
- Organize related logic and presentation into a single unit
  - Includes necessary state and the logic for updating this state
  - Includes presentation for rendering this state into HTML
- Synchronizes state and visual presentation
  - Whenever state changes, HTML should be rendered again



bowiespacecat Sometimes I wonder how Bowie doesn't get cramps from sleeping the way he does 😹

Components
Example: Like button comp

- 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



bowiespacecat Sometimes I wonder how Bowie doesn't get cramps from sleeping the way he does 😹

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

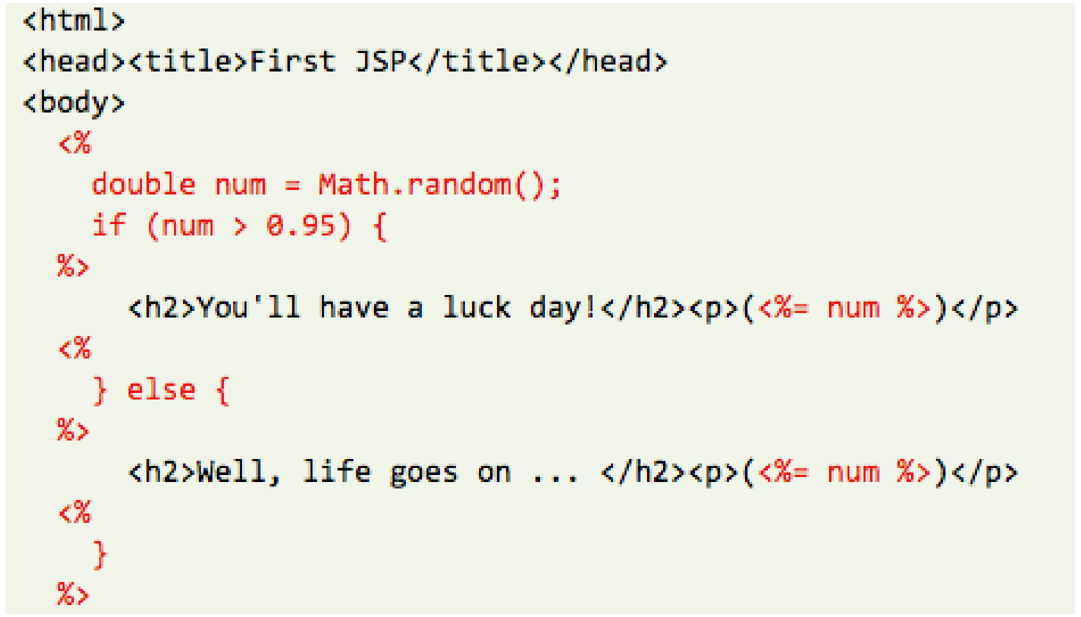
Expressing Logic

- 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

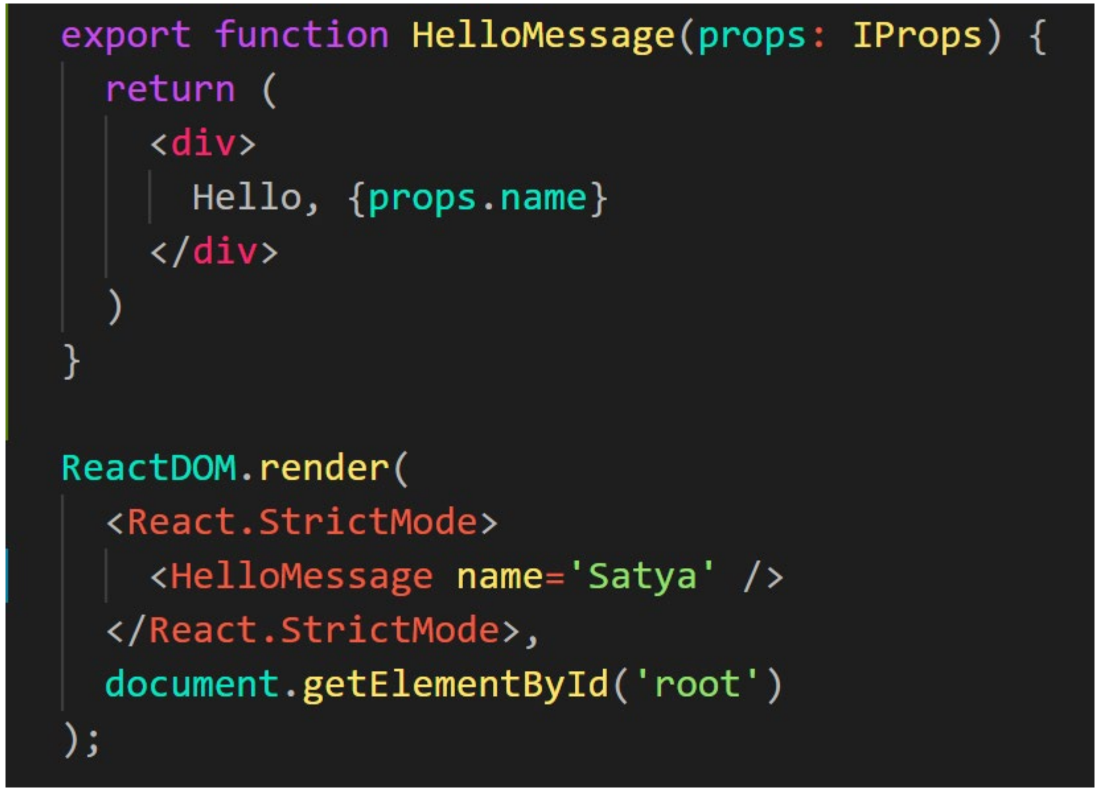
- Template takes the form of an HTML file, with extensions
  - Popular for server-side frameworks
  - Uses another language (e.g., Java, C) or custom language to express logic
  - Found in frameworks such as PHP, Angular, ColdFusion, ASP (NOT React)
  - Can't type check anything



```
<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

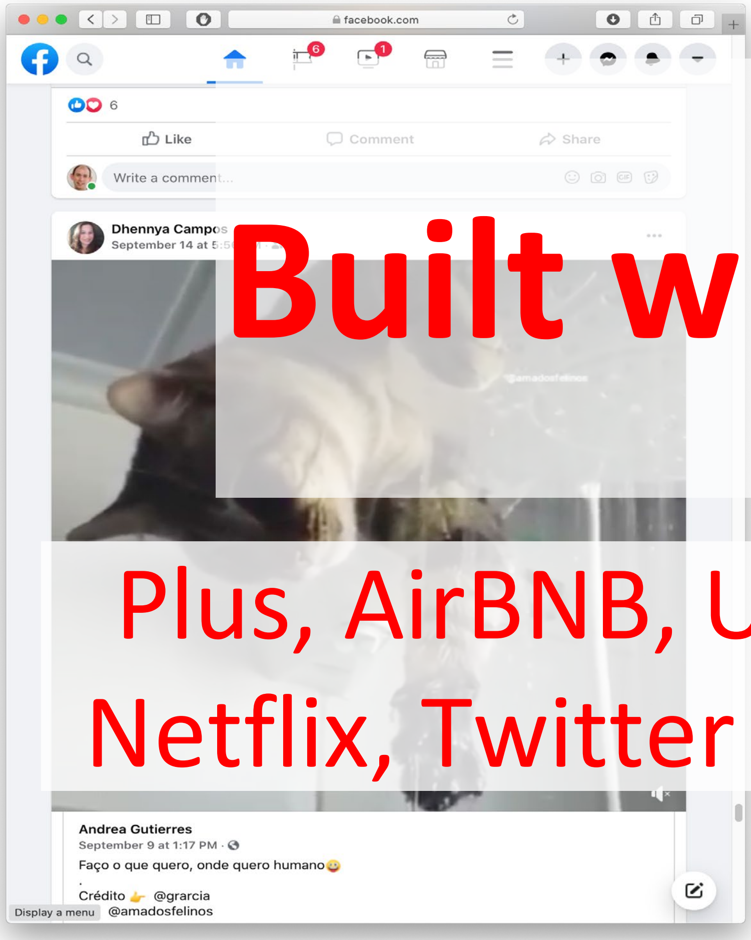


```
export function HelloMessage(props: IProps) {
  return (
    <div>
      Hello, {props.name}
    </div>
  )
}

ReactDOM.render(
  <React.StrictMode>
    <HelloMessage name='Satya' />
  </React.StrictMode>,
  document.getElementById('root')
);
```

React: Front End Framework for Components

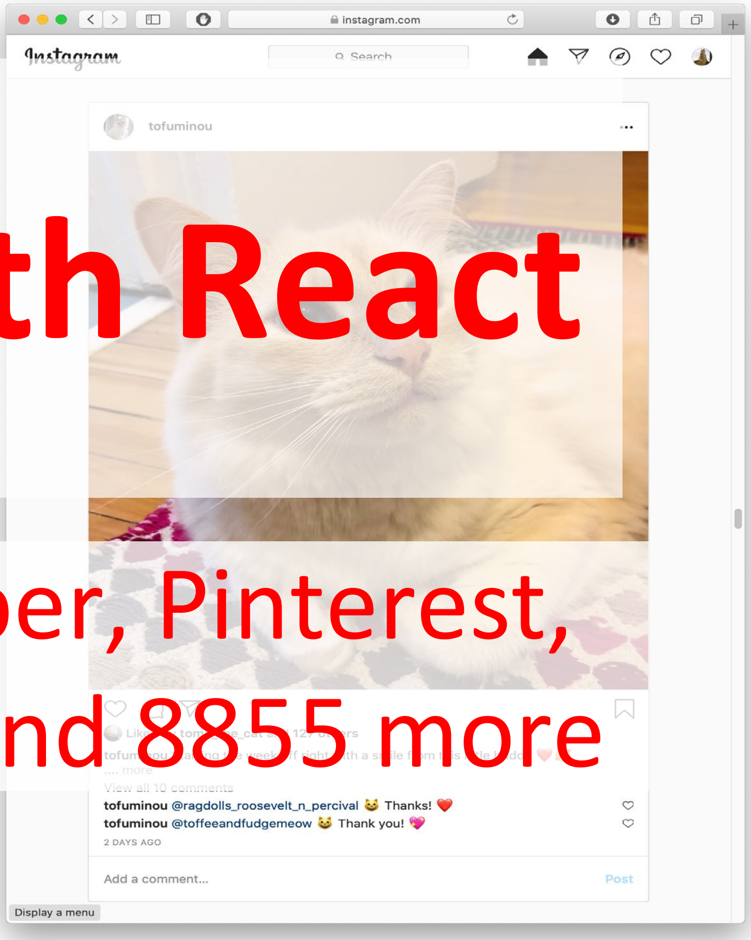
- Created by Facebook
- Powerful abstractions for describing frontend UI components
- Official documentation & tutorials: https://reactjs.org/
- Key concepts:
  - Embed HTML in TypeScript
  - Track application “state”
  - Automatically and efficiently re-render page in browser based on changes to state

Rich, interactive web apps
Infinite scrolling of cats



Dhennya Campos
September 14 at 5:5

Andrea Gutierres
September 9 at 1:17 PM ·
Faço o que quero, onde quero humano😀
.
Crédito 👉 @grarcia
@amadosfelinos



tofuminou @ragdolls_roosevelt_n_percival 😺 Thanks! ❤️
tofuminou @toffeeandfudgemeow 😺 Thank you! 💖
2 DAYS AGO

Built with React

Plus, AirBNB, Uber, Pinterest, Netflix, Twitter and 8855 more

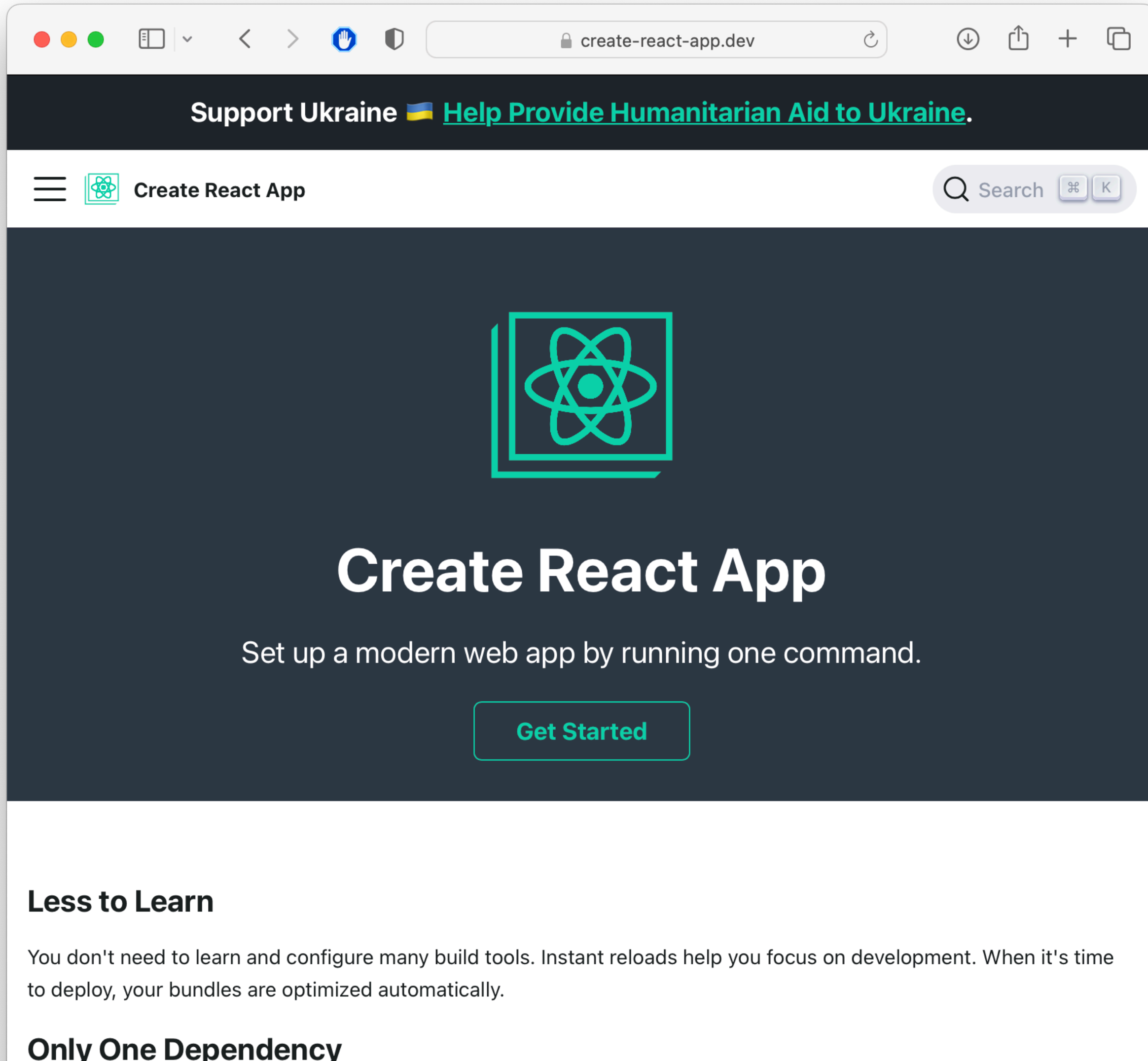
Embedding HTML in TypeScript

```
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 in (parentheses)
- Output of expression is HTML

Creating New React Applications

- React applications must be "transpiled" into a format that browsers can understand
- "Create React App" is a set of scripts to automate this all
- Get started: `npx create-react-app my-app --template typescript`
- Implement in App.tsx, run `npm start` to run in web browser



Support Ukraine 🇺🇦 Help Provide Humanitarian Aid to Ukraine.

Create React App

Create React App

Set up a modern web app by running one command.

Get Started

Less to Learn

You don't need to learn and configure many build tools. Instant reloads help you focus on development. When it's time to deploy, your bundles are optimized automatically.

Only One Dependency

Hello World in React

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

“Declare a Hello component”

Declares a new component that can be rendered by React

“Return the following HTML whenever the component is rendered”

The HTML is dynamically generated by the library.

```
function App(){
  return <HelloMessage />;
}
```

“Render a Hello Component”

Components are rendered as if they were HTML tags

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>
}
```

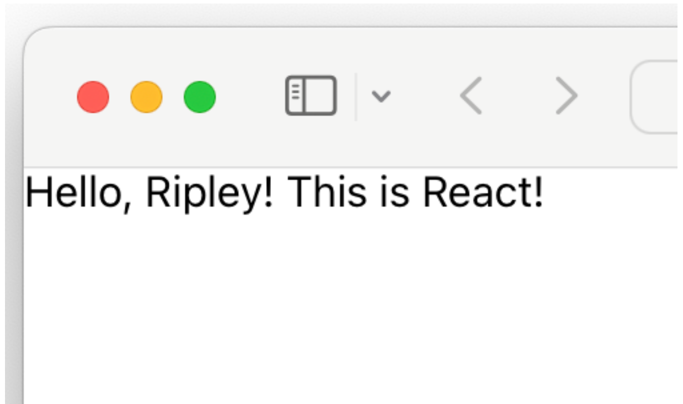
Hello World, Circa 2022
(Defined as a function)

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
- Reminder: inside of HTML code, execute TypeScript code using {mustaches}

```
export function PersonalizedHello(props: {name: string}){
  return <div>Hello, {props.name}! This is React!</div>
}
```

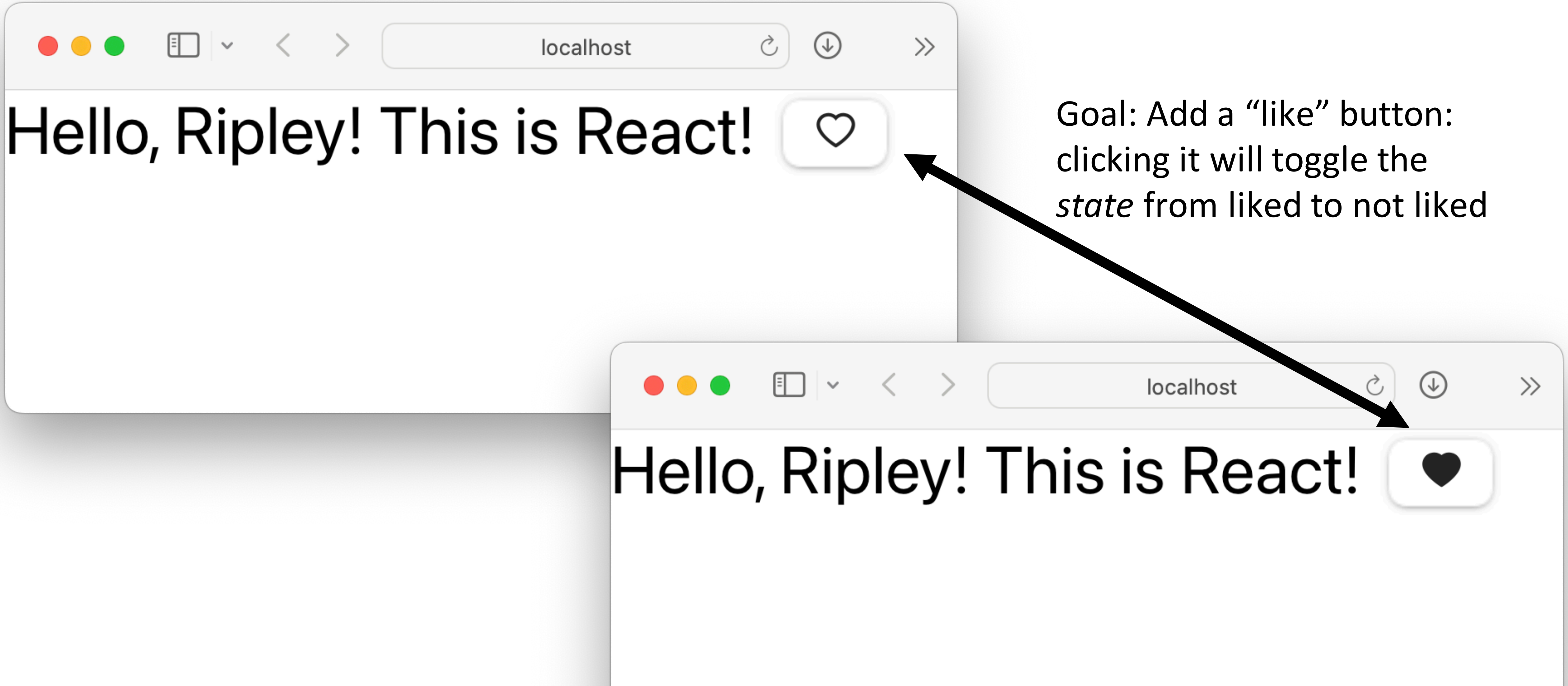
```
<PersonalizedHello name="Ripley" />
```


Hello, Ripley! This is React!

Component State is Data That Changes

- All internal component data that, when changed, should trigger UI update
  - Stored as state variables in the component
    - Created using useState<stateType>(defaultValue)
    - E.g. const [isLiked, setIsLiked] = *useState*(false);
    - Import *useState* from React
  - The only way to change the value of a state variable is with the setter
  - You *could* choose any names for the variable and its setter; for this class, please follow the convention of const [goodVariableName, setGoodVariableName]

React State Example: “Like” Button



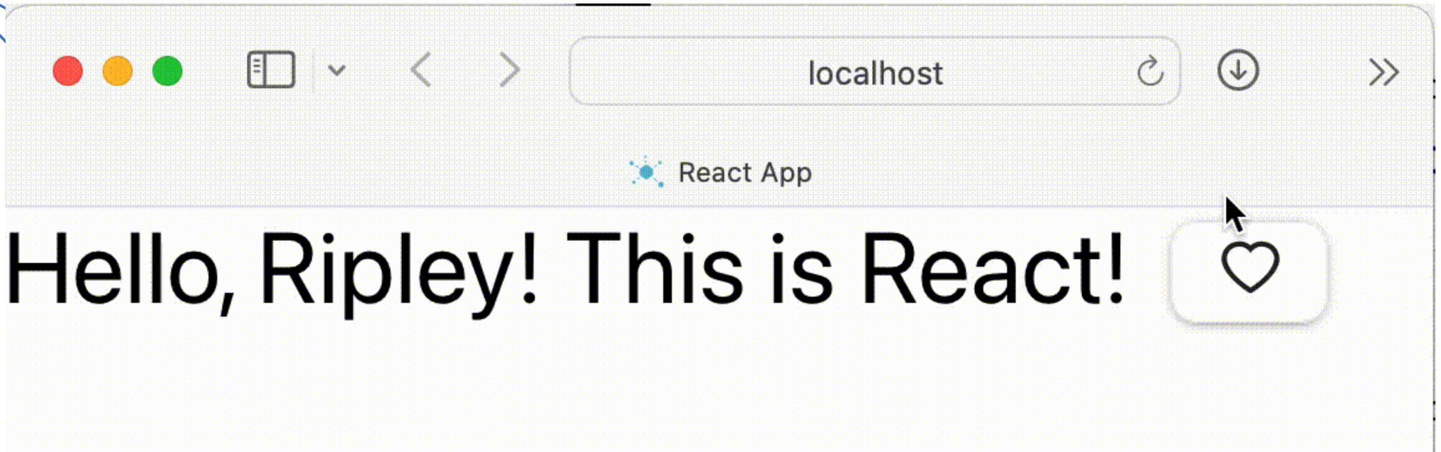
Goal: Add a “like” button: clicking it will toggle the *state* from liked to not liked

React State Example: “Like” Button

Create a state variable called isLiked, and a *state setter*, defaulting to *false*

Depending on the state, show a filled-in or outlined button

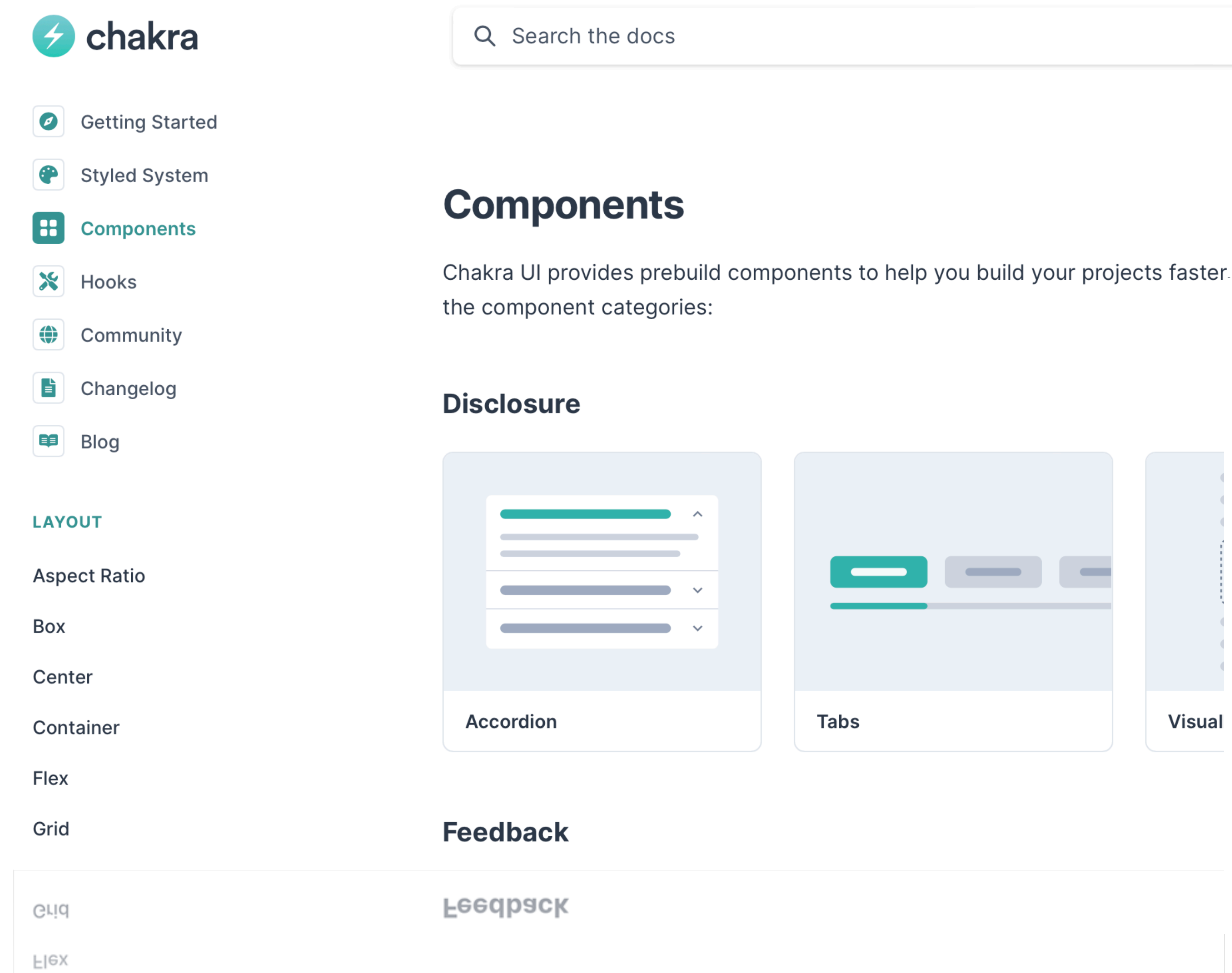
```
function PersonalizedLikableHello(props: { name: string }) {
  const [isLiked, setIsLiked] = useState(false);
  let likeButton;
  if (isLiked) {
    likeButton = (<IconButton aria-label="unlike"
        icon={<AiFillHeart />} onClick={() => setIsLiked(false)} /> );
  } else {
    likeButton = (<IconButton aria-label="like"
        icon={<AiOutlineHeart />} onClick={() => setIsLiked(true)} /> );
  }
  return (
    <div>
      Hello, {props.name}! This is
    </div>
  );
}
```


localhost

React App

Hello, Ripley! This is React!

Sidebar: React Has a Rich Component Library



chakra

Search the docs ⌘ K v2.2.9

Getting Started
Styled System
Components
Hooks
Community
Changelog
Blog

LAYOUT

Aspect Ratio
Box
Center
Container
Flex
Grid

Components

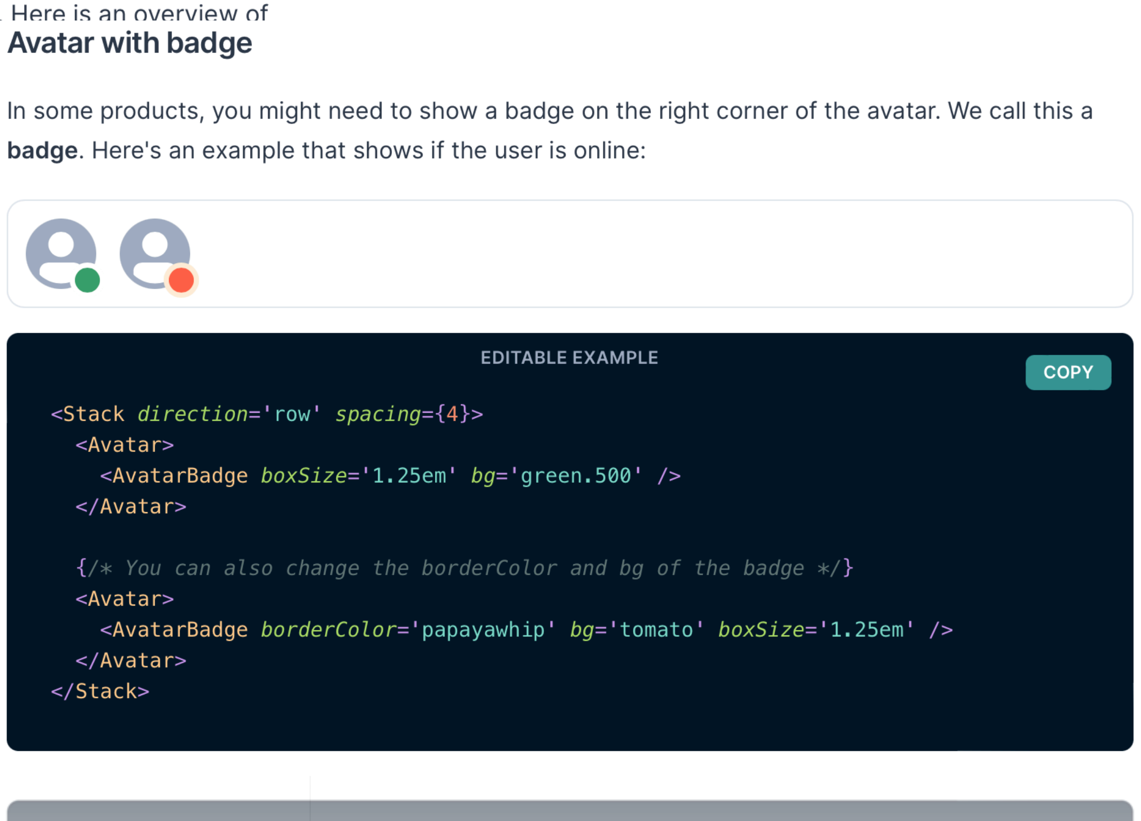
Chakra UI provides prebuild components to help you build your projects faster. Here is an overview of the component categories:

Disclosure

Accordion | Tabs | Visual

Feedback

Install UI libraries from NPM just like any other kind of module, e.g.
`npm install --save @chakra-ui/react`



Avatar with badge

In some products, you might need to show a badge on the right corner of the avatar. We call this a **badge**. Here's an example that shows if the user is online:

EDITABLE EXAMPLE COPY

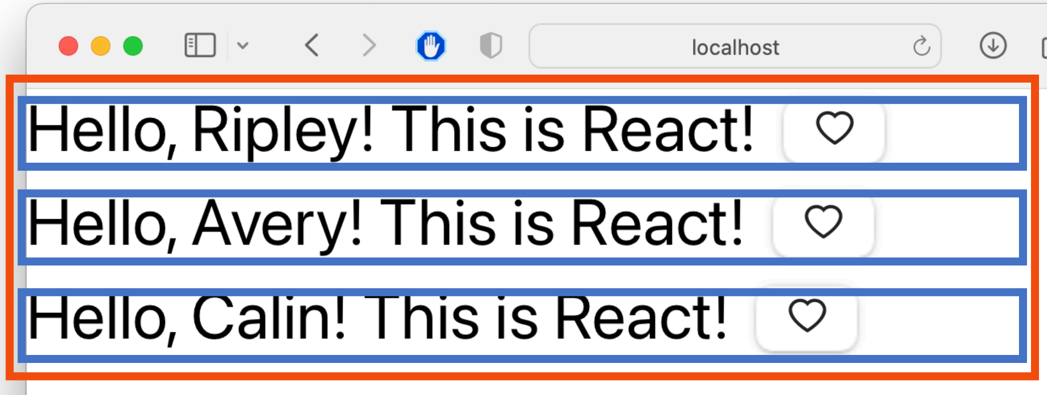
```
<Stack direction='row' spacing={4}>
  <Avatar>
    <AvatarBadge boxSize='1.25em' bg='green.500' />
  </Avatar>

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

Nest Components, Passing State as Properties

- A common pattern in React is to store state in one component, and nest others in it, passing properties
- Example: Creating multiple PersonalizedHello's:

```
export function MultiHellos() {
const [names, setNames] = useState(["Ripley", "Avery", "Calin"]);
  return (
    <div>
      {names.map((eachName) => (
         <PersonalizedLikableHello name={eachName} />
     ))}
    </div>
  );
}
```

- Problem: How to add "delete" buttons?



Hello, Ripley! This is React!

Hello, Avery! This is React!

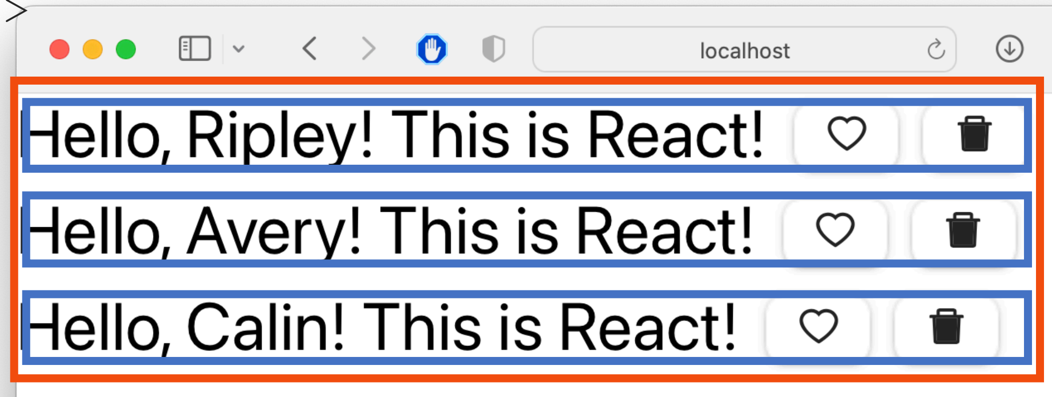
Hello, Calin! This is React!

Do not reference this slide for study (spoiler alert!)

Nest Components, Passing State (and setter) as Properties

- Add a “delete” button inside of each Hello Message
- What should the delete button do? The state with the list of names is stored in the MultiHellos component
- Solution: Pass an “onDelete” handler to each

```
export function MultiHellos() {
  const [names, setNames] = useState(["Ripley", "Avery", "Calin"]);
  return (<div>
        {names.map((eachName) => (
            <PersonalizedLikableDeletableHello name={eachName}
            onDelete={()=> setNames(names.filter(
          filteredName => filteredName !== eachName))}/>
        ))}
        </div>
  );
}
```


localhost

Hello, Ripley! This is React!

Hello, Avery! This is React!

Hello, Calin! This is React!

Do not reference this slide for study (spoiler alert!)

React State Example: “Delete” Button

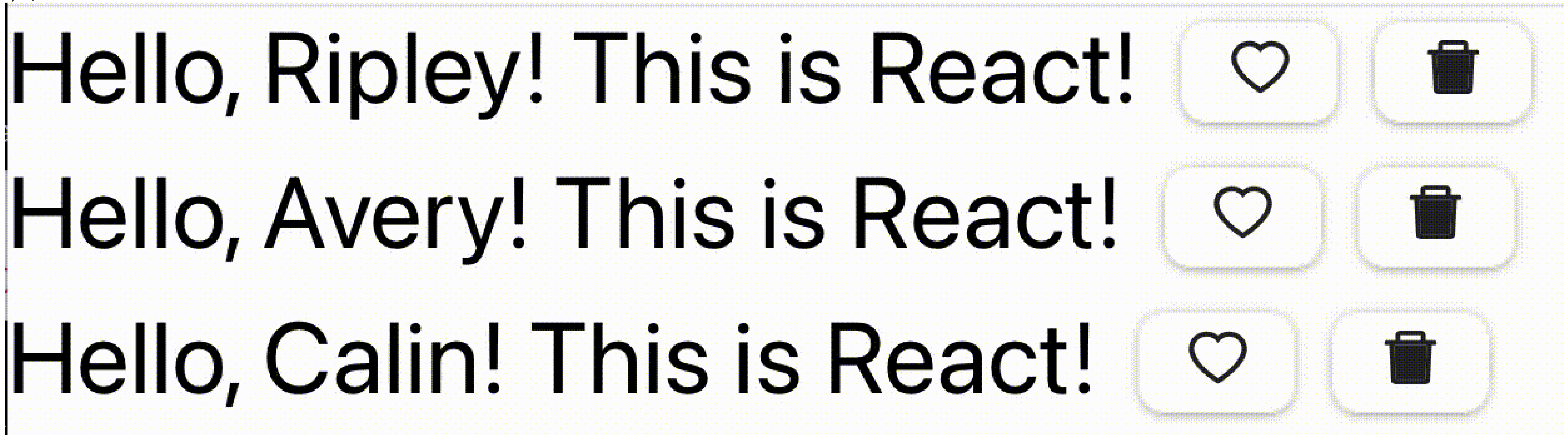
Create a state variable called isLiked, and a *state setter*, defaulting to *false*

onDelete prop of this button is connected to the onClick handler

```
function PersonalizedLikableDeletableHello (
                                           props: {name: string, onDelete: ()=> void }
                                                )
{
  const [isLiked, setIsLiked] = useState(false);
  let likeButton;
  ……

  return (
 <div>
       Hello, {props.name}! This is React! {likeButton}
       <IconButton aria-label='delete' icon={<AiTwotoneDelete />} onClick={props.onDelete} />
     </div>
  );
}
```

Testing the “Delete” button

```
export function MultiHellos() {
  const [names, setNames] = useState(["Ripley", "Avery", "Calin"]);
  return (<div>
      {names.map((eachName) => (
          <PersonalizedLikableDeletableHello name={eachName}
          onDelete={()=> setNames(names.filter(
         filteredName => filteredName !== eachName))}/>
      ))}
      </div>
  );
}
```

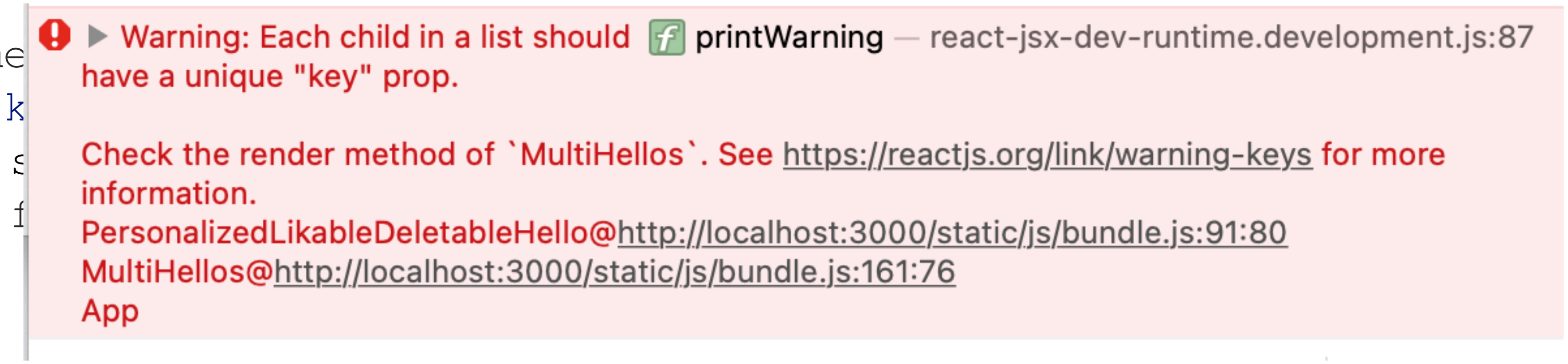

Hello, Ripley! This is React!

Hello, Avery! This is React!

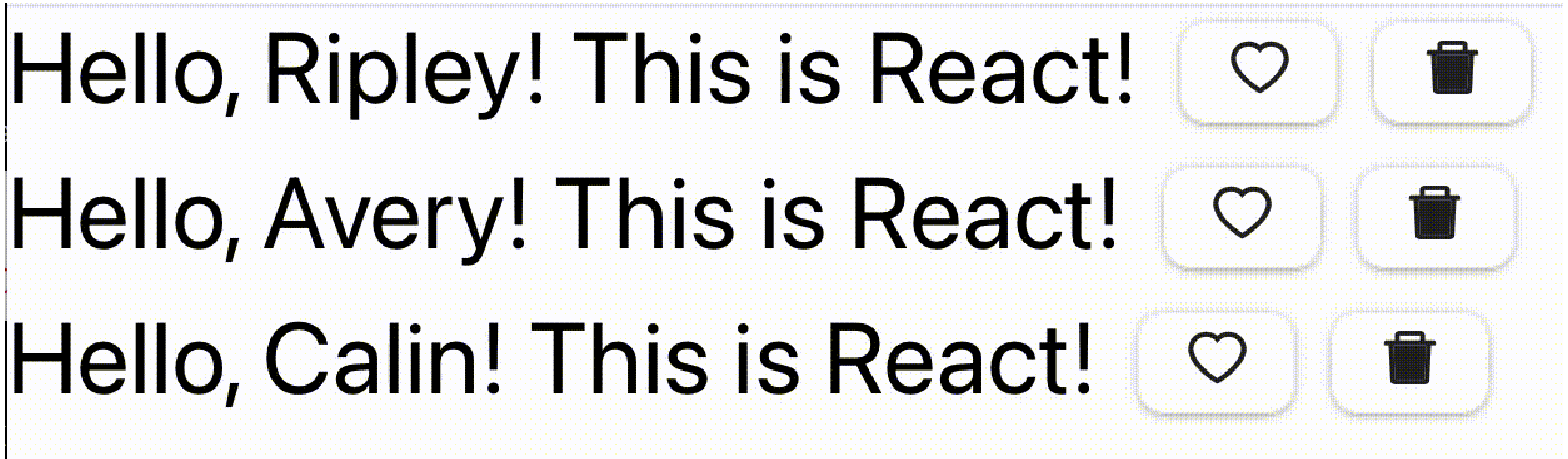
Hello, Calin! This is React!

Testing the Delete AND Like Buttons

```
export function MultiHellos() {
  const [names, setNames] = useState(["Ripley", "Avery", "Calin"]);
  return (<div>
        {names.map((eachName
             <PersonalizedLik
             onDelete={()=> s
            filteredName => f
        ))}
        </div>
  );
}
```



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

Check the render method of `MultiHellos`. See https://reactjs.org/link/warning-keys for more information.
PersonalizedLikableDeletableHello@http://localhost:3000/static/js/bundle.js:91:80
MultiHellos@http://localhost:3000/static/js/bundle.js:161:76
App
```



Hello, Ripley! This is React!

Hello, Avery! This is React!

Hello, Calin! This is React!

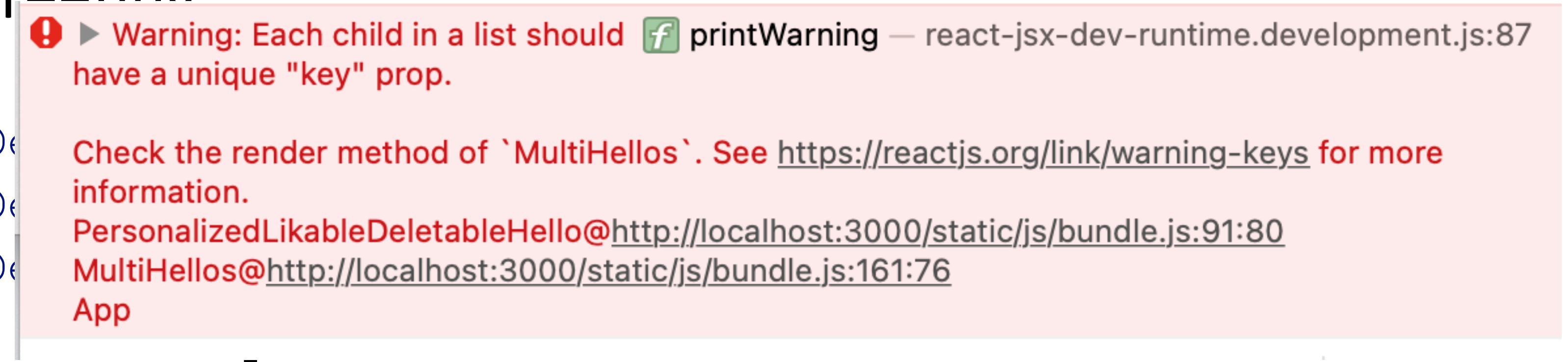
Reacting to change: How does the page update automatically?

- Re-rendering is *asynchronous*: do not happen immediately upon calling a state setter
- Reconciliation: Framework diffs the previously rendered DOM with the new DOM, updating only part of DOM that changed
- 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

Reconciliation Must Differentiate Updates from Deletions/Additions

Before deleting Ripley's Gr

```
<div>
  <PersonalizedLikableDe
  <PersonalizedLikableDe
  <PersonalizedLikableDe
</div>
```



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

Check the render method of `MultiHellos`. See https://reactjs.org/link/warning-keys for more
information.
PersonalizedLikableDeletableHello@http://localhost:3000/static/js/bundle.js:91:80
MultiHellos@http://localhost:3000/static/js/bundle.js:161:76
App
```

After deleting Ripley's Greeting:

```
<div>
  <PersonalizedLikableDeletableHello name="Avery" />
  <PersonalizedLikableDeletableHello name="Calin" /> /* isLiked=true */
</div>
```

React processed this change as:
Ripley's greeting becomes Avery's greeting
Avery's greeting becomes Calin's greeting
Calin's greeting is deleted

Reconciliation with Keys

- Add the “key” attribute to each component in a list
- Keys must be unique
- React will use the “key” to determine which elements are added, deleted, or re-ordered when re-rendered

```
export function MultiHellos() {
  const [names, setNames] = useState(["Ripley", "Avery", "Calin"]);
  return (<div>
      {names.map((eachName) => (
          <PersonalizedLikableDeletableHello name={eachName}
          key={eachName}
          onDelete={()=> setNames(names.filter(filteredName => filteredName !== eachName))}/>
      ))}
      </div>
  );
}
```

Write UI component tests just like any other test

Follow the generic testing model from Module 2:

- Assemble the situation:
  - Set up system under test (SUT) to get the state ready
  - [Optional: Prepare collaborators]
- Act - Apply the operation inputs.
- Assess - Check the outputs, verify the state change, handle the behavior

1: Render component into a testing DOM tree

2: Interact with the rendered component

3: Check the rendered result

UI Testing Libraries make Component Tests Lightweight

- Render components into a "virtual DOM"
  - Just like browser would, but no browser
- Interact with components by "firing events" like a user would
  - Click, enter text, etc. on DOM nodes, just like a user would in a browser
- Inspect components that are rendered
  - Tests specify how to "find" a component in that virtual DOM

"Testing Library"
https://testing-library.com
Compatible with many UI libraries and many testing frameworks

Rendering Components in Virtual DOM

```
let deleteCalled = false;
beforeEach(() => {
  deleteCalled = false;
  render(
    <PersonalizedLikableDeletableHello name="Ripley"
      onDelete={() => { deleteCalled = true; }} /> );
});
```

- The *render* function prepares our component for testing:
  - Creates a virtual DOM
  - Instantiates our component, mounts it in DOM
  - Mocks all behavior of the core of React
  - Allows us to inspect the rendered result in the *screen* import

https://testing-library.com/docs/react-testing-library/api#render

Inspecting Rendered Components: By Text

SUT

```
return (
    <div>
      Hello, {props.name}! This is React! {likeButton}
      <IconButton aria-label='delete' icon={<AiTwotoneDelete />}
           onClick={props.onDelete} />
    </div>
);
```

Test

```
test("It renders the greeting", ()=>{
  const greeting = screen.getByText(/Hello, Ripley!/);
  expect(greeting).toBeInTheDocument();
})
```

First approach to inspect rendered components: match by text

Inspecting Rendered Components: ARIA label

SUT

```
if (isLiked) {
  likeButton = (<IconButton aria-label="unlike"
      icon={<AiFillHeart />} onClick={() => setIsLiked(false)} /> );
} else {
  likeButton = (<IconButton aria-label="like"
      icon={<AiOutlineHeart />} onClick={() => setIsLiked(true)} /> );
}
```

Test

```
test("Like button defaults to not liked, clicking it likes, clicking again
unlikes", () => {
  const likeButton = screen.getByLabelText("like");
  fireEvent.click(likeButton);
  const unLikeButton = screen.getByLabelText("unlike");
  fireEvent.click(unLikeButton);
  expect(screen.getByLabelText("like")).toBeInTheDocument();
});
```

3 Tiers for Inspecting Rendered Components

- Queries that reflect how every users interacts with your app
  - byRole – Using accessibility tree
  - byLabelText – Using label on form fields
  - byPlaceHolderText – Using placeholder text on form field
  - byText – By exact text in an element
  - byDisplayValue – By current value in a form field
- Queries that reflect how some users interact with your app
  - byAltText – By alt text, usually not presented to sighted users
  - byTitle - By a “title” attribute, usually not presented to sighted users
- Queries that have nothing to do with how a user interacts with app
  - byTestId

More: https://testing-library.com/docs/queries/about

Testing Library Cheat Sheet

	No Match	1 Match	1+ Match	Await?
getBy	throw	return	throw	No
findBy	throw	return	throw	Yes
queryBy	null	return	throw	No
getAllBy	throw	array	array	No
findAllBy	throw	array	array	Yes
queryAllBy	[]	array	array	No

- Get and query have different behavior when there are different numbers of matches
- Find is *async* and will return a promise to wait for all rendering to complete

https://testing-library.com/docs/react-testing-library/cheatsheet

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
  - Be able to map the three core steps of a test (construct, act, check) to UI component testing
- The next lesson will include a deep-dive on patterns of React, including useState and its friend, useEffect