

CS 4530: Fundamentals of Software Engineering

Lesson 6.2 Introduction to React

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand
Khoury College of Computer Sciences

Learning Goals

By the end of this lesson, you should...

- Be able to explain how component reuse simplifies application development
- Understand how the React framework binds data (and changes to it) to a UI

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?

The Guardian · Monday 12 September 2011

**Read more on
MediaGuardian.co.uk**

Digital economy or bust
Part 33. In which the team turn up the volume with made-to-order TV on The X Factor – and get a glimpse of the future.

Coming up this week
Monday: Shortlist for Student Media Awards announced
Wednesday to Friday: Coverage of the HTLs Cambridge Convention

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 Sweeney**

I **Rio Caraeff** succeeds, perhaps only the first time music video rights part owners for the two largest music companies in the world will have the same impact as MTV and its partners did in the 1980s. His thesis is with that of making the music industry relevant again, a classic of recordings, and the Caraeff thesis is that it can be done.

Caraeff is the youthful chief executive of Vevo, the joint venture of the four major groups, Sony Music, Universal Music, and EMI, who have joined forces to compete with a robust market in the US and elsewhere. The new service, which launches on 27 September, is the only entertainment company on the planet to have a global reach and audiences at the scale of billions of people. "It's a massive audience," says Caraeff, "2 billion people watching."

Is this the last year, Caraeff, you'll be involved in this? "I don't know. I'm not sure I didn't need to," he says. "But I'm fully committed to this. I'm fully committed to making sure that we're successful. If Vevo can do what we set out to do, then I think that the shareholders of Vevo will be happy. It would be irresponsible for me to say, 'We can't do this any longer'."

Vevo's relationship with Google, the owner of the world's largest search engine, is key to its success. Google has agreed to duplicate copies of music videos on YouTube, and the official versions are now available from us. They don't threaten us; they're not trying to compete with us. We're not trying to compete with them. All the traffic to Vevo's website comes from YouTube search, and Google is the only search engine that has videos that users might like to watch that appear on the side of the YouTube search results. So it's great to be working with them.

Free access

Vevo's business model is all about providing free access to music. You can access free, funded by advertising – or go to another way: give away the music for free. "Deliver the future is access, not ownership, not it's mine. As a society, we've got to move away from selling people music, our customers are not the ones that are going to pay for that music. We are about providing access, it is the only scalable model for the music industry. So, how do you do that and make money?"

That's a question of how well Vevo is actually doing. Caraeff doesn't seem to be worried about that financially but says it is already making "hundreds of millions of dollars" in revenue, though he declines to put a figure on it. More than half of gross revenue goes to content partners, and the rest is split between Vevo and "significantly ahead" of its original business plan – about six times what it was in "the very early part of next year".

He is clear that Vevo's future is not dependent on advertising, and he is thinking of ways to generate more revenue. His contention is that advertising won't work for Vevo because it's not clear what Vevo's role is in "the new" the prime concern will be how to monetise the new music product. Think the free-to-air television equivalent of BSkyB's own music channel.

"The audience that we want to satiate and keep growing: it should be treated as

Video creation... Rio Caraeff says 'if MTV was born in 2011'

If a user uploads a video to YouTube, YouTube's system automatically generates a thumbnail image and a transcript. It also provides a link to the video on the guardian.co.uk website, where it can be viewed in a larger frame. The video is also linked to other related content on the site.

Video creation... Rio Caraeff says 'if MTV was born in 2011'

Rio Caraeff: Vevo revolutionary
The former Universal mobile chief reveals his frustration with MTV and explains why no one need own music if his site succeeds

Mark Sweeney
Music editor, *The Guardian*
Sunday 11 September 2011 20:00 BST

Comments
26 Responses | Add your own

Topics
Music | Online TV | Digital industry | Technology | Rio Caraeff | Internet | More Interviews

More
See also
21 Aug 2011 Argos to make its own digital music service
26 Apr 2011 Chuck Ghosh: how to win that race of online TV
27 Aug 2010 YouTube: how to search
14 Jul 2011 Our music service: back to basics

Curriculum vitae

Age 36

Education [did not go to school] Started my first job at 12

Career

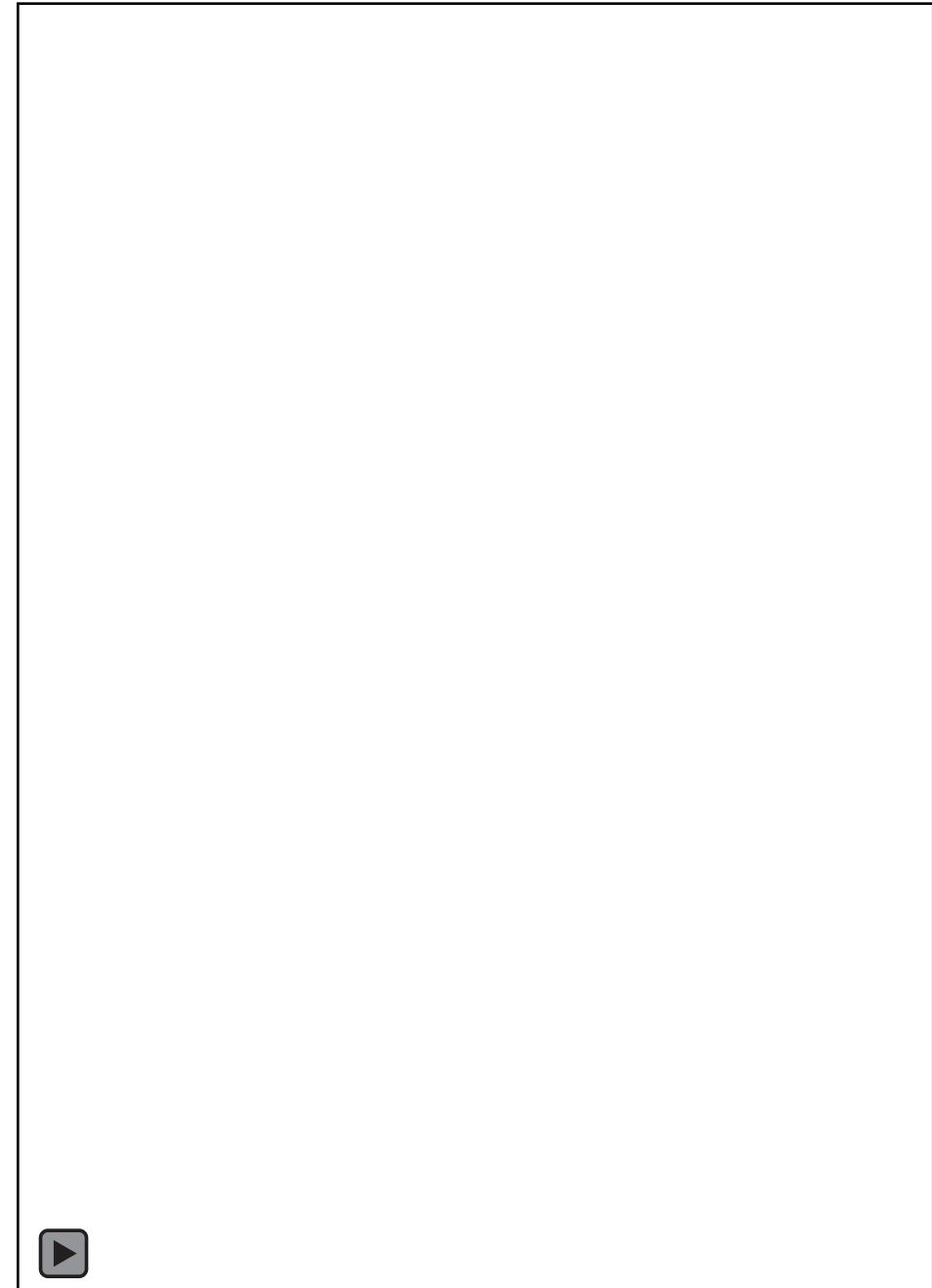
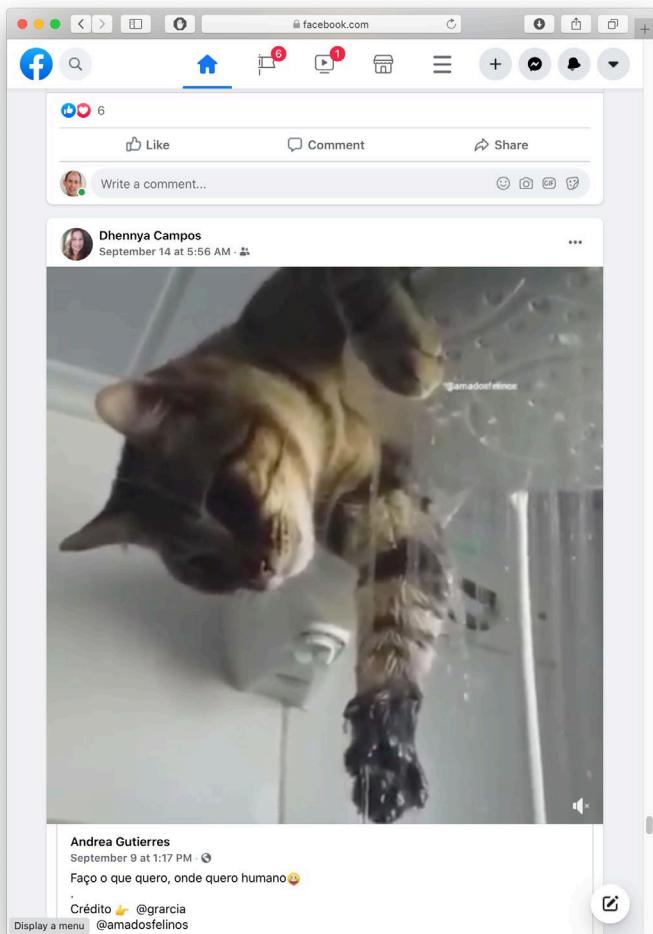
2004 Vice-president, marketing, Music Mail (a private-label, UK-based music retailer) and new technologies. Now

Caraeff is the youthful chief executive of Vevo – launched in late 2009

If MTV succeeds, perhaps only content fees will need to be paid to the video arm; music video site, part-owned by the two largest record companies, hopes to bring in a similar amount of revenue as MTV to be an alternative to YouTube. Chuck Ghosh goes in wait of that outcome, and to see how Vevo's new dependence on the purchase of recordings, and the fact that there is already plenty to do.

Rich, interactive web apps

Infinite scrolling of cats



Typical properties of web app Uis

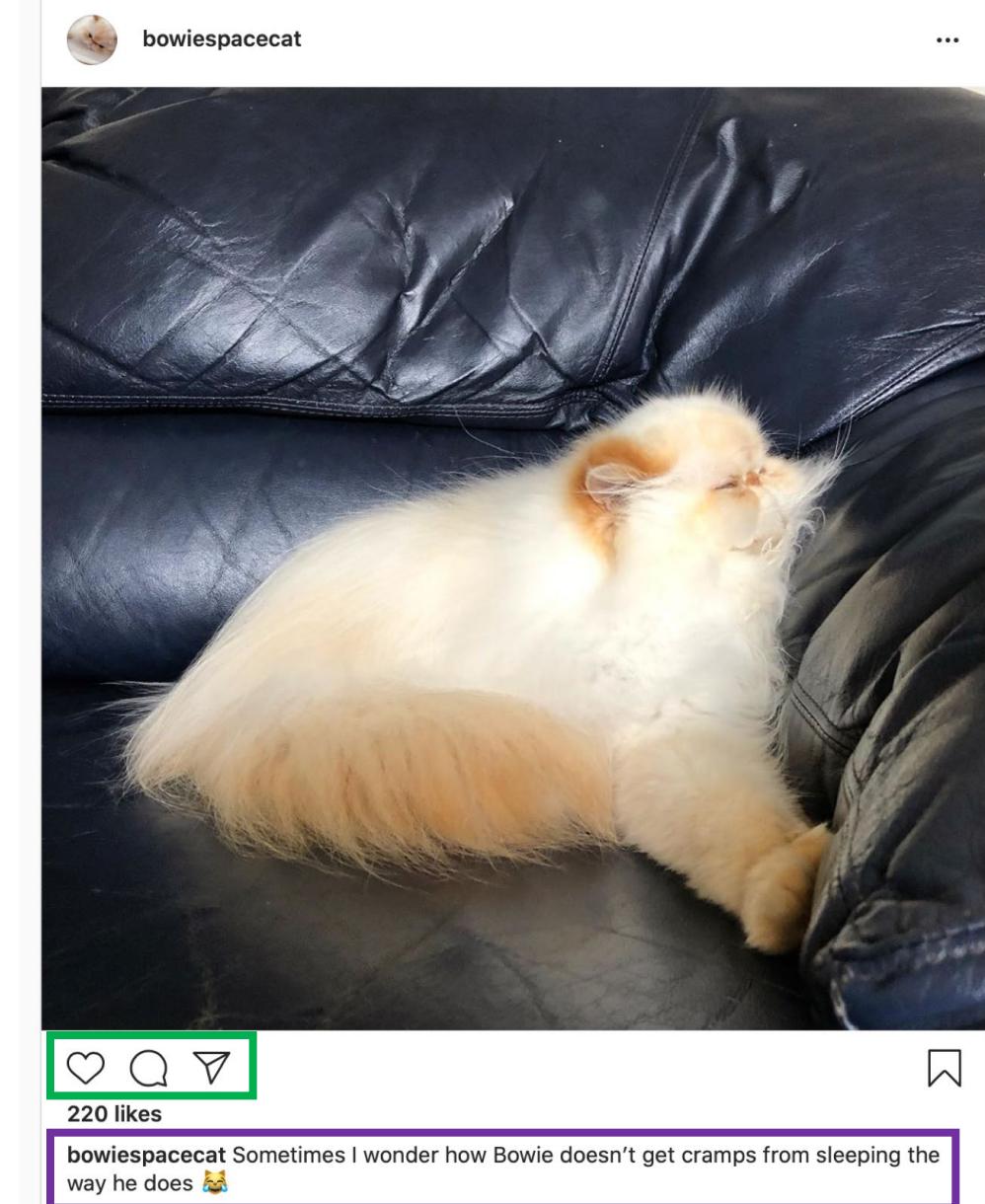
Building abstractions for web app development?

- 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



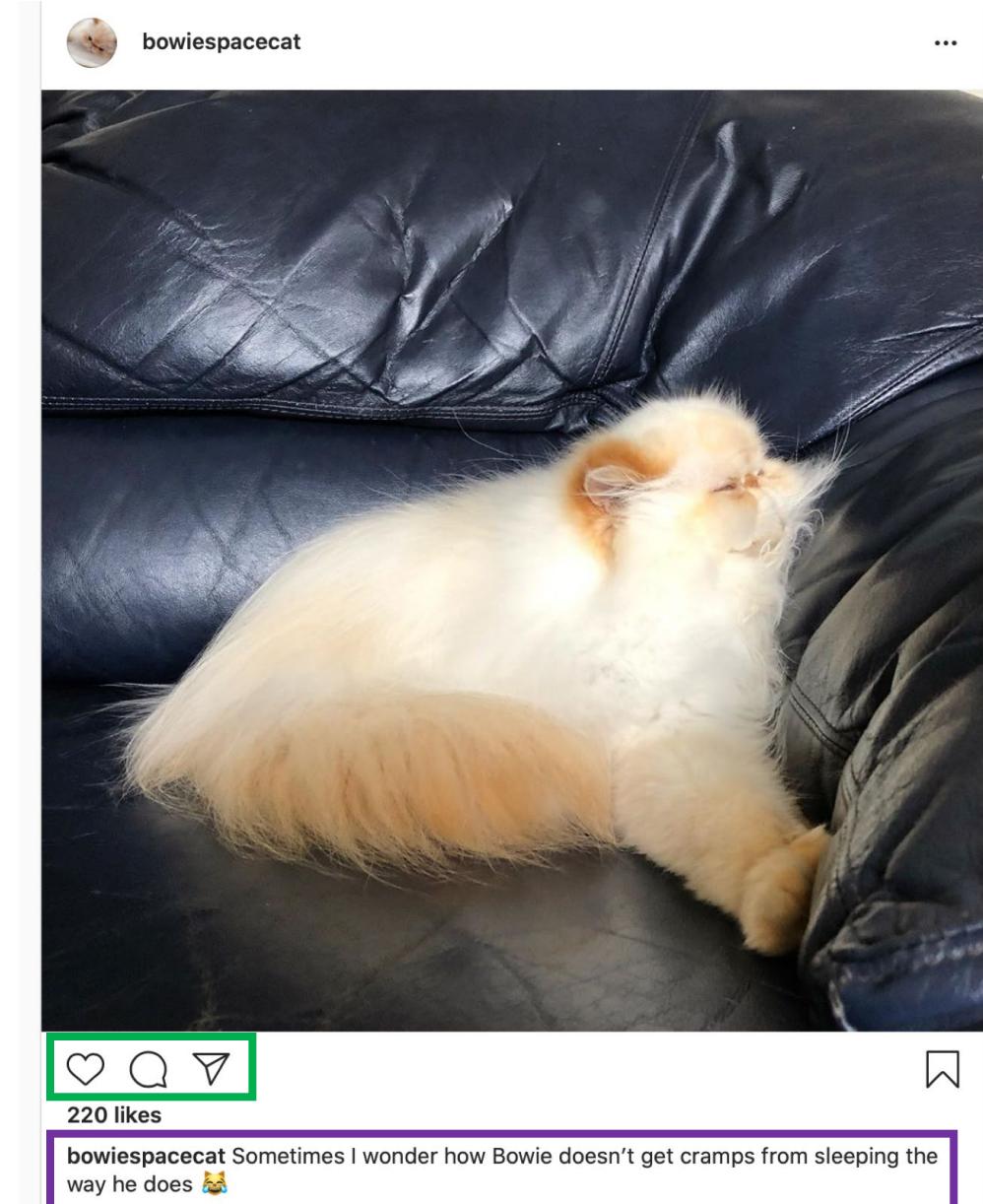
Key Idea: Components

- 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)



Components

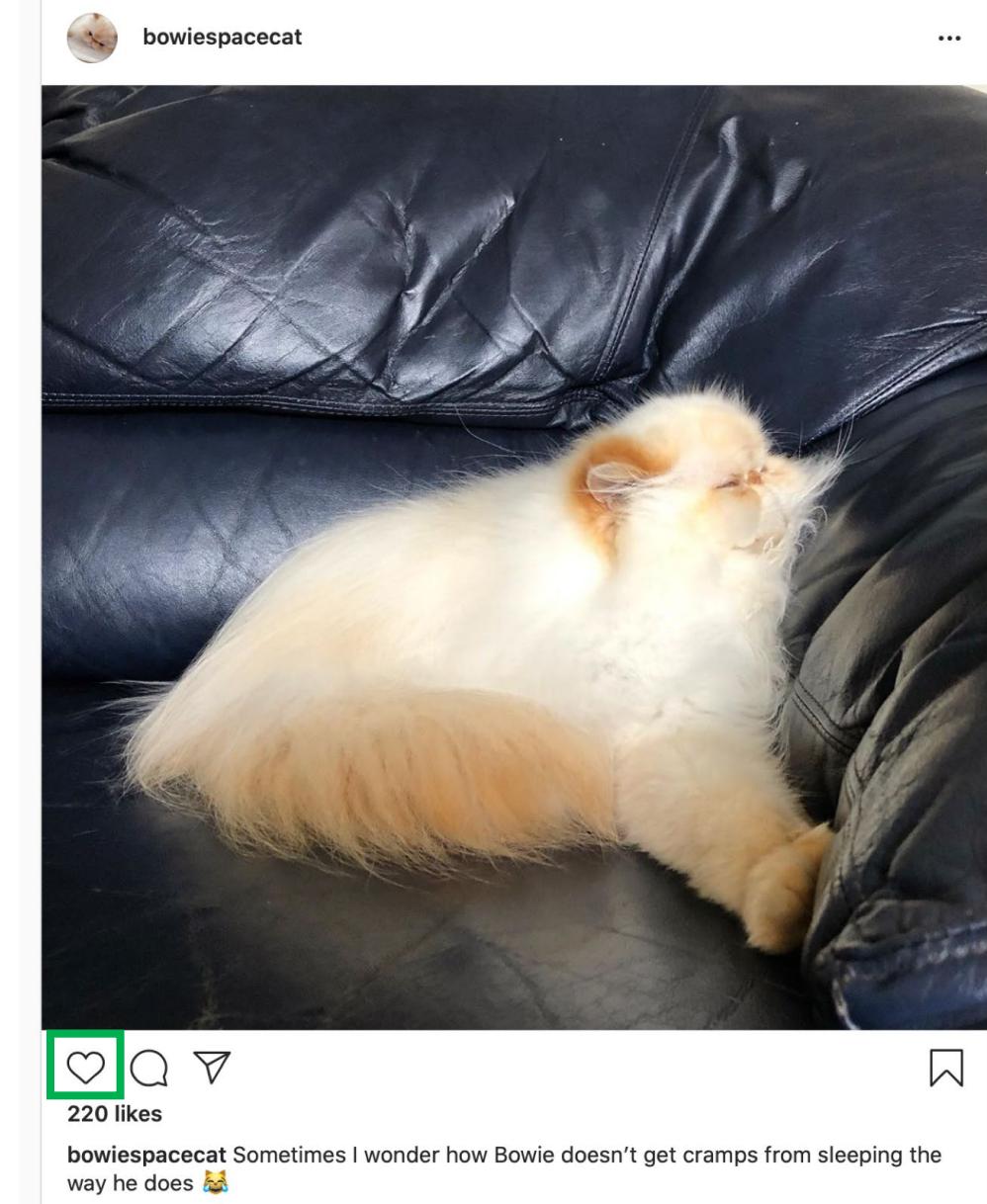
- Organize related logic and presentation into a single unit
 - Includes necessary state and the logic for updating this state
 - Includes presentation for rendering this state into HTML
- Synchronizes state and visual presentation
 - Whenever state changes, HTML should be rendered again



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

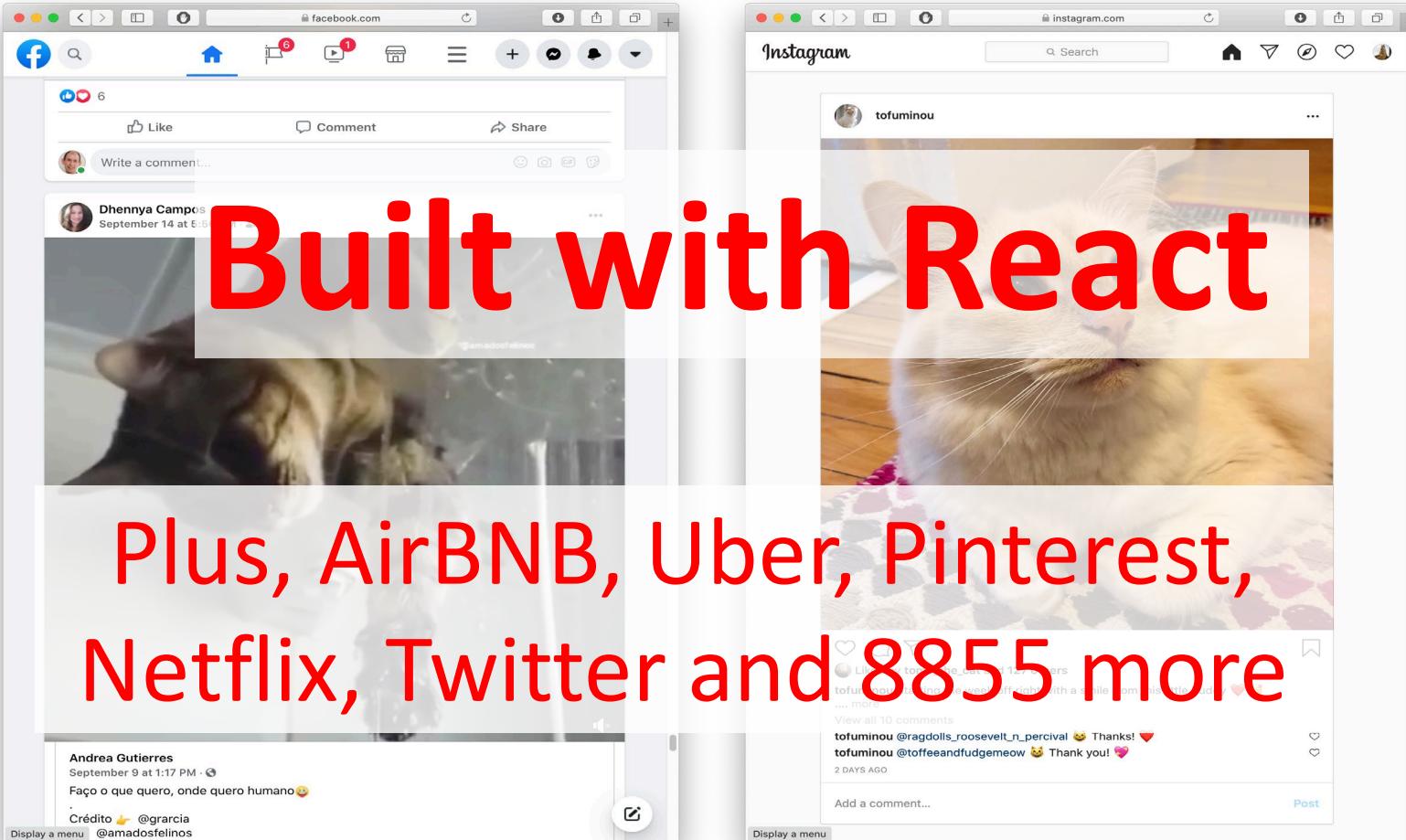
```
export function HelloMessage(props: IProps) {  
  return (  
    <div>  
      Hello, {props.name}  
    </div>  
  )  
}  
  
ReactDOM.render(  
  <React.StrictMode>  
    <HelloMessage name='Satya' />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

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



React Evolution

From classes to functional components

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

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

- Hooks were added to functional components in React 16.8.
- Recommended using functional components instead of class components.
- Will have more features added.
- Neither approach is wrong.

Embedding HTML in TypeScript

```
return <div>Hello {name}</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() }
- Output of expression is HTML

Example Component

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

“Return the following
HTML whenever the
component is rendered”

The HTML is dynamically
generated by the library.

“Declare a Hello
component”

Declares a new component to
which state and other
functionality can be added.

Properties vs. State

- Properties should be immutable.
 - Created through attributes when component is instantiated.
 - Should never update within component
 - Parent may create a new instance of component with new properties

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

```
<HelloMessage name='Satya' />
```

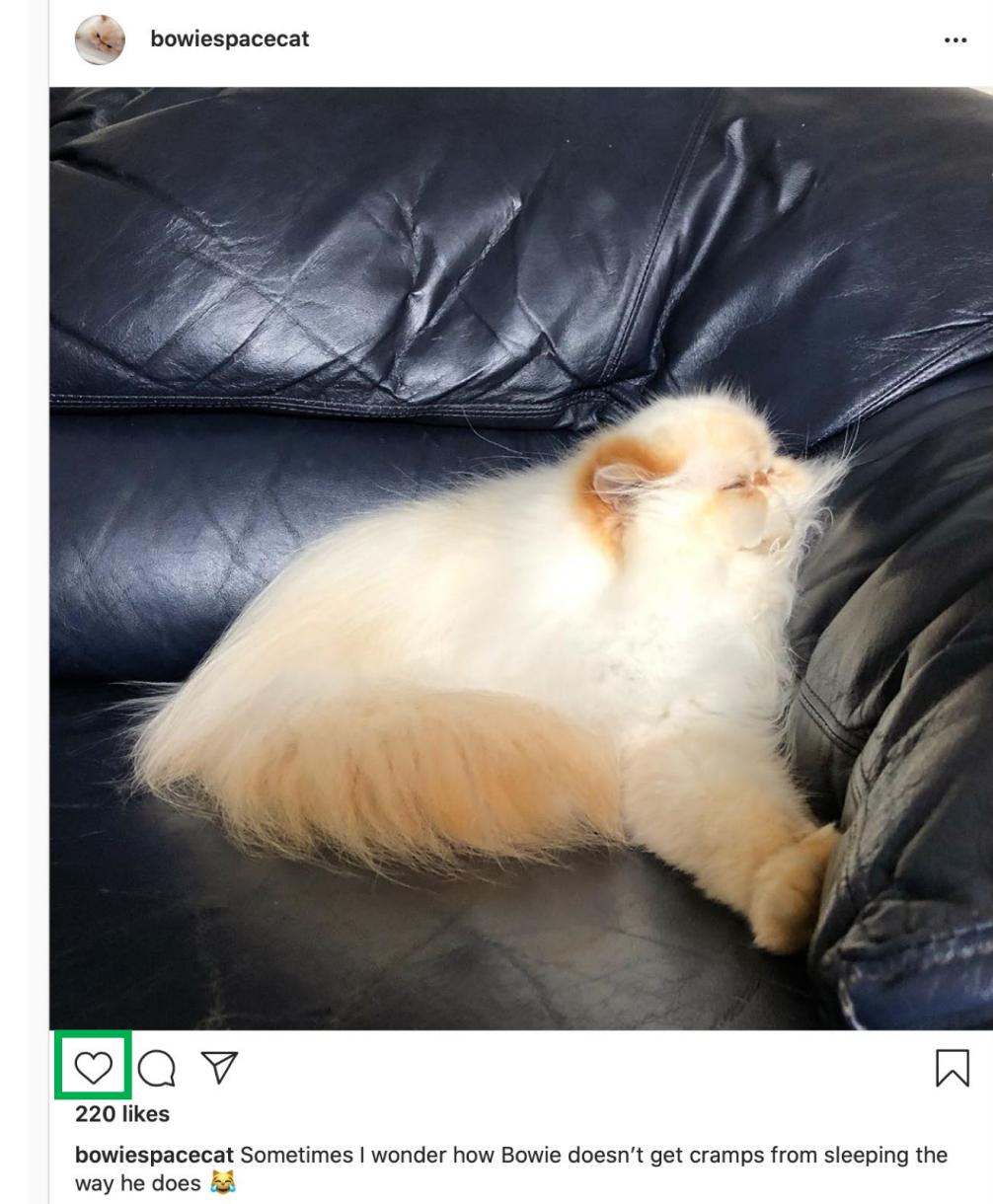
- State changes to reflect the current state of the component.
 - Can (and should) change based on the current internal data of your component.

Components

Example: Like button component

- What does the button keep track of?
 - Is it liked or not (state)
 - What post this is associated with (property)

```
if(state.isLiked){  
  return <HeartFilled onClick={toggleLike} />  
} else {  
  return <HeartOutlined onClick={toggleLike} />  
}
```



What is state?

- All internal component data that, when changed, should trigger UI update
 - Stored as state variables in the component
 - Created using useState(defaultValue)
 - E.g. let [state, setState] = useState({});
 - Only can set directly before a component is created (in useState()). Otherwise must call setState()
- Import useState from react

```
import { useState } from 'react';
```

Reacting to change

How does the page update automatically?

- Your code updates the state of component when event(s) occur (e.g., user enters data, get data from network)
- Updating state causes the html to be re-rendered by the framework (must call setter, not update variable directly)
- Reconciliation: Framework diffs the previously rendered DOM with the new DOM, updating only part of DOM that changed

Working with state

- useState() should initialize state of object inside component

```
let [date, setDate] = useState(new Date());
```

- Use setState to update state (setDate in example)

```
setDate(new Date());
```

- Doing this will (asynchronously) eventually result in render being invoked
- Multiple state updates can be automatically batched together and result in a single render call

Nesting components

```
return (  
  <div>  
    <LikeButton post={post} />  
    <CommentButton post={post} />  
  </div>  
)
```

Establishes ownership by creating in returned template

Sets **post** property of child to value of **post** property of parent

The data flows down

- State that is common to multiple components should be owned by a common ancestor
 - State can be passed into descendants as properties
- When this state can be manipulated by descendants (e.g., a control), change events should invoke a handler on common ancestor
- Handler function should be passed to descendants

The data flows down

```
export function Counter() {  
  
  let [count, setCount] = useState(0);  
  
  function incrementCount() {  
    setCount(count + 1);  
  }  
  
  return (  
    <div>  
      <Display count={count} />  
      <Button incrementCount={incrementCount} />  
    </div>  
  );  
}
```

```
export function Display(props: any) {  
  return (  
    <h1>Count: {props.count}</h1>  
  )  
}  
  
export function Button(props: any) {  
  return (  
    <button onClick={props.incrementCount}>  
      Increment Count  
    </button>  
  )  
}
```

Component Lifecycle

- Traditionally, the React Component Lifecycle consists of 3 phases
 - Mounting: When a component first loads
 - componentDidMount()
 - Updating: When the component is updated
 - componentDidUpdate()
 - Unmounting: When the component is about to be removed
 - componentWillUnmount()
- In functional components, these are replaced by hooks.
 - Specifically, the useEffect() hook, imported from react

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

Working with Hooks

Self incrementing timer

```
export function Timer() {  
  let [seconds, setSeconds] = useState(0);  
  
  function tick() {  
    setSeconds((nrSeconds) => nrSeconds + 1);  
  }  
  
  // Some magic to make it work.  
  
  return (  
    <div>  
      | Seconds: {seconds}  
    </div>  
  );  
}
```

Working with Hooks

Self incrementing timer

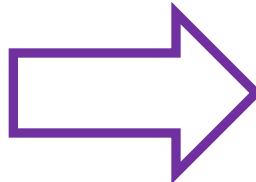
```
useEffect(() => {
  // set interval when component loads.
  let interval: NodeJS.Timeout = setInterval(tick, 1000);
  return () => {
    // clear interval when component is about to be removed.
    clearInterval(interval as NodeJS.Timeout);
  }
}, []); // Empty array to prevent execution when state is updated.

useEffect(() => {
  // Executes every time seconds is updated.
  console.log(seconds);
}, [seconds]); // will only run when seconds is updated.
```

Reconciliation

Efficiently updating browser's view of the app

```
<Card>
  <p> Paragraph 1 </p>
  <p> Paragraph 2 </p>
</Card>
```



```
<Card>
  <p> Paragraph 2 </p>
</Card>
```

- Process by which React updates the DOM with each new render pass
- Occurs based on order of components
 - Second child of Card is destroyed.
 - First child of Card has text mutated.

Reconciliation with Keys

- Problem: what if children are dynamically generated and have their own state that must be persisted across render passes?
 - Don't want children to be randomly transformed into other child with different state
- Solution: give children identity using keys
 - Children with keys will always keep identity, as updates will reorder them or destroy them if gone

Reconciliation with Keys

```
export function NumberList(props: any) {
  const numbers = props.numbers;
  const listItems = numbers.map((number: any) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Summary - React

- Component-based framework
- Automatically re-render components based on changes to data
- Maps each component to some HTML elements and efficiently updates them