Templates, Databinding and HTML

SWE 432, Fall 2018

Web Application Development

Review: What forms of data might you have

- Key / value pairs
- JSON objects
- Tabular arrays of data
- Files

Review: Options for backend persistence

- Where it is stored
  - On your server or another server you own
    - SQL databases, NoSQL databases
    - File system
  - Storage provider (not on a server you own)
    - NoSQL databases
    - BLOB store

Today

- Storing files
- HTML
- Intro to templating
- Reminder: HW2 due in one week!

Blobs: Storing uploaded files

- Example: User uploads picture
  - … and then?
  - … somehow process the file?

How do we store our files?

- Dealing with text is easy - we already figured out firebase
    - Could use other databases too… but that's another class!
- But
    - What about pictures?
    - What about movies?
    - What about big huge text files?
- Aka…Binary Large OBject (BLOB)
    - Collection of binary data stored as a single entity
    - Generic terms for an entity that is array of bytes

Working with Blobs

- Module: multer

- Simplest case: take a file, save it on the server

```
app.post('/upload',upload.single("upload"), function(req, res) {
        var sampleFile = req.file.filename;
      //sampleFile is the name of the file that now is living on our server
        res.send('File uploaded!');
    });
});
```

- Long story... can't easily have file uploads and JSON requests at the same time

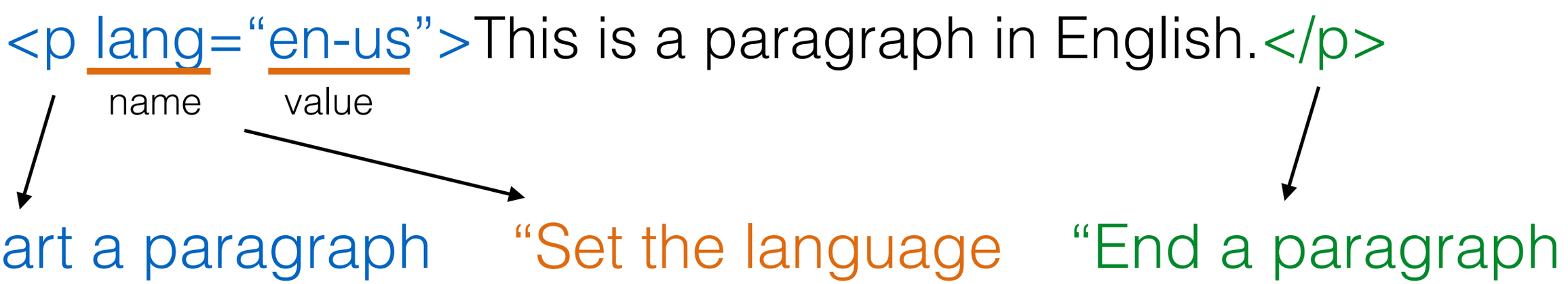
HTML: HyperText Markup Language

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

HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866
- 1997: HTML 4.0 Standardized most modern HTML element w/ W3C recommendation
    - Encouraged use of CSS for styling elements over HTML attributes
- 2000: XHTML 1.0
    - Imposed stricter rules on HTML format
        - e.g., elements needed closing tag, attribute names in lowercase
- 2014: HTML5 published as W3C recommendation
    - New features for capturing more *semantic* information and *declarative* description of behavior
        - e.g., Input constraints
        - e.g., New tags that explain *purpose* of content
    - Important changes to DOM (will see these later….)

HTML Elements



Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

HTML attributes are name / value pairs that provide additional information about the contents of an element.

Closing tag ends an HTML element. All content between the tags and the tags themselves compromise an HTML element.

HTML Elements

<input type="text" />

"Begin and end input element"

Some HTML tags can be self closing, including a built-in closing tag.

<!-- This is a comment. Comments can be multiline. -->

A starter HTML document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

"Title"

Used by browser for title bar or tab.

"Document content"

HTML Example

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="main.css">
    <title>Prof Bell's Webpage</title>
</head>
<body>
<h1>
    Prof Jonathan Bell
</h1>
```

Use <h1>, <h2>, …, <h5> for headings

```
    <h2>Welcome, students!</h2>
        <p>
            <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">See how to make
this page</a>
        </p>
    <h2>
        Some funny links
    </h2>
    <p>
        <ul>
            <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
            <li><a
href="http://www.wb3w.net/The%20Original%20Hamsterdance.htm">Hamster Dance</a></li>
        </ul>
    </p>
    <h3>
        About Prof Bell
    </h3>
    <p>
        Prof Bell's office is at 4422 Engineering Building. His email address is <a
href="mailto:bellj@gmu.edu">bellj@gmu.edu</a>.
    </p>
    <p>
        Last updated: September 4th, 1999
    </p>
</div>
</body>
</html>
```

https://seecode.run/#-KQgR7vG9Ds7IUJS1kdq

HTML Example

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="main.css">
    <title>Prof Bell's Webpage</title>
</head>
<body>
<h1>
    Prof Jonathan Bell
</h1>
<div>
    <p>
        <img alt="My really cool laptop" src="http://www.wonder-
tonic.com/geocitiesizer/images/laptop-01.gif" /> <br />
        <div class="marquee">
            This is Prof Bell's ACTUAL homepage from 1999!
        </div>
    </p>
    <h2>Welcome, students!</h2>
        <p>
            <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">See how to make
this page</a>
        </p>
    <h2>
        Some funny links
    </h2>
    <p>
        <ul>
            <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
            <li><a
href="http://www.wb3w.net/The%20Original%20Hamsterdance.htm">Hamster Dance</a></li>
        </ul>
    </p>
    <h3>
        About Prof Bell
    </h3>
    <p>
        Prof Bell's office is at 4422 Engineering Building. His email address is <a
href="mailto:bellj@gmu.edu">bellj@gmu.edu</a>.
    </p>
```

Paragraphs (<p>) consist of related content. By default, each paragraph starts on a new line.

n/#-KQgR7vG9Ds7IUJS1kdq

HTML Example

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="main.css">
    <title>Prof Bell's Webpage</title>
</head>
<body>
<h1>
    Prof Jonathan Bell
</h1>
<div>
    <p>
        <img alt="My really cool laptop" src="http://www.wonder-
tonic.com/geocitiesizer/images/laptop-01.gif" /> <br />
        <div class="marquee">
            This is Prof Bell's ACTUAL homepage from 1999!
        </div>
    </p>
    <h2>Welcome, students!</h2>
        <p>
            <a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">See how to make
this page</a>
        </p>
    <h2>
        Some funny links
    </h2>
    <p>
        <ul>
            <li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
            <li><a
href="http://www.wb3w.net/The%20Original%20Hamsterdance.htm">Hamster Dance</a></li>
        </ul>
hr
    </p>
</div>
</body>
</html>
```

Unordered lists (<ul>) consist of list items (<li>) that each start on a new line. Lists can be nested arbitrarily deep.

https://seecode.run/#-KQgR7vG9Ds7IUJS1kdq

Text

```
<h1>Level 1 Heading</h1>
<h2>Level 2 Heading</h2>
<h3>Level 3 Heading</h3>
<h4>Level 4 Heading</h4>
<h5>Level 5 Heading</h5>
<h6>Level 5 Heading</h6>
Text can be made <b>bold</b> and
    <i>italic</i>, or <sup>super</sup>
    and <sub>sub</sub>scripts. White
    space collapsing removes all
    sequences of two more more spaces
    and line breaks, allowing
    the markup to use tabs
    and whitespace for
    organization.
    Spaces can be added with
        &amp;nbsp;.
<br/>New lines can be added with &lt
    ;BR/&gt;.

<p>A paragraph conssists of one or
    more sentences that form a self
    -contained unit of discourse. By
    default, a browser will show each
    paragraph on a new line.</p>

<hr/>
Text can also be offest with
horizontal rules.
```

<h1>Level 1 Heading</h1>
<h2>Level 2 Heading</h2>
<h3>Level 3 Heading</h3>
<h4>Level 4 Heading</h4>
<h5>Level 5 Heading</h5>
<h6>Level 5 Heading</h6>

Text can be made **bold** and *italic*, or ^{super} and $_{sub}$scripts.
White space collapsing removes all sequences of two more more spaces and line breaks, allowing the markup to use tabs and whitespace for organization. Spaces can be added with .
New lines can be added with
.

A paragraph conssists of one or more sentences that form a self-contained unit of discourse. By default, a browser will show each paragraph on a new line.

Text can also be offest with horizontal rules.

Semantic markup

- Tags that can be used to denote the *meaning* of specific content
- Examples
  - An element that has importance.
  - <blockquote> An element that is a longer quote.
  - <q> A shorter quote inline in paragraph.
  - <abbr> Abbreviation
  - <cite> Reference to a work.
  - <dfn> The definition of a term.
  - <address> Contact information.
  - <ins><del> Content that was inserted or deleted.
  - <s> Something that is no longer accurate.

Links

```
<a href="http://www.google.com">Absolute link</a><br/>
<a href="movies.html">Relative URL</a><br/>
<a href="mailto:tlatoza@gmu.edu">Email Prof. LaToza</a><br/>
<a href="http://www.google.com" target="_blank">Opens in new
    window</a><br/>
<a href="#idName">Navigate to HTML element idName</a>
```

Absolute link
Relative URL
Email Prof. LaToza
Opens in new window
Navigate to HTML element idName

Controls

```
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
    <input type="checkbox" checked="checked" /> Checked
    <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
    <select>
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>Multiple Select Box:
    <select multiple="multiple">
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatoza
    /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
    value="30" /></p>
```

Search input provides clear button

Text Input:

Password Input:

Search Input:

Text Area: Initial text

Checkbox: Checked Unchecked

Drop Down List Box: Option2

Multiple Select Box: Option1 Option2

File Input Box: Choose File No file chosen

Image Button:

Button: Button

Range Input:

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

```
<h1>Hiroshi Sugimoto</h1>
<p>The dates for the ORIGIN OF ART exhbibition are as
    follows:</p>
<ul>
    <li>Science: 21 Nov- 20 Feb 2010/2011</li>
    <li>Architecture: 6 Mar - 15 May 2011</li>
</ul>
```

Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhbibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

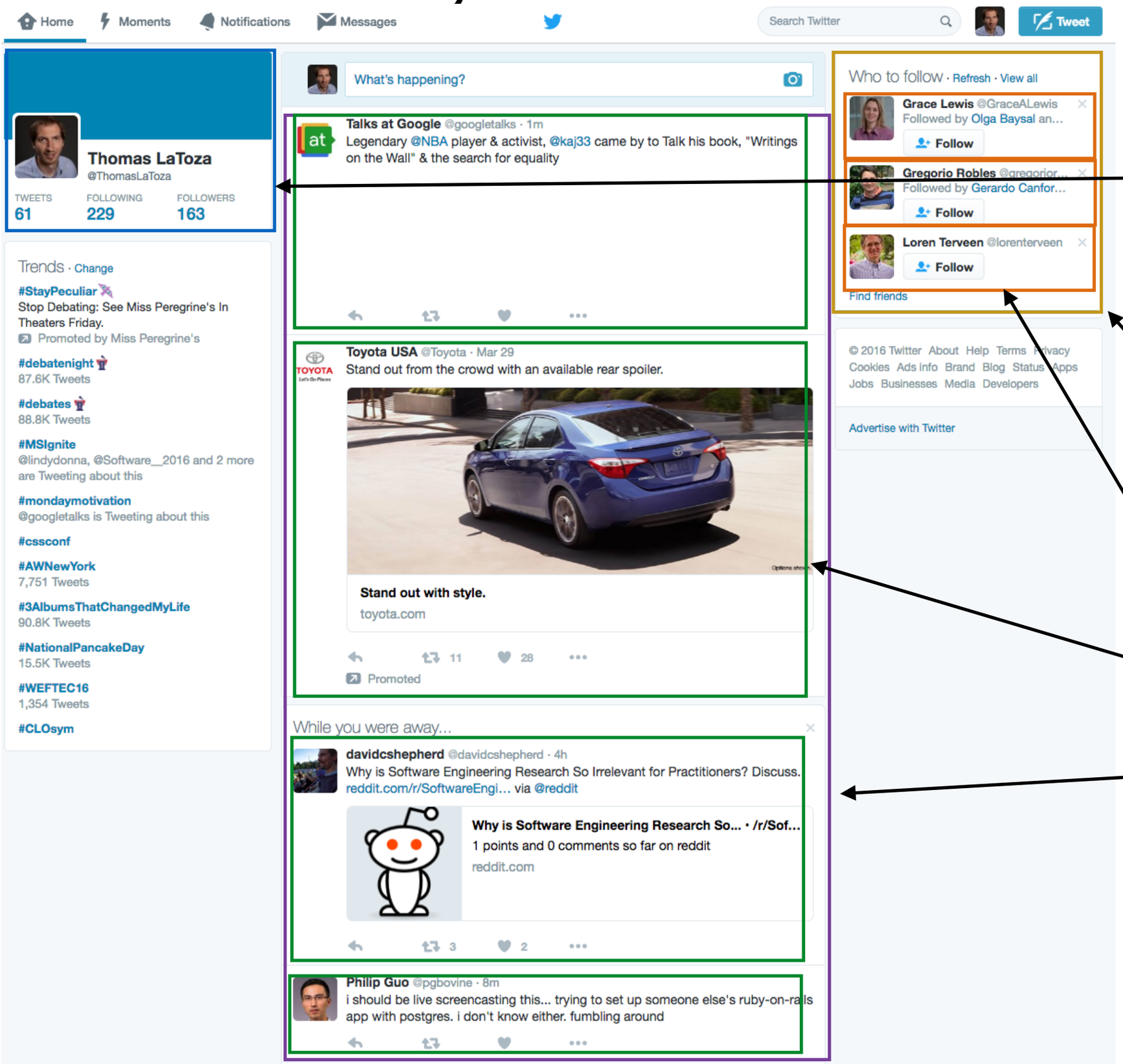
Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>

```
Timed to a single revolution of the planet around the sun
at a 23.4 degrees tilt that plays out the rhythm of the
seasons, this <em>Origins of Art</em> cycle is organized
around four themes: <b>science, architecture, history</b>,
and <b>relgion</b>.
```

Timed to a single revolution of the planet around the sun at a 23.4 degrees tilt that plays out the rhythm of the seasons, this *Origins of Art* cycle is organized around four themes: **science, architecture, history**, and **relgion**.

Anatomy of a Non-Trivial Web App



User profile widget

Who to follow widget

Follow widget

Feed widget

Feed item widget

Typical properties of web app UIs

- Each widget has both visual presentation & logic
  - e.g., clicking on follow 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., Follow widget occurs multiple times in Who to Follow Widget
  - Need to generate a copy of widget based on data
- Changes to data should cause changes to widget
  - e.g., following person should update UI to show that the person is followed. Should work even if person becomes followed through other UI
- Widgets are hierarchical, with parent and child
  - Seen this already with container elements in HTML…

Idea 1: Templates

```
document.getElementById('todoItems').innerHTML +=
        '<div class="todoItem" data-index="' + key
        + '"><input type="text" onchange="itemChanged(this)" value="'
+ value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>';
```

- Templates describe repeated HTML through a single common representation
- May have variables that describe variations in the template
- May have logic that describes what values are used or when to instantiate template
- Template may be instantiated by binding variables to values, creating HTML that can be used to update DOM

Templates with template literals

```
document.getElementById('todoItems').innerHTML +=
      `<div class="todoItem" data-index="${key}">
          <input type="text" onchange="itemChanged(this)" value="${value}">
          <button onclick="deleteItem(this.parentElement)">&#x2716;</button>
       </div>`;
```

- Template literals reduce confusion of nested strings

Server side vs. client side

- Where should template 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

```
<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>
  <%
    }
  %>
```

- *Front-end* framework: Template runs in web browser
  - Examples: React, Angular, Meteor, Ember, Aurelia, …
  - Server passes template to browser, browser generates HTML on demand

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

Server side vs. client side

- Server side
  - Oldest solution.
  - True when “real” code ran on server, Javascript
- Client side
  - Enables presentation logic to exist entirely in browser
  - e.g., can make call to remote web service, no need for server to be involved
  - (What we are looking at in this course).

Logic

- Templates 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)

Embed code in HTML

```
<cfcomponent name = "PersonalChef">
    <cffunction name = "makeToast" returnType = "component">
        <cfargument name = "color" required="yes">

        <cfset this.makeToast = "Making your toast #arguments.color#!" />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```

```
<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>
  <%
    }
  %>
```

- Template takes the form of an HTML file, with extensions
  - Custom tags (e.g., <% %>) enable logic to be embedded in HTML
  - Uses another language (e.g., Java, C) or custom language to express logic
  - Found in frameworks such as PHP, Angular, ColdFusion, ASP, ...

Embed HTML in code

- Template takes the form of an HTML fragment, embedded in a code file

    - HTML instantiated as part of an expression, becomes a value that can be stored to variables

    - Uses another language (e.g., Javascript) to express logic

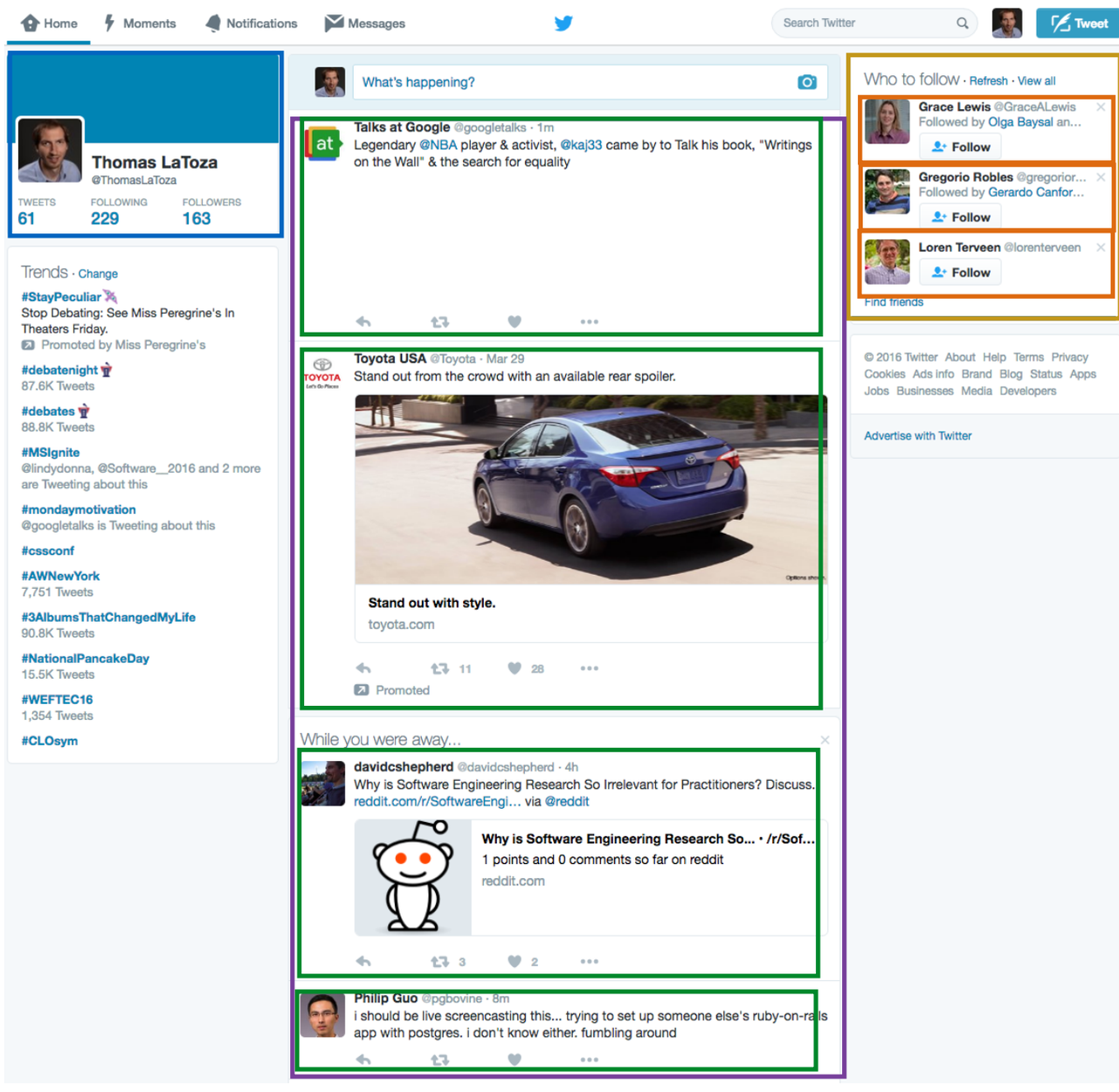
    - This course: **React**

Templates enable HTML to be rendered multiple times

- Rendering takes a template, instantiates the template, outputs HTML
- Logic determines which part(s) of templates are rendered
- Expressions are evaluated to instantiate values
    - e.g., { this.props.name }
    - Different variable values ==> different HTML output

Idea 2: Components

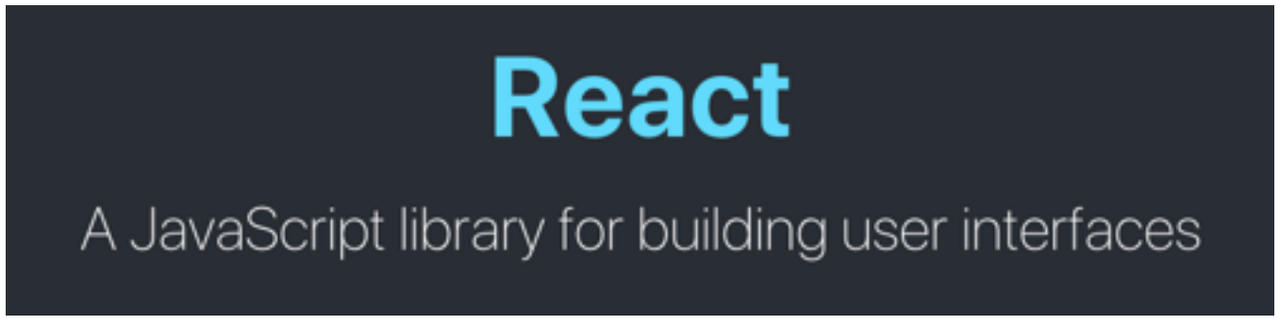
- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
  - Templates that correspond to a specific widget
  - Encapsulates related logic & presentation using language construct (e.g., class)



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
  - Outside world *must* interact with state through accessors, enabling access to be controlled
- Synchronizes state and visual presentation
  - Whenever state changes, HTML should be rendered again
- Components instantiated through custom HTML tag

React: Front End Framework for Components



- Originally build by Facebook
- Opensource frontend framework
- Powerful abstractions for describing frontend UI components
- Official documentation & tutorials
  - https://reactjs.org/

Example

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

ReactDOM.render(
  <HelloMessage/>, mountNode
);
```

"Declare a HelloMessage component"

Declares a new component with the provided functions.

"Return the following HTML whenever the component is rendered"

Render generates the HTML for the component. The HTML is dynamically generated by the library.

"Render HelloMessage and insert in mountNode"

Instantiates component, replaces mountNode innerHTML with rendered HTML. Second parameter should always be a DOM element.

Example - Properties

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

“Read this.props.name and output the value”

Evaluates the expression to a value.

“Set the name property of HelloMessage to John”

Components have a this.props collection that contains a set of properties instantiated for each component.

Embedding HTML in Javascript

- HTML embedded in JavaScript

```
return <div>Hello {this.props.name}</div>;
```

  - HTML can be used as an expression

  - HTML is checked for correct syntax

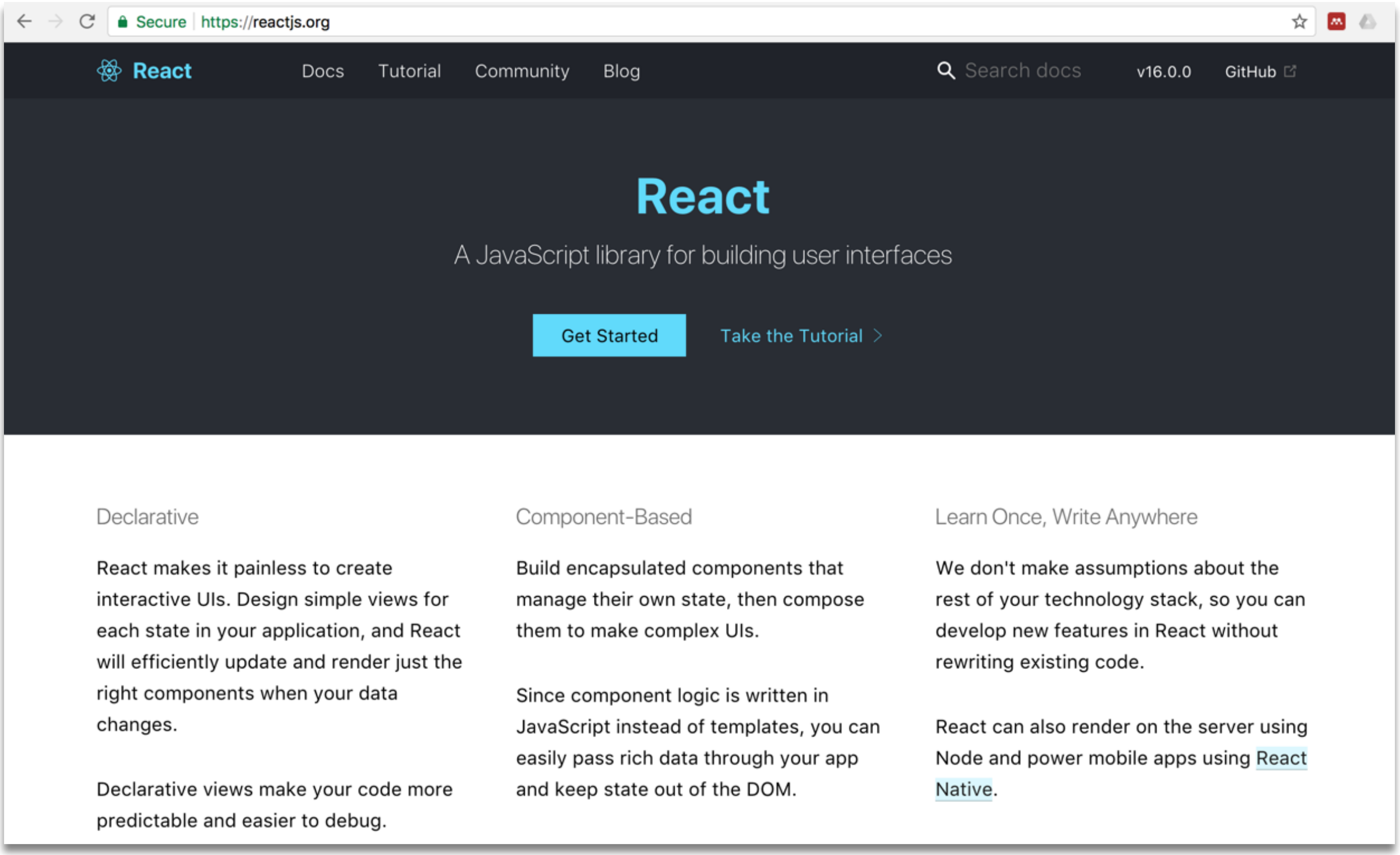
  - Can use { expr } to evaluate an expression and return a value

    - e.g., { 5 + 2 }, { foo() }

- Output of expression is HTML

JSX

- How do you embed HTML in JavaScript and get syntax checking??
- Idea: extend the language: JSX
  - Javascript language, with additional feature that expressions may be HTML
  - Can be used with ES6 or traditional JS (ES5)
- It's a new language
  - Browsers *do not* natively run JSX
  - If you include a JSX file as source, you will get an error


Secure https://reactjs.org
React
Docs
Tutorial
Community
Blog
Search docs
v16.0.0
GitHub
React
A JavaScript library for building user interfaces
Get Started
Take the Tutorial
Declarative
React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.
Declarative views make your code more predictable and easier to debug.
Component-Based
Build encapsulated components that manage their own state, then compose them to make complex UIs.
Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.
Learn Once, Write Anywhere
We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.
React can also render on the server using Node and power mobile apps using React Native.

Secure | https://codepen.io/gaearon/pen/gWWZgR?editors=1010

Tic Tac Toe

A PEN BY Dan Abramov PRO

Fork | Settings | Change View

HTML

```
<div id="errors" style="
  background: #c00;
  color: #fff;
  display: none;
  margin: -20px -20px 20px;
  padding: 20px;
  white-space: pre-wrap;
"></div>
<div id="root"></div>
<script>
window.addEventListener('mousedown',
function(e) {
  document.body.classList.add('mouse-
navigation');
  document.body.classList.remove('kbd-
navigation');
});
window.addEventListener('keydown',
function(e) {
  if (e.keyCode === 9) {
    document.body.classList.add('kbd-
```

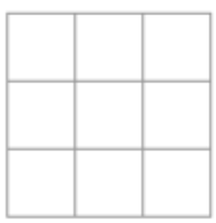
CSS

JS (Babel)

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}

class Board extends React.Component {
  renderSquare(i) {
    return (
      <Square
        value={this.props.squares[i]}
        onClick={() => this.props.onClick(i)}
      />
    );
  }

  render() {
    return (
      <div>
```



Next player: X

1. Go to game start

- Pastebin sites such as JSFiddle and codepen.io work with React
- Just need to include React first

Create React App

```
npx create-react-app my-app
cd my-app
npm start
```

(npx comes with npm 5.2+ and higher, see instructions for older npm versions)

Then open http://localhost:3000/ to see your app.
When you're ready to deploy to production, create a minified bundle with `npm run build`.

```
cd my-app
yarn start

Happy hacking!
λ cd my-app
λ npm start

> my-app@0.1.0 start ~/my-app
> react-scripts start

Starting the development server...

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.37.106:3000/
```

https://github.com/facebook/create-react-app