Today’s Agenda

• Quicktest
• Web Components
• Roundup Quiz
It’s Quicktest time!
Imperative vs. Declarative

Imperative: Specify *how* to do something
Declarative: Specify *what* should be done

Other definition: a programming paradigm that expresses the logic of a computation without describing its control flow

Often (not always), you’ll find these concepts alongside declarative programming
- Functional programming
- Reactive programming
- Databinding

http://latentflip.com/imperative-vs-declarative/
Rendering

• Model
  Data (e.g. todo-list items)

• View
  Graphical Interface (e.g. webpage)

• Controller
  Defines behavior (e.g. events)

• Often imperative

Adapted from Jake Archibald’s Talk: In The Loop
Breakout: Code-Along MVC

• Let’s create a UI that follows an MVC pattern.
• Using TypeScript
• Imperative Programming Style.
• use the breakout skeleton from GitHub
Notes on the Breakout

• The MVC Pattern makes sense for bigger applications (not so much for our example).
  • State managers are a viable solution
  • “Data binding” can turn MVC into a MVVM pattern (Model View ViewModel)

• Even for such a small example, the amount of code is vast.
  • we need something slimmer and directly understandable
Goal: Reusable, Declarative Components

```html
<body>
  <header><h1>Counter MVC</h1></header>
  <my-counter></my-counter>
</body>
```
Web Components
Web Components

• Approach to more declarative web programming style.

• Goal: re-use “things”, that we would have to write over and over (reducing boilerplate code on the web)

• Driven by Google, Mozilla, Vaadin and many more.

• Concepts:
  • Custom Elements
  • HTML Imports
  • Templates
  • Shadow DOM

Web Components

Custom Elements

<my-element></my-element>

HTML Templates

<template>
  This content is rendered later on.
</template>

HTML Imports

<link rel="import" href="my-code.html">

Shadow DOM

https://mdn.mozillademos.org/files/15788/shadow-dom.png
Web Components and Custom Elements

• Advantages:
  • More declarative approach than using nested classes etc.
  • Encapsulation
  • Separation of concerns
  • Readable
Without Custom Elements

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Memes</title>
  <style>
  ... 
  </style>
</head>
<body>
<div class="meme">
  <p class="caption captionTop">CALLBACKS</p>
  <p class="caption captionBottom">CALLBACKS EVERYWHERE</p>
</div>
<div class="meme">
  <p class="caption captionTop">ONE DOES NOT SIMPLY</p>
  <p class="caption captionBottom">WRITE BEAUTIFUL JAVASCRIPT</p>
</div>
<div class="meme">
  <img src="https://imgflip.com/s/meme/Spiderman-Peter-Parker.jpg"/>
  <p class="caption captionTop">IT WILL BE FUN</p>
  <p class="caption captionBottom">I PROMISE</p>
</div>
</body>
</html>
meme-component/my-memes.html
With Custom Elements

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>My Memes</title>

  <!-- Imports polyfill -->
  <script src="webcomponents.min.js"></script>

  <!-- Imports custom element -->
  <link rel="import" href="my-meme.html">
</head>
<body>

  <!-- Runs custom element -->
  <my-meme captionTop="JAVASCRIPT" captionBottom="JAVASCRIPT EVERYWHERE" memeTemplateUrl="https://imgflip.com/s/meme/X-X-Everywhere.jpg"></my-meme>

  <my-meme captionTop="ONE DOES NOT SIMPLY" captionBottom="WRITE BEAUTIFUL JAVASCRIPT CODE" memeTemplateUrl="https://imgflip.com/s/meme/One-Does-Not-Simply.jpg"></my-meme>

  <my-meme captionTop="IT WILL BE FUN" captionBottom="I PROMISE" memeTemplateUrl="https://imgflip.com/s/meme/Spiderman-Peter-Parker.jpg"></my-meme>

</body>
</html>
```

meme-component-custom-element/index.html

Inspired by: [https://github.com/webcomponents/hello-world-element](https://github.com/webcomponents/hello-world-element)
With Custom Elements

```html
<template> <!-- Defines element markup -->
  <img src=""/>
  <p class="caption captionTop"></p>
  <p class="caption captionBottom"></p>
  <style>...</style>
</template>

<script>
(function(window, document, undefined) {
    // ... some code to initialize your element goes here
})(window, document);
</script>
```

meme-component-custom-element/my-meme.html

Inspired by: https://github.com/webcomponents/hello-world-element
Polyfills

• Not all browsers support web components yet
• A Polyfill is a script that implements a workaround, to support new features in web browsers that do not support that feature.

<!-- Imports polyfill -->
<script src="webcomponents.min.js"></script>

Script source: https://www.webcomponents.org/polyfills
Can I use **html imports**?  
1 result found

### HTML Imports

Method of including and reusing HTML documents in other HTML documents.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Usage % of All Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current aligned</td>
<td></td>
</tr>
<tr>
<td>Edge</td>
<td>2.31</td>
</tr>
<tr>
<td>Firefox</td>
<td>4.29</td>
</tr>
<tr>
<td>Chrome</td>
<td>30.34</td>
</tr>
<tr>
<td>Safari</td>
<td>32.55</td>
</tr>
<tr>
<td>Opera</td>
<td>17.21</td>
</tr>
<tr>
<td>iOS Safari</td>
<td>22</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>6.16</td>
</tr>
<tr>
<td>Android Browser</td>
<td>3.2-12.4</td>
</tr>
<tr>
<td>Opera Mobile</td>
<td>12-12.1</td>
</tr>
<tr>
<td>Chrome for Android</td>
<td>21-4.4.4</td>
</tr>
<tr>
<td>Firefox for Android</td>
<td>12.12</td>
</tr>
<tr>
<td>UC Browser for Android</td>
<td>10.1</td>
</tr>
<tr>
<td>Samsung Internet</td>
<td>1.2</td>
</tr>
<tr>
<td>QQ Browser</td>
<td>7.1</td>
</tr>
</tbody>
</table>

[www.caniuse.com](http://www.caniuse.com)
Shadow DOM

- Include DOM elements into the rendering, but not into the main document
- Hides implementation details => encapsulation

What’s visible in the main document:

```html
<body>
  <my-meme captionTop="JAVASCRIPT" captionBottom="JAVASCRIPT EVERYWHERE"
    memeTemplateUrl="https://imgflip.com/....jpg"></my-meme>
</body>
</html>
```

What the browser renders:

```html
<!doctype html>
<html>
  <head>
    <!-- Runs custom element -->
    <my-meme captionTop="JAVASCRIPT" captionBottom="JAVASCRIPT EVERYWHERE"
      memeTemplateUrl="https://imgflip.com/s/meme/X-X-Everywhere.jpg">
      <shadow-root (open)
        <img src="https://imgflip.com/s/meme/X-X-Everywhere.jpg">
        <p class="caption captionTop">JAVASCRIPT</p>
        <p class="caption captionBottom">JAVASCRIPT EVERYWHERE</p>
      </shadow-root>
    </my-meme>
  </head>
  <body>
    <div>
      <style>...<style>
    </div>
  </body>
</html>
```
What it really looks like...

Get references to
- The importer (index.html)
- The importee (my-meme.html)

Create a new DOM object, extending prototype

Create Shadow DOM

Callback executed when instance of the element is created

Access attribute values of <my-meme> declaration

Get reference to sub-element in Shadow DOM, and set content

Register <my-meme> in the main document
Breakout #2

The counter app again:

• Use the your solution of the first breakout, or breakout1-solution from GitHub

• Encapsulate the counter card into a web component

• Since modules have to be loaded with CORS, you'll need a simple HTTP server, e.g. via
  
  *npm install -g http-server*
  *http-server*
How to use Web Components in Practice

• Lots of code for a simple functionality

• So of course there are useful libraries and frameworks for that!

• Different “Flavors”:
  • ReactJS
  • Angular
  • Polymer
Roundup Quiz

1. Which of these are (mostly) imperative, which are declarative languages:
   1. HTML
   2. JavaScript
   3. Java
   4. SQL
   5. TypeScript

2. What is a Polyfill for?

3. Which 4 concepts do WebComponents encompass?