Web Components

the future of web development?

Gaia Trecarichi

ICT4G Seminar
Povo, 14 April 2015

Outline

1. Web Components
   - intro
   - specs
   - libraries

2. Polymer
   - intro
   - a quick tutorial

3. Conclusions

The Web Development Mess

- It's hard to develop web app…lot's of libraries that might not like each other

- Hard to maintain the code when the application scales
  - lack of coding standards
  - cumbersome code
  - few html elements
  - complex DOM structure

Web Components

- W3C's emerging standard
- A way of standardizing widgets and plugins
- Teach new elements to the browser
- Enable to *create web applications as a set of reusable components*
- Live in *self-defined encapsulated unit with corresponding style and behavior logic*
- Four *foundational specifications*:
    1. Custom Elements
    2. HTML Templates
    3. Shadow DOM
    4. HTML Imports

Web Components Specs:
Custom Elements

Enable developers to *define and use new types of DOM elements in a document* with the ability to *style/script them just like any other HTML tag*

- *Less code to write.*
- *Express the function of the code.*
- *Encapsulate internal details.*
- *Allow you to reuse elements.*

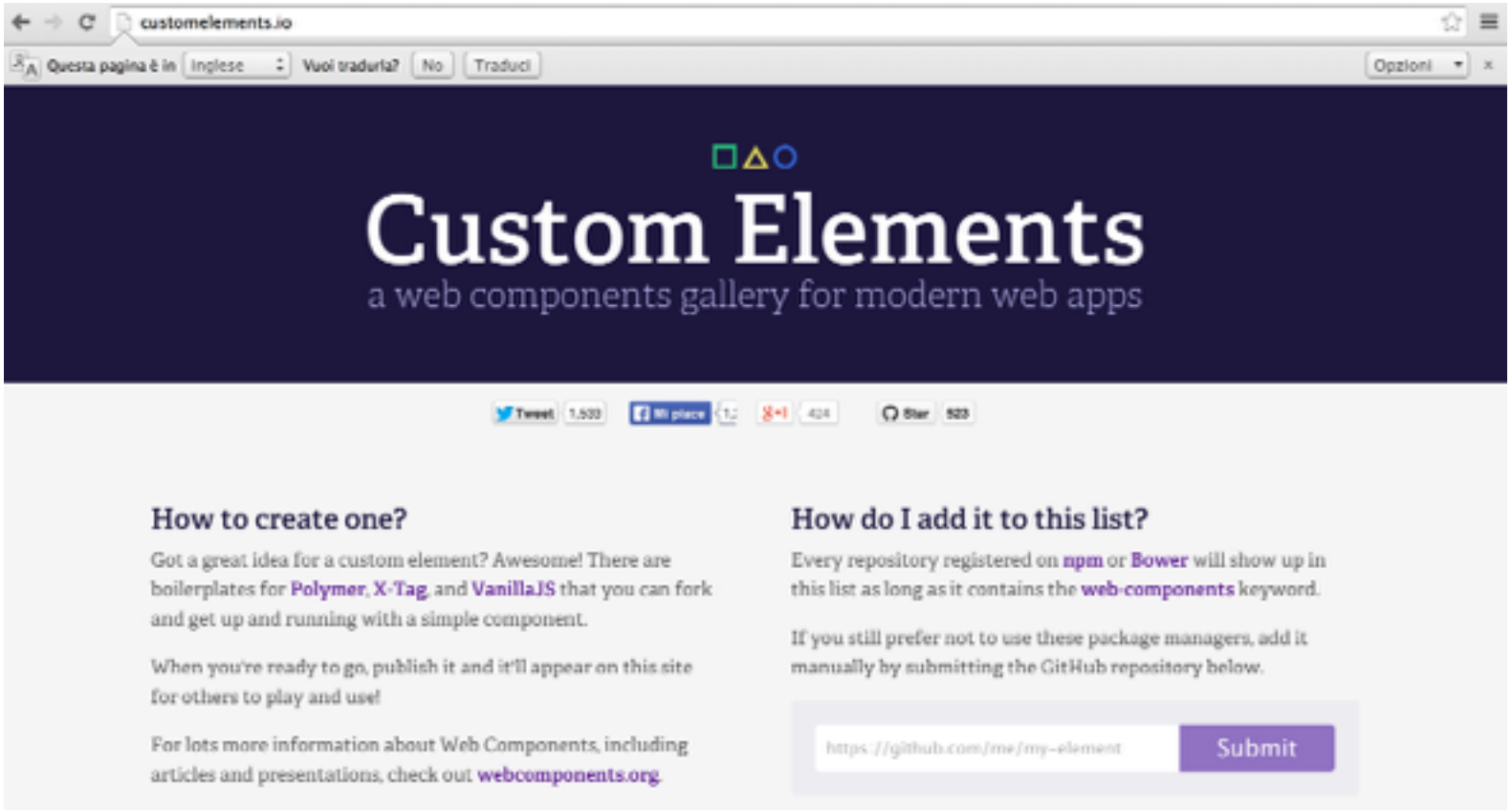
```
<polymer-element name="post-card">
    <template>
        <style>


        </style>
        <!-- CARD CONTENTS GO HERE -->

    </template>
    <script>



    </script>
</polymer-element>
```

Web Components Specs:
Custom Elements

Each instance of a custom element:

- *Is a DOM element*
- *Behaves like other DOM elements*
- *Lives in the DOM tree with the rest of other elements*
- *Can be accessed and manipulated with DOM methods…*
  or UI libraries
- *Is also a JavaScript object*

Custom Elements Gallery



Web Components Specs:
HTML Templates - properties

- Clonable DOM that can be reused on the page
- Inert HTML chunks until they are activated
  - <script> not run, stylesheets/image not loaded, media not played
- Hidden from document. Cannot traverse into its DOM
- Templates can be placed anywhere inside of <head>, <body>, and can contain any type of content which is

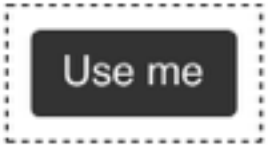
```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>
```

```
var t = document.querySelector('#mytemplate');
// Populate the src at runtime.
t.content.querySelector('img').src = 'logo.png';

var clone = document.importNode(t.content, true);
document.body.appendChild(clone);
```

Web Components Specs:
HTML Templates - demo

```
<button onclick="useIt()">Use me</button>
<div id="container"></div>
<script>
  function useIt() {
    var content = document.querySelector('template').content;
    // Update something in the template DOM.
    var span = content.querySelector('span');
    span.textContent = parseInt(span.textContent) + 1;
    document.querySelector('#container').appendChild(
        document.importNode(content, true));
  }
</script>

<template>
  <div>Template used: <span>0</span></div>
  <script>alert('Thanks!')</script>
</template>
```

LIVE DEMO:



Web Components Specs:
HTML Templates - old trick

- Using DOM
- Nothing is rendered

```
<div id="mytemplate" hidden>
  <img src="logo.png">
  <div class="comment"></div>
</div>
```

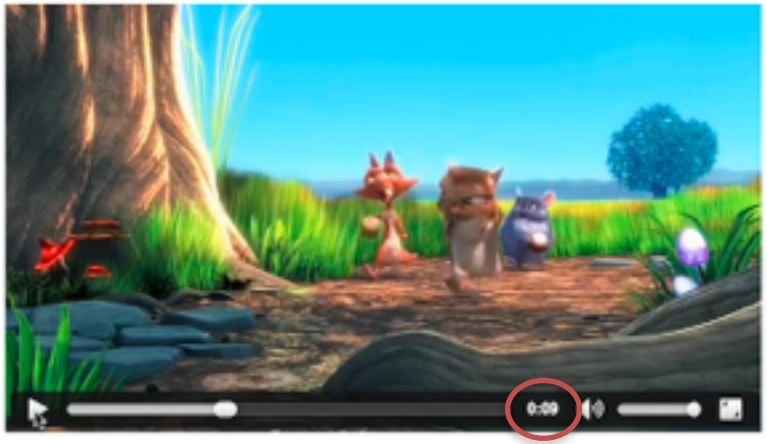
- Not inert
- Bad styling and theming

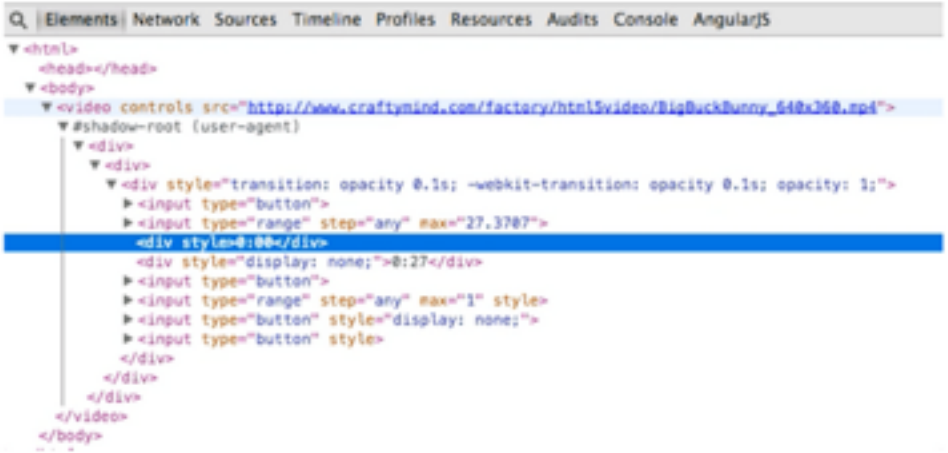
Web Components Specs: Shadow DOM

- Markup encapsulation
- Style boundaries
- Exposes (to web developers) the same *mechanics browsers vendors have been using* to implement their internal UI

Web Components Specs:
Shadow DOM

- **addresses the DOM tree encapsulation problem**
- abstract all the complexities from the markup by defining functional boundaries between the DOM tree and the subtrees hidden behind a shadow root

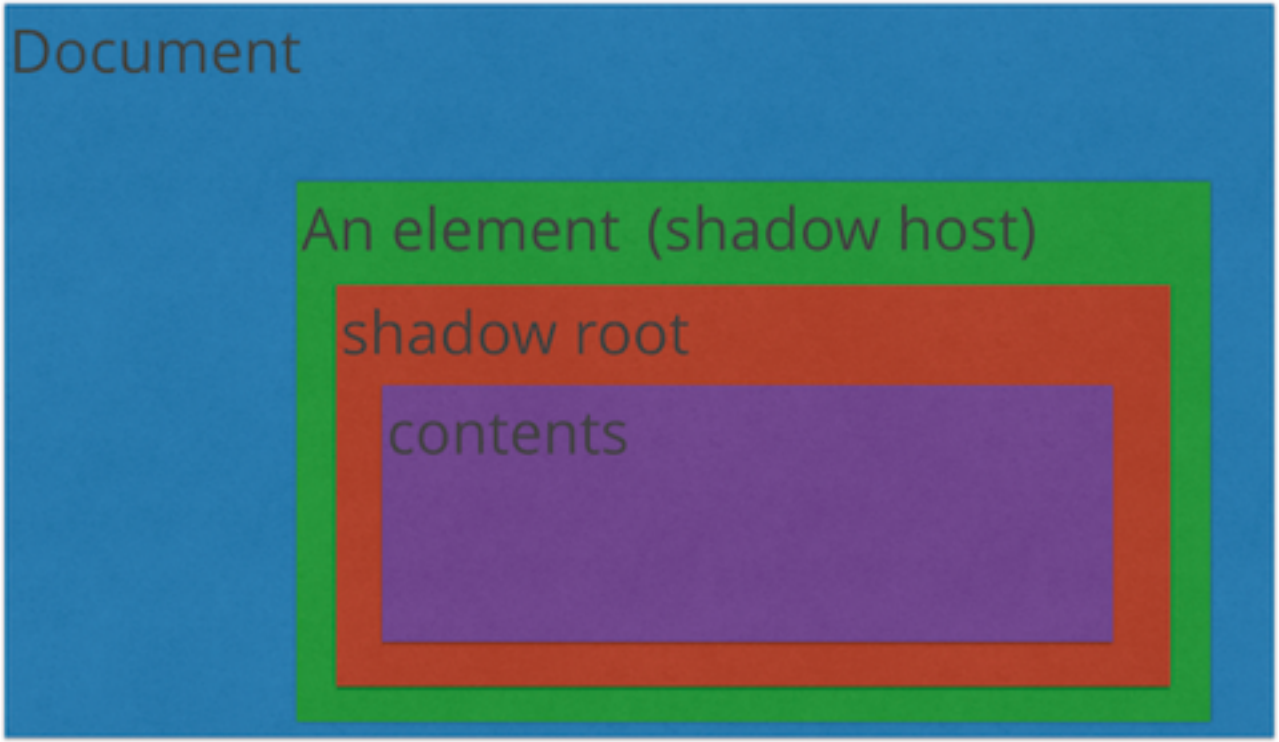





```
<video
src="http://craftymind.com/factory/html5video/BigBuckBunny_640x360.mp4"
controls></video>
```

Web Components Specs:
Shadow DOM - structure

- shadow root can be treated as an ordinary DOM element so you can append arbitrary nodes to it
- markup and CSS are scoped to the host element



Web Components Specs:
Shadow DOM - creation

1. By adding elements to the Shadow Root

```
<div id="host"></div>
```

```
var host = document.querySelector('#host');
var root = host.createShadowRoot(); // Create a Shadow Root
var div = document.createElement('div');
div.textContent = 'This is Shadow DOM';
root.appendChild(div); // Append elements to the Shadow Root
```

2. Declaratively with HTML

```
<!-- Content of <template> will be appended to the Shadow Root -->
<template id="template">
  <style>
    ...
  </style>
  <div id="container">
    <img src="http://webcomponents.org/img/logo.svg">
    <content select="h1"></content> // Insert h1 here
  </div>
</template>

<div id="host">
  <h1>This is Shadow DOM</h1>
</div>
```

hide presentation details

content // presentation

distribution mechanism

Web Components Specs:
Shadow DOM - distribution mechanism

- Reflecting the Shadow Host's content to a Shadow DOM
- The content is in the document; the presentation is in the Shadow DOM.
- Simplify the code that manipulates the content

The name update code doesn’t need to know the structure used for rendering.

Web Components Specs:
HTML Imports

Similar to import one CSS file into another, these allow you to include and reuse HTML documents in other HTML documents

```
<link rel="import" href="../components/polymer/polymer.html">
<link rel="import" href="../components/core-icon-button/core-icon-button.html">


<polymer-element name="post-card">



</polymer-element>
```

WCs Browser Support

	Specced	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	IE
Templates			Stable	Stable	8	Vote
HTML Imports			Stable	On Hold		Vote
Custom Elements			Stable	Flag		Vote
Shadow DOM			Stable	Flag		Vote

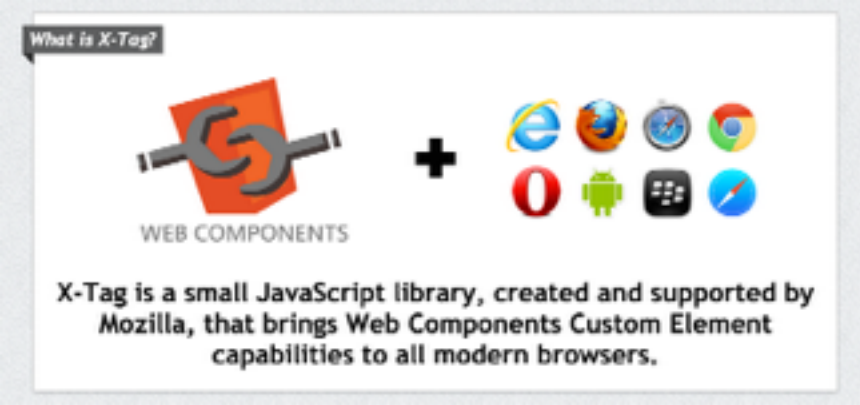
@are-we-componentized-yet?

@caniuse.com

Why to invest on Web Components?

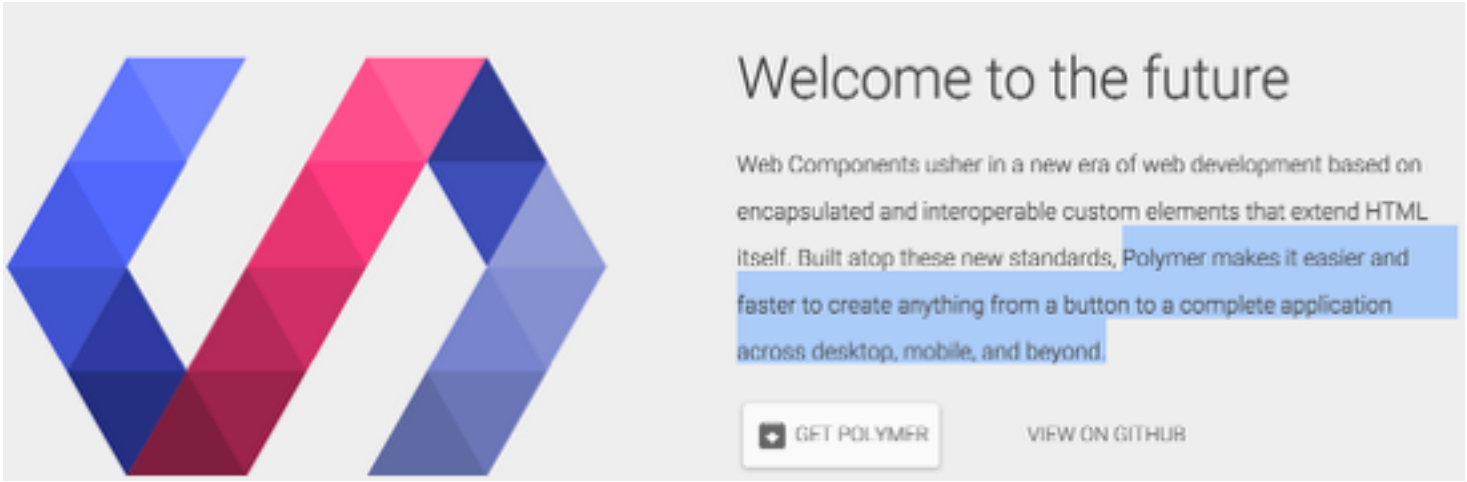
- “In two to three years every web application being built will be using them extensively…”
- Interoperability: shared across a single web application but can also be distributed on the web for use by others
- Make it possible to write modular, encapsulated code
- Work in all modern desktop and mobile browsers (Chrome, Firefox, Safari, IE >=10)
  - Google has created a compatibility library called **webcomponents.js** that lets web components work on any browser

WCs oriented libraries



[X-Tag by Mozilla]

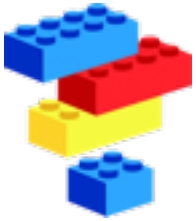
[Bosonic]



[Polymer by Google]

What is Polymer

Pioneering library to build modern, modular and maintainable web applications.



Built on top of a set of new W3C web platform primitives called Web Components

Currently in "developer preview" but many used it in production

Implements material design for the web.

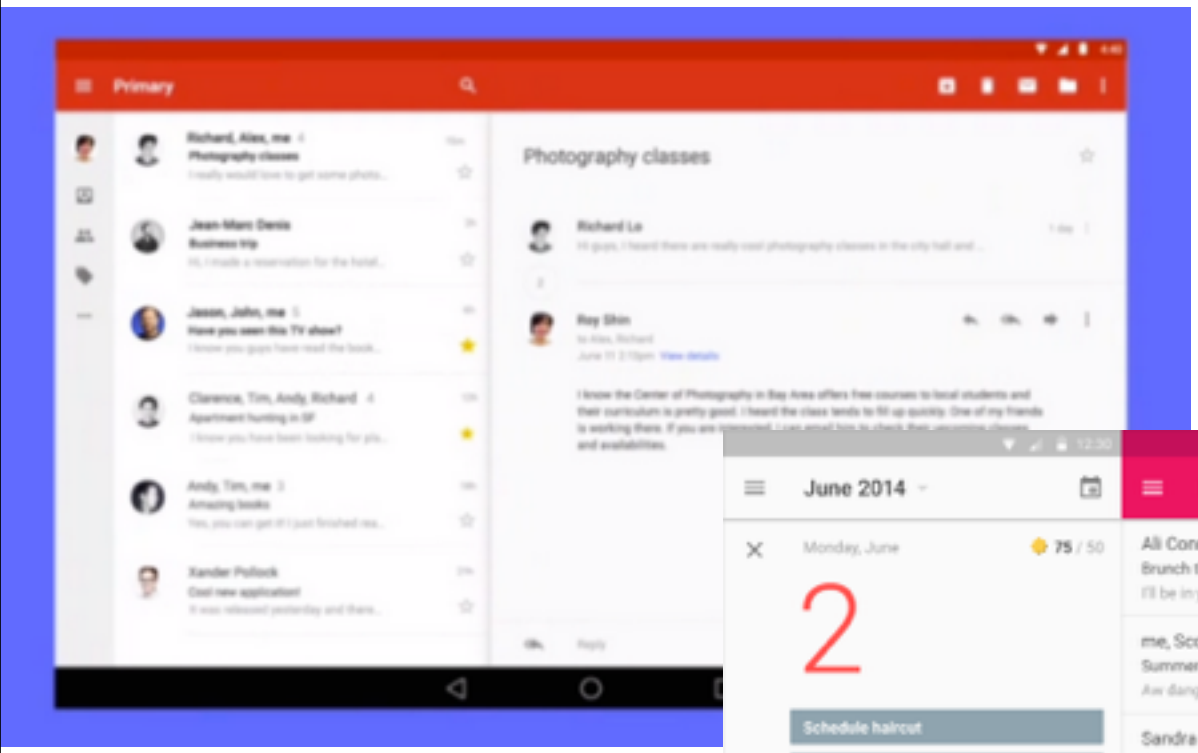
Polymer Conceptual Layers

- ***Web Components:*** a collection of libraries (or "polyfills") for new web technologies that haven't shipped yet across all browsers. The web components polyfills make it possible for developers to use these standards today across all modern browsers;

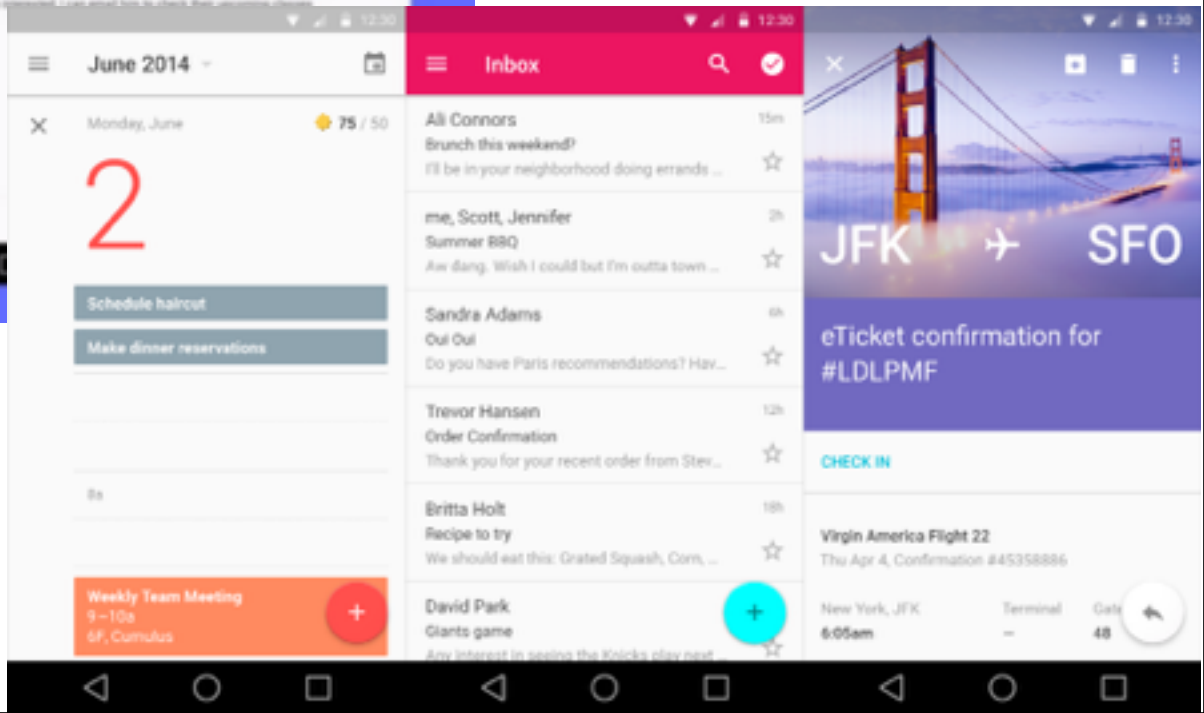
- **Polymer library**: provides a declarative syntax that makes it simpler to define custom elements. And it adds features like two-way data binding, event handling, property observation, and gesture support to help you build powerful, reusable elements;

- ***Elements*** *provide a suite of:*
  - ***Core Elements*** *– These are a set of visual and non-visual elements designed to work with the layout, user interaction, selection, and scaffolding applications.*
  - ***Paper Elements*** *– Implements the material design philosophy launched by Google recently at Google I/O 2014, and these include everything from a simple button to a dialog box with neat visual effects.*

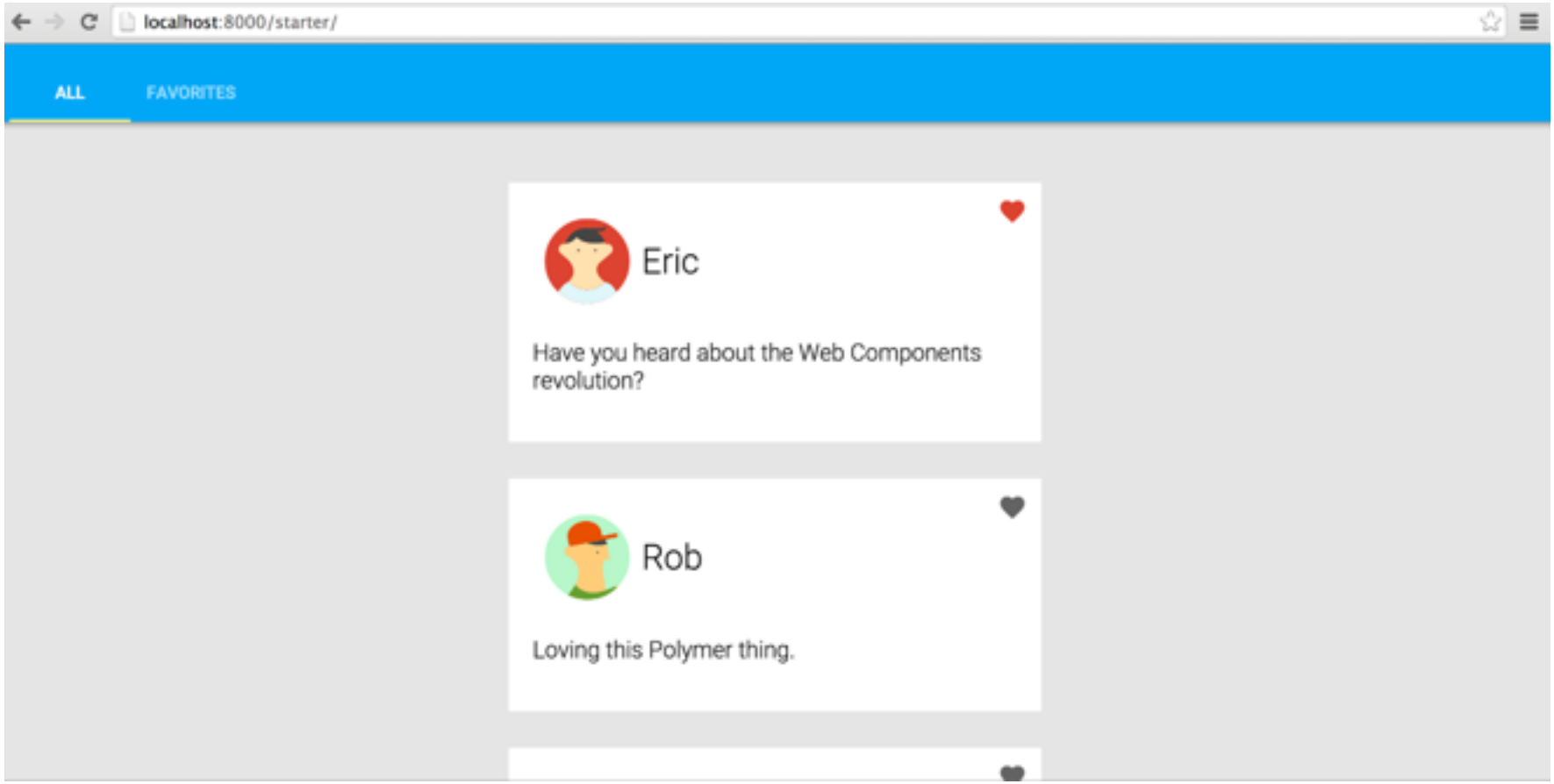
Material Design



Google

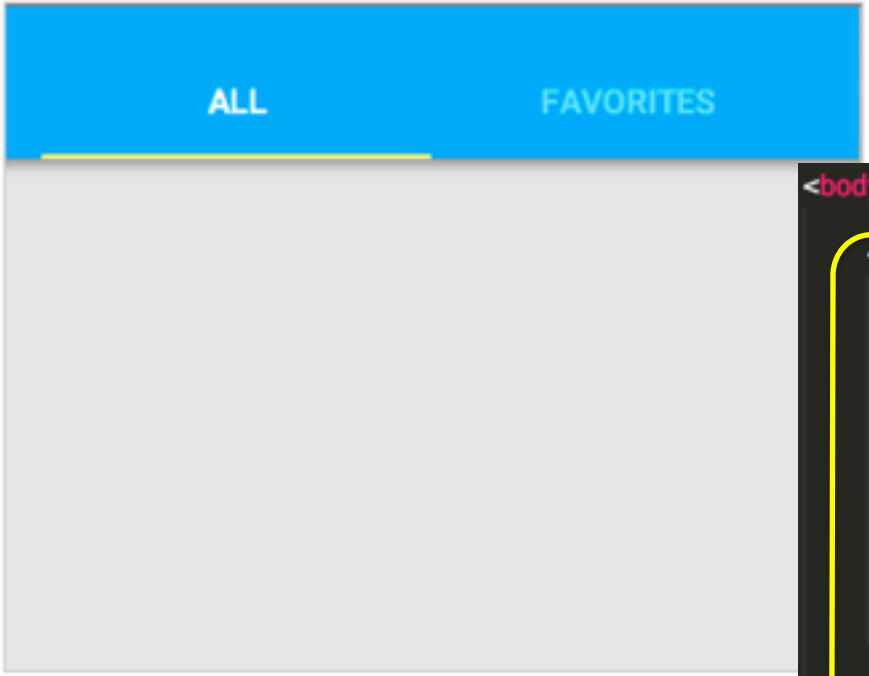


Polymer 0.5 Tutorial



Visit [Polymer Tutorial page]

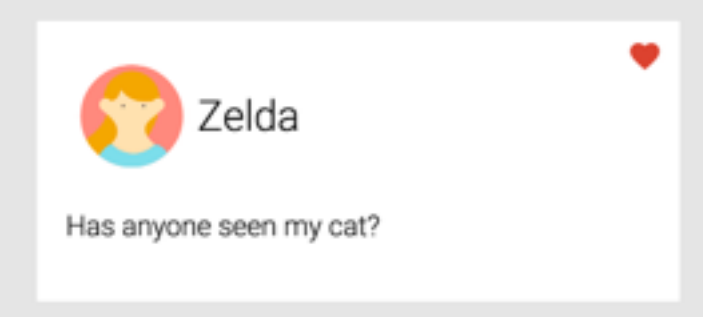
Polymer Tutorial
Step 1: app structure




```
<body unresolved>

    <core-header-panel>
        <core-toolbar>
            <paper-tabs id="tabs" selected="all" self-end>
                <paper-tab name="all">All</paper-tab>
                <paper-tab name="favorites">Favorites</paper-tab>
            </paper-tabs>
        </core-toolbar>

        <!-- main page content will go here -->
        <div class="container" layout vertical center>
            <post-list show="all"></post-list>
        </div>

    </core-header-panel>

    <script>
    var tabs = document.querySelector('paper-tabs');
    var list = document.querySelector('post-list');

    tabs.addEventListener('core-select', function() {
        console.log("Selected: " + tabs.selected);
        list.show = tabs.selected;
    });
    </script>
</body>
```

- *core-header-panel*
- *core-toolbar*
- *paper-tabs*

Polymer Tutorial
Step 2: post-card custom element




```
polyfill-next-selector {
    content: '.card-header h2';
}

.card-header ::content h2 {
    margin: 0;
    font-size: 1.8rem;
    font-weight: 300;
}

polyfill-next-selector {
    content: '.card-header img';
}

.card-header ::content img {
    width: 70px;
    border-radius: 50%;
    margin: 10px;
}
```

```
<polymer-element name="post-card">
    <template>
        <style>


        ....

        :host([favorite]) core-icon-button {
            color: #da4336;
        }

        </style>
        <!-- CARD CONTENTS GO HERE -->
        <div class="card-header" layout horizontal center>
            <content select="img"></content>
            <content select="h2"></content>
        </div>
        <core-icon-button id="favicon" icon="favorite" on-tap="{{favoriteTapped}}">
        </core-icon-button>
        <content></content>
    </template>

    <script>
    Polymer({
        publish: {
            favorite: {
                value: false,
                reflect: true
            }
        },
        favoriteTapped: function(event, detail, sender) {
            this.favorite = !this.favorite;
            this.fire('favorite-tap');
        }
    });
    </script>

</polymer-element>
```

Polymer Tutorial
Step 3: post-list custom element

```
<polymer-element name="post-service" attributes="posts">
  <template>
    <style>
    :host {
      display: none;
    }
    </style>
    <core-ajax id="ajax"
      auto
      url="../api/posts.json"
      on-core-response="{{postsLoaded}}"
      handleAs="json">
    </core-ajax>
  </template>
  <script>
  Polymer('post-service', {
    created: function() {
      this.posts = [];
    },
    postsLoaded: function() {
      // Make a copy of the loaded data
      this.posts = this.$.ajax.response.slice(0);
    },


    setFavorite: function(uid, isFavorite) {
      // no service backend, just log the change
      console.log('Favorite changed: ' + uid + ", now: " + isFavorite);
    }
  });
  </script>
</polymer-element>
```

```
<polymer-element name="post-list" attributes="show">
    <template>
        <style>
        :host {
            display: block;
            width: 100%;
        }

        post-card {
            margin-bottom: 30px;
        }
        </style>
        <!-- add markup here -->
        <post-service id="service" posts="{{posts}}">
        </post-service>

        <div layout vertical center>
            <template repeat="{{post in posts}}">
                <post-card favorite="{{post.favorite}}" on-favorite-tap="{{handleFavorite}}"
                {{show == 'favorites' && !post.favorite}}">
                    <img src="{{post.avatar}}" width="70" height="70">
                    <h2>{{post.username}}</h2>
                    <p>{{post.text}}</p>
                </post-card>
            </template>
        </div>
    </template>
    <script>
    Polymer({
        handleFavorite: function(event, detail, sender) {
            var post = sender.templateInstance.model.post;
            this.$.service.setFavorite(post.uid, post.favorite);
        }
    });
    </script>
</polymer-element>
```

- *post-service*
- *data binding*

Templates in Polymer

- In a Polymer element declaration, the first (top-level) <template> element is used to define the custom element's shadow DOM.

- Inside a Polymer element, you can use templates with data binding to render dynamic content.

- Data binding: assign, or bind, a JavaScript object as the template's data model.
    A. Single Templates (*bind*)
    B. Iterative Templates (*repeat*)
    C. Conditional Templates (*if*)

Polymer Tutorial
Step 4: the favorite button

```
lymer-element name="post-list" attributes="show">
 <template>
     <style>
     :host {
         display: block;
         width: 100%;
     }

     post-card {
         margin-bottom: 30px;
     }
     </style>
     <!-- add markup here -->
     <post-service id="service" posts="{{posts}}">
     </post-service>

     <div layout vertical center>
         <template repeat="{{post in posts}}">
             <post-card favorite="{{post.favorite}}"
              on-favorite-tap="{{handleFavorite}}"
              hidden?="{{show == 'favorites' && !post.favorite}}">
                 <img src="{{post.avatar}}" width="70" height="70">
                 <h2>{{post.username}}</h2>
                 <p>{{post.text}}</p>
             </post-card>
         </template>
     </div>
 </template>
 <script>
 Polymer({
     handleFavorite: function(event, detail, sender) {
         var post = sender.templateInstance.model.post;
         this.$.service.setFavorite(post.uid, post.favorite);
     }
 });
 </script>
olymer-element>
```

```
<polymer-element name="post-card">
    <template>
        <style>


        ....

        :host([favorite]) core-icon-button {
            color: #da4336;
        }

        </style>
        <!-- CARD CONTENTS GO HERE -->
        <div class="card-header" layout horizontal center>
            <content select="img"></content>
            <content select="h2"></content>
        </div>
        <core-icon-button id="favicon" icon="favorite" on-tap="{{favoriteTapped}}">
        </core-icon-button>
        <content></content>
    </template>

    <script>
    Polymer({
        publish: {
            favorite: {
                value: false,
                reflect: true
            }
        },
        favoriteTapped: function(event, detail, sender) {
            this.favorite = !this.favorite;
            this.fire('favorite-tap');
        }
    });
    </script>

</polymer-element>
```

- *Event handling*
- *Adding properties and methods to the element's prototype*
- *Automatic node finding*

Polymer 0.8

polymer

- Getting Started (0.5)
- Guides & Resources (0.5)
- Elements (0.5)
- Polymer 0.8 (alpha)
  - About this release
  - Getting the code
  - Developer guide
  - Migration Guide

About this release

0.8 ALPHA

The 0.8 release of the Polymer core library is now out.

Consider the 0.8 release to be our **proposed API for 1.0**. It is an "alpha" release — we fully expect some breaking changes as a result of the feedback we get.

This release is **intended for early adopters** who want to **test out the new APIs and provide feedback.** This release is optimized for performance and size, and is not yet a feature-complete replacement for 0.5. We're working hard on getting to feature parity. See the roadmap for more detailed timelines.

BREAKING CHANGES. This release is **not compatible with the 0.5 APIs.** For guidance on migrating an existing 0.5 element to the 0.8 APIs, see the Migration guide.

Highlights

- Dramatically faster startup time and runtime performance than 0.5, even in Chrome where web components are natively supported.
- Significantly smaller payload than 0.5.

WCs Alternatives

- **React** has its own “Virtual DOM” and allows the developer to use something very similar to Web Components. Since it doesn’t try to simulate Web Components, browser support is much better (Internet Explorer 6+). React is currently used on the Instagram and Facebook commenting system.

- **AngularJS** directives are very similar to web components but don't use the Web Component standard in order to achieve better browser support (Internet Explorer 8+). Since AngularJS is Google's playground for future features, it will surely move to real web components at some point.

Conclusions

- WCs are becoming a W3C standard

- Google is investing on them both with Polymer and AngularJS (2.0)

- Browser vendors are adhering to the specs

- No “side effects” mentioned so far

WCs seem a promising way to simplify the development and maintenance of web pages and apps but…

…who live will see!

References

- An Introduction to Web Components
- Web Components: A Tectonic Shift for Web Development
- How to Create Your Own HTML Elements With Web Components
- Are We Componentized Yet?
- Web Components - building blocks of the future web
- Introduction to the template elements
- HTML's New Template Tag
- Shadow DOM 101
- Introduction to Shadow DOM
- Polymer project
- An Introduction to Web Components and Polymer (Tutorial)
- AngularJS (2.0) and Polymer
- Getting Started with Polymer in Ruby on Rails
- Jarrod Overson and Jason Strimpel. "Developing Web Components: UI from jQuery to Polymer". O'Reilly, 2015.