



startups funding prospective
startup contest ^{html5}
cloud mobility **DIY** design

Web Components avec Polymer



Horacio Gonzalez

@LostInBrittany

Cityzen Data
<http://cityzendata.com>

Spaniard lost in Brittany,
developer, dreamer and all-
around geek

#Webcomponents



@LostInBrittany



Introduction

Because I love to tell old stories



Warning : I'm a web developer

And when I tell stories, I do it from the webdev POV



So please, thick-client devs, allow me some oversimplifications

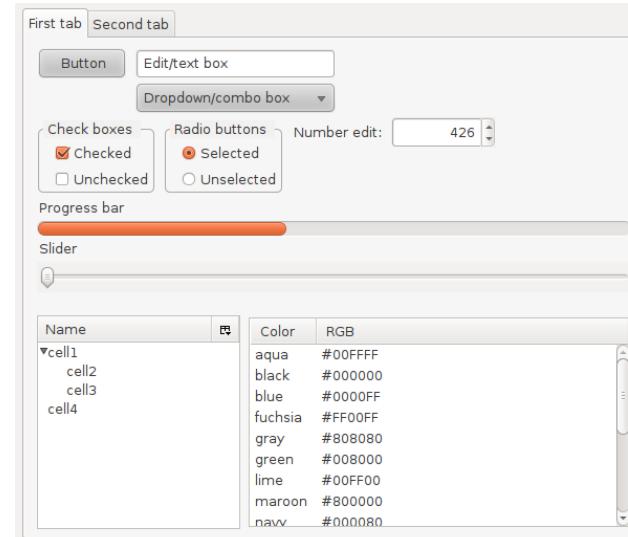
Image : TylkoMrok.pl



At the beginning we had the thick client

- Widget : basic building blocks of your UI

- Encapsulation
- Reutilisation



Name	Color	RGB
cell1	aqua	#00FFFF
cell2	black	#000000
cell3	blue	#0000FF
cell4	fuchsia	#FF00FF
	gray	#808080
	green	#008000
	lime	#00FF00
	maroon	#800000
	navv	#000080

In thick client you create your UI by assembling widgets

Image : [Wikipedia](#)



Web development was unlike thick-client

- HTML/CSS/JS didn't support widgets
 - Pages were the building blocks

A screenshot of a web browser window titled "Web Browser". The address bar shows the URL "http://localhost:9080/SimpleStrutsWeb/submitpage.jsp". The main content area displays a "Pizza Order Page" with the following form fields:

Pizza Order Page

Name:

Address:

Size: Small Medium Large Toppings:

Pepperoni

Onion

Mushroom

Hot Pepper

Bacon

Done

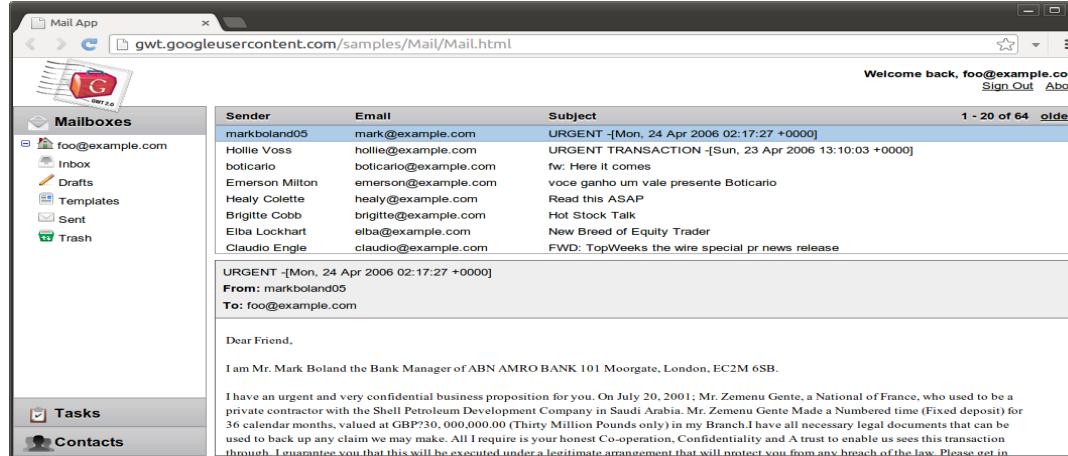
Web apps were page oriented

Image : [IBM](#)



GWT gave back widgets to web devs

- GWT used a thick-client-like development paradigm
 - Widgets, properties, events



GWT web apps were widget oriented :
Single-page apps

Image : [GWT Mail sample app](#)

@LostInBrittany



Single-page apps are a current trend

- From UX POV single-page apps are richer
 - But making them without widgets is risky and difficult



Image : [Ken Schultz comedy juggler](#)



Web Components

Reinventing the wheel... and this time making it round

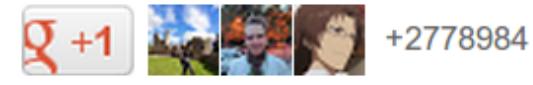


Example : the Google+ button

- If you want to place a Google+ button in your page

```
<!-- Place this tag where you want the +1 button to render. -->
<div class="g-plusone" data-annotation="inline" data-width="300"
></div>
```

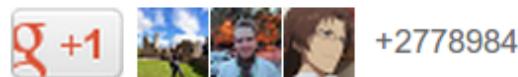
```
<!-- Place this tag after the last +1 button tag. -->
<script type="text/javascript">
  (function() {
    var po = document.createElement('script');
    po.type = 'text/javascript';
    po.async = true;
    po.src = 'https://apis.google.com/js/plusone.js';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(po, s);
  })();
</script>
```



Example : the Google+ button

- And what I would like is simple

```
<g:plusone></g:plusone>
```



Example : the Google+ button

- To be fair, Google already makes it simpler

```
<script type="text/javascript" src="https://apis.google.com/js/plusone.js">
</script>...
<g:plusone></g:plusone>
```



- They create directives with JS to emulate components
 - AngularJS approach
 - Respecting the spirit of the future standard
 - Working in current browsers

Totally non standard...



Another example : the RIB

- If you're French, you know what a RIB is
 - A banking account identification number

Banque	Guichet	N° compte	Clé
58496	87451	00014500269	74

- To show a RIB in HTML:
 - All styling & surface control must be done elsewhere by CSS and JS

```
<div class="rib">58496 87451 00014500269 74</div>
```

- What I would like
 - A semantic tag
 - With encapsulated styling and surface controlling

```
<x-rib banque="58496" guichet="87451" compte="00014500269" cle="74" />
```



But we already can do that!

- In most modern frameworks we can already do components, in a way or another
 - And all those ways are different!
 - Using different JavaScript libraries
 - Almost no component sharing between frameworks
- W3C's works aim to make a standard way
 - Supported natively by all browsers
 - Allowing for component reuse



Web Components : a W3C standard

- Web Components standard is being worked at W3C
 - We all know what this means
 - Clue : HTML5



They will work for years, bickering and fighting
Browsers and devs will use the WiP document



The 4 pillars of the Web Components

- Templates
- Shadow DOM
- Custom Elements
- Imports



Templates

The clone wars



[Instructables](#)



Templates before <template>

- How did we do templating before

- Using `display:none` or `hidden`

```
<div id="commentTemplate" class="comment" hidden>
    <img src=""> <div class="comment-text"></div>
</div>
```

- Putting it inside a `script`

- Type unknown to browser, it isn't interpreted
 - Markup easily recovered via `.innerHTML` and reused
 - Approach used by many template engines

```
<script id="commentTemplate" type="text/template">
    <div class="comment">
        <img src=""> <div class="comment-text"></div>
    </div>
</script>
```



The <template> tag

- Uniformising those approach with a new tag

```
<template id="commentTemplate">
  <div>
    <img src="">
    <div class="comment-text"></div>
  </div>
</template>
```

- Content inside the tag is parsed but not interpreted
 - HTML not shown
 - Resources are not loaded
 - Scripts not executed



Template instantiation

- Create the elements in page by cloning the template

```
<template id="commentTemplate">
  <div class="comment">
    <img src=""> <div class="comment-text"></div>
  </div>
</template>

<script>
  function addComment(imageUrl, text) {
    var t = document.querySelector("#commentTemplate");
    var comment = t.content.cloneNode(true);

    // Populate content.
    comment.querySelector('img').src = imageUrl;
    comment.querySelector('.comment-text').textContent = text;
    document.body.appendChild(comment);
  }
</script>
```



Shadow DOM

Join the shadowy side,
young padawan



[Springfield Punx](#)



Encapsulation

- Each component should have
 - Public interface
 - Private inner code
- When using a component
 - You manipulate the interface only
 - You don't need to know anything about inner code
 - No conflict between inner code and outside code



Encapsulation before Shadow DOM

- Only a way :
`<innerFrame>`

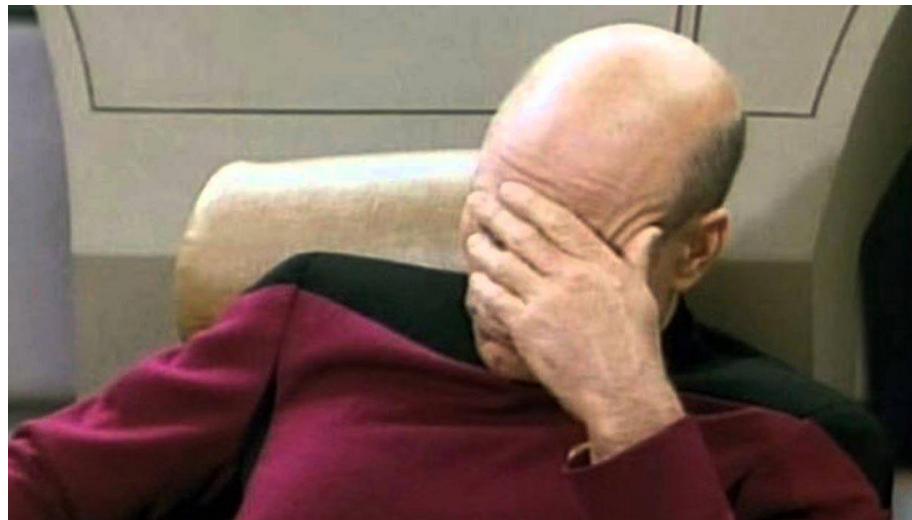


Image : Star Trek the Next Generation



Your browser cheats on you

- Considerer this simple slider

```
<input id="foo" type="range">
```



- How does the browser deal with it?
 - With HTML, CSS and JS!
- It has a movable element, I can recover it's position
 - Why cannot see it in DOM tree?

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
</video>
```

Browsers hide DOM sub-trees for standard components
They have a public interface and hidden inner code

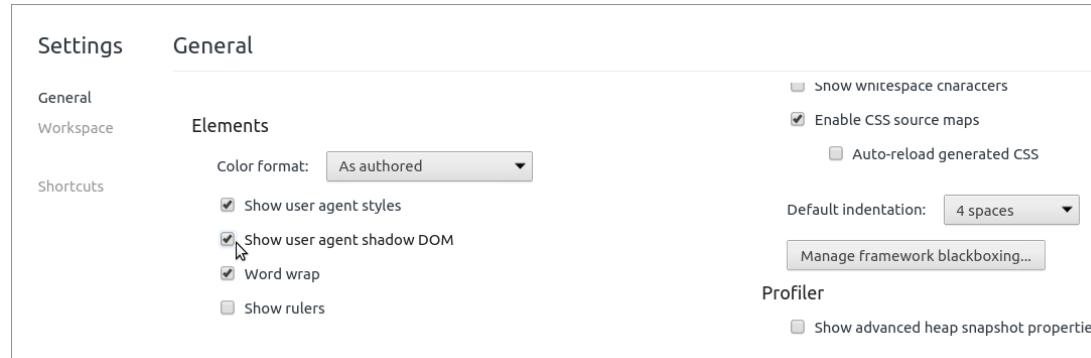


Image: Springfield Punx



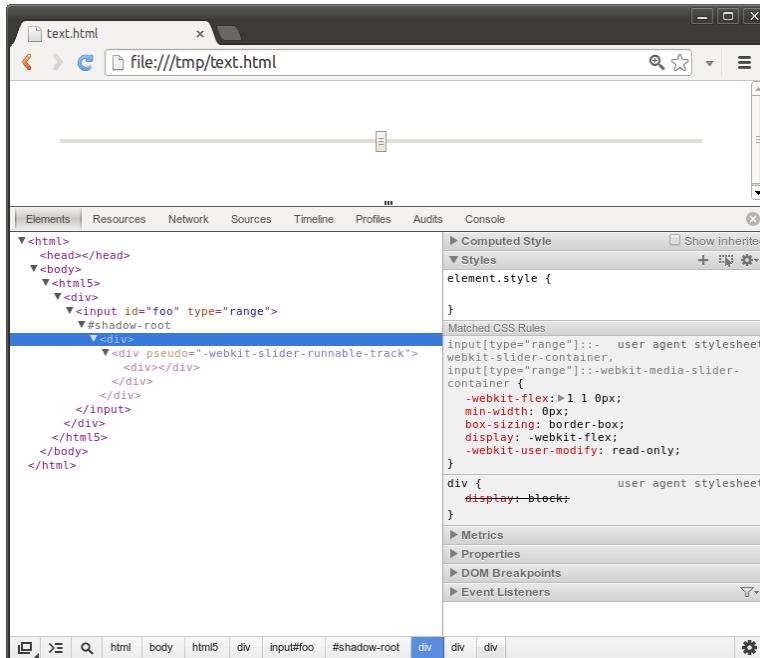
My name is DOM, Shadow DOM

- Shadow DOM: ability of the browser to
 - Include a DOM subtree into the rendering
 - But not into the main document DOM tree
- In Chrome you can see the Shadow DOM
 - By activating the option in Inspector



Looking into the Shadow

- For the slider :



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The page content is a simple slider with a horizontal track and a thumb. The DevTools sidebar displays the DOM tree:

```
<html>
  <head></head>
  <body>
    <div>
      <input id="foo" type="range">
      <div>
        <div pseudo="-webkit-slider-runnable-track">
          <div></div>
        </div>
      </div>
    </div>
  </body>
</html>
```

The 'Computed Style' panel on the right shows the styles applied to the input element, including vendor-specific properties like `-webkit-user-modify: read-only;`. The 'Styles' panel lists the element's style rule, and the 'Matched CSS Rules' panel lists the user agent stylesheets for the input element.



Shadow DOM is already here

- Browser use it everyday...
 - For their inner needs
 - Not exposed to developers
- Web Components makes Shadow DOM available
 - You can use it for your own components



Image: [Springfield Punx](#)



Using Shadow DOM

- There is a host element
 - A normal element in DOM tree
- A shadow root is associated to the host
 - Using the `createShadowRoot` method
 - The shadow root is the root of the hidden DOM tree

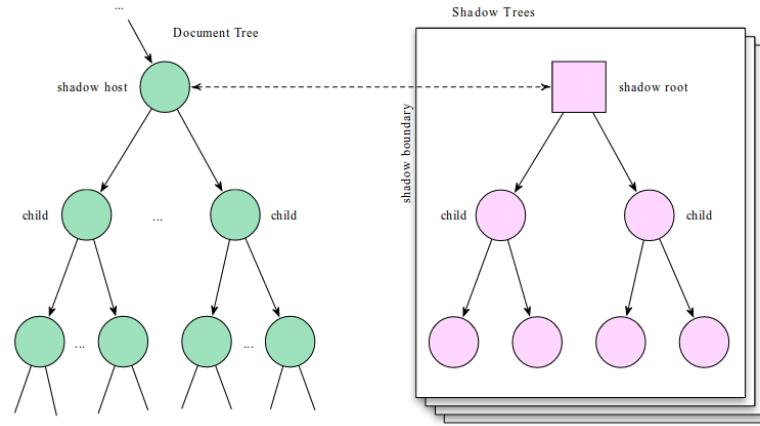


Image: [W3C](#)



Using Shadow DOM

- Quick and dirty Shadow DOM

```
<div id="emptyHost"></div>
<script>
    var host = document.querySelector('#emptyHost');
    var root = host.createShadowRoot();
    root.innerHTML = "<h1>Not empty anymore!</h1>";
</script>
```

- DOM tree only shows

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

- Rendered HTML shows

Not empty anymore!

- Markup in `innerHTML` is ugly



Using Shadow DOM

- Shadow DOM with templates

```
<div id="emptyHost"></div>
<template id="commentTemplate"> [...] </template>

<script>
  var host = document.querySelector('#emptyHost');
  var shadowRoot = host.webkitCreateShadowRoot();

  function addComment(imageUrl, text) { [...] }

  function addShadowedElement() {
    var instanceTemplate =
      addComment("http://lostintbrittany.org/avatar.png",
                 "This is a nice comment made by a nice guy");
    shadowRoot.appendChild(instanceTemplate);
  }
</script>
```



Shadow DOM et CSS

- CSS defined in the Shadow DOM remains there
- Outside styles don't affect Shadowed content

```
<h1>This is a title</h1>
<div id="widget">
  #document-fragment
  <style>
    div {border: solid 1px red;}
    h1 {color: blue;}
  </style>
  <h1>And this is widget title</h1>
  <div>Widget content here</div>
</div>
```

This is a title

And this is widget title

Widget content here



Shadow DOM et CSS

- Styling the host element : @host

```
<template id="template">
  <style>
    @host {
      button { border-radius: 5px; }
    }
  </style>
  <content></content>
</template>

<button id="host">My Button</button>

<script>
  var host = document.querySelector('#host');
  var root = host.createShadowRoot();
  var shadowContent =
    document.querySelector("#template").content.cloneNode(true);
  root.appendChild(shadowContent);
</script>
```

My Button



Example

The screenshot shows a web application interface with the following components:

- Header:** A nice banking application with navigation links: Home, About, Contact, Email, Password, and Sign in.
- Action menu (Left Column):**
 - Alpha actions:** Action #1, Action #2, Action #3, Action #4.
 - Beta actions:** Action #1, Action #2, Action #3, Action #4.
- Action #1 workbench (Right Column):**
 - RIB du client sans formattage:** 1234, 5678, xxxxxxxxx, 69.
 - RIB du client avec formattage:**

Code banque 1234	Code guichet 5678	Numéro de compte xxxxxxxxxx	Clé RIB 69
---------------------	----------------------	--------------------------------	---------------
- Code HTML pour les deux:**

```
<div id="code_html_rib">
  <div class="code_banque">1234</div>
  <div class="code_guichet">5678</div>
  <div class="numero_compte">xxxxxxxxxx</div>
  <div class="cle_rib">69</div>
</div>
```



Custom Elements

Elemental mayhem !



[The Brick Blogger](#)

#Webcomponents

@LostInBrittany



Custom elements : the HTML side

- An element encloses template, lifecycle and behaviour
 - Templates done with `<template>` tag

```
<!-- Template Definition -->
<template id="template">
  <style>
    ...
  </style>
  <div id="container">
    
    <content select="h1"></content>
  </div>
</template>

<!-- Custom Element usage -->
<x-component>
  <h1>This is Custom Element</h1>
</x-component>
```



Custom elements : the JavaScript side

- An element encloses template, lifecycle and behaviour
 - JavaScript to define behaviour and register the element

```
var proto = Object.create(HTMLElement.prototype);
proto.createdCallback = function() {
    // Adding a Shadow DOM
    var root = this.createShadowRoot();
    // Adding a template
    var template = document.querySelector('#template');
    var clone = document.importNode(template.content, true);
    root.appendChild(clone);
}
var XComponent = document.registerElement('x-component', {
    prototype: proto
});
```



Extending other elements

- To create element A that extends element B,
element A must inherit the prototype of element B

```
var MegaButton = document.registerElement('mega-button'  
  prototype: Object.create(HTMLButtonElement.prototype),  
  extends: 'button'  
});
```



Imports

Because you hate long files



Brikipedia



Imports

- Custom elements can be loaded from external files
 - Using the link tag:

```
<link rel="import" href="goodies.html">
```

- Only `<decorator>` and `<element>` are interpreted
- The DOM of this document is available to script
- Documents are retrieved according to cross-origin policies



Can I use?

If not, why are you telling us all this sh*t?



[Christoph Hauf](#)



Are we componentized yet?

Are We Componentized Yet?

Tracking the progress of [Web Components](#) through standardisation, polyfillification¹, and implementation.

	Specged	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	IE
Templates			Stable	Stable	8	
HTML Imports			Stable	Flag		
Custom Elements			Stable	Flag		
Shadow DOM			Stable	Flag		

Cells are clickable! Yellow means in-progress; green means mostly finished.



WebComponents.org



The screenshot shows the homepage of WebComponents.org. At the top, there's a navigation bar with links for HOME, ARTICLES, PRESENTATIONS, RESOURCES, and SANDBOX. To the right of the navigation is a Google Custom Search bar and a search icon. The main header features a large orange logo consisting of two interlocking white 'x' shapes forming a circular pattern, followed by the text "WebComponents.org" in a large, bold, white sans-serif font. Below the header, a subtext reads "a place to discuss and evolve web component best-practices". The page is divided into several sections: "WHAT?", "SPECS", and "ARTICLES". The "WHAT?" section contains a brief description of the site's purpose and a "BROWSER SUPPORT" chart showing compatibility across various browsers. The "SPECS" section lists three specifications: "WEB COMPONENTS", "CUSTOM ELEMENTS", and "HTML IMPORTS", each with a brief description and an orange circular icon. The "ARTICLES" section features a photo of a child in a superhero costume flexing their muscles, with a link to "WEB COMPONENTS BEST PRACTICES". A "Read More >" link and a "see all articles" button are also present.

WHAT?

WebComponents.org is where pioneers and community-members of the Web Components ecosystem (like Polymer, X-tags, and other interested parties) document web components best practices so that others can follow the same path instead of needlessly striking out on their own.

BROWSER SUPPORT

CHROME	OPERA	FIREFOX	SAFARI	IE
Green	Green	Green	Red	Red

SPECS

WEB COMPONENTS
This document is a non-normative reference, which provides an overview of Web Components. It summarizes the normative information in the respective specifications in easy-to-digest prose with illustrations.

CUSTOM ELEMENTS
This specification describes the method for enabling the author to define and use new types of DOM elements in a document.

HTML IMPORTS
HTML Imports are a way to include and reuse HTML documents in other HTML documents.

ARTICLES

WEB COMPONENTS BEST PRACTICES
Web Components (WC) are a new set of web platform features that enable developers to build applications in a declarative, composable way. The following is an initial list of best practices we advocate component authors consider to ensure their elements are good citizens in the Web Component ecosystem.

[Read More >](#)

[see all articles](#)

#Webcomponents

@LostInBrittany



Polymer

Webcomponents for today's web



Polymer

- A Google project
 - Introduced in Google I/O 2013
 - New type of library for the web
 - Built on top of Web Components
 - Designed to leverage the evolving web platform

Version 1.0 just released last week at Google IO

Oh yeah!

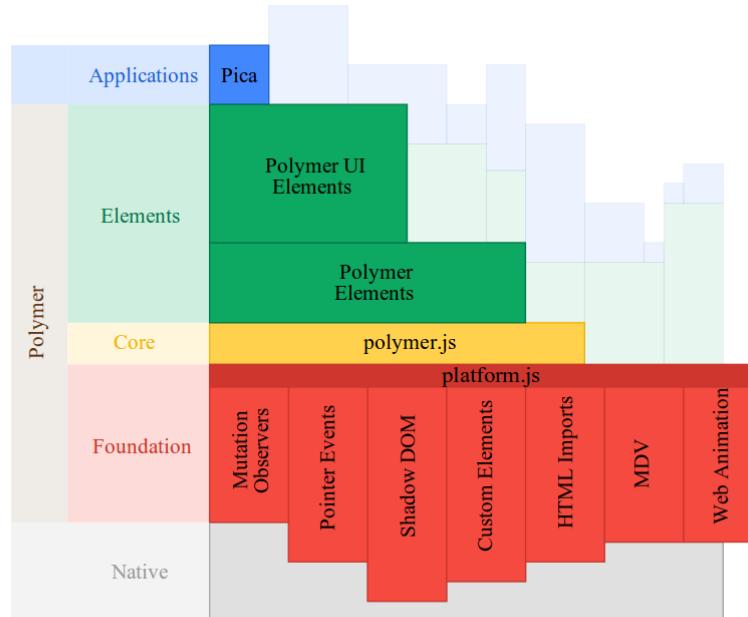


Polymer

- What does it means ?
 - Polymer is comprised of two efforts :
 - A paper platform to give Web Component capabilities to modern browsers
 - Shims for all modern browsers
 - Shared with Mozilla x-tag project
 - A next-generation web framework built upon this core platform
 - Called Polymer Elements



Polymer



Architectural Diagram



Polymer elements



Fe 1.0.0
Iron Elements
Polymer core elements

Md 1.0.1
Paper Elements
Material design elements

Go 1.0.0
Google Web Components
Components for Google's APIs and services

Au 1.0.0
Gold Elements
Ecommerce Elements

Ne 1.0.0
Neon Elements
Animation and Special Effects

Pt 1.0.0
Platinum Elements
Offline, push, and more

Mo 1.0.0
Molecules
Wrappers for third-party libraries



Polymer

- Principles:
 - Everything is a component
 - Encapsulation is needed for a component oriented application
 - Extreme pragmatism
 - Boilerplate is bad
 - Anything repetitive should be re-factored into a component
 - Handled by Polymer itself or
 - Added into the browser platform itself



Polymer

- Principles:
 - Salt to taste
 - Use as much or as little of the framework as you wish.
 - You want polyfills only : load `polymer-all/platform/platform.js`
 - You want extra bits : load `polymer-all/polymer/polymer.js`
 - Polymer elements



Polymer

- Platform technologies are already functional
 - You can use it to add templates, shadow DOM, custom elements and imports to your app
- Lots of examples in the site



What about X-Tag?

- X-Tag is a small JavaScript library
 - created and supported by Mozilla
 - that brings Web Components capabilities
 - to all modern browsers
- Polymer vs X-Tag?
 - Different features and approaches
 - Mozilla and Google are collaborating
 - building the shared polyfills platform



And AngularJS?

- AngularJS directives allow to create custom elements
 - Same spirit, different implementation

```
<!doctype html>
<html>
  <head>
    <title>Directive Test</title>
    <script type="text/javascript" src="jquery.min.js" ></script>
    <script type="text/javascript" src="angular.min.js"></script>
    <script type="text/javascript" src="mydirectives.js"></script>
  </head>
  <body ng-app>
    <div test-elem></div>
  </body>
</html>
```



Polymer today

Because you can already play!



Step-1 : get Polymer

The screenshot shows the official Polymer website's "Get the code" page. The left sidebar has a light gray background with various navigation links. The main content area has a blue header with the title "Get the code" and a "Basics" sub-section. The right side of the main content area includes a "v. 1.0" dropdown and a search icon.

polymer

About this release

Get started

What is Polymer?

Quick tour of Polymer

Get the code

Reusable elements

Developer guide

API Reference

Articles

Catalog

v. 1.0 ▾

Get the code

Basics

Edit on GitHub

▶ Table of contents

Installing Polymer

If you're ready to start your own project, you can install the Polymer library in one of several ways:

- [Bower](#). Bower manages dependencies, so installing a component also installs any missing dependencies. Bower also handles updating installed components. For more information, see [Installing with Bower](#).
- [ZIP file](#). Includes all dependencies, so you can unzip it and start using it immediately. The ZIP file requires no extra tools, but doesn't provide a built-in method for updating dependencies. For more information, see [Installing from ZIP files](#).

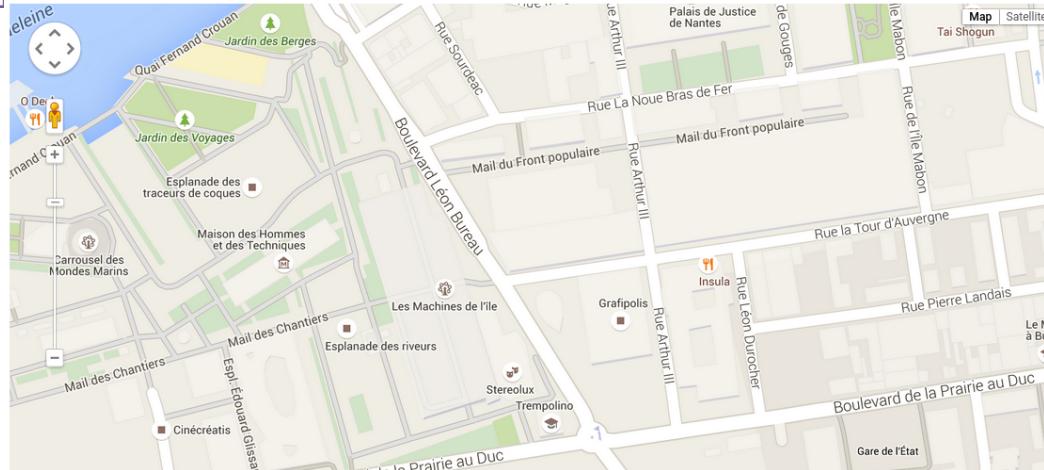


Step 2 - Use an element

```
<!-- 1. Polyfill Web Components support for older browsers -->
<script src="bower_components/webcomponentsjs/webcomponents-lite.min.js"
></script>

<!-- 2. Import element -->
<link rel="import" href="bower_components/google-map/google-map.html">

<!-- 3. Use element -->
<google-map latitude="47.205402" longitude="1.5621060" zoom="17"></google-map>
```



Step-3.1 : define an element

```
<dom-module name="x-web2day">
  <template>
    <style>
      @host { /*...*/ }
    </style>
    <div id="box">
      <h1>Bonsoir web2day !</h1>
      <p><content></content></p>
    </div>
  </template>
</dom-module>
<script>
  Polymer({ is: 'x-web2day' });
</script>
```



Step-3.2 : load Polymer, import your element

```
<!DOCTYPE html>
<html>
  <head>
    <!-- 1. Load Polymer -->
    <script src="bower_components/webcomponentsjs/webcomponents.js">
    </script>
    <!-- 2. Load a component -->
    <link rel="import" href="x-web2day.html">
  </head>
  <body>
    <!-- 3. Declare the component by its tag. -->
    <x-web2day>Et je peux mettre mon texte ici
    </x-web2day>
  </body>
</html>
```



Step-3.3 : enjoy

Bonsoir web2day !

Et je peux mettre mon texte ici



Step-4.1 : Add properties/methods

```
<dom-module id="x-web2day">
  <template>
    <style> /.../ </style>
    <div id="box">
      <h1>{{greet}}</h1>
      <p><content></content></p>
      <div><button on-click="alertHello">Click me!</button></div>
    </div>
  </template>
</dom-module>

<script>
  Polymer({
    is: 'x-web2day',
    properties: {
      whoami: { type: String, value: "web2day" },
      greet: { type: String, computed: 'doGreet(whoami)' }
    },
    alertHello: function() { alert("Hello "+this.whoami); },
    doGreet: function(who) { return "Hello "+who}
  });
</script>
```



Step-4.2 : enjoy



Step-4.1 : Two-ways databinding

```
<dom-module id="x-web2day">
  <template>
    <style>/*...*/</style>
    <div id="box">
      <h1>Bonjour <span>{{whoami}}</span></h1>
      <p><content></content></p>
      <!-- iron-input exposes a two-way bindable input value -->
      <input is="iron-input" bind-value="{{whoami}}" placeholder="Your name here..." />
    </div>
  </template>
</dom-module>

<script>
Polymer({
  is: 'x-web2day',
  properties: {
    whoami: { type: String, value: "web2day", notify: true
    }
  }
});
</script>
```



Step-5.2 : enjoy

Bonjour web2day

Et je peux mettre mon texte ici

web2day



Catalog of elements



Fe 1.0.0
Iron Elements
Polymer core elements

Md 1.0.1
Paper Elements
Material design elements

Go 1.0.0
Google Web Components
Components for Google's APIs and services

Au 1.0.0
Gold Elements
Ecommerce Elements

Ne 1.0.0
Neon Elements
Animation and Special Effects

Pt 1.0.0
Platinum Elements
Offline, push, and more

Mo 1.0.0
Molecules
Wrappers for third-party libraries



Polymer iron elements

The screenshot shows the Polymer Catalog interface. On the left, there's a sidebar with a navigation menu. The 'Iron Elements' option is highlighted with a green background and a green border. The main content area is titled 'Iron Elements (33)' and contains a table listing seven utility elements: iron-a11y-announcer, iron-a11y-keys, iron-a11y-keys-behavior, iron-ajax, iron-autogrow-textarea, iron-behaviors, and iron-collapse. Each row in the table includes a 'Name', 'Description', 'Tags', and an 'Action' column with three icons.

Name	Description	Tags	Action
iron-a11y-announcer	A singleton element that simplifies announcing text to screen readers.	a11y, live,	
iron-a11y-keys	A basic element implementation of iron-a11y-keys-behavior, matching the legacy core-a11y-keys.	a11y, input	
iron-a11y-keys-behavior	A behavior that enables keybindings for greater a11y.	a11y, input	
iron-ajax	Makes it easy to make ajax calls and parse the response		
iron-autogrow-textarea	A textarea element that automatically grows with input	input, textarea,	
iron-behaviors	Provides a set of behaviors for the iron elements		
iron-collapse	Provides a collapsable container	container,	

Set of visual and non-visual utility elements



Example: iron-ajax

iron-ajax (1.0.0)

Documentation

The iron-ajax element exposes network request functionality.

```
<iron-ajax
  auto
  url="http://gdata.youtube.com/feeds/api/videos/"
  params='{"alt":"json", "q":"chrome"}'
  handle-as="json"
  on-response="handleResponse"
  debounce-duration="300"></iron-ajax>
```

With auto set to true, the element performs a request whenever its url, params or body properties are changed. Automatically generated requests will be debounced in the case that multiple attributes are changed sequentially.

Note: The params attribute must be double quoted JSON.

You can trigger a request explicitly by calling generateRequest on the element.



Paper Elements

Material Design on Polymer, oh yeah!



ToyPro



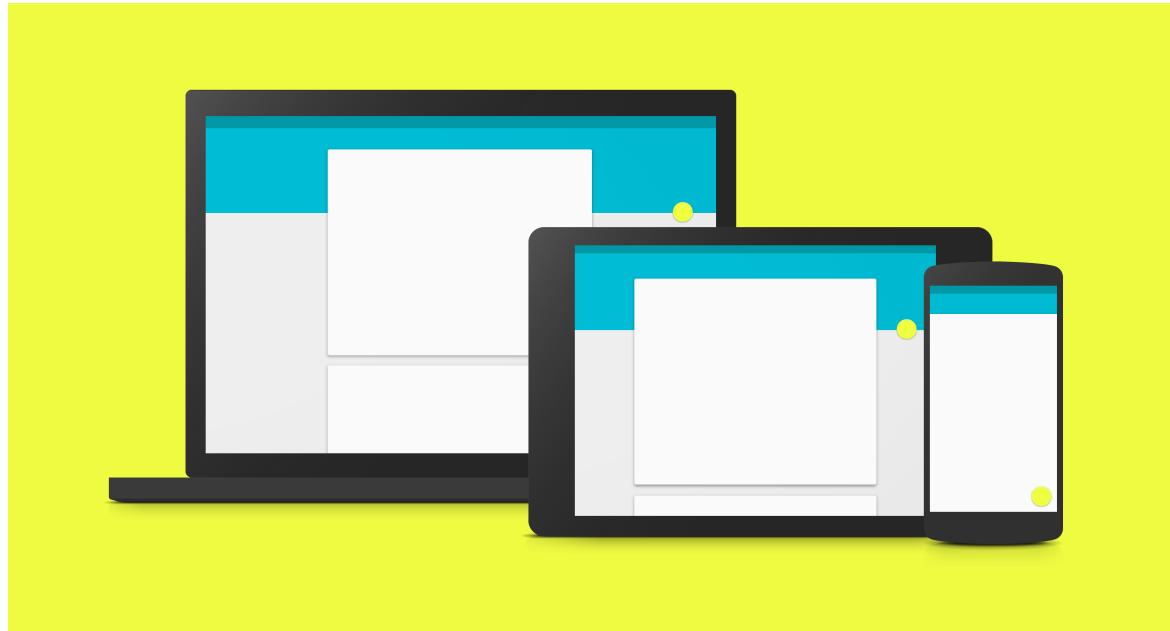
Android Lollipop's look & feel



Material Design is a visual language that synthesizes the classic principles of good design with the innovation and possibility of technology and science



Material Design

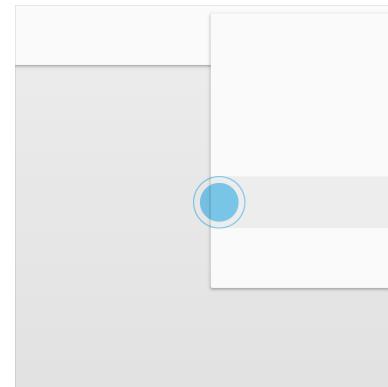
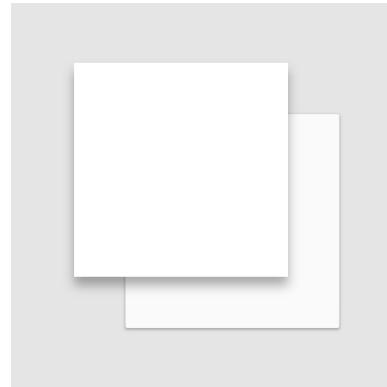
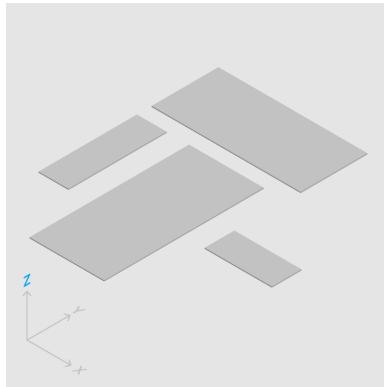


Visual language joining good design with technical innovation to create an unified experience across platforms and device sizes



What's material design?

UIs implementing a 3D metaphor : lights & shadows, stacking



<http://www.google.com/design/spec/material-design/>



Polymer paper elements

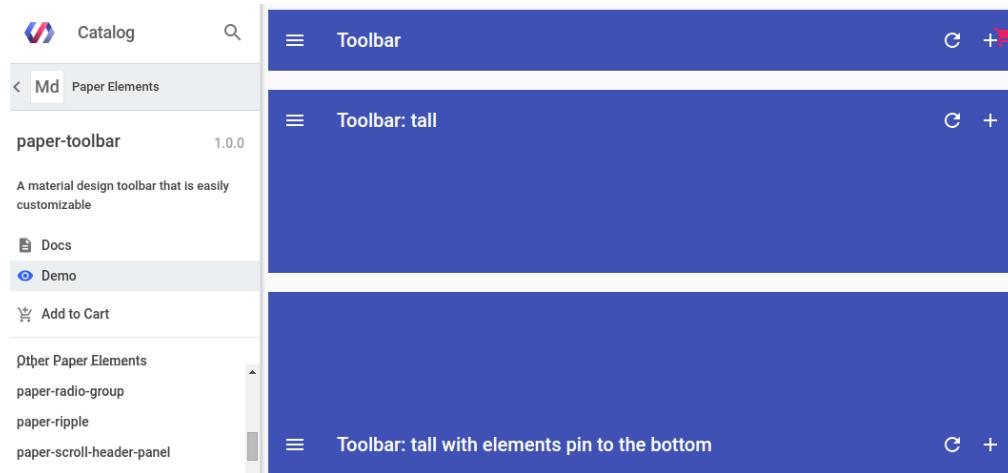
The screenshot shows the Polymer Catalog interface. On the left, there's a sidebar with categories: Products, All Elements, Iron Elements (highlighted in green), Paper Elements (highlighted in grey), Google Web Components, Gold Elements, Neon Elements, and Platinum Elements. The main area is titled "Paper Elements (26)" and contains a table with the following data:

Name	Description	Tags	Action
paper-behaviors	Common behaviors across the paper elements	behavior,	
paper-button	Material design button	button,	
paper-checkbox	A material design checkbox	checkbox, control,	
paper-dialog	A Material Design dialog	dialog, overlay,	
paper-dialog-behavior	Implements a behavior used for material design dialogs	dialog, overlay, behavior,	

Set of visual elements that implement Material Design



<paper-toolbar>

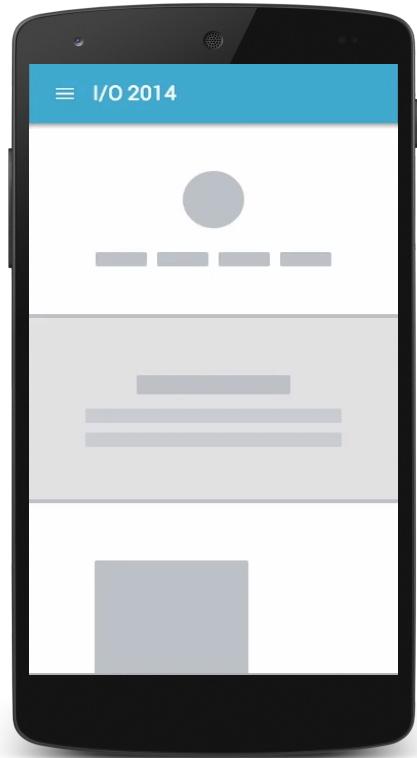


```
<link rel="import"  
      href="paper-toolbar.html">
```

```
<paper-toolbar>  
  <div>Title</div>  
  <paper-icon-button icon="menu">  
    </paper-icon-button>  
</paper-toolbar>
```



<paper-header-panel>

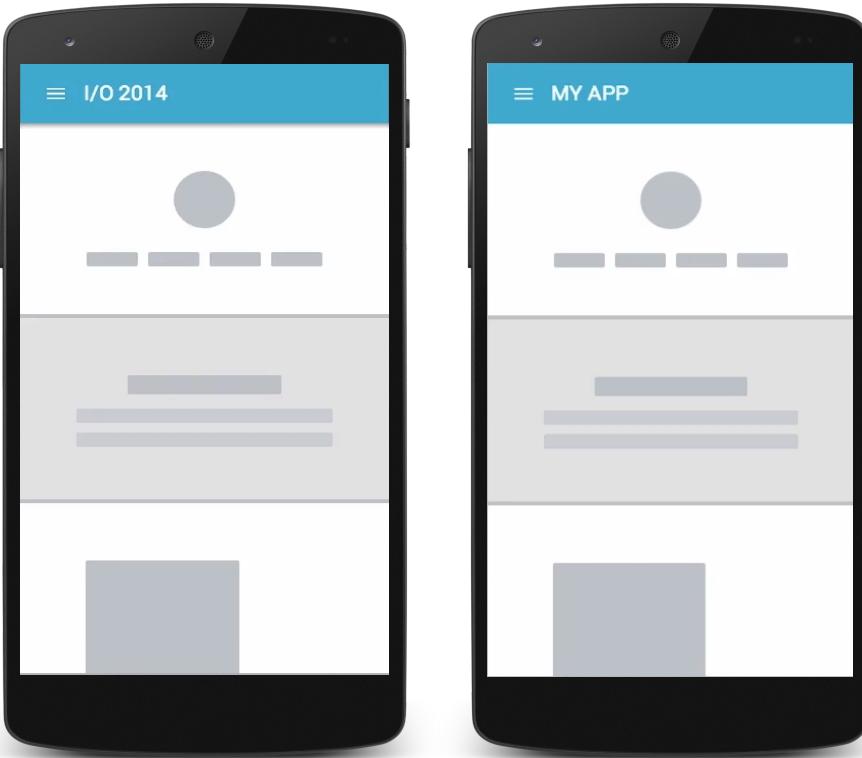


```
<link rel="import"  
      href="paper-toolbar.html">  
<link rel="import"  
      href="paper-header-panel.html">
```

```
<paper-header-panel flex>  
  <paper-toolbar>  
    <paper-icon-button icon="menu">  
    </paper-icon-button>  
    <div>I/O 2014</div>  
  </paper-toolbar>  
  <div class="content">...</div>  
</paper-header-panel>
```



Elements are configurable



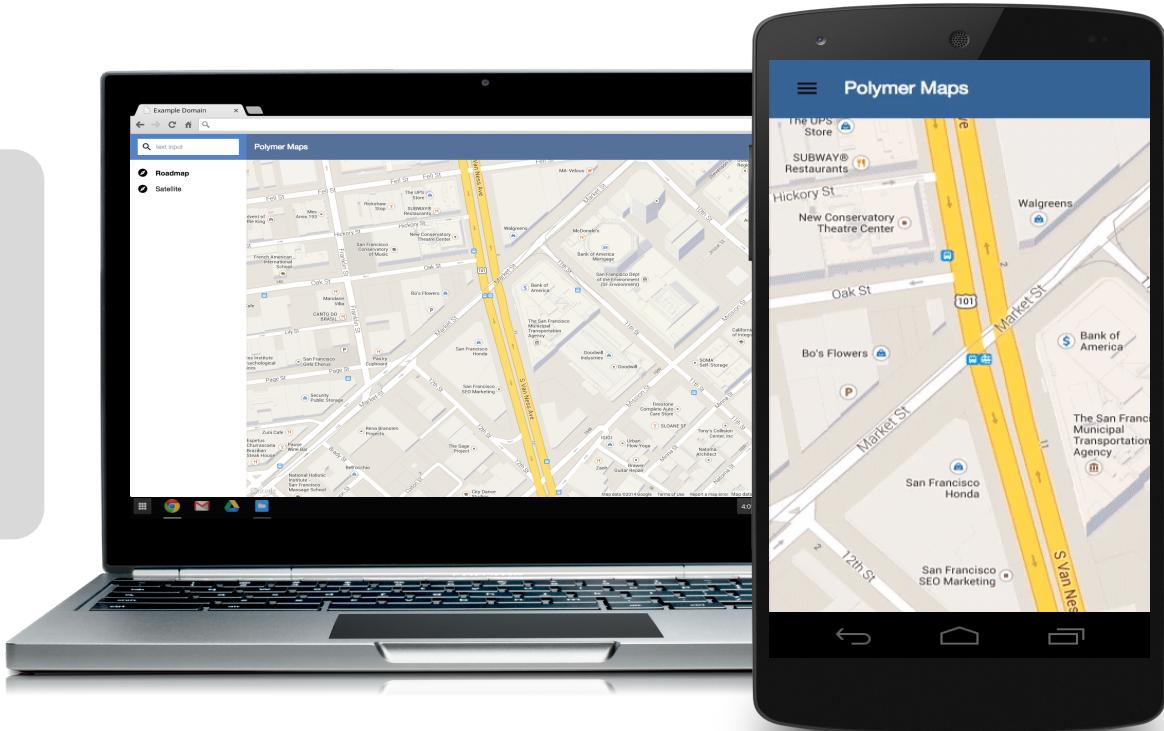
Toolbar will scroll with the page

```
<paper-header-panel mode="scroll" flex>
  <paper-toolbar>
    <paper-icon-button icon="menu">
    </paper-icon-button>
    <div>I/O 2014</div>
  </paper-toolbar>
  <div class="content">...</div>
</paper-header-panel>
```



Responsivity from the beginning

```
<paper-drawer-panel>
  <div drawer>
    Drawer panel...
  </div>
  <div main>
    Main panel...
  </div>
</paper-drawer-panel>
```



<paper-ripple>

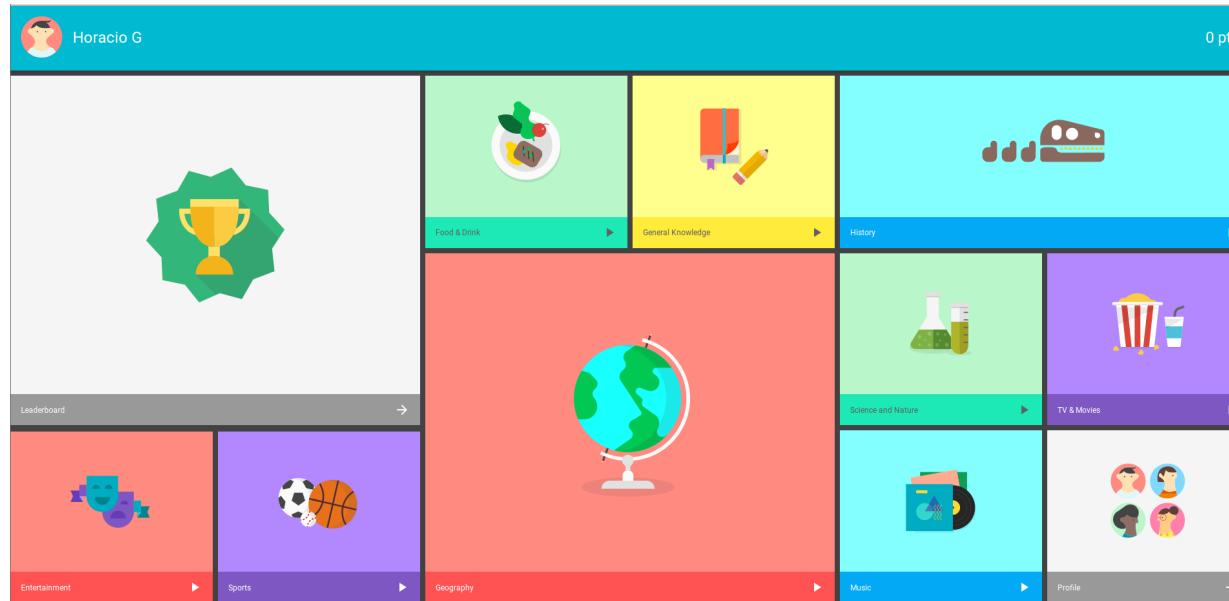


A reactive ink effect for indicating touch and mouse actions

```
<div class="card">
  
  <paper-ripple fit></paper-ripple>
</div>
```



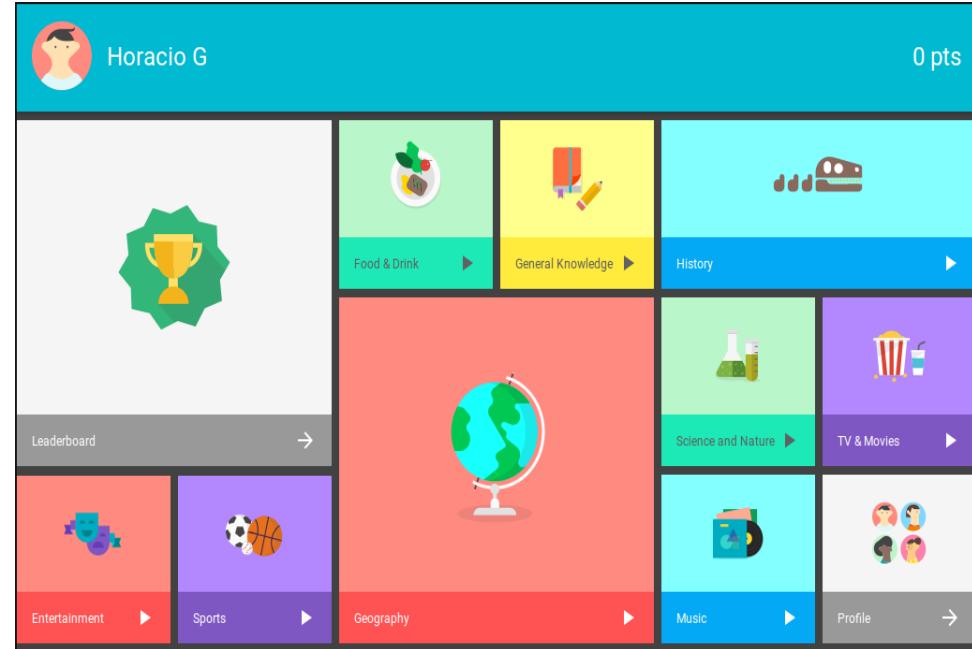
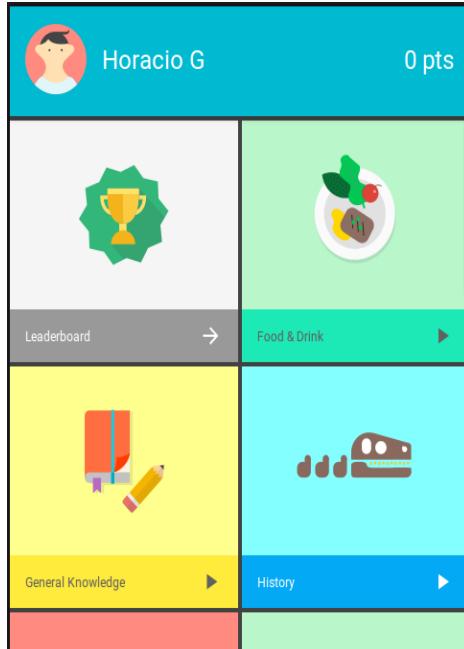
A full demo of Polymer Paper



<https://www.polymer-project.org/apps/topeka/>



Fully responsive



Gold Elements

For e-commerce use-cases



[Pinterest](#)



Polymer gold elements

The screenshot shows the Polymer Catalog interface. On the left, there's a sidebar with categories: Products, All Elements, Iron Elements, Paper Elements, Google Web Components, Gold Elements (which is currently selected and highlighted in grey), and Neon Elements. The main area is titled "Gold Elements (6)" and contains a table with six rows, each representing a gold element. The table has columns for Name, Description, Tags, and Action. The "Tags" column for all elements in this section is consistently listed as "input". The "Action" column for each row includes icons for edit, delete, and view.

Name	Description	Tags	Action
gold-cc-cvc-input	Provides an input field for a credit card cvc number	input	
gold-cc-expiration-input	A validating input for a credit card expiration date	input	
gold-cc-input	A credit card input field	input	
gold-email-input	An email input field	input	
gold-phone-input	A validating Input for a phone number	input	
gold-zip-input	An input field for a zip code	input	

Set of elements for e-commerce use-cases, like checkout flows



Google Web Components

From widgets to full apps



[LikeCool](#)



Google Web Components

The screenshot shows a catalog interface for Google Web Components. On the left, there's a sidebar with categories: Products, All Elements, Iron Elements, Paper Elements, Google Web Components (which is selected and highlighted in blue), Gold Elements, and Neon Elements. The main area is titled "Google Web Components (14)" and contains a table with the following data:

Name	Description	Tags	Action
firebase-element			
google-analytics	Encapsulates Google Analytics dashboard features into web components	analytics, dashboard,	
google-apis	Web components to load Google API libraries	apis,	
google-calendar	Web components for working with Google Calendar	calendar,	
google-castable-video	HTML5 Video Element with extended Chromecast	video, chromecast,	

From small widgets to full-scoped apps



Other available web components

The more, the merrier!



[KNP Labs](#)

#Webcomponents

@LostInBrittany



Mozilla Brick

mozillabrick Documentation Support Blog Search v2.0 Fork me on GitHub

DOCUMENTATION

- What Is Brick?
- Installation
- Introducing Brick 2.0
- Development

COMPONENTS

- brick-action
- brick-appbar
- brick-calendar [Coming Soon]
- brick-deck
- brick-flipbox
- brick-form
- brick-layout
- brick-menu
- brick-tabbar
- brick-storage-indexdb

STAY UP TO DATE

What Is Brick?

Brick is a collection of UI components designed for the easy and quick building of web application UIs. Brick components are built using the Web Components standard to allow developers to describe the UI of their app using the HTML syntax they already know.

Suggest Edits

Keep Up To Date

Twitter @mozbrick Github mozbrick/brick

Installation

Brick can be installed using the Bower package manager:

```
bower install mozbrick/brick
```

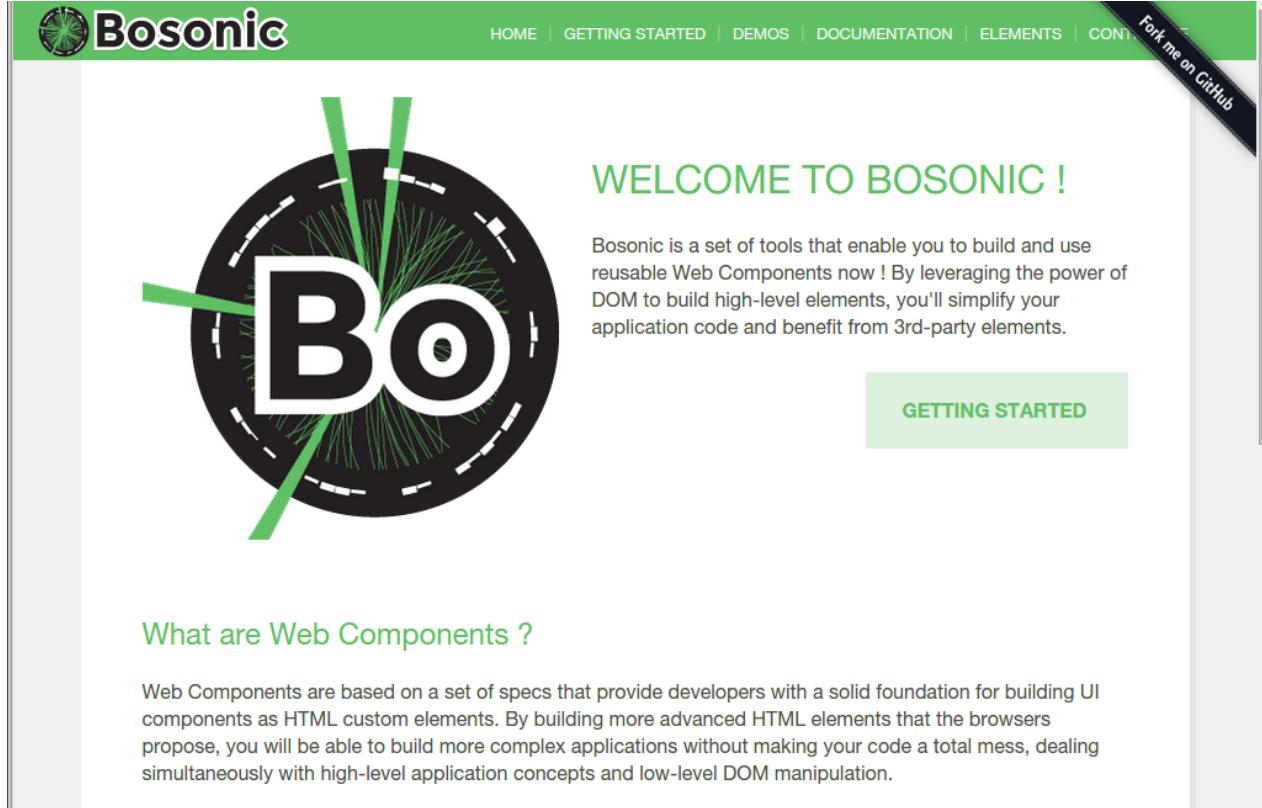


#Webcomponents

@LostInBrittany



Bosomic



The screenshot shows the Bosonic website homepage. At the top, there's a green navigation bar with the word "Bosonic" in white. Below it is a large circular logo featuring the letters "BO" in white on a black background, surrounded by green sprouts and dashed lines. To the right of the logo, the text "WELCOME TO BOSONIC !" is displayed in green. Below this, a paragraph explains what Bosonic is: "Bosonic is a set of tools that enable you to build and use reusable Web Components now ! By leveraging the power of DOM to build high-level elements, you'll simplify your application code and benefit from 3rd-party elements." A green button labeled "GETTING STARTED" is located below this text. In the top right corner of the page, there's a dark ribbon with the text "Fork me on GitHub".

What are Web Components ?

Web Components are based on a set of specs that provide developers with a solid foundation for building UI components as HTML custom elements. By building more advanced HTML elements that the browsers propose, you will be able to build more complex applications without making your code a total mess, dealing simultaneously with high-level application concepts and low-level DOM manipulation.



Conclusion

That's all folks!



dcplanet.fr



The code samples (W.i.P)

- You can get current version on github
 - branch [web2day](#)

<https://github.com/LostInBrittany/webcomponents-polymer-talk>



You want to know more?

- [W3C's Introduction to Web Components](#)
- [HTML5 Rocks' Shadow DOM 101](#) & [HTML5 Rocks' Shadow DOM 201: CSS](#)
- [WebComponents.org's Introduction to Shadow DOM](#)
- [Polymer](#), [X-Tags](#), [Mozilla Brick](#)
- [MutationObserver](#) & [Object.observe](#)



Would you like to know more?

Then go to the
Cloud endpoints, Polymer, material design hand's on lab

DEVOXX WWW.DEVOXX.FR ACCUEIL PROGRAMME SPONSORS FAQ INSCRIPTION

Home > All Hand's on labs > Thursday >

Cloud endpoints, Polymer, material design: the Google stack, infinitely scalable, positively beautiful

Hand's on Labs

 Web, Mobile & UX

Paris 224M-225M Lab Thursday at 13:00 - 16:00

Google has been pushing the web forward for several years and designing cloud architectures for as long as it exists. Now it all comes together. In this lab you will use material design elements to design, develop and deploy an end-to end web application, front-end and back-end, ready to scale to millions of users.

You will learn to use the following technologies: - Google Cloud Endpoints (Java) and Cloud Datastore (used here with a web front-end but this part is also applicable to Android and iOS development) - Polymer and Web Components (for mobile and desktop) - The Paper Elements for Polymer (material design)

Mandatory Installs prior to lab:

+JDK 7 or 8 (200MB) +Eclipse (4.4 - Luna, 160MB)) +“Google Plugin for Eclipse” and “Google App Engine SDK” to install into Eclipse through “Help > Install New Software ...” from source “<http://dl.google.com/eclipse/plugin/4.4>” (157MB) +Bower (optional but recommended) +Lab starter code from Git: “git clone <https://github.com/martin-gorner/endpoints-polymer-material-tutorial/>” (90MB)

 Martin Gorner

Martin Gorner, Google Developer Relations. Martin se passionne pour la science, la technologie, l'informatique, les algorithmes et tout ce qui s'en rapproche. Après avoir obtenu son diplôme d'ingénieur à Mines Paris-Tech, Martin a commencé sa carrière dans le groupe "computer architecture" chez ST Microelectronics. Il a ensuite passé les 11 années suivantes dans le domaine naissant des livres électroniques, d'abord avec la start-up mobipocket.com, qui est ensuite devenue la partie logicielle du Kindle d'Amazon et ses versions mobiles. Il a rejoint Google en 2011.

 Horacio Gonzalez

Spaniard lost in Brittany, unconformist coder, Java craftsman, dreamer and all-around geek



Thank you !





startups funding prospective
startup contest ^{html5}
cloud mobility DIY design

Web Components avec Polymer

