Web Components avec Polymer

Horacio Gonzalez

@LostInBrittany

Cityzen Data
http://cityzendata.com

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

Introduction

Because I love to tell old stories

Warning : I’m a web developer

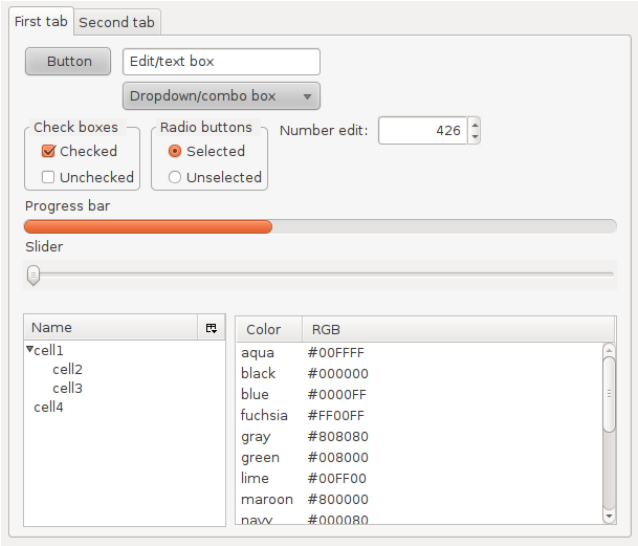
And when I tell stories, I do it from the webdev POV

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

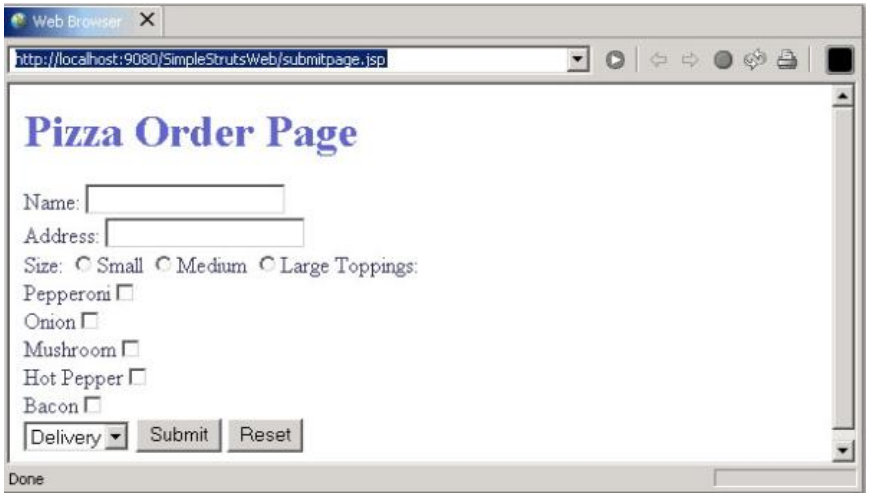


In thick client you create your UI by assembling widgets

Image : Wikipedia

Web development was unlike thick-client

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

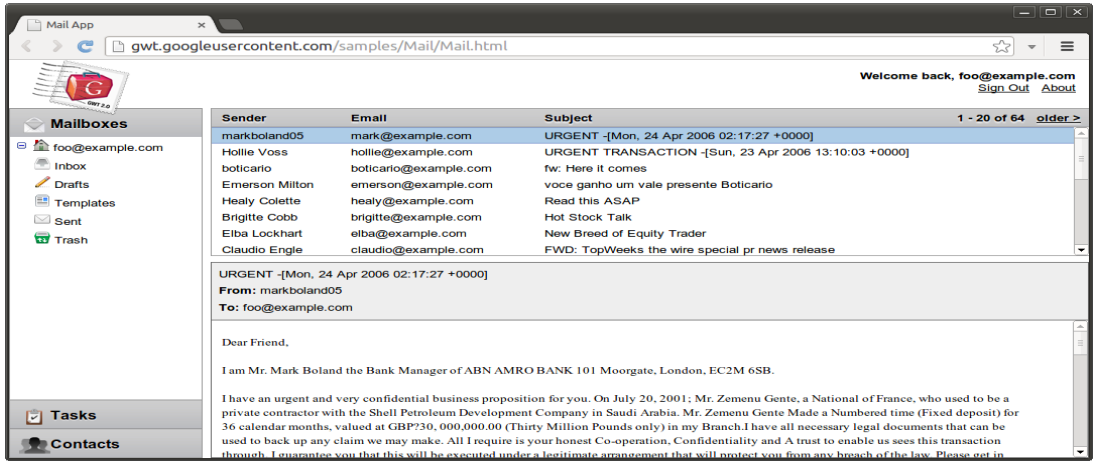


Web apps were page oriented

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

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

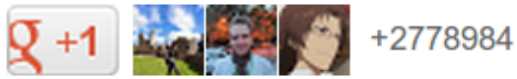
Reinventing the wheel... and this time making it round

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

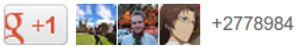
- And what I would like is simple

```
<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>
```

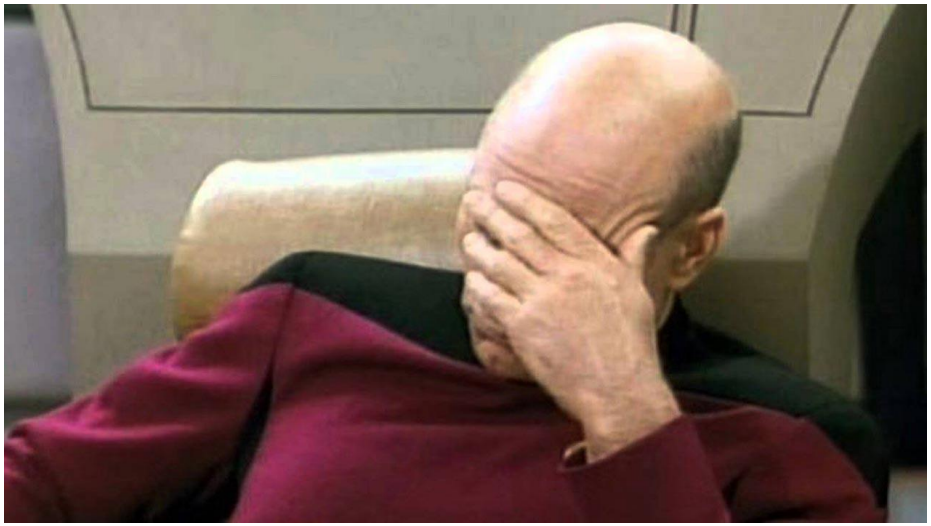


Image : Star Trek the Next Generation

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>
```

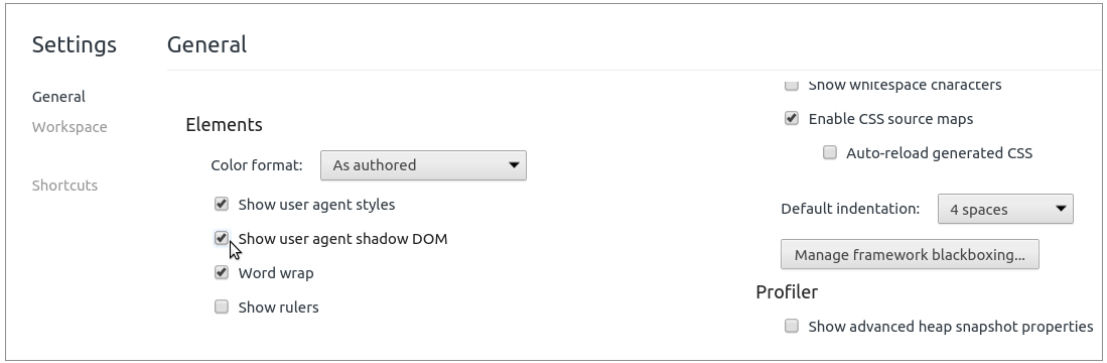
Browsers hide DOM sub-trees for standard components
They have a public interface and hidden inner code

Image: Springfield Punx

My name is DOM, Shadow DOM

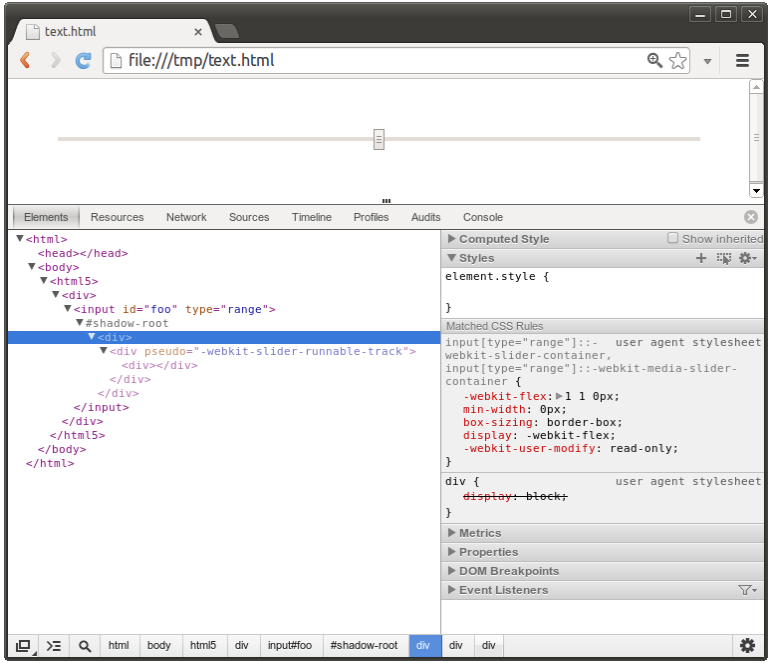
- Shadow DOM: ability of the browser to
  - Include a DOM subtree into the rendering
  - But not into the main document DOM tree

- In Chome you can see the Shadow DOM
  - By activating the option in Inspector



Looking into the Shadow

- For the slider :



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

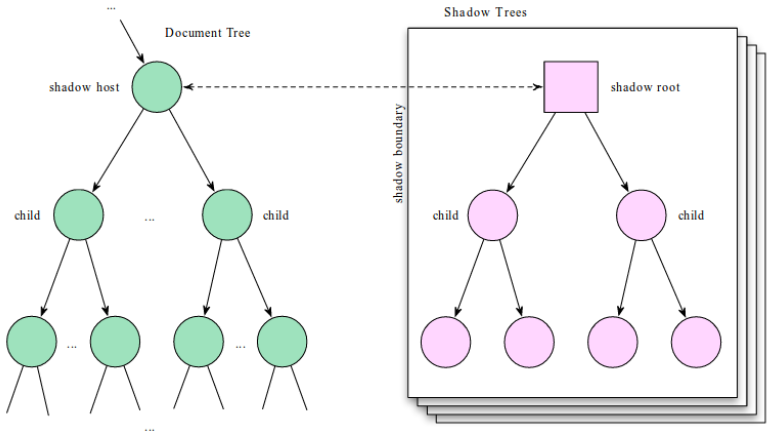


Image: W3C

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!</h4>";
</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://lostinbrittany.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

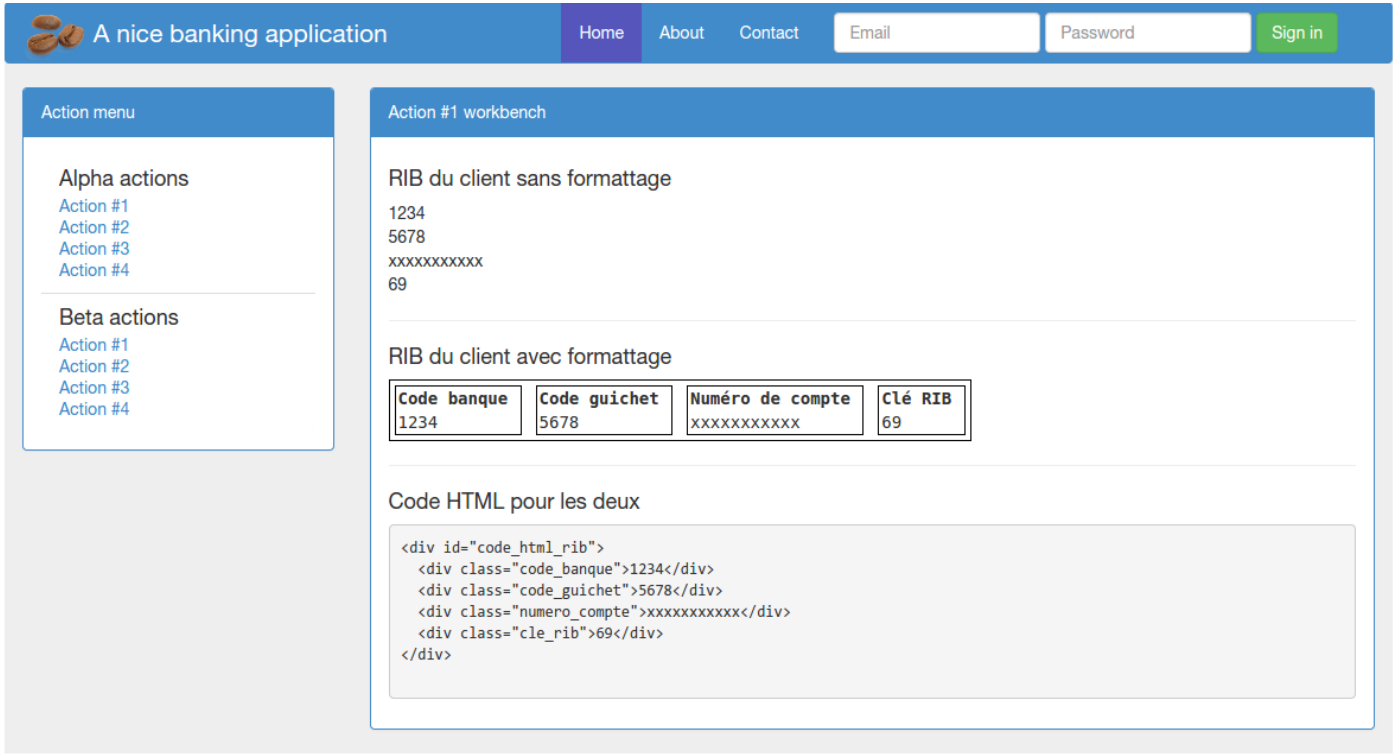
- Styling the host element : @host

```
<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>
```

Example


A nice banking application
Home
About
Contact
Email
Password
Sign in
Action menu
Alpha actions
Action #1
Action #2
Action #3
Action #4
Beta actions
Action #1
Action #2
Action #3
Action #4
Action #1 workbench
RIB du client sans formattage
1234
5678
xxxxxxxxxx
69
RIB du client avec formattage
Code banque
1234
Code guichet
5678
Numéro de compte
xxxxxxxxxx
Clé RIB
69
Code HTML pour les deux

1234
5678
xxxxxxxxxx
69

Custom Elements

Elemental mayhem !

The Brick Blogger

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">
    <img src="http://webcomponents.org/img/logo.svg">
    <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, polyfillification[1], and implementation.

	Specced	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

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

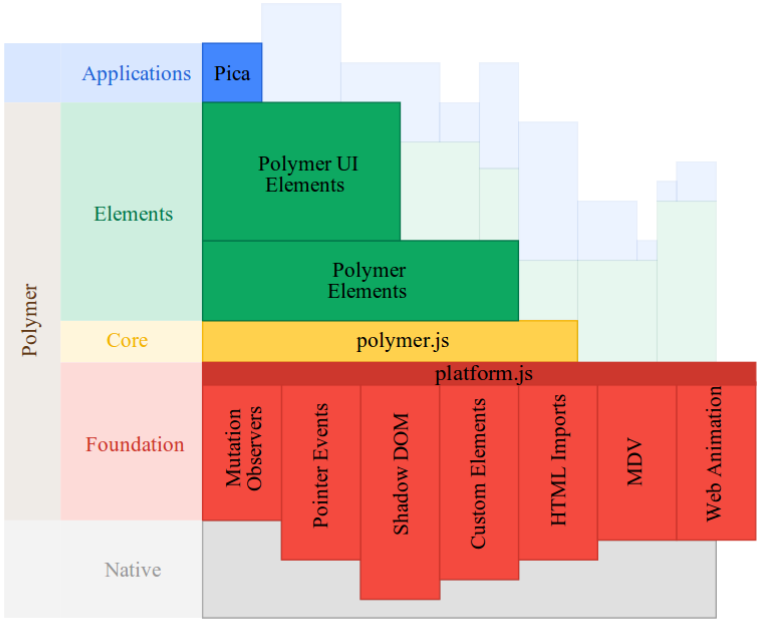
Version 1.0 just released last week at Google IO

Oh yeah!

Polymer

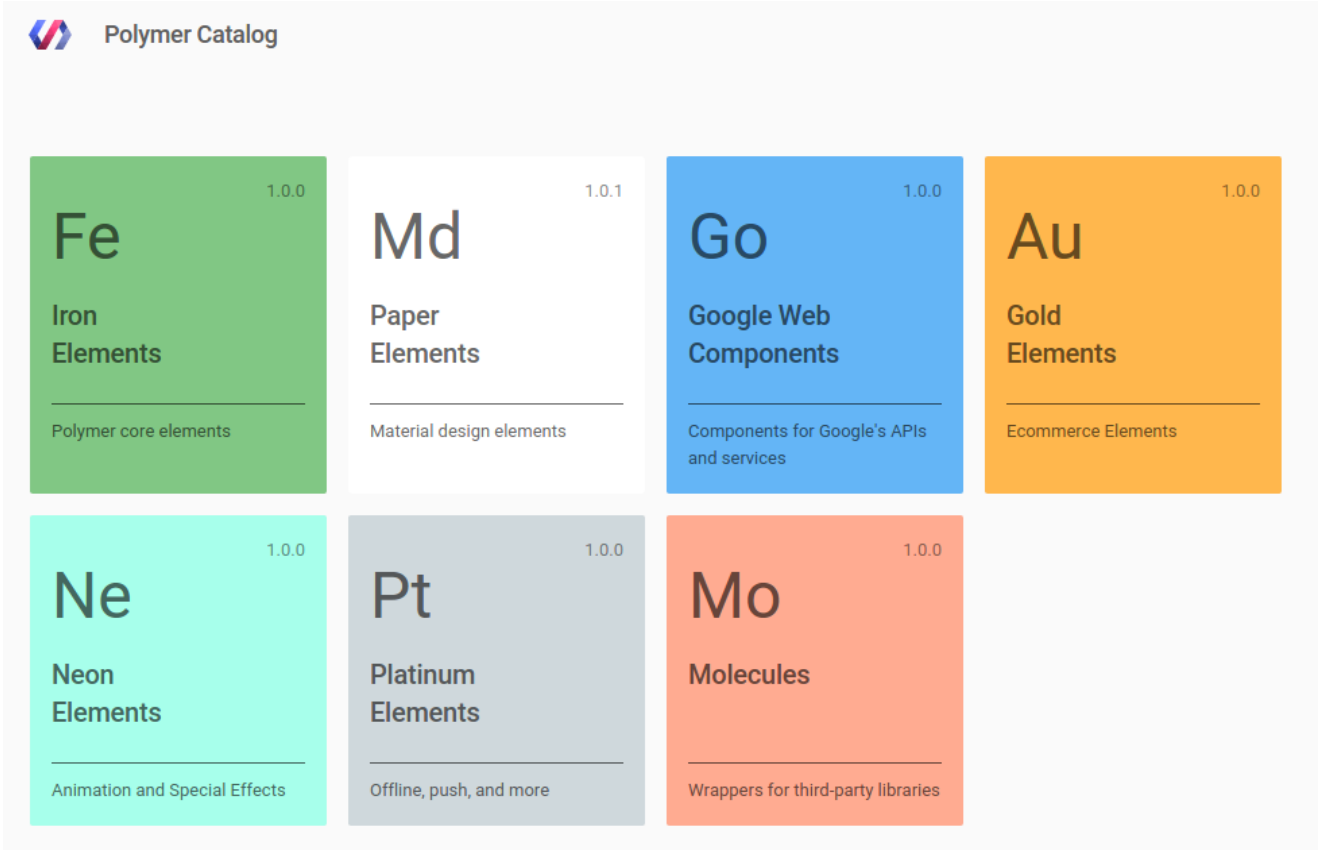
- What does it means ?
  - Polymer is comprised of two efforts :
    - A paper platform to give Web Component capabilities to modern browsers
      - Shims for all modern browsers
      - Shared with Mozilla x-tag project
    - A next-generation web framework built upon this core platform
      - Called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

Polymer



Architectural Diagram

Polymer elements



Polymer

- Principes:
  - 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

- Principes:
    - 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?

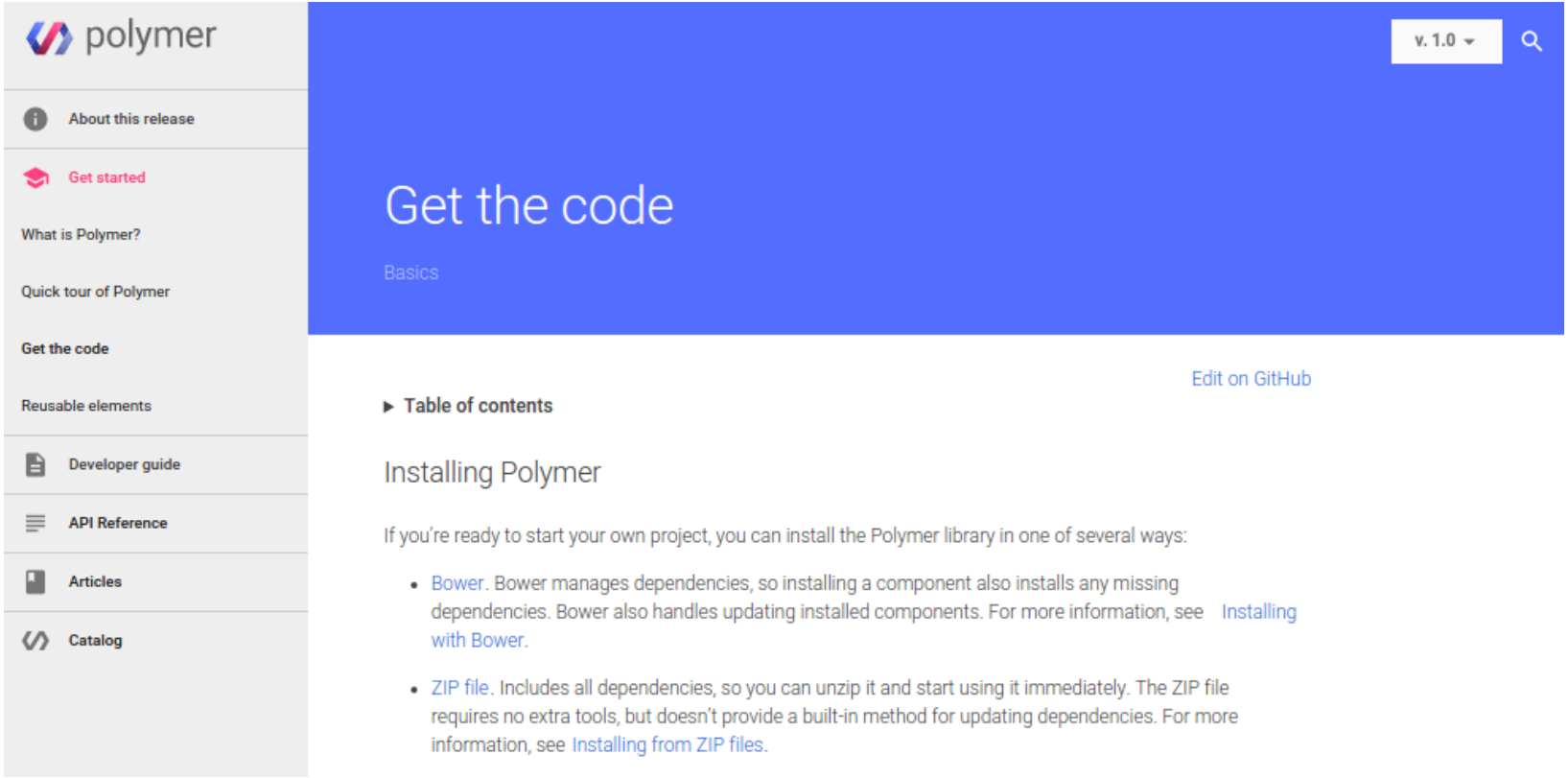
- AngularJS directives allow to create custom elements
  - Same spirit, different implementation

```
<!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

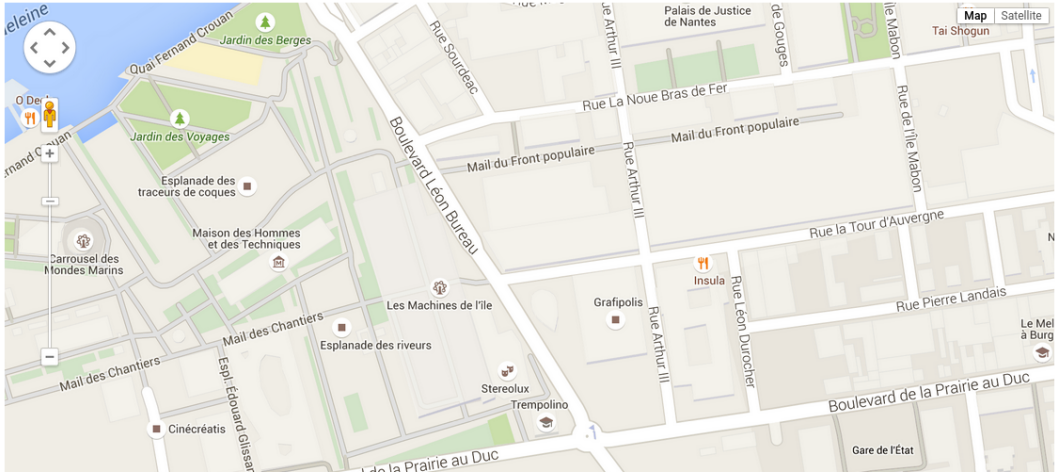


Step 2 - Use an element

```
<!-- 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="47.205403" longitude="-1.5631069" 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

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>
```

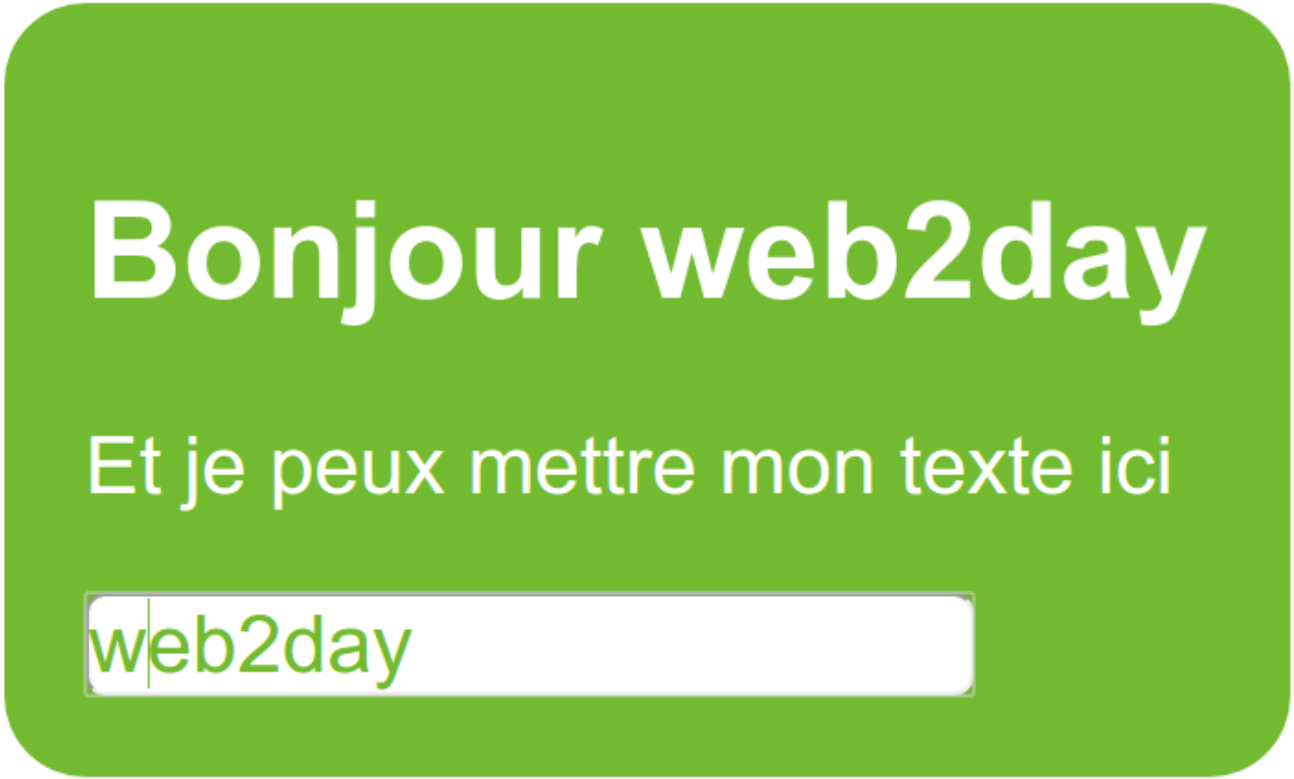

Step-4.2 : enjoy

Step-4.1 : Two-ways databinding

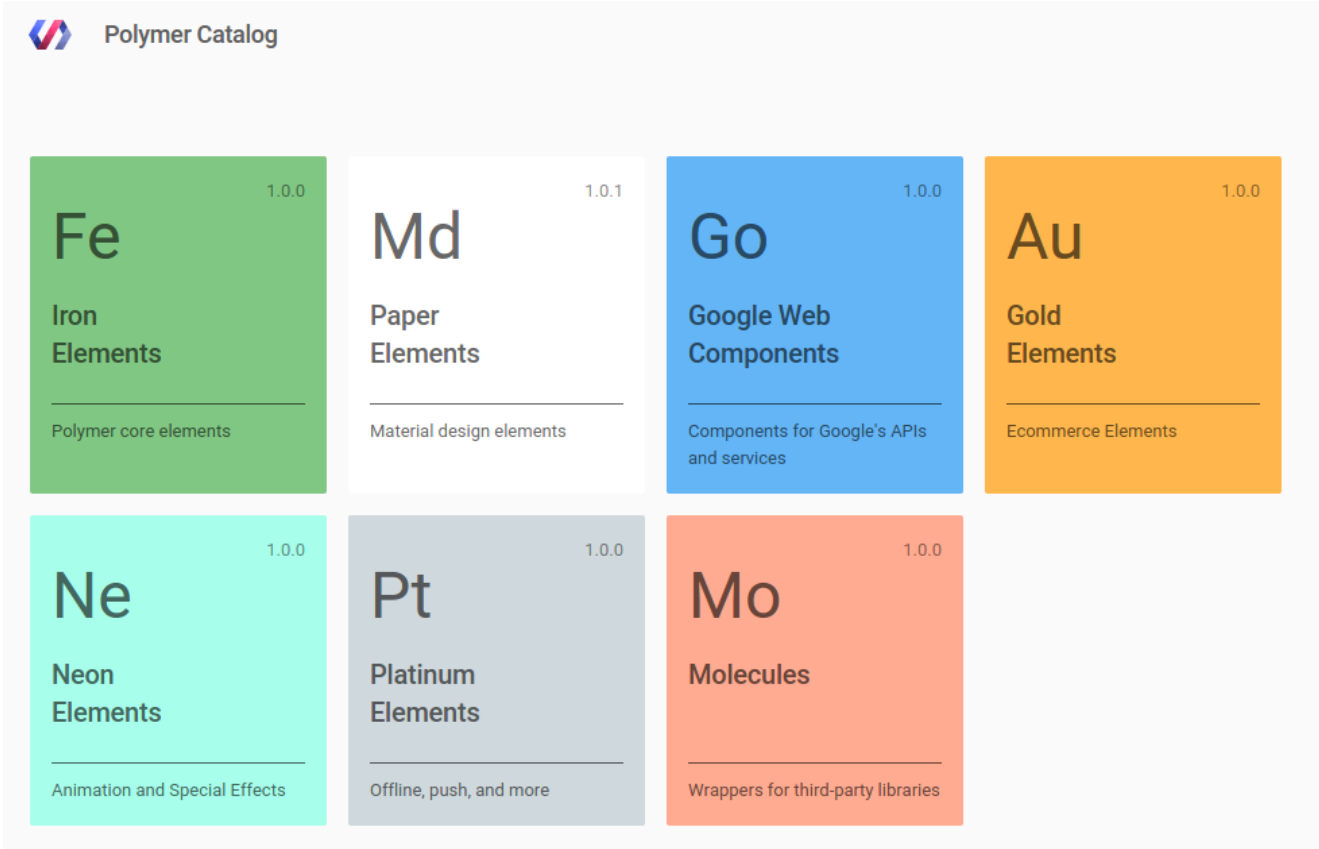
```
<dom-module id="x-web2day">
 <template>
   <style>/*...*/</style>
   <div id="box">
      <h1>Bonjour <span>{{whoami}}</span></h1>
      <p><content></content></p>
      <!-- iron-input exposes a two-way bindable input value -->
      <input is="iron-input" bind-value="{{whoami}}" placeholder="Your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

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

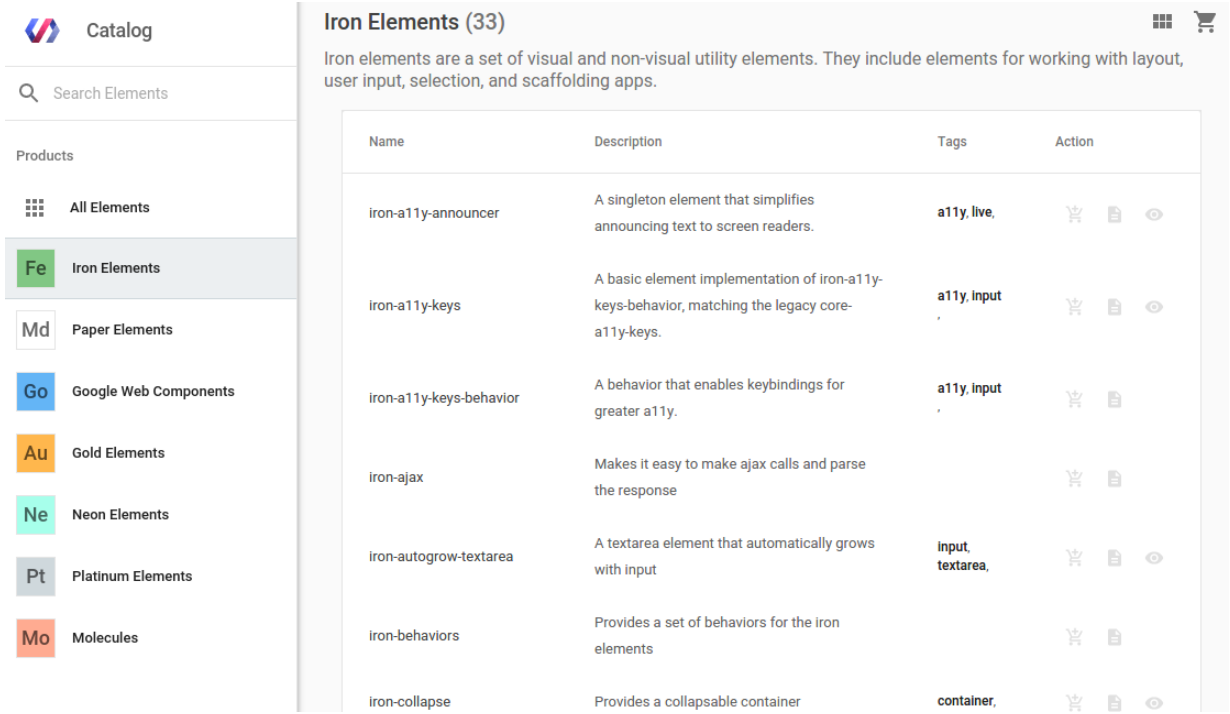

Step-5.2 : enjoy



Catalog of elements



Polymer iron elements



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

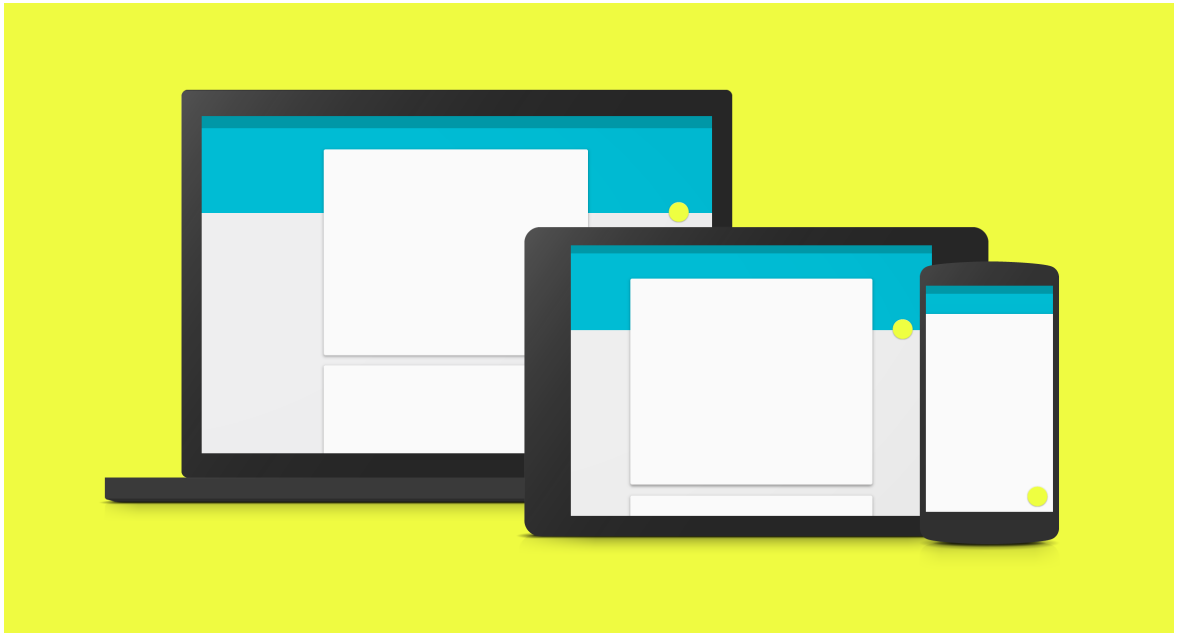
Nice visual effects!

ANDROID 5
Lollipop

It's like a flat design, but in 3D!

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

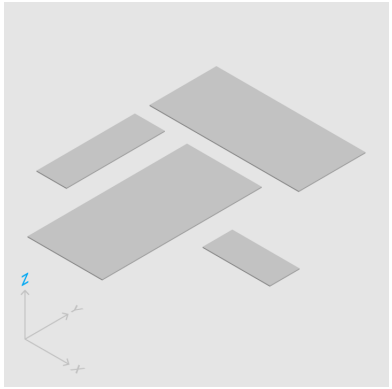


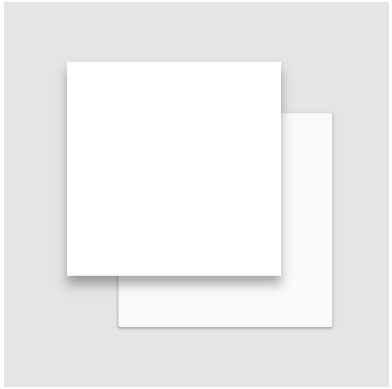
RESPONSIVE!
RESPONSIVE!

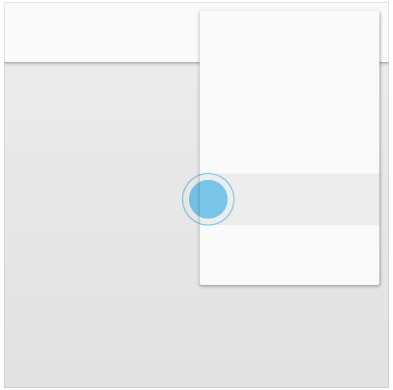
Visual language joining good design with technical innovation to create an unified experience across platforms and device sizes

What's material design?

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

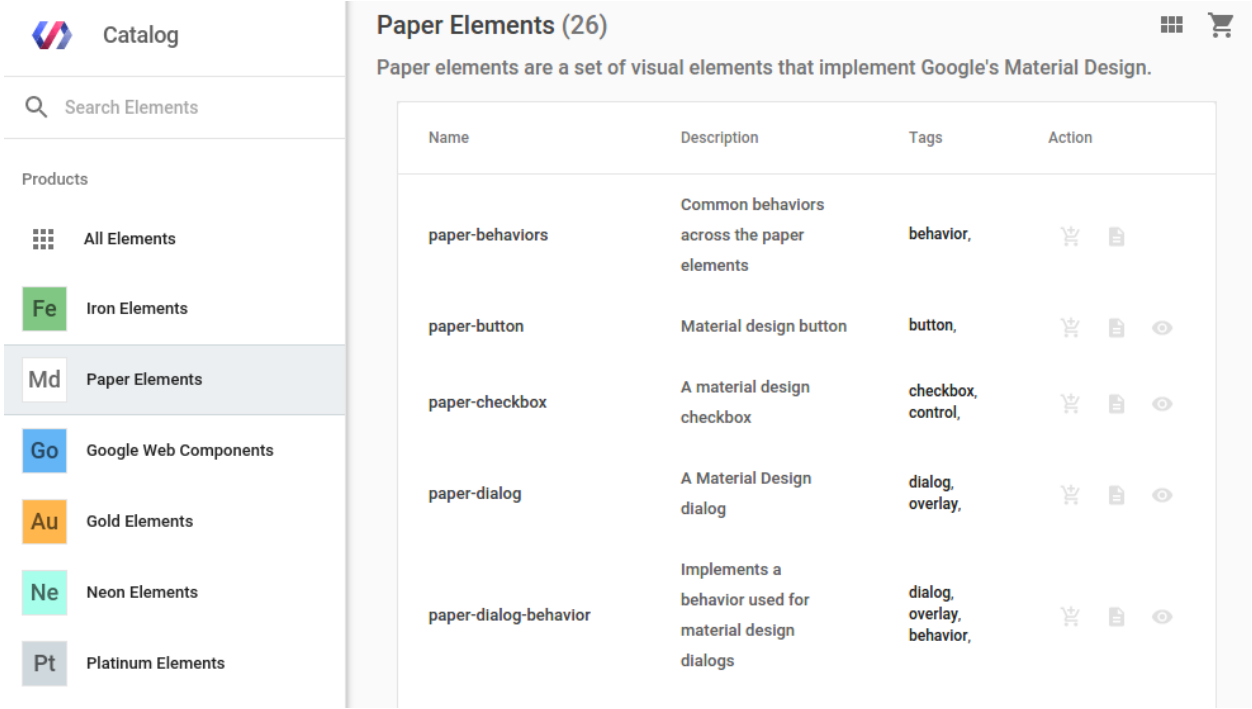






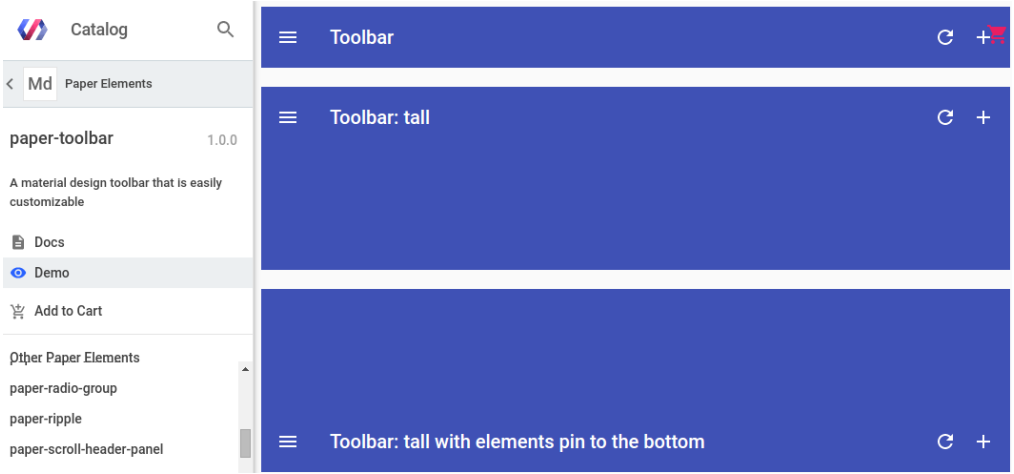
http://www.google.com/design/spec/material-design/

Polymer paper elements



Set of visual elements that implement Material Design

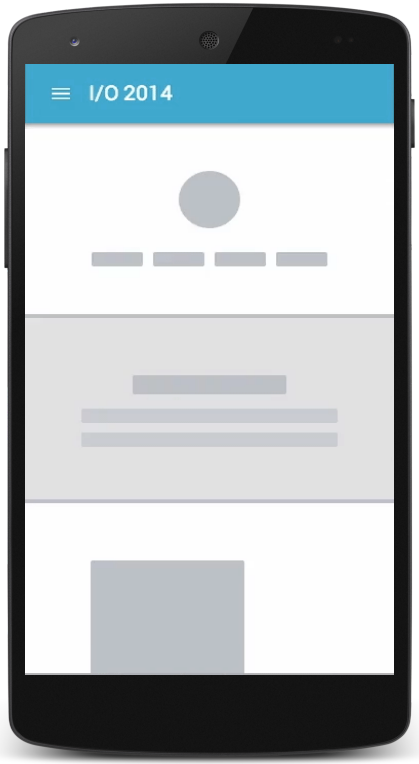
<paper-toolbar>




```
<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>
```

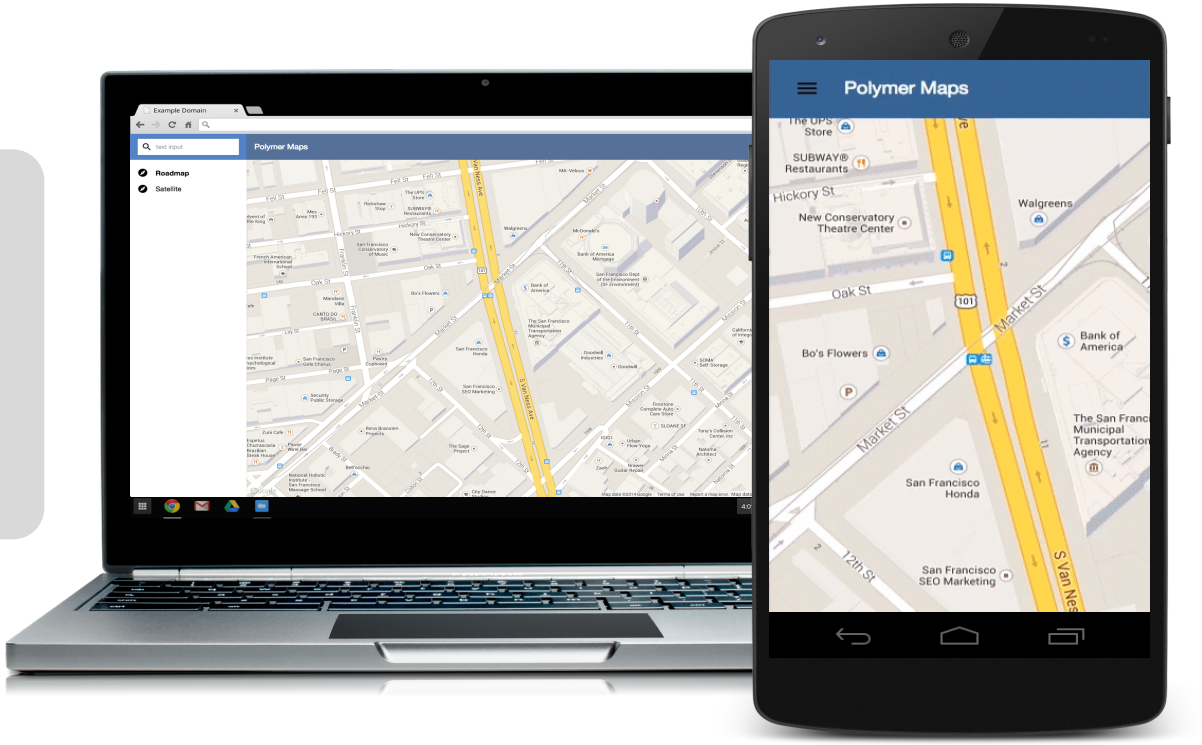

Elements are configurable

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>
```

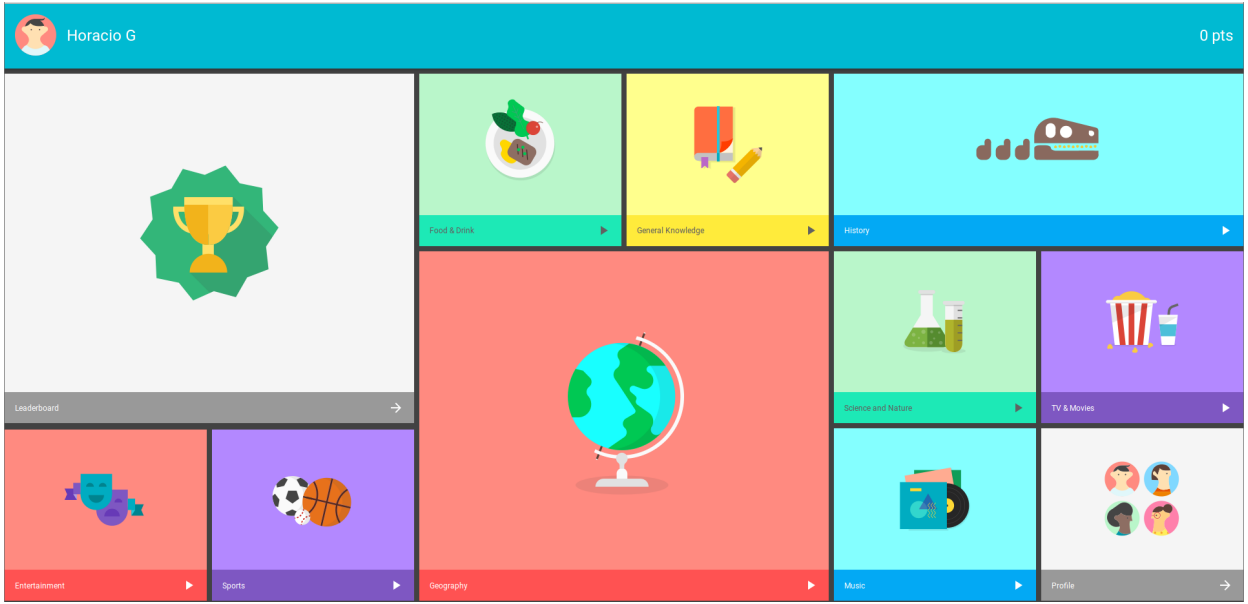


<paper-ripple>

A reactive ink effect for indicating touch and mouse actions

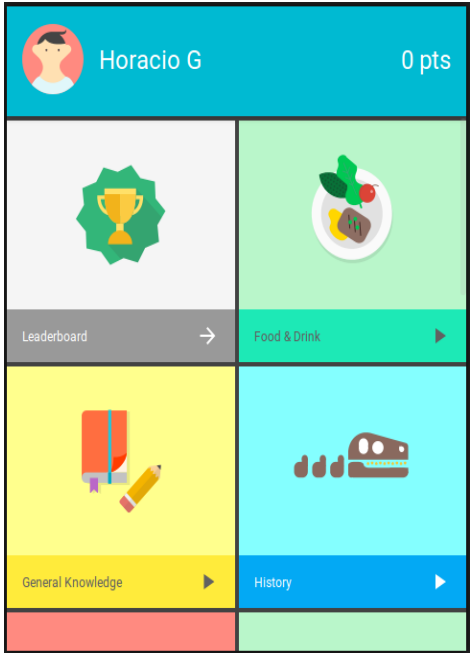
```
<div class="card">
  <img src="learning.svg">
  <paper-ripple fit></paper-ripple>
</div>
<div class="card">
  <img src="science.svg">
  <paper-ripple fit></paper-ripple>
</div>
<div class="card">
  <img src="food.svg">
  <paper-ripple fit></paper-ripple>
</div>
<div class="card">
  <img src="earth.svg">
  <paper-ripple fit></paper-ripple>
</div>
```

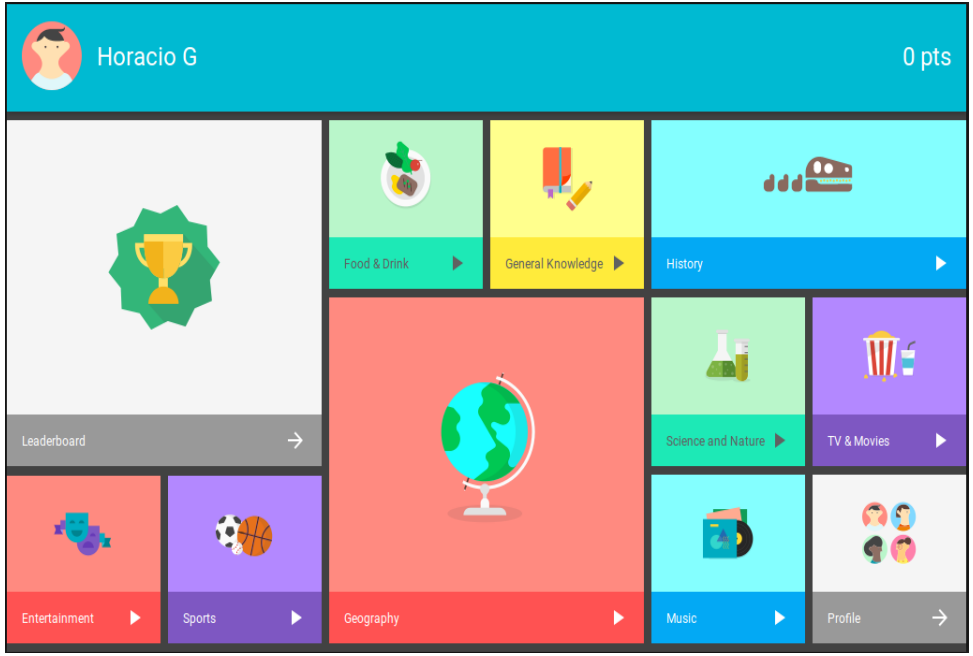

A full demo of Polymer Paper



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

Fully responsive



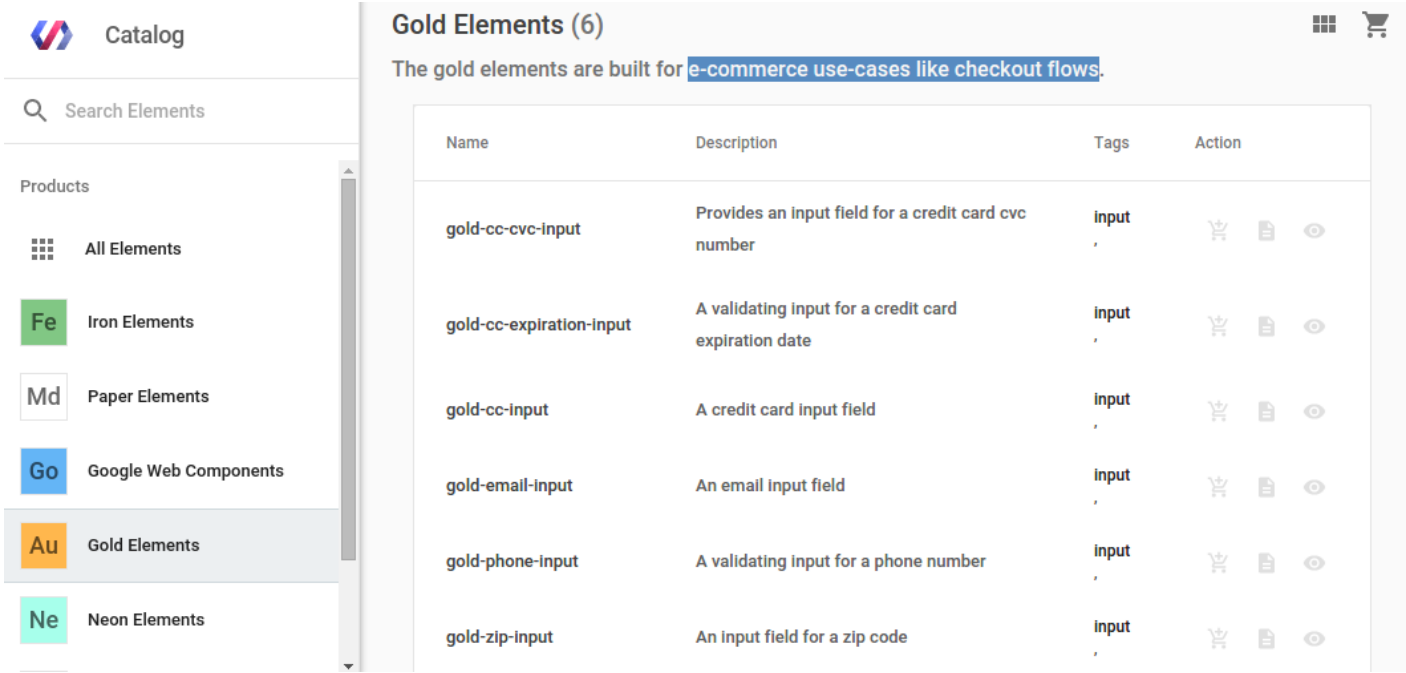


Gold Elements

For e-commerce use-cases

Pinterest

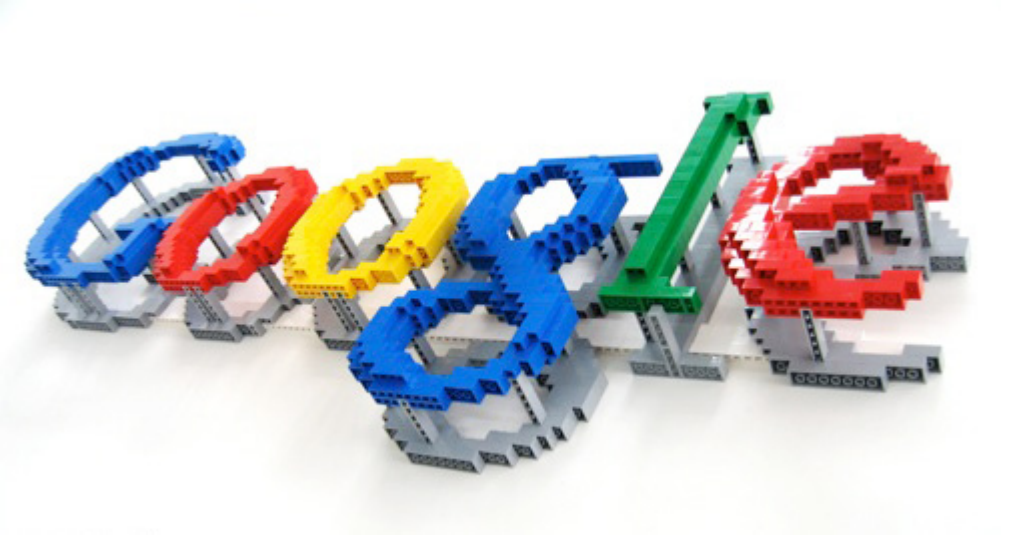
Polymer gold elements



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

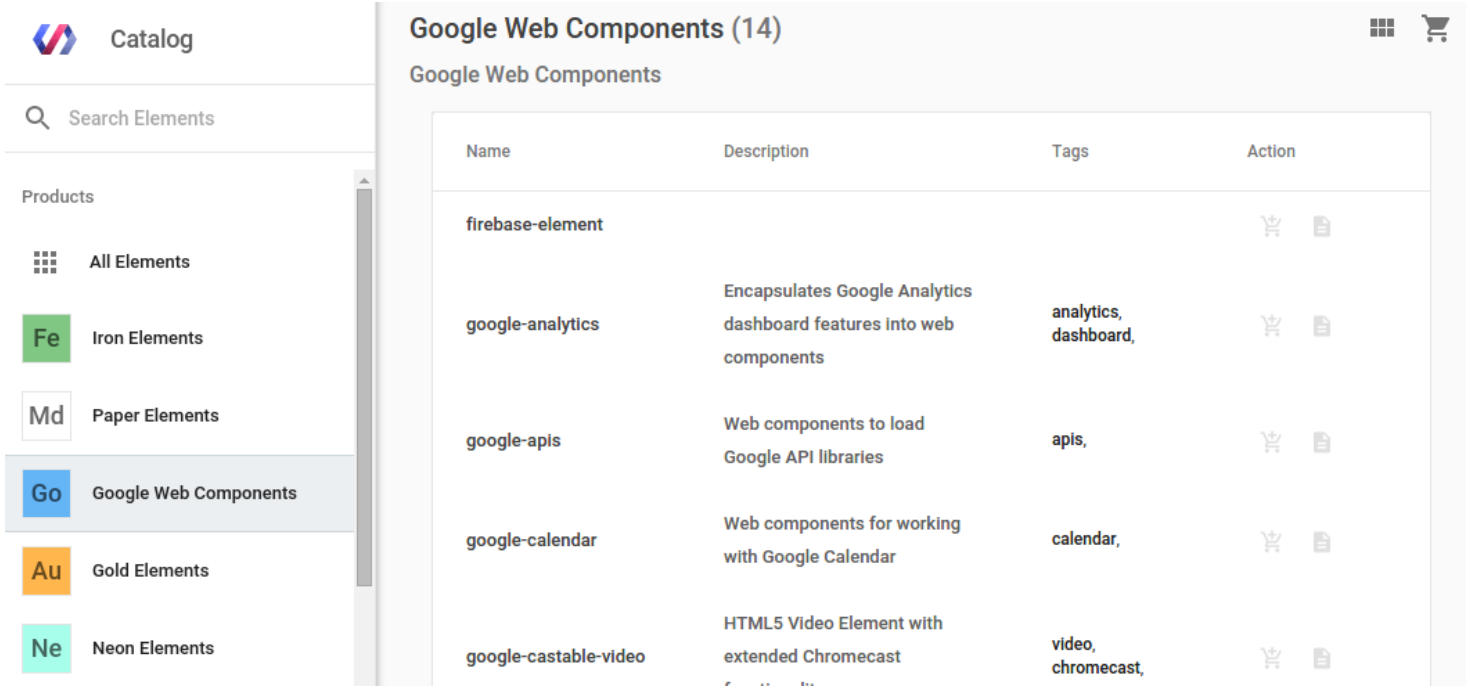
Google Web Components

From widgets to full apps



LikeCool

Google Web Components



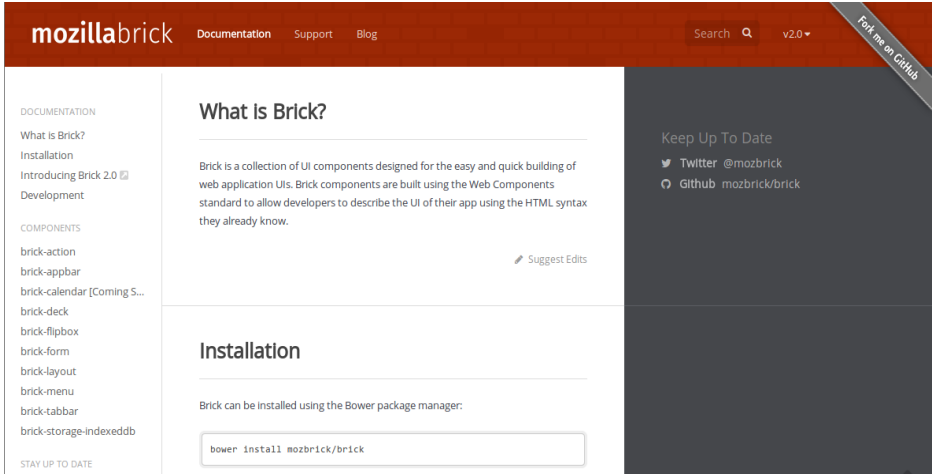
From small widgets to full-scoped apps

Other available web components

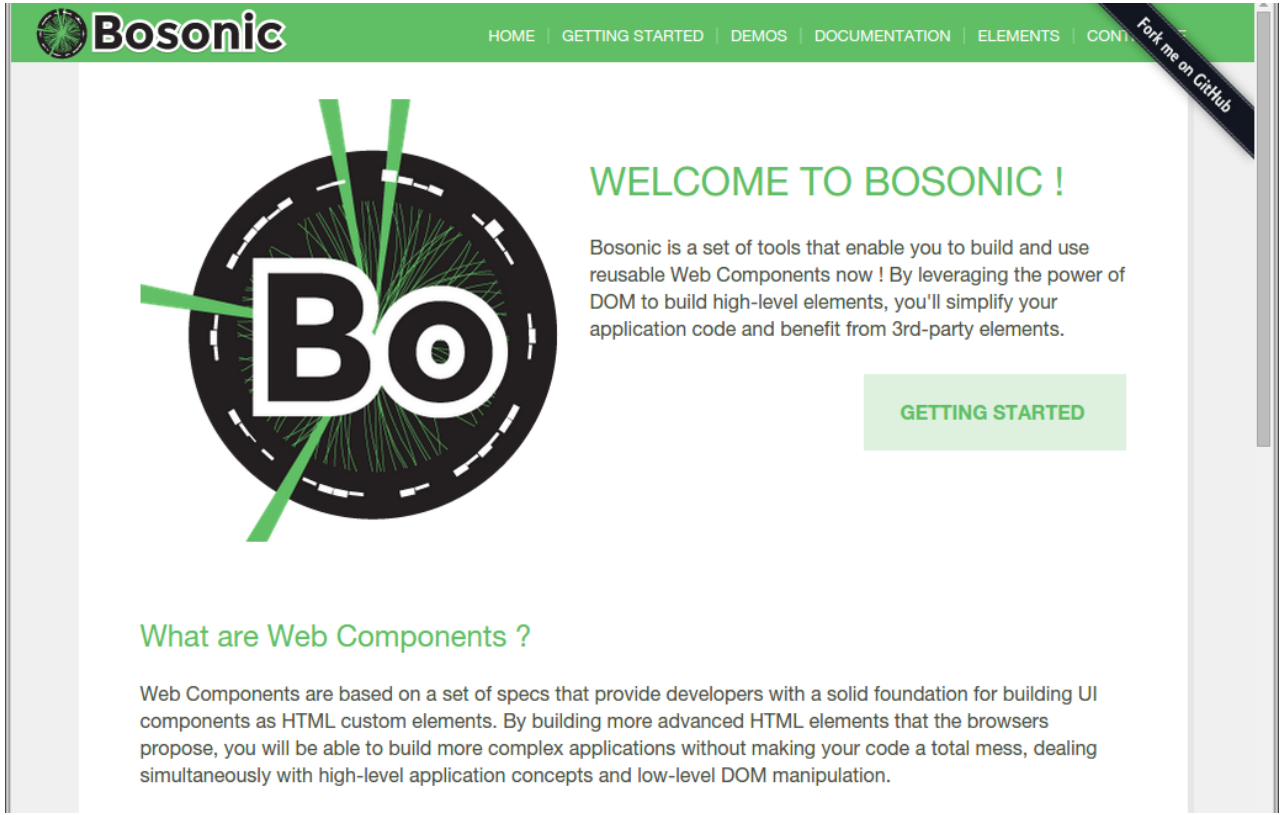
The more, the merrier!

KNP Labs

Mozilla Brick



Bosomic


Bosonic
HOME | GETTING STARTED | DEMOS | DOCUMENTATION | ELEMENTS
Fork me on GitHub
WELCOME TO BOSONIC !
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.
GETTING STARTED
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

Thank you !


Thank
you
Merci
Gracias
Arigato
bolzin
Tashakkur
Shukuria
Shukria
Grazie
Brigyan
Juspaxar
Mehrbani
atu
Dankscheen

Web Components avec Polymer