



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

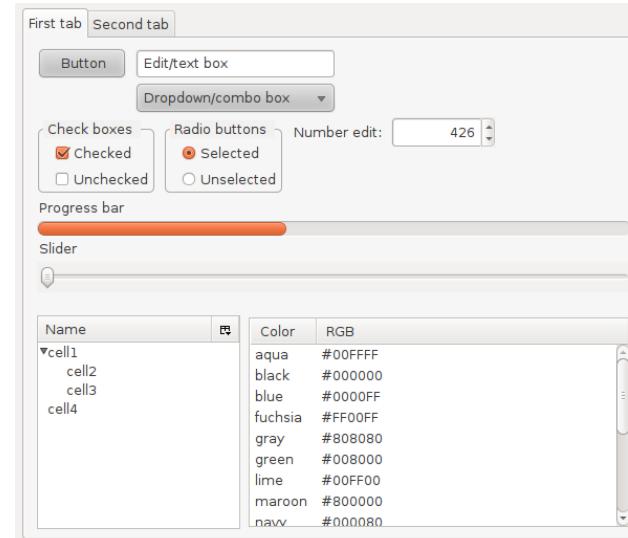
Image : TylkoMrok.pl



At the beginning we had the thick client

- Widget : basic building blocks of your UI

- Encapsulation
- Reutilisation



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 form titled "Pizza Order Page". The form includes fields for "Name" and "Address", a radio button group for "Size" (Small, Medium, Large), and a checkbox group for "Toppings" (Pepperoni, Onion, Mushroom, Hot Pepper, Bacon). At the bottom are "Delivery" dropdown, "Submit", and "Reset" buttons.

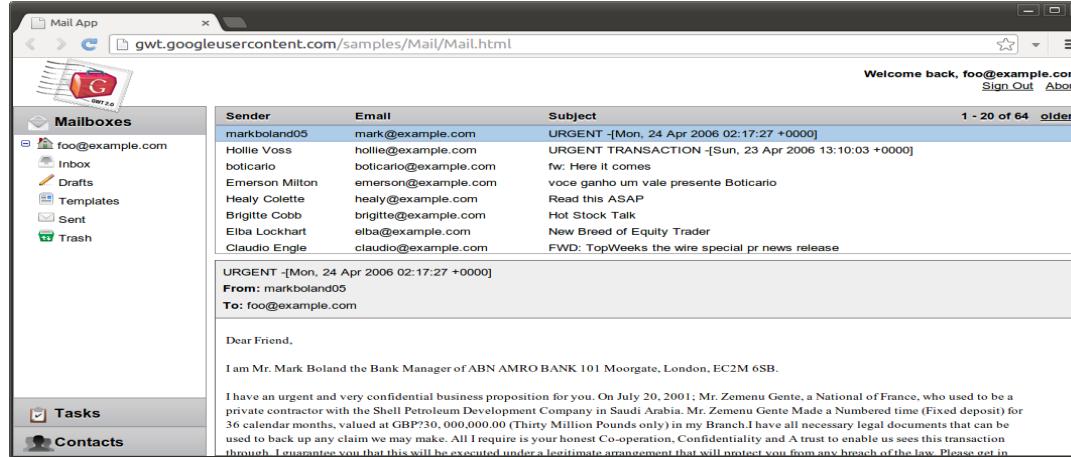
Web apps were page oriented

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

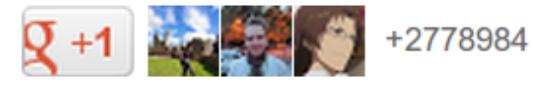


Example : the Google+ button

- 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>
```

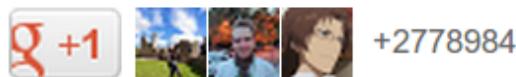
```
<!-- Place this tag after the last +1 button tag. -->
<script type="text/javascript">
  (function() {
    var po = document.createElement('script');
    po.type = 'text/javascript';
    po.async = true;
    po.src = 'https://apis.google.com/js/plusone.js';
    var s = document.getElementsByTagName('script')[0];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

#Webcomponents

@LostInBrittany



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>`

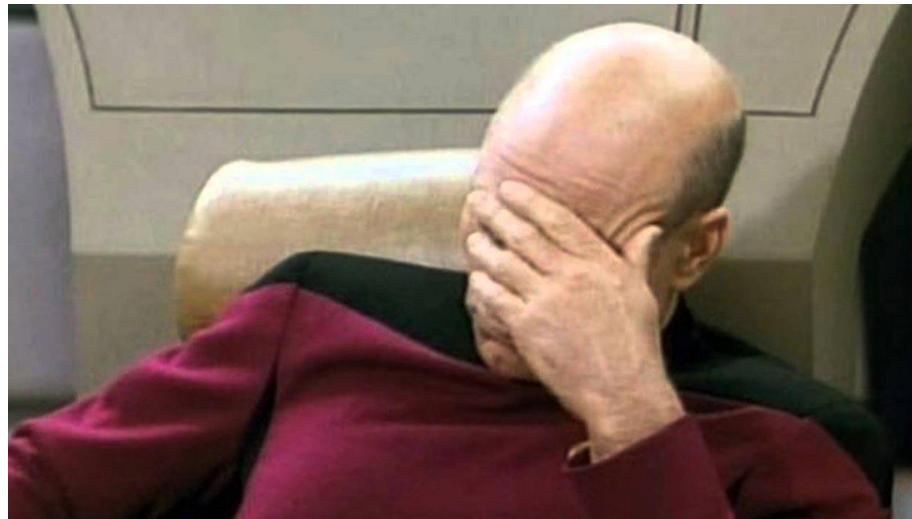


Image : Star Trek the Next Generation



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

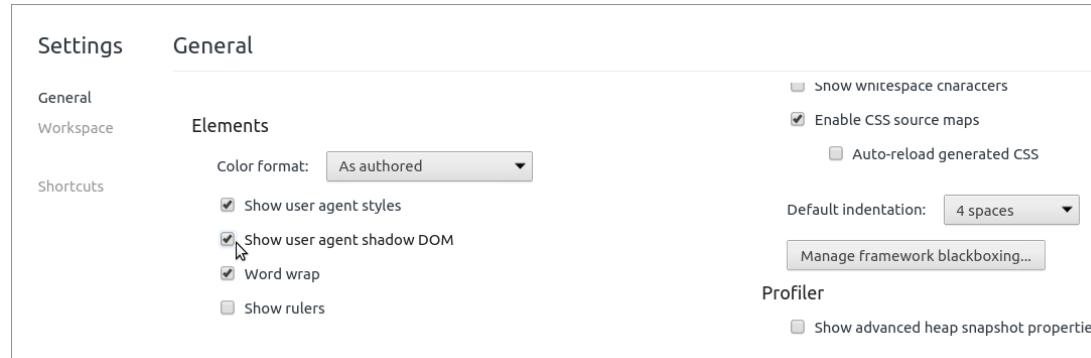


Image: Springfield Punx



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 a browser window titled "text.html" displaying a simple range slider. The developer tools are open, specifically the Element tab of the DevTools sidebar. The DOM tree on the left shows the following structure:

```
<html>
  <head>
  <body>
    <div>
      <input id="foo" type="range">
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

The right pane displays the "Computed Style" and "Styles" sections. The "Computed Style" section lists properties like `display: block`. The "Styles" section shows the following CSS rules:

```
element.style { }  
Matched CSS Rules  
input[type="range"]::-webkit-slider-runnable-track {  
  user agent stylesheet  
  -webkit-slider-container,  
  -webkit-media-slider-container {  
    -webkit-flex: 1 1 0px;  
    min-width: 0px;  
    box-sizing: border-box;  
    display: -webkit-flex;  
    -webkit-user-modify: read-only;  
  }  
  div {  
    user agent stylesheet  
    display: block;  
  }
```



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

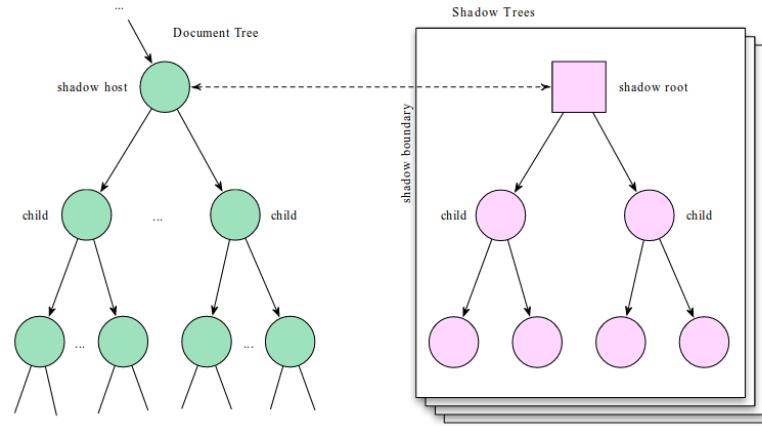


Image: [W3C](#)



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

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 formattege

1234
5678
xxxxxxxxxx
69

RIB du client avec formattege

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 magnifying glass icon. The main header features a large orange logo consisting of two interlocking wrenches forming a circular shape, positioned above the text "WebComponents.org" and the subtitle "a place to discuss and evolve web component best-practices". Below the header, there are three main sections: "WHAT?", "SPECS", and "ARTICLES". The "WHAT?" section contains text about the purpose of the site and a "BROWSER SUPPORT" chart. The "SPECS" section lists "WEB COMPONENTS", "CUSTOM ELEMENTS", and "HTML IMPORTS". The "ARTICLES" section features an image of a child flexing their muscles and a summary of "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

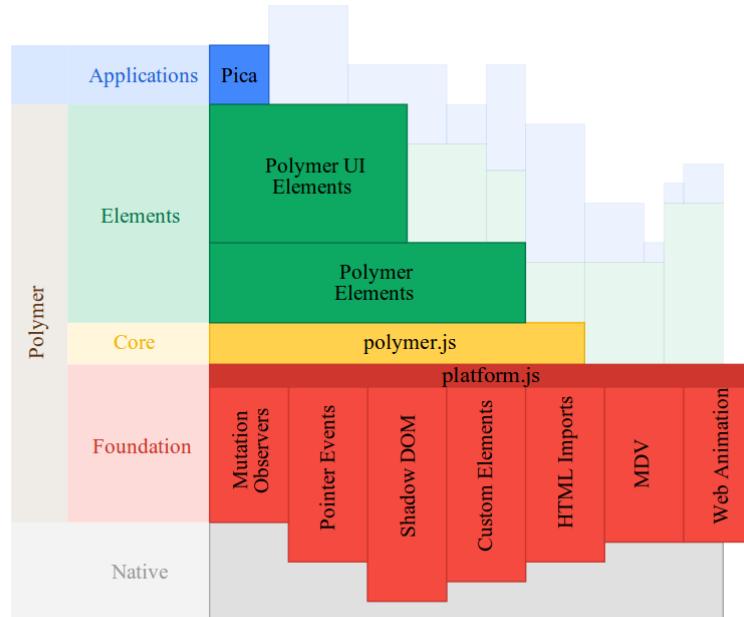


Polymer

- What does it means ?
 - Polymer is comprised of two efforts :
 - A core 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

- 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



X-Tags

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



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 Polymer website's "Getting the code" page. The left sidebar has sections like "Getting started" (with "Get the Code" selected), "Polymer", "Platform", "Articles", and "Videos". The main content area has a title "Getting the code" and a "Basics" section. It includes a "Table of contents" with links to "Installing Polymer", "Installing components", and "Next steps". Below that is a "Installing Polymer" section with a "DOWNLOAD STARTER PROJECT" button. A note says the project includes Polymer, elements, and a starter app. There's also a "START THE TUTORIAL" button and a note about picking components. The footer has "Edit on Github".

Getting the code

Basics

Table of contents

- Installing Polymer
- Installing components
 - Installing with Bower
 - Installing from ZIP files
 - Using git
- Next steps

Installing Polymer

If you want to learn Polymer, the easiest way to get started is to download the starter project:

[DOWNLOAD STARTER PROJECT](#)

The starter project includes Polymer, a set of elements, and a starter app. Work through the tutorial for an introduction to Polymer APIs and concepts, or work through the finished app on your own.

[→ START THE TUTORIAL](#)

If you're ready to start your own project, you can pick and choose the components you want to install, or install a whole set of components, like the Paper element collection.

Installing components



Step 2 - Use an element

```
<!-- 1. Polyfill Web Components support for older browsers -->
```

```
<script src="/bower_components/webcomponentsjs/webcomponents.js"></script>
```

```
<!-- 2. Import element -->
```

```
<link rel="import" href="bower_components/google-map/google-map.html">
```

```
<!-- 3. Use element -->
```

```
<google-map latitude="47.3834309" longitude="0.6986325" zoom="17"></google-map>
```



Step-3.1 : define an element

```
<polymer-element name="x-tours-jug">
  <template>
    <style>
      @host { /*...*/ }
    </style>
    <div id="box">
      <h1>Bonsoir Tours JUG !</h1>
      <p><content></content></p>
    </div>
  </template>
  <script>
    Polymer(x-tours-jug);
  </script>
</polymer-element>
```



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-tours-jug.html">
  </head>
  <body>
    <!-- 3. Declare the component by its tag. -->
    <x-tours-jug>Et je peux mettre mon texte ici
    </x-tours-jug>
  </body>
</html>
```



Step-3.3 : enjoy

Bonsoir Tours JUG !

Et je peux mettre mon texte ici

Elements Network Sources Timeline Profiles Resources Audits Console

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <!-- 3. Declare the component by its tag. -->
    <x-tours-jug>
      #shadow-root
        <style>...</style>
        <div id="box">
          <h1>Bonsoir Tours JUG !</h1>
          <p>...</p>
        </div>
        "Et je peux mettre mon texte ici"
    </x-tours-jug>
```



Step-4.1 : Add properties/methods

```
<polymer-element name="x-tours-jug">
  <template>
    <style>/*...*/ </style>
    <div id="box">
      <h1>Bonsoir {{whoami}}</h1>
      <p><content></content></p>
      <button on-click="{{sayHello}}>Click me!</button>
    </div>
  </template>
  <script>
    Polymer('x-tours-jug', {
      whoami: "Tours JUG",
      sayHello: function() { alert("Hello "+this.whoami); }
    });
  </script>
</polymer-element>
```



Step-4.2 : enjoy



A screenshot of the Chrome DevTools Elements tab. The component tree shows the following structure:

```
<!DOCTYPE html>
▼<html>
  ▶<head>...</head>
  ▼<body>
    <!-- 3. Declare the component by its tag. -->
    ▼<x-tours-jug>
      ▼#shadow-root
        ▶<style>...</style>
        ▼<div id="box">
          <h1>Bonsoir Tours JUG</h1>
          ▶<p>...</p>
          ▶<div>...</div>
        </div>
        "Et je peux mettre mon texte ici"
```



Step-5.1 : Declarative two-ways databinding!

```
<polymer-element name="x-tours-jug">
  <template>
    <style>/*...*/ </style>
    <div id="box">
      <h1>Bonsoir {{whoami}}</h1>
      <p><content></content></p>
      <input value="{{whoami}}" placeholder="Your name here..." />
    </div>
  </template>
  <script>
    Polymer('x-tours-jug', {
      whoami: "Tours JUG"
    });
  </script>
</polymer-element>
```

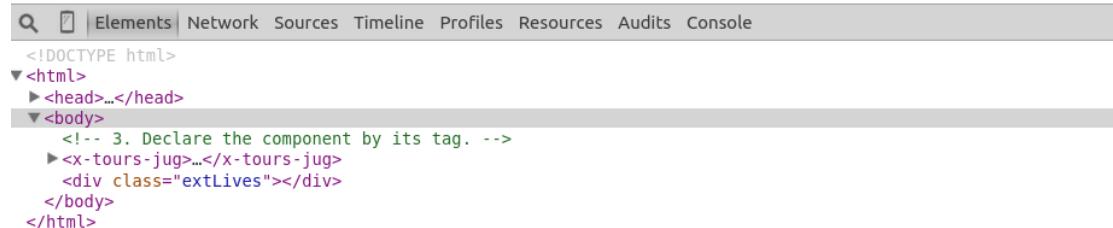


Step-3.2 : enjoy

Bonsoir Tours JUG

Et je peux mettre mon texte ici

Tours JUG



The screenshot shows the browser's developer tools with the "Elements" tab selected. The page source code is displayed, highlighting the rendered HTML structure. The code includes the DOCTYPE declaration, an HTML element, a head element containing meta tags, and a body element. Inside the body, there is a comment indicating where the component should be declared, followed by the custom element tag <x-tours-jug>, a div element with a class attribute, and finally closing body and html tags.

```
<!DOCTYPE html>
▼<html>
►<head>...</head>
▼<body>
  <!-- 3. Declare the component by its tag. --&gt;
  ▶&lt;x-tours-jug&gt;...&lt;/x-tours-jug&gt;
  &lt;div class="extLives"&gt;&lt;/div&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```



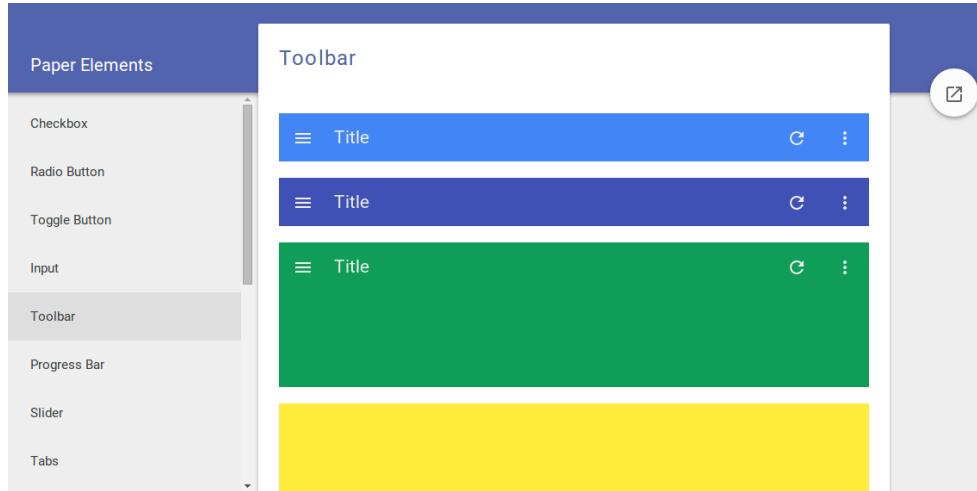
Polymer core elements

The screenshot shows the Polymer Elements documentation for the `core-list` element. The left sidebar lists various core elements: `core-a11y-keys`, `core-ajax`, `core-xhr`, `core-animated-pages`, `core-transition-pages`, `core-animation-group`, `core-animation`, `core-animation-keyframe`, `core-animation-prop`, `core-collapse`, and `core-drag-drop`. The main content area has a blue header with the title "Polymer core elements" and a sub-header "Elements". It features a large pink heading "core-list" and two buttons: "GET CORE-LIST" and "DEMO". Below these are sections for "Summary" and "Details". The summary explains that `core-list` displays a virtual, 'infinite' list using templates and provides details about the `data` and `height` properties. The details section continues with information about managing scroll position and performance.

Set of visual and non-visual utility elements



<core-toolbar>

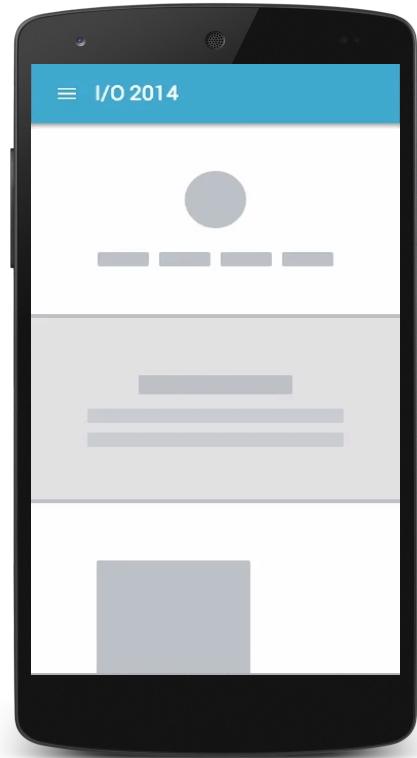


```
<link rel="import"  
      href="core-toolbar.html">
```

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



<core-header-panel>

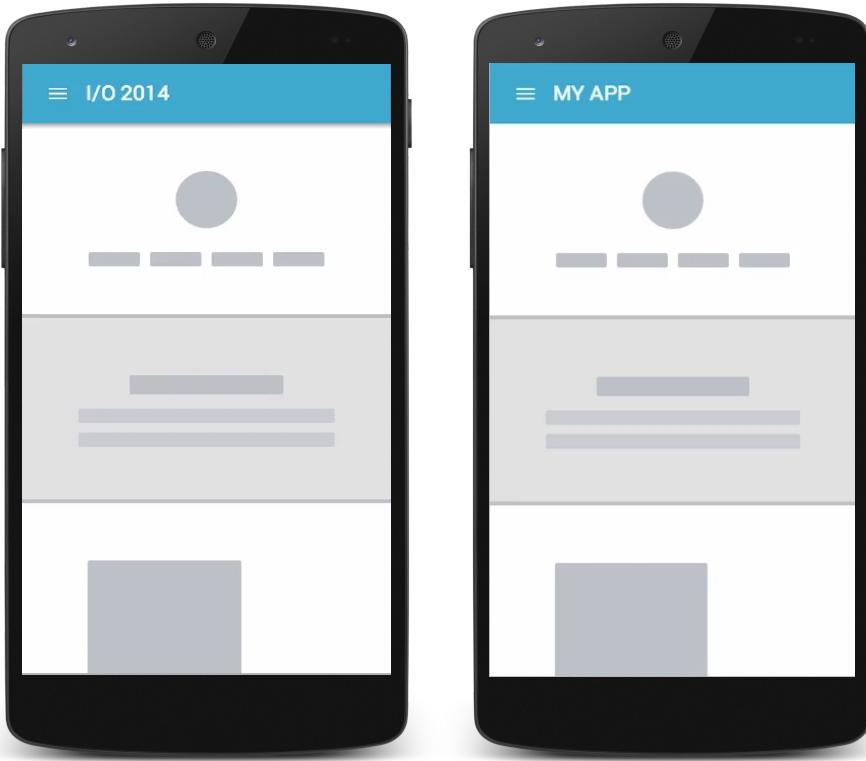


```
<link rel="import"  
      href="core-toolbar.html">  
<link rel="import"  
      href="core-header-panel.html">
```

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



Elements are configurable



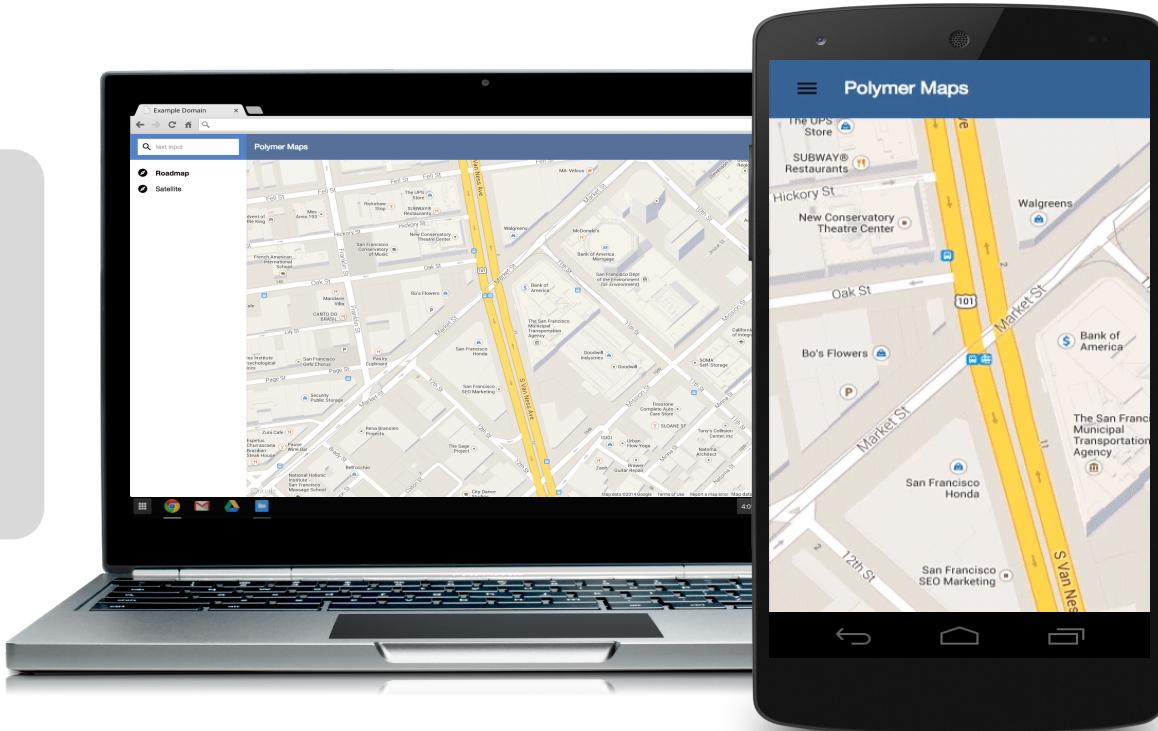
Toolbar will scroll with the page

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



Responsivity from the beginning

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



Polymer Paper

Material Design on Polymer, oh yeah!



ToyPro



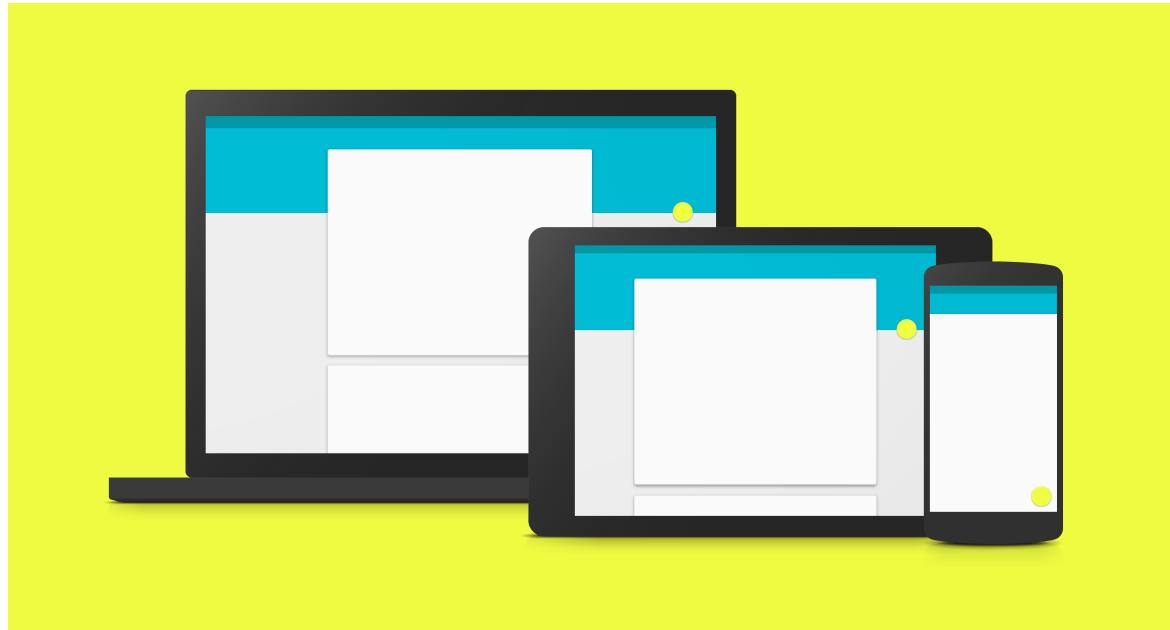
Android Lollipop's look & feel



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

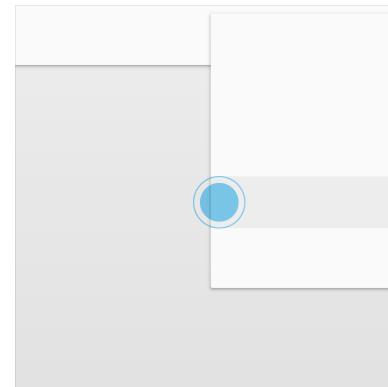
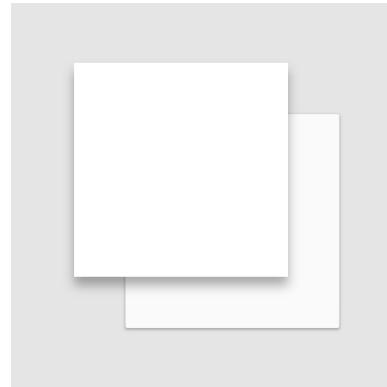
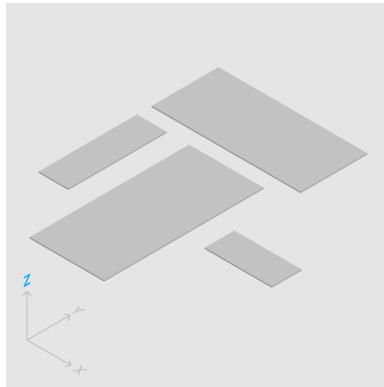


Visual language joining good design with technical innovation to create an unified experience across platforms and device sizes



What's material design?

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



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



Polymer Paper

A set of Polymer elements implementing Material Design guidelines

The screenshot shows a web browser window displaying the Polymer paper-elements documentation. The left sidebar lists various Polymer elements under the 'Paper elements' category, including paper-button-base, paper-button, paper-checkbox, paper-dialog-transition, paper-dialog, paper-dropdown-menu, paper-fab, paper-focusable, paper-icon-button, and paper-input. The main content area has a blue header with the text 'Polymer paper elements' and 'material design'. Below the header, the 'paper-button' element is highlighted in pink. It is described as extending 'paper-button-base'. There are two buttons: 'GET PAPER-BUTTON' with a download icon and 'DEMO'. Below these buttons is a 'Summary' section and a link to 'Material Design: Buttons'. The top navigation bar includes 'Learn', 'Elements' (which is selected), 'Resources', and a search icon.



<paper-tabs>

The screenshot shows the 'Paper Elements' sidebar with 'Tabs' selected. The main content area is titled 'Tabs' and contains three examples labeled A, B, and C. Example A shows a horizontal bar with three items: 'ITEM ONE' (highlighted in yellow), 'ITEM TWO', and 'ITEM THREE'. Example B shows a horizontal bar where the first item 'ITEM ONE' is highlighted in yellow, and the bar has shifted to the left. Example C shows a horizontal bar with three items: 'ITEM ONE', 'ITEM TWO', and 'ITEM THREE', all in white text on a blue background.

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

```
<paper-tabs selected="0">  
  <paper-tab>  
    ITEM ONE  
  </paper-tab>  
  <paper-tab>  
    ITEM-TWO  
  </paper-tab>  
  <paper-tab>  
    ITEM THREE  
  </paper-tab>  
</paper-tabs>
```



<paper-input>

Type only numbers... (floating)

```
<paper-input floatinglabel  
    label="Type only numbers... (floating)"  
    validate="^[0-9]*$"  
    error="Input is not a number!">  
</paper-input>
```



<paper-ripple>

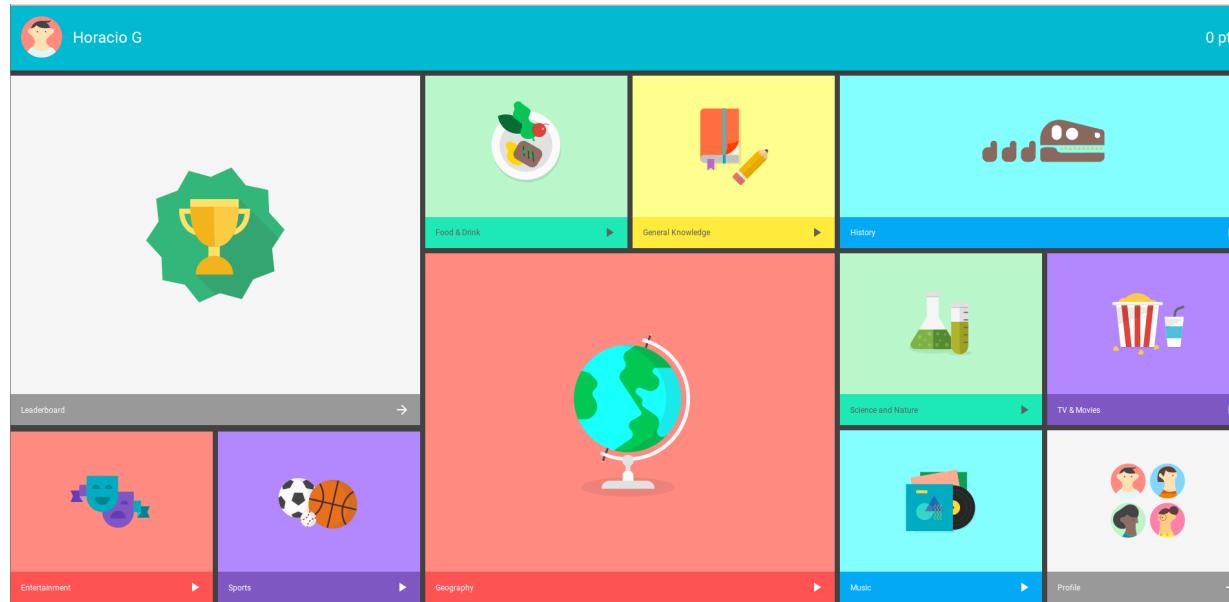


A reactive ink effect for indicating touch and mouse actions

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



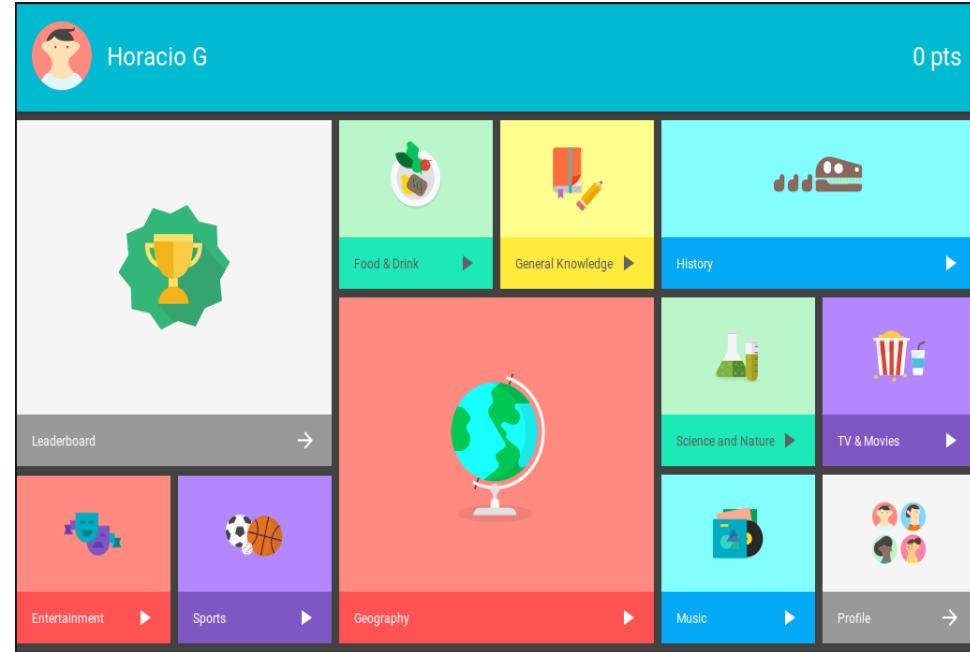
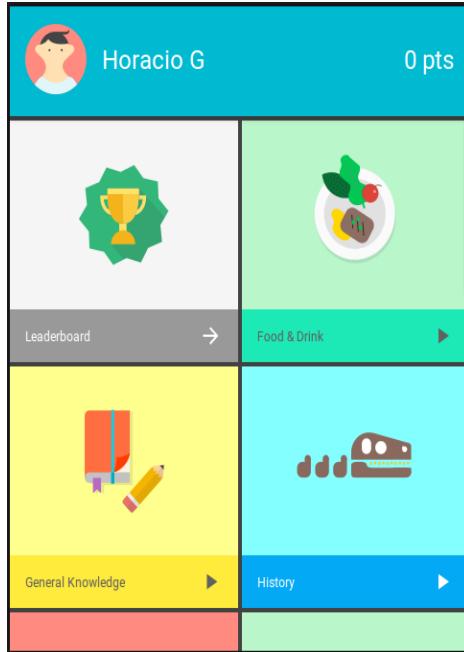
A full demo of Polymer Paper



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



Fully responsive



Other available web components

The more, the merrier!



[KNP Labs](#)

#Webcomponents

@LostInBrittany



GoogleWebComponents.github.io

Google Web Components

A collection of web components for Google APIs & services. Built with Polymer.

8+ 387 Tweet 280 Like 110

`<google-analytics>`

[DEMO](#) [DOCS](#) [GITHUB](#)

```
bower install GoogleWebComponents/google-analytics [--save]
```

`<google-apis>`

[DEMO](#) [DOCS](#) [GITHUB](#)

```
bower install GoogleWebComponents/google-apis [--save]
```

`<google-calendar>`

[DEMO](#) [DOCS](#) [GITHUB](#)

```
bower install GoogleWebComponents/google-calendar [--save]
```



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

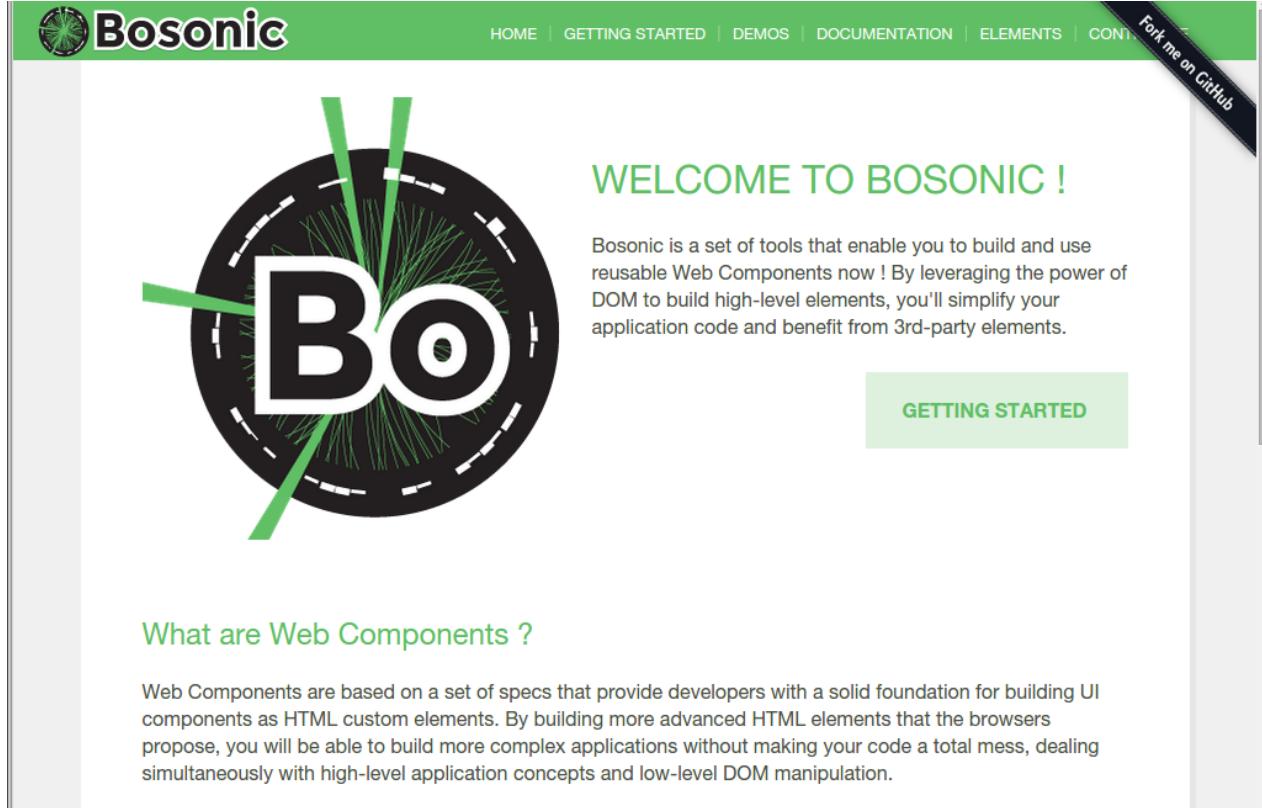
Installation

Brick can be installed using the Bower package manager:

```
bower install mozbrick/brick
```



Bosomic



The screenshot shows the Bosonic website homepage. At the top, there's a green navigation bar with the title "Bosonic" and a small icon. Below the bar, the main content area features a large circular logo with the letters "BO" in the center, surrounded by green branches and a dashed circle. 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 "GETTING STARTED" button is located below this text. In the top right corner of the main content area, there's a dark ribbon with the text "Fork me on GitHub". At the bottom left, there's a section titled "What are Web Components ?" with a descriptive paragraph about them.

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 [tours-jug](#)

<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 !





Web Components avec Polymer

