EBook Gratis

APRENDIZAJE polymer

Free unaffiliated eBook created from
Stack Overflow contributors.

#polymer

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: polymer

It is an unofficial and free polymer ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official polymer.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el polímero

Observaciones

El proyecto Polymer consiste en:

- Biblioteca de polímeros : Polymer es una biblioteca liviana que te ayuda a aprovechar al máximo los componentes web. Con los componentes web, puede crear elementos personalizados reutilizables que interactúan a la perfección con los elementos incorporados del navegador, o dividir su aplicación en componentes del tamaño correcto, haciendo que su código sea más limpio y menos costoso de mantener.
- WebComponents Polyfill : WebComponents Polyfill es una futura biblioteca dirigida a cumplir con las especificaciones de los componentes web del W3C. Los navegadores que implementan completamente la especificación no necesitan webcomponents.js. Sin embargo, a la mayoría de los navegadores aún les falta alguna parte de la especificación, por lo que esta será una dependencia durante bastante tiempo.
- Polymer App Toolbox : Polymer App Toolbox lo ayuda a crear y ofrecer aplicaciones web progresivas de vanguardia con una sobrecarga y una carga útil mínimas, aprovechando las potentes funciones de la plataforma web como Web Components, Service Worker y HTTP / 2. Toolbox proporciona una arquitectura basada en componentes, diseños sensibles, un enrutador modular, soporte de localización, soporte llave en mano para almacenamiento local y almacenamiento en caché sin conexión, y entrega eficiente de recursos de aplicaciones desagregados. Adopte estas funciones individualmente, o utilícelas juntas para crear una aplicación web progresiva con todas las funciones. El polímero esparce un poco de azúcar sobre las API de componentes web estándar, lo que facilita la obtención de excelentes resultados. La biblioteca de polímeros proporciona un conjunto de características para crear elementos personalizados. Estas características están diseñadas para que sea más fácil y rápido hacer elementos personalizados que funcionen como elementos DOM estándar. Similar a los elementos estándar de DOM, los elementos de polímero pueden ser:

Versiones

Versión	Fecha de lanzamiento
v2.0.0	2017-05-15
v1.7.0	2016-09-28
v1.6.1	2016-08-01
v1.6.0	2016-06-29
v1.5.0	2016-05-31
v1.4.0	2016-05-18

Versión	Fecha de lanzamiento
v1.0.0	2015-05-27

Examples

Hola Mundo

Este ejemplo crea un elemento de polímero llamado `x-foo` , cuya plantilla se enlaza a una propiedad de cadena, llamada "mensaje". El HTML del elemento se importa al documento principal, lo que permite el uso de etiquetas `<x-foo>` en `<body>` .

x-foo.html

```
<dom-module id="x-foo">
  <template>
    <span>{{message}}</span>
  </template>

  <script>
    Polymer({
      is: 'x-foo',
      properties : {
        message: {
          type: String,
          value: "Hello world!"
        }
      }
    });
  </script>
</dom-module>
```

index.html

```
<head>
  <!-- PolyGit used here as CDN for demo purposes only. In production,
       it's recommended to import Polymer and Web Components locally
       from Bower. -->
  <base href="https://polygit.org/polymer+1.6.0/components/">

  <script src="webcomponentsjs/webcomponents-lite.min.js"></script>
  <link rel="import" href="polymer/polymer.html">

  <link rel="import" href="x-foo.html">
</head>
<body>
  <x-foo></x-foo>
</body>
```

Ver demo en CodePen

Usando elementos del catálogo de polímeros

El polímero produce muchos elementos bien construidos para que los uses en tu aplicación.

Hojéalos en su Catálogo de Elementos .

Revisemos el flujo de trabajo de usar un elemento incluyendo `paper-input` (Documentación)

Descargar el elemento

Para descargar un elemento hay dos formas:

Cenador

La forma más conveniente es usar la línea de comandos usando el comando bower `install` :

```
bower install --save PolymerElements/paper-input
```

Nota: `--save` agrega el elemento como una dependencia al `bower.json` de su aplicación.

archivo zip

La otra forma es agregar el elemento seleccionado (`paper-input` en este caso) a su colección (en el Catálogo de polímeros) usando "Agregar a la colección" en la navegación y descargar su colección utilizando el icono de estrella en la esquina superior derecha.

Esto generará un archivo .zip que contiene el elemento y todas sus dependencias. Luego puede copiar la carpeta `bower_components` dentro de .zip / components en el directorio raíz de su aplicación.

Importa el elemento en tu aplicación

Para importar el elemento que acaba de instalar, importe el archivo `.html` correspondiente:

```
<link rel="import" href="bower_components/paper-input/paper-input.html">
```

Usa el elemento

Ahora puede utilizar `paper-input` dentro del documento que lo importó a:

```
<paper-input></paper-input>
```

Estructura básica del elemento

Obtuvimos el siguiente elemento muy básico `my-element` guardado como `src/my-element.html`

```
<link rel="import" href="bower_components/polymer/polymer.html">

<dom-module id="my-element">

  <template>
    <style>
      /* local styles go here */
      :host {
        display: block;
      }
    </style>
    <!-- local DOM goes here -->
    <content></content>
  </template>

  <script>
    Polymer({
      /* this is the element's prototype */
      is: 'my-element'
    });
  </script>

</dom-module>
```

- El `<link>` incluye la biblioteca de polímeros utilizando una importación HTML.
- El `<dom-module>` es el contenedor DOM local para el elemento (en este caso, `my-element`).
- `<template>` es la definición de DOM local real.
- `<style>` dentro de `<template>` permite definir estilos que están incluidos en este elemento y su DOM local y no afectarán a nada más en el documento.
- El `<content>` guardará cualquier cosa que coloques dentro de tu elemento.
- La pseudo clase del `:host` coincide con el elemento personalizado (`my-element`).
- La llamada `Polymer` registra el elemento.
- El `is` la propiedad es el nombre del elemento **(tiene** que coincidir con el `<dom-module>` 's `id`)

Puedes importarlo en tu aplicación usando:

```
<link rel="import" href="src/my-element.html">
```

Y utilízalo como una etiqueta:

```
<my-element>Content</my-element>
```

Configurando tu primera aplicación de polímero desde una plantilla

¡Preparémonos para construir su propia aplicación web progresiva impresionante con Polymer!

Antes de que puedas comenzar a instalar Polymer, necesitas lo siguiente:

- Node.js: echa un vistazo a la documentación de instalación de Node.js de StackOverflow
- Bower: puede instalar Bower utilizando Node Package Manager instalado con Node.js:

```
npm install -g bower
```

Instalación de la interfaz de línea de comandos de polímero

El CLI de polímeros le proporciona todas las herramientas necesarias para los proyectos de polímeros:

```
npm install -g polymer-cli
```

Inicializa tu aplicación desde una plantilla de aplicación

Use el iniciador de `polymer init` para inicializar su aplicación desde una plantilla de aplicación .

Una plantilla genial es la `--app-drawer-template` . Vamos a usar eso:

```
polymer init app-drawer-template
```

Sirve tu aplicación

No se necesita ningún edificio para servir tu primera aplicación de polímero impresionante. Sólo `serve` que `serve` :

```
polymer serve --open
```

Esto abrirá la aplicación en su navegador predeterminado en `http://localhost:8080` .

Lea Empezando con el polímero en línea: https://riptutorial.com/es/polymer/topic/949/empezando-con-el-polimero

Capítulo 2: Bucle, la plantilla dom-repetir.

Examples

Una lista basica

Este es un elemento polímero básico que muestra una lista de nombres.

```
<link rel="import" href="../bower_components/polymer/polymer.html">

<dom-module id="basic-list">
  <template>
    <style>
    </style>

    <div>Name's list</div>
    <template is="dom-repeat" items="{{list}}">
        <div>{{item.lastName}}, {{item.firstName}}</div>
    </template >

  </template>

  <script>
    Polymer({
      is: 'basic-list',

      properties:{
        list:{
          type: Array,
          value: function(){
            let list = [
              {firstName: "Alice", lastName: "Boarque"},
              {firstNName: "Carlos, lastName: "Dutra"}
            ]

            return list
          },
        },
      },
    });
  </script>
</dom-module>
```

Lea Bucle, la plantilla dom-repetir. en línea: https://riptutorial.com/es/polymer/topic/6160/bucle--la-plantilla-dom-repetir-

Capítulo 3: Creando la aplicación usando el kit de inicio de polímero

Introducción

Polymer Starter Kit es una excelente solución para comenzar a crear aplicaciones web progresivas . Ofrece un diseño de material sensible Interfaz de usuario, enrutamiento de página, servicio sin conexión.

Sintaxis

- Polímero: Comando para iniciar el polímero cli
- Polymer init: presenta una lista de plantillas para elegir para construir su elemento / aplicación
- npm: inicia la interfaz CLI del administrador de paquetes Node.js
- npm install: instala un paquete
- npm install -g: instala un paquete globalmente en su sistema. Útil para herramientas CLI

Examples

Instalación del kit de inicio de polímero

1. Asegúrese de que tiene Polymer CLI instalado en su sistema a nivel mundial. Si no es así, abra la línea de comandos / interfaz de terminal y ejecute este comando: `npm install -g polymer-cli`

   Nota: Si es un usuario de Ubuntu, es posible que tenga que prefijar el código anterior con la palabra clave `sudo` .

2. Después de instalar la CLI de Polymer, ejecute este comando

   cd [SU DIRECTORIO EN EL QUE DESEA INICIAR SU PROYECTO]

```
polymer init
```

3. Se le darán 4 opciones. Use las teclas de flechas para ir al 4º, es decir **, el kit de inicio** . Presiona la tecla `Enter` .

   Todos los archivos requeridos serán descargados en su directorio seleccionado.

Lea Creando la aplicación usando el kit de inicio de polímero en línea: https://riptutorial.com/es/polymer/topic/9330/creando-la-aplicacion-usando-el-kit-de-inicio-de-polimero

Capítulo 4: Depuración

Examples

Deshabilitar el almacenamiento en caché de importación HTML

El almacenamiento en caché de importación HTML a veces significa que los cambios realizados en los archivos HTML que se importan no se reflejan en la actualización del navegador. Tome la siguiente importación como ejemplo:

```
<link rel="import" href="./my-element.html">
```

Si se realiza un cambio en `my-element.html` después de cargar la página anteriormente, es posible que el archivo modificado no se descargue ni se use en el documento actual cuando se actualice (ya que se importó y guardó en caché). Esto puede ser excelente para una producción, pero podría obstaculizar el desarrollo.

Para deshabilitar esto en Google Chrome:

- Abre DevTools de Google Chrome
- Seleccione el Menú principal > Configuración
- Ir a la sección de Red
- Seleccione "Deshabilitar caché (mientras DevTools está abierto)"

Esto evitará el almacenamiento en caché de las importaciones HTML, pero solo cuando DevTools esté abierto.

Lea Depuración en línea: https://riptutorial.com/es/polymer/topic/5864/depuracion

Capítulo 5: Ejemplo de Polymer toggleAttribute

Sintaxis

1. toggleAttribute (nombre, bool, nodo)

Parámetros

Nombre	Detalles
nombre	Cadena: nombre del atributo HTML que debe cambiarse
bool	booleano: booleano para forzar el atributo de activación o desactivación. Cuando no se especifique, el estado del atributo se invertirá.
nodo	HTMLElement: nombre del nodo que contiene el atributo HTML. Por defecto a esto

Observaciones

Un buen ejemplo de esto será el formulario, donde el botón de envío solo debe estar activo si todos los campos obligatorios tienen entrada.

Examples

Ejemplo basico

```
<script src='bower_components/webcomponentsjs/webcomponents-lite.min.js'></script>
<link rel='import' href='bower_components/polymer/polymer.html'>
<link rel='import' href='bower_components/paper-button/paper-button.html'>
<link rel='import' href='bower_components/paper-input/paper-input.html'>
<dom-module id='toggle-attribute'>
  <template>
    <style>
    </style>
    <paper-input id='input'></paper-input>
    <paper-button on-tap='_toggle'>Tap me</paper-button>
  </template>
</dom-module>
<script>
  Polymer({
    is:'toggle-attribute',
    properties:{
      isTrue:{
        type:Boolean,
```

```
        value:false
      }
    },
    _toggle:function(){
      this.isTrue = !this.isTrue;
      this.toggleAttribute('disabled',this.isTrue,this.$.input);
    }
  })
</script>
```

Aquí hay un plunker corriendo

Lea Ejemplo de Polymer toggleAttribute en línea:
https://riptutorial.com/es/polymer/topic/5978/ejemplo-de-polymer-toggleattribute

Capítulo 6: Examen de la unidad

Observaciones

Probador de componentes web : la herramienta para probar unidades en aplicaciones creadas con Polymer. Usted obtiene un entorno de prueba basado en el navegador, configurado fuera de la caja con mocha , chai , async , lodash , sinon y sinon-chai , test-fixture , accessibility-developer-tools . WCT ejecutará sus pruebas en cualquier navegador que haya instalado localmente o de forma remota a través de Sauce Labs.

Examples

Ejemplo simple usando el componente web-tester

instalando

```
npm install web-component-tester --save-dev
```

estableciendo

wct.conf.js

```
module.exports = {
    verbose: true,
    plugins: {
        local: {
            browsers: ['chrome']
        }
    }
};
```

corriendo

```
node node_modules/web-component-tester/bin/wct
```

prueba / index.html

```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, minimum-scale=1.0, initial-
```

```
scale=1">

        <script src="../bower_components/webcomponentsjs/webcomponents.min.js"></script>
        <script src="../node_modules/web-component-tester/browser.js"></script>
        <link rel="import" href="../src/utils.html">

    </head>
    <body>

        <my-utils id="utils"></my-utils>

        <script>

            test('utils.isNumeric', function(){
                var utils = document.getElementById('utils');
                [1,0,-1,1.83,-9.87].forEach(function(d){
                    assert.isTrue(utils.isNumeric(d));
                });
                [true, false, null, {}, 'a', new Date()].forEach(function(d){
                    assert.isFalse(utils.isNumeric(d));
                });
            });

        </script>

    </body>
</html>
```

src / utils.html

```
<link rel="import" href="../bower_components/polymer/polymer.html">

<dom-module id="my-utils">
    <template></template>
</dom-module>

Polymer({

    is: "my-utils",

    isNumeric: function(n) {
        return !isNaN(parseFloat(n)) && isFinite(n);
    }

});
```

Lea Examen de la unidad en línea: https://riptutorial.com/es/polymer/topic/3898/examen-de-la-unidad

Capítulo 7: Hoja de trucos de polímero

Introducción

Esta es una hoja de trucos para la biblioteca Polymer 2.x. Horquilla de correos de monica dinculescu.

Examples

Definiendo un elemento

Docs: 1.x -> 2.x guía de actualización , registro de un elemento , módulos de estilo compartidos .

```
<link rel="import" href="bower_components/polymer/polymer-element.html">
<dom-module id="element-name">
  <template>
    <!-- Use one of these style declarations, but not both -->
    <!-- Use this if you don't want to include a shared style -->
    <style></style>
    <!-- Use this if you want to include a shared style -->
    <style include="some-style-module-name"></style>
  </template>
  <script>
    class MyElement extends Polymer.Element {
      static get is() { return 'element-name'; }
      // All of these are optional. Only keep the ones you need.
      static get properties() { ... }
      static get observers() { ... }
    }

    // Associate the new class with an element name
    customElements.define(MyElement.is, MyElement);
  </script>
</dom-module>
```

Para obtener la definición de clase para una etiqueta personalizada en particular, puede usar `customElements.get('element-name')` .

Extendiendo un elemento

Docs: elementos extensibles , plantillas heredadas .

En lugar de `Polymer.Element` , un elemento personalizado puede extender un elemento diferente:

```
class ParentElement extends Polymer.Element {
  /* ... */
}
class ChildElement extends ParentElement {
  /* ... */
}
```

Para cambiar o agregar a la plantilla de los padres, anule el captador de `template` :

```
<dom-module id="child-element">
  <template>
    <style> /* ... */ </style>
    <span>bonus!</span>
   </template>
  <script>
    var childTemplate;
    var childTemplate = Polymer.DomModule.import('child-element', 'template');
    var parentTemplate = ParentElement.template.cloneNode(true);
    // Or however you want to assemble these.
    childTemplate.content.insertBefore(parentTemplate.firstChild, parentTemplate);

    class ChildElement extends ParentElement {
      static get is() { return 'child-element'; }
      // Note: the more work you do here, the slower your element is to
      // boot up. You should probably do the template assembling once, in a
      // static method outside your class (like above).
      static get template() {
        return childTemplate;
      }
    }
    customElements.define(ChildElement.is, ChildElement);
  </script>
</dom-module>
```

Si no conoce la clase para padres, también puede usar:

```
class ChildElement extends customElements.get('parent-element') {
  /* ... */
}
```

Definiendo un mixin

Documentos: mixins , elementos híbridos .

Definición de una combinación de expresiones de clase para compartir la implementación entre diferentes elementos:

```
<script>
  MyMixin = function(superClass) {
    return class extends superClass {
      // Code that you want common to elements.
      // If you're going to override a lifecycle method, remember that a) you
      // might need to call super but b) it might not exist.
      connectedCallback() {
        if (super.connectedCallback) {
          super.connectedCallback();
        }
        /* ... */
      }
    }
  }
</script>
```

Usando el mixin en una definición de elemento:

```
<dom-module id="element-name">
  <template><!-- ... --></template>
  <script>
    // This could also be a sequence:
    // class MyElement extends AnotherMixin(MyMixin(Polymer.Element)) { … }
    class MyElement extends MyMixin(Polymer.Element) {
      static get is() { return 'element-name' }
      /* ... */
    }
    customElements.define(MyElement.is, MyElement);
  </script>
</dom-module>
```

Usando comportamientos híbridos (definidos en la sintaxis 1.x) como mixins:

```
<dom-module id="element-name">
  <template><!-- ... --></template>
  <script>
    class MyElement extends Polymer.mixinBehaviors([MyBehavior, MyBehavior2], Polymer.Element)
{
     static get is() { return 'element-name' }
     /* ... */
    }
    customElements.define('element-name', MyElement);
  </script>
</dom-module>
```

Métodos de ciclo de vida

Documentos: devoluciones de llamada de ciclo de vida , listo .

```
class MyElement extends Polymer.Element {
 constructor() { super(); /* ... */ }
 ready() { super.ready(); /* ... */ }
 connectedCallback() { super.connectedCallback(); /* ... */ }
 disconnectedCallback() { super.disconnectedCallback(); /* ... */ }
 attributeChangedCallback() { super.attributeChangedCallback(); /* ... */ }
}
```

El enlace de datos

Documentos: enlace de datos , enlace de atributos , enlace a elementos de matriz , enlaces calculados .

No olvide: las propiedades de las fundas de camello de polímero, así que si en JavaScript usa `myProperty` , en HTML usaría `my-property` .

`myProperty` **una manera** : cuando `myProperty` cambia, su `theirProperty` se actualiza:

```
<some-element their-property="[[myProperty]]"></some-element>
```

`myProperty` **bidireccional** : cuando `myProperty` cambia, su `theirProperty` se actualiza y viceversa:

```
<some-element their-property="{{myProperty}}"></some-element>
```

`myProperty` **atributos** : cuando `myProperty` es `true` , el elemento está oculto; Cuando es `false` , el elemento es visible. La diferencia entre atributo y enlace de propiedad es que el enlace de propiedad es equivalente a `someElement.someProp = value` , mientras que el enlace de atributo es equivalente a: `someElement.setAttribute(someProp, value)`

```
<some-element hidden$="[[myProperty]]"></some-element>
```

Enlace computado : el enlace al atributo de `class` recompilará los estilos cuando cambie `myProperty` :

```
<some-element class$="[[_computeSomething(myProperty)]]"></some-element>
<script>
_computeSomething: function(prop) {
  return prop ? 'a-class-name' : 'another-class-name';
}
</script>
```

Observadores

Documentos: observadores , observadores de múltiples propiedades , observando mutaciones de matriz , agregando observadores dinámicamente .

Agregar un `observer` en el bloque de `properties` permite observar cambios en el valor de una propiedad:

```
static get properties() {
  return {
    myProperty: {
      observer: '_myPropertyChanged'
    }
  }
}

// The second argument is optional, and gives you the
// previous value of the property, before the update:
_myPropertyChanged(value, /*oldValue */) { /* ... */ }
```

En el bloque de `observers` :

```
static get observers() {
  return [
    '_doSomething(myProperty)',
    '_multiPropertyObserver(myProperty, anotherProperty)',
    '_observerForASubProperty(user.name)',
    // Below, items can be an array or an object:
    '_observerForABunchOfSubPaths(items.*)'
  ]
}
```

Agregar un observador dinámicamente para una propiedad `otherProperty` :

```
// Define a method.
_otherPropertyChanged(value) { /* ... */ }
// Call it when `otherPropety` changes.
this._createPropertyObserver('otherProperty', '_otherPropertyChanged', true);
```

Lea Hoja de trucos de polímero en línea: https://riptutorial.com/es/polymer/topic/10710/hoja-de-trucos-de-polimero

Capítulo 8: Manejo de eventos

Observaciones

- Aquí hay un plunker para todos los ejemplos
- El nombre del atributo no distingue entre mayúsculas y minúsculas y siempre se convertirá a minúsculas, por ejemplo, si tiene un atributo, `on-myListener` escucha de `on-myListener` se configurará en el evento `mylistener` .
- Al igual que para `listen` , también puede usar el método `unlisten` para eliminar cualquier escucha.
- El cambio de Poperty dispara un evento por el nombre de la `property-changed` por ejemplo, si la propiedad es `myProperty` evento se cambiará de `my-propert-changed` (la envoltura del camello a '-'). Puede escucharlos usando `on-event` atributo `on-event` del objeto de `listener` .
- El nuevo valor siempre se almacena en `e.detail.value` para eventos de cambio de propiedad.

Examples

Escucha de eventos usando el objeto oyente

```
<dom-module id="using-listeners-obj">
  <template>
    <style>
      :host{
        width: 220px;
        height: 100px;
        border: 1px solid black;
        display: block;
      }

      #inner{
        width: calc(100% - 10px);
        height: 50px;
        border: 1px solid blue;
        margin: auto;
        margin-top: 15px;
      }
    </style>
    <div>Tap me for alert message</div>
    <div id="inner">Tap me for different alert</div>
  </template>
</dom-module>
<script>
  Polymer({
    is:'using-listeners-obj',

    //Listener Object
    listeners:{
      //tap a special gesture event in Polymer. Its like click event the main difference being
it is more mobile deivce friendly
      'tap':'tapped',  //this will get executed on host of the element
```

```
      'inner.tap':'divTapped' //this tap will get executed only on the mentioned node (inner
in this case)
    },

    tapped:function(){
      alert('you have tapped host element');
    },

    divTapped:function(e){
      event.stopPropagation();  //this is only to stop bubbling of event. If you comment this
then tap event will bubble and parent's(host) listener will also get executed.
      console.log(e);
      alert('you have tapped inner div');
    }
  })
</script>
```

Oyente anotado

Otra forma de agregar un detector de eventos es usar la anotación en el `on-event` en DOM. A continuación se muestra un ejemplo utilizando `up` evento, que se dispara cuando el dedo / ratón sube

```
<dom-module id="annotated-listener">
  <template>
    <style>
      .inner{
        width: calc(200px);
        height: 50px;
        border: 1px solid blue;
        margin-top: 15px;
      }
    </style>
    <!-- As up is the name of the event annotation will be on-up -->
    <div class="inner" on-up='upEventOccurs'>Tap me for different alert</div>
  </template>
</dom-module>
<script>
  Polymer({
    is:'annotated-listener',
    upEventOccurs:function(e){
      //detail Object in event contains x and y co-ordinate of event
      alert('up event occurs at x:'+e.detail.x+' y:'+e.detail.y);
    }
  })
</script>
```

Oyente imperativo

También puede agregar / eliminar el oyente de forma imperativa usando el método de escuchar y deseleccionar de Polymer

```
    <dom-module id="imparative-listener">
      <template>
        <style>
          #inner{
```

```
            width: 200px;
            height: 50px;
            border: 1px solid blue;
            margin-top: 15px;
          }
        </style>
        <div id="inner">I've imparative listener attached</div>
      </template>
    </dom-module>
    <script>
      Polymer({
        is:'imparative-listener',

        //keeping it in attached to make sure elements with their own shadow root are attached
        attached:function(){
          this.listen(this.$.inner,'track','trackDetails'); // For more details on method
check https://www.polymer-project.org/1.0/docs/api/Polymer.Base#method-listen
        },

        //all the listener functions have event as first parameter and detail (event.detail)
as second parameter to the function
        trackDetails:function(e,detail){
          if(detail.state=='end'){
            alert('Track distance x:'+detail.dx+' y:'+detail.dy);// For more details on track
event check https://www.polymer-project.org/1.0/docs/devguide/gesture-events
      }
    }
  })
</script>
```

Eventos personalizados

También puede activar sus propios eventos y luego escucharlos desde cualquiera de los elementos Polymer de la página HTML

Este elemento dispara el evento personalizado.

```
    <dom-module id="custom-event">
  <template>
    <style>
      #inner{
        width: 200px;
        height: 50px;
        border: 1px solid blue;
        margin-top: 15px;
      }
    </style>
    <div id="inner" on-tap="firing">I'll fire a custom event</div>
  </template>
</dom-module>
<script>
  Polymer({
    is:'custom-event',
    firing:function(){
      this.fire('my-event',{value:"Yeah! i'm being listened"}) //fire is the method which is
use to fire custom events, second parameter can also be null if no data is required
    }
  })
```

```
</script>
```

Aquí hay un elemento que está escuchando ese evento personalizado.

```
  <link rel="import" href="custom-event.html">
 <dom-module id="custom-event-listener">
  <template>
    <style></style>
    <custom-event on-my-event="_myListen"></custom-event>
  </template>
</dom-module>
<script>
  Polymer({
    is:'custom-event-listener',
/*    listeners:{ //you can also use listener object instead of on-event attribute
      'my-event':'listen'
    },*/
    _myListen:function(e,detail){
      alert(detail.value+"from Polymer Element");
    },
  })
</script>
```

Escuchando desde HTML

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Events</title>
    <script src='bower_components/webcomponentsjs/webcomponents-lite.min.js'></script>
    <link rel="import" href="./bower_components/polymer/polymer.html">
    <link rel="import" href="custom-event-listener.html">
  </head>
  <body>
    <!--As event bubbles by default we can also listen to event from custom-event-listener
instead of custom-event-->
    <custom-event-listener></custom-event-listener>
  </body>
  <script>
    document.querySelector('custom-event-listener').addEventListener('my-event',function(e){
      console.log(e);
      alert(e.detail.value+"from HTML");
    })
  </script>
</html>
```

Evento de cambio de propiedad

Propiedades con `notify:true` también dispara un evento.

```
<link rel="import" href="../bower_components/paper-input/paper-input.html">
<dom-module id="property-change-event">
  <template>
    <style></style>
    <paper-input id="input" label="type here to fire value change event" on-value-
changed='changeFired'></paper-input>
```

```
  </template>
</dom-module>
<script>
  Polymer({
    is:'property-change-event',
    changeFired:function(e){
      if(e.detail.value!="")
        alert("new value is: "+e.detail.value);
    }
  })
</script>
```

Retargeting de eventos

Es posible reorientar un evento en Polymer, es decir, puede cambiar los detalles del evento, como la `path` , ocultando así los detalles reales de un evento / elemento del usuario.

Por ejemplo, si un `div` dentro `event-retargeting` elemento está disparando el evento, pero desarrollador no quiere que el usuario sepa que puede reorientar el evento a `event-retargeting` elemento utilizando el siguiente código.

```
var targetEl = document.querySelector('event-retargeting');
var normalizedEvent = Polymer.dom(event);
normalizedEvent.rootTarget = targetEl;
normalizedEvent.localTarget =targetEl
normalizedEvent.path = [];
normalizedEvent.path.push(targetEl);
normalizedEvent.path.push(document.querySelector('body'));
normalizedEvent.path.push(document.querySelector('html'));
```

Para ver el ejemplo de funcionamiento, consulte `plunker` en la sección de `remarks` .

Lea Manejo de eventos en línea: https://riptutorial.com/es/polymer/topic/4778/manejo-de-eventos

Capítulo 9: Marca de mapa de Google con construido en caché

Sintaxis

- <g-map

```
api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

locations='["Boston","NewYork","California","Pennsylvania"]'></g-map>
```

Parámetros

Clave API	javascript api clave de google
localizaciones	Lista de ubicaciones que desea marcar en el mapa de google
----------	----------

Observaciones

Para todo el código, vaya al repositorio

Para ver el mapa de google en acción, mira aquí.

Examples

A - Comenzando

En pocas palabras, nuestro elemento personalizado debe

- Tener una funcionalidad de búsqueda integrada, que busca y marca lugares en el mapa.
- Aceptar Un atributo llamado array de ubicación, que es una lista de lugares
- Tenga un Oyente, para escuchar un evento llamado "google-map-ready". Este evento se activa cuando el elemento se ha cargado.
    - Este oyente, debe recorrer la matriz de ubicación y asignar el siguiente elemento en su matriz de ubicación como la "consulta de búsqueda" actual
- Tener un método llamado caché
    - Este método almacena en caché la latitud y la longitud de cada lugar en el arreglo de ubicación, tan pronto como la búsqueda devuelve un resultado
- Tener una plantilla de repetición de Dom, que recorre los resultados almacenados en caché y suelta un marcador en cada ubicación

Por lo tanto, el propósito de nuestro elemento personalizado en esencia significa, si pasa una serie de ubicaciones como:

```
<g-map location-array='["Norway","Sweden"]'></g-map>
```

El elemento personalizado

- No solo debe marcar Noruega y Suecia en el mapa, sino
- También debe marcar cualquier búsqueda posterior en el mapa, es decir, si busca Boston, después de que el mapa haya marcado Noruega y Suecia, Boston también debe tener un pin en el mapa.

Importando Dependencias

Primero, importemos lo que necesitemos a nuestro elemento. Este elemento está bajo

elementos / g-map.html

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

Nota

Algunas de las importaciones anteriores no son necesarias, pero es bueno tenerlas, en caso de que le interese agregar funcionalidad en sus marcadores

Necesitamos necesariamente,

- mapa de Google
- marcador-mapa-google
- google-map-search

Además, se presume que usted sabe cómo instalar los elementos poliméricos individuales a través de Bower

Registrando nuestro elemento con polímero

A continuación, registramos nuestro elemento personalizado como "g-map"

```
Polymer({
       is:"g-map"
});
```

¡Asi que! Nuestro elemento personalizado está registrado con Polymer.

Agregaremos las propiedades específicas de la búsqueda en el mapa de Google y cualquier otra

propiedad que necesitemos, más adelante.

Agregando un Blueprint a nuestro elemento personalizado - a través de plantillas

A continuación, agregamos la plantilla para nuestro elemento personalizado.

Abra elementos / g-map.html y agregue una plantilla que vincule los elementos personalizados DOM

```
<template id="google-map-with-search">
</template>
```

Ahora, envuelva todo el código HTML y javascript que escribió para nuestro elemento personalizado, dentro de un módulo dom.

```
<dom-module id="g-map">

    <template id="google-map-with-search">

    </template>
    <script>
    Polymer({
        is:"g-map",
        properties: {}
 });
     </script>
</dom-module>
```

¡Muy agradable! Tenemos un plano básico.

B - Representación del mapa básico de google

Preparando su navegador - Polyfill it

Cree una nueva página de destino para nuestro elemento, en index.html.

Incluya polyfills para que funcione un elemento personalizado. También importe el elemento personalizado g-map, que registramos con Polymer.

Nota:

- Los componentes web son el polyfill necesario para el soporte del navegador.
- Polímero, está incluido, para que podamos usar la biblioteca para crear nuestro elemento personalizado

Así que abre index.html, e incluye los Polyfills y Polymer necesarios. Además, **importa** el elemento personalizado que hayamos registrado.

```
<head>
    <title>Google Map</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
     <script src="bower_components/webcomponentsjs/webcomponents.min.js"></script>
     <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.min.css">
     <link rel="import" href="bower_components/polymer/polymer.html">
     <link rel="import" href="elements/g-map.html">

</head>
```

Así que con el navegador Polyfilled,

Vamos a invocar el elemento polímero de google-map dentro de nuestro elemento personalizado.

```
<template id="google-map-with-search">

    <google-map></google-map>

</template>
```

y luego, invoque el elemento personalizado dentro de index.html, nuestra página de destino.

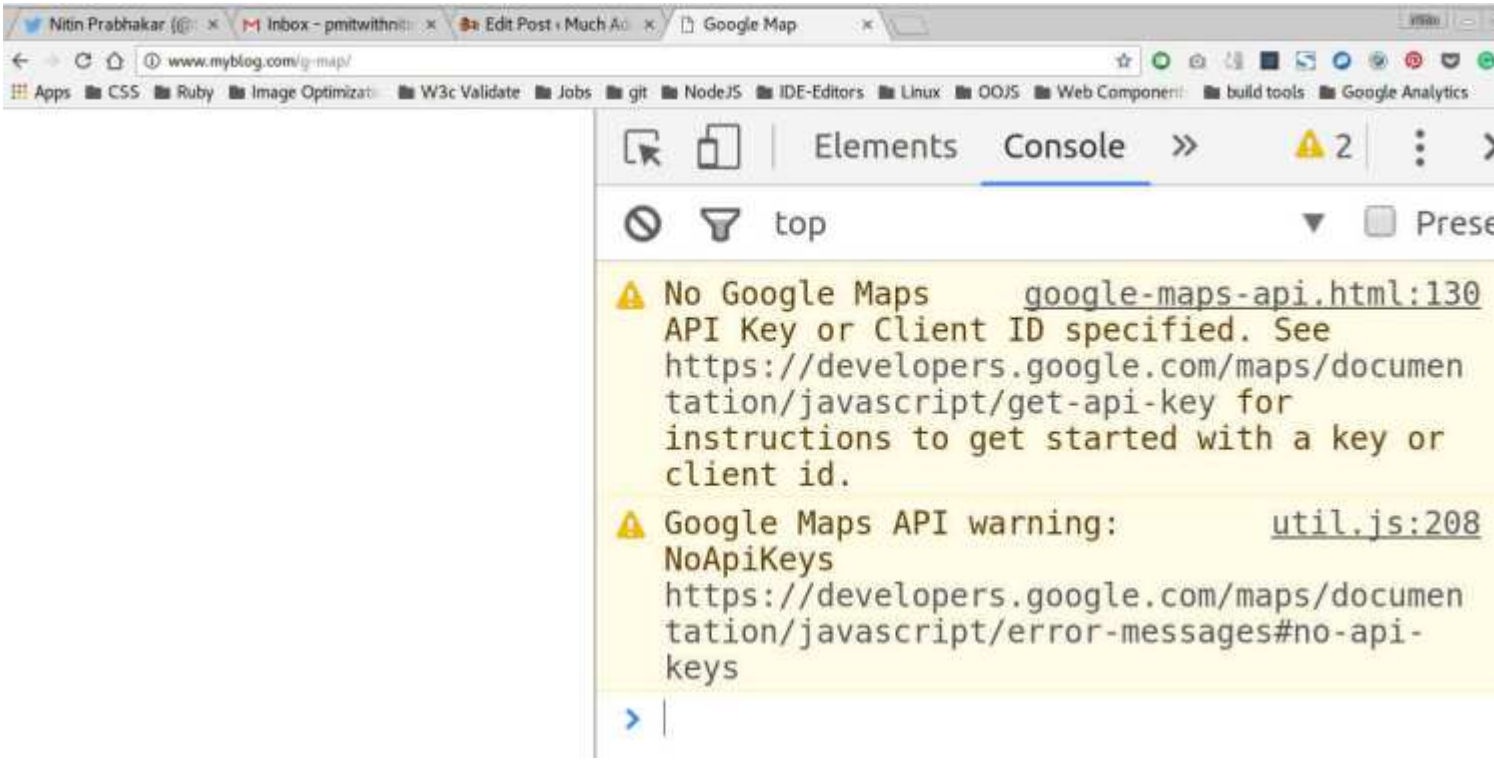
Así que esto va dentro del index.html

```
<body>

    <div class="container">

        <div class="row">

            <div class="col-xs-12">
                <g-map></g-map>
            </div>

        </div>

    </div>

</body>
```

Intenté renderizar mi página con el elemento g-map personalizado en este punto, ¡y encontré una página VACÍA!

¿¿POR QUÉ??

Cuando veo el registro de la consola, veo esto



Ah!

Entonces, necesitamos una clave api para renderizar un mapa de google.

Nota:

Obtener una clave de API de la API de javascript de Google es bastante simple.

Simplemente siga el enlace de abajo, para generar su clave personal por proyecto.

Clave API de Google

Ahora, ya que se va a utilizar la clave API, tenemos dos opciones.

- Deja que el usuario lo pase como un atributo
- Tener una clave de API predeterminada configurada

Ahora, me gustaría ir por la primera, simplemente porque, cualquier clave API tiene sus límites impuestos. es decir, solo se puede usar tantas veces. Por lo tanto, una clave configurada y enviada con el elemento que desarrollamos, pronto podría agotarse.

Quien quiera usar el elemento personalizado, puede generar una nueva clave y pasarla como un atributo.

al igual que:

```
<g-map api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"></g-map>
```

Así que estamos pasando la clave api como un atributo y dentro de nuestro elemento

personalizado, la vinculamos al atributo api-key del elemento Polymer.

al igual que:

```
<google-map api-key=[[apiKey]]></google-map>
```

Entonces, una vez que pasamos la clave de API y la actualización, ¡todavía no vemos el mapa!

> ## Nota:
>
> Google maps, necesita algunos css para renderizar. Necesitamos la altura del conjunto de mapas explícitamente.
>
> Entonces, haga un archivo css dentro del directorio css, diga app.css y agregue estas líneas
>
> ```
> google-map{
>
>     min-height:30vmax;
> }
> ```

Y ahora, al actualizar, vemos esto.

¡Hurra! ¡Acabamos de renderizar un mapa con un mínimo de marcas! ¡Acabamos de escribir esto!

```
<google-map api-key="[[apiKey]]"></google-map>
```

C - Agregar capacidad de búsqueda a nuestro elemento personalizado

Para buscar un lugar, usamos el poderoso elemento que envía Polymer, llamado google-map-search.

Todo lo que necesitas hacer, es pasar.

- un objeto de mapa y
- una cadena de consulta al elemento como tal:

```
<google-map-search map=[[map]] query=[[query]]></google-map-search>
```

¿Cómo pasamos el objeto de mapa? ¿De dónde sacamos eso?

¡Sencillo!

Cuando invoque el elemento google-map, enlace el mapa de propiedades de su elemento personalizado al mapa de propiedades del elemento.

Entonces, invocamos el elemento google-map así:

```
<google-map api-key="whatever key you generated for google's javascript API"
map={{map}}></google-map>
```

Nota:

Hacemos un enlace de datos bidireccional para el mapa de arriba, al invocar el polímero google-map.

Está representado por las llaves {{}} en lugar de una unión de una manera indicada por llaves [[]].

Cuando hacemos un enlace de dos vías,

- Cada vez que se modifica la propiedad del mapa de nuestro elemento personalizado g-map, se notifica el elemento google-map

y,

- Cada vez que se modifica la propiedad del mapa de google-map, se nos notifica en nuestro elemento personalizado g-map.

Por lo tanto, con el enlace de datos bidireccional establecido, cada vez que el objeto de mapa cambia en google-map, nuestro elemento personalizado se actualiza con los cambios.

y pasamos el objeto de mapa como un atributo al elemento google-map-search, de modo que, cualquier resultado de búsqueda se marque en el mapa actual representado.

¿Cómo se ve el DOM de nuestro elemento personalizado en este momento?

```
<template id="google-map-custom-element">

<google-map-search map=[[map]] query=[[query]]></google-map-search>
<google-map api-key=[[apiKey]] map={{map}}></google-map>
</template>
```

¡Nunca agregamos las propiedades de mapa y consulta para nuestro elemento personalizado!

Agregue las propiedades en la llamada de registro a Polymer

```
Polymer({

    is:"g-map",

    properties:{

        map:{

            type: Object

        },

        query:{

            type:String,

            value:""

        }
    }

});
```

Ahora que hemos registrado las propiedades, notamos que

- ¿Está obteniendo el mapa actualmente representado como un objeto - a través de enlace de datos y almacenándolo como una propiedad local también llamada mapa,
- Tener una consulta de propiedad, pasar a google-map-search

Aceptar entrada de usuario como consulta de búsqueda

¿Pero quién nos da la consulta para buscar? ¡Errar! ¿ahora mismo? NADIE.

Eso es precisamente lo que hacemos a continuación.

Agreguemos un formulario de búsqueda, que el usuario que representa el mapa usaría para ingresar una consulta. Necesitamos entradas para el cuadro de búsqueda, y usamos la entrada de hierro de Polymer.

Agregar un formulario de búsqueda

Incluya los elementos de hierro necesarios al principio del archivo g-map.html

```
<link rel=import href=../bower_components/iron-input/iron-input.html>
<link rel=import href=../bower_components/iron-icon/iron-icon.html>
<link rel=import href=../bower_components/iron-icons/iron-icons.html>
<link rel=import href=../bower_components/iron-icons/maps-icons.html>
```

Queremos el formulario de búsqueda, en la esquina superior izquierda, así que agregamos una entrada de hierro con la posición absoluta, y algunos css para fijarla allí.

Entonces, envuelva el DOM para nuestro elemento personalizado, dentro de un div llamado "map-container", y agregue un div hijo llamado "search_form".

```
<template id="google-map-custom-element">
    <div class="map-container">
         <google-map-search map="[[map]]" query="[[query]]"></google-map-search>
         <google-map api-key="[[apiKey]]"></google-map>
         <div class="search_form">
             <iron-icon icon="icons:search" on-tap=__search></iron-icon>
             <input is="iron-input" placeholder="Search Places" type=text name="search" bind-
value="{{query}}">
         </div>
     </div>
 </template>
```

Agregue un poco de CSS básico para fijar el formulario de búsqueda que escribimos en la parte superior derecha del mapa

```
.search_form{
    position: absolute;
    top:10px;
    right:10px;
}
.search_form iron-icon{
    position: absolute;
    right: 1px;
    top:1px;
    cursor: pointer;
}
```

Y luego, cuando refrescamos, tenemos esto. El formulario de búsqueda en la parte superior derecha.

Todo lo que queda por hacer es, para nosotros, proporcionar una entrada al elemento google-map-search, a través de la propiedad de consulta de nuestro elemento personalizado.

Por lo tanto, vinculamos nuestra propiedad "consulta", a la entrada de hierro en el formulario de búsqueda. al igual que:

```
<input is="iron-input" placeholder="Search" type=text name="search" bind-value="{{query}}">
```

Nota:

Enlazamos la consulta de propiedades de nuestro elemento personalizado, a la entrada de búsqueda usando,

Atributo de "valor de enlace" de una entrada de hierro.

Entonces, bind-value = {{query}} en la entrada de hierro anterior, se asegura de que cualquier cambio en el texto dentro de la entrada de hierro se notifique a la propiedad de consulta.

Con nuestra configuración actual,

Para cada alfabeto que un usuario escribe en el cuadro de búsqueda, se realiza una búsqueda, lo que ralentiza el mapa y consume el límite de nuestra clave API.

¿Por qué?

Porque, la consulta de propiedades de nuestro elemento personalizado, está vinculada a la entrada de hierro, y cada letra escrita en la entrada de hierro, cambia el valor de "consulta", que luego afecta el html en

```
<google-map-search map="[[map]]" query="[[query]]">
```

causando una búsqueda para suceder

Nuestra solución es simple! Solo enlace el atributo de "consulta" del elemento de búsqueda de mapa de google, a una propiedad separada.

Digamos que creamos una propiedad "searchQuery" para nuestro elemento personalizado.

Agregue esto, debajo de la propiedad "consulta" en la llamada Polymer

```
searchQuery :{

    type:String

}
```

A continuación, cambie el enlace en la llamada del elemento google-map-search en el DOM de nuestro elemento personalizado

al igual que:

```
<google-map-search map="[[map]]" query="[[searchQuery]]">
```

Por último, ¿nota que agregamos un icono de hierro (icono de búsqueda) a nuestro formulario de

búsqueda?

```
<iron-icon icon="icons:search" on-tap=search></iron-icon>
```

Agreguemos esa función de búsqueda a nuestro elemento personalizado. Se llama al tocar el icono de búsqueda.

En nuestro método de búsqueda, asignamos lo que haya en el cuadro de búsqueda, al atributo de "consulta" de google-map-search!

Agregue esta función, después del objeto de "propiedades" en la llamada de polímero para el registro.

```
//paste this after, the properties object

search: function() {
this.query = this.searchQuery;
}
```

¡Muy agradable! Por lo tanto, ahora, una búsqueda ocurre solo al tocar el ícono de búsqueda.

Mostrar los resultados de la búsqueda

Tenemos

- Rindió un mapa
- Pineado un formulario de búsqueda en el mapa
- Escribió la funcionalidad para buscar en el toque del icono de búsqueda

Ahora tenemos que mostrar los resultados.

Para eso, debemos vincular los resultados del elemento de búsqueda de mapa de google, en una propiedad local.

Entonces, ¡Vamos a crear una propiedad para nuestro elemento g-map personalizado, y llamémosle resultados duh!

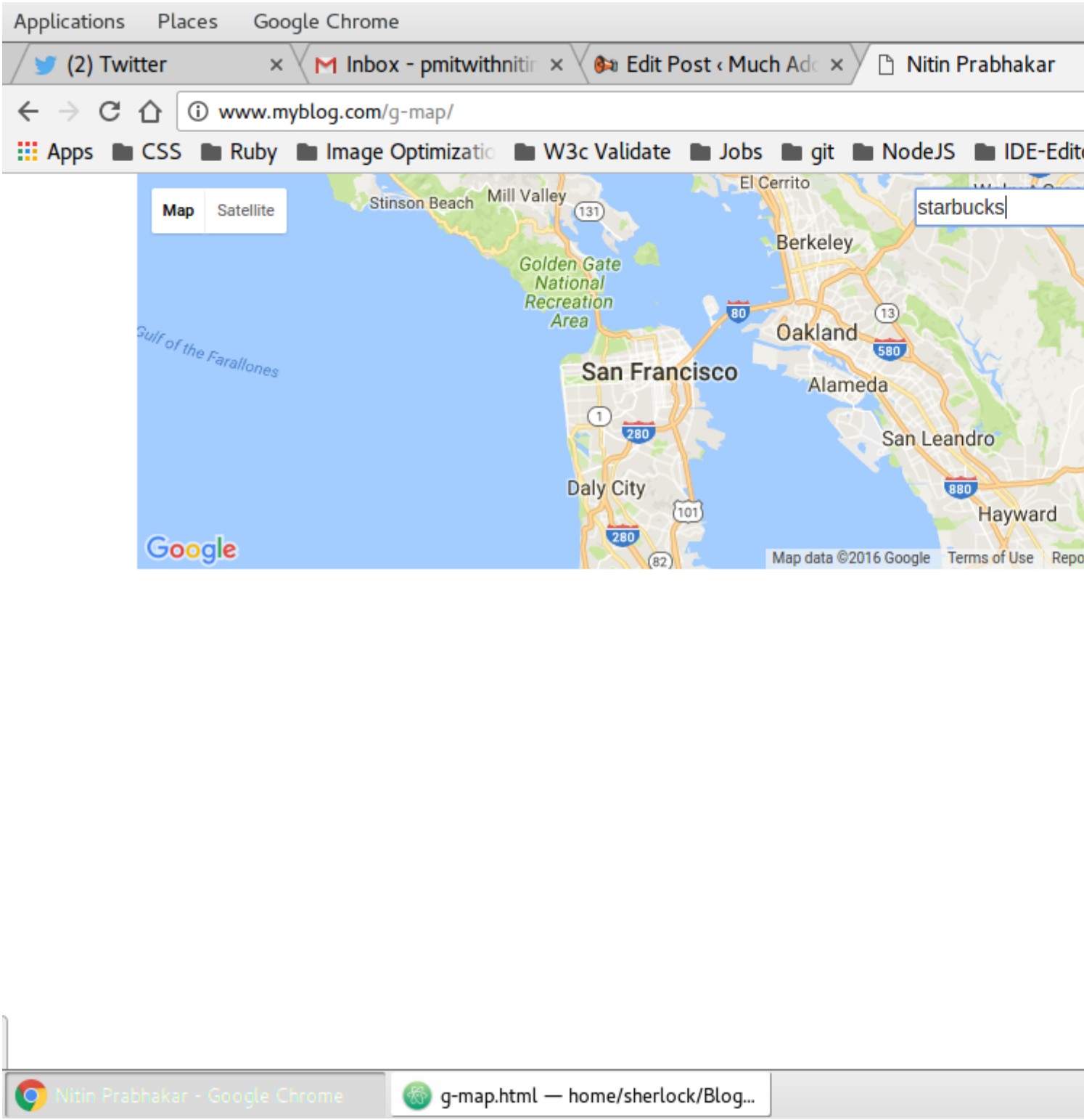
En la llamada de registro de Polymer, agregue los resultados de propiedad de tipo objeto en la declaración de propiedades

```
results:{

type:Object

}
```

Y enlace la propiedad de resultados de google-map-search a la propiedad de resultados local que definió anteriormente.

```
<google-map-search results={{results}} map=[[map]] query=[[query]]></google-map-search>
```

Con eso en su lugar, vamos a actualizar la página y buscar **Starbucks** en el mapa.



¡No pasó nada!

Bien duh Acabamos de enlazar los resultados de google-map-search a la propiedad de resultados. ¿Escribimos algo para mostrarlos en el mapa? ¡NO!

Necesitamos que.

Por lo tanto, sabemos que el elemento Polymer google-map-search devuelve una matriz de

marcadores como resultados.

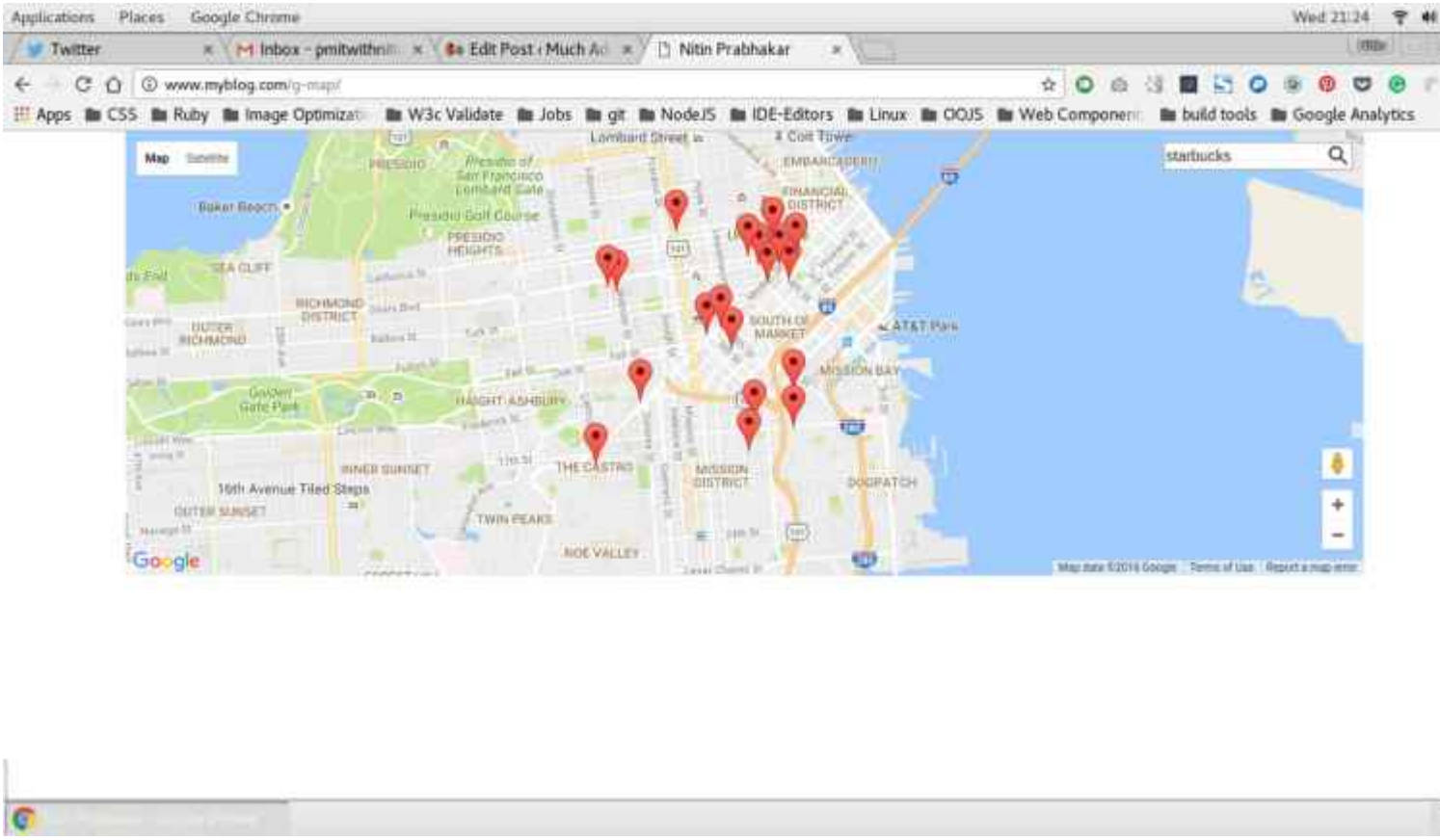
Necesitamos recorrer la matriz y colocar marcadores en el mapa.

Entonces, escribimos una plantilla de repetición de dom, dentro de nuestro elemento, para mostrar los marcadores para cada resultado.

al igual que:

```
<google-map api-key=[[apiKey]] map=[[map]]>

    <template is="dom-repeat" items="{{results}}" as="marker">
        <google-map-marker latitude="{{marker.latitude}}" longitude="{{marker.longitude}}">
            <h2>{{marker.name}}</h2>
            <span>{{marker.formatted_address}}</span>
        </google-map-marker>
    </template>

</google-map>
```

Y ahora en la actualización, vemos esto



¡Muy agradable!

Hemos agregado con éxito la funcionalidad de búsqueda a nuestro elemento personalizado.

Todo lo que necesitamos hacer es

escribe esto en nuestro index.html

```
<g-map api-key="Whatever API key you generated"></g-map>
```

Y obtienes un mapa en el que puedes buscar!

D - Caching Search Results

Nuestro objetivo, sin embargo, era

- hacer una lista de lugares
- marcarlos todos en el mapa y
- Marque cualquier otro lugar de nuestra elección, utilizando el cuadro de búsqueda

Tenemos un cuadro de búsqueda, donde el usuario puede buscar una consulta a la vez. Para poder acomodar varias consultas, debemos almacenar los resultados de cada consulta, antes de mostrar los marcadores y colocarlos en el mapa.

Entonces, agreguemos

- Una propiedad dice ubicaciones que aceptan una variedad de ubicaciones
- Un caché de propiedades que almacena los resultados de cada consulta en la matriz de ubicación pasada a nuestro elemento personalizado

```
cache:{

type:Object

},

locations:{

type:Object

}
```

A continuación, escribimos una función para almacenar en caché cada resultado de búsqueda, como resultado de una búsqueda para cada lugar en la matriz de ubicación.

¡Es bastante simple!

Ya estamos reuniendo los resultados en una propiedad llamada "resultados", que luego estamos haciendo un bucle para mostrar los marcadores.

Todo lo que tenemos que hacer ahora es insertar cada resultado de búsqueda en una matriz, ¡y nuestra caché está lista!

Necesitamos usar un evento para activar el almacenamiento en caché. Ese evento, se llama "on-google-map-search-result" y se dispara, después de que el elemento google-map-search termine una búsqueda.

Por lo tanto, "on-google-map-search-result", llamamos a una función decir "cacher", para almacenar en caché los resultados de la búsqueda actual en una propiedad llamada caché.

```
<google-map-search map=[[map]] query=[[searchQuery]] on-google-map-search-results=cacher>
```

Nota:

Normalmente, para insertar un elemento en una matriz, usamos el método de empuje de vainilla

por ejemplo: tengo un nuevo lugar para agregar a mi lista de lugares que quiero marcar, digamos NewYork.

Si la matriz a la que quiero agregarla se llama ubicaciones, escribiría

```
locations.push('New York');
```

Sin embargo, en nuestro caso, debemos insertar cualquier resultado de google-map-search en una matriz llamada caché. Luego debemos recorrer el caché con los últimos resultados incluidos. No el caché existente.

Ahora, usamos la plantilla dom-repeat para recorrer los resultados y colocar marcadores en el mapa.

Sin embargo, si pasamos una matriz de ubicación como tal:

```
<g-map locations=["Iceland","Argentina","London"]></g-map>
```

Luego, la propiedad de resultados de nuestro elemento personalizado, se sobrescribirá para cada elemento en esa matriz.

Por lo tanto, llamamos al método cacher en el evento de google-map-search-results, e insertamos el resultado actual en la propiedad de caché.

Esto no puede ser un empuje de vainilla como en la nota anterior.

Necesitamos hacerle saber al dom-repetidor, que nuestra memoria caché se ha sobrescrito con cada nueva búsqueda.

Por lo tanto, utilizamos un impulso especial que Polymer envía, así:

```
cacher: function() {
this.push('cache', this.results);
}
```

Agregue esa función, después de la función de búsqueda que escribimos anteriormente.

La sintaxis implica que la memoria caché de la propiedad se va a mutar, mediante la adición del valor actual de los resultados de la propiedad y cualquier repetidor de dom que use la propiedad de la memoria caché para realizar un bucle, se debe notificar la mutación.

Por último, cambiamos la plantilla dom-repeat para recorrer en bucle en lugar de resultados

Nota:

caché es una matriz de matrices.

Cada elemento de la matriz de caché, es una matriz de resultados.

Así nuestro dom-repetidor, será así:

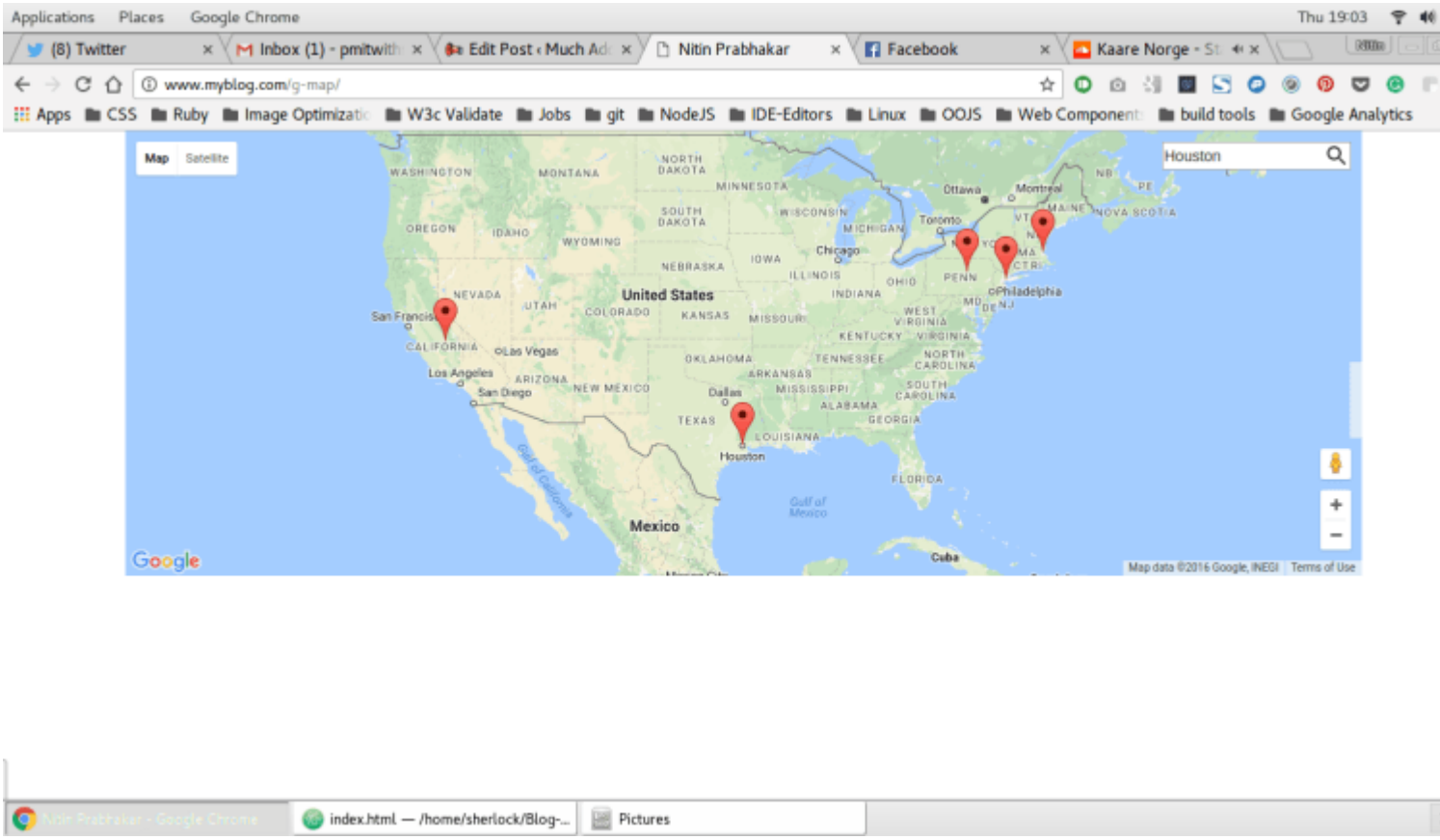
```
<template id="resultList" is="dom-repeat" items="[[cache]]">

    <template is="dom-repeat" items="[[item]]" as="marker">

        <google-map-marker latitude="{{marker.latitude}}" longitude="{{marker.longitude}}">
            <h2>{{marker.name}}</h2>
            <span>{{marker.formatted_address}}</span>
        </google-map-marker>
    </template>
</template>
```

¡Muy agradable! ¡Tenemos el elemento personalizado todo codificado y listo para ser utilizado!

Probémoslo pasando una matriz de ubicaciones desde index.html ¿lo haremos?

```
<g-map

api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

locations='["Boston","NewYork","California","Pennsylvania"]'>
</g-map>
```

y ahí vamos!



El polímero personalizado completo de Google Map con el caché incorporado

```
<link rel=import href="../bower_components/google-map/google-map.html">
<link rel=import href="../bower_components/google-map/google-map-marker.html">
<link rel=import href="../bower_components/google-map/google-map-search.html">
<link rel=import href="../bower_components/google-map/google-map-directions.html">
<link rel=import href="../bower_components/iron-input/iron-input.html">
<link rel=import href="../bower_components/iron-icon/iron-icon.html">
<link rel=import href="../bower_components/iron-icons/iron-icons.html">
<link rel=import href="../bower_components/iron-icons/maps-icons.html">
<dom-module id="g-map">
    <template id="google-map">
        <div class="map-container">
            <google-map-search map="[[map]]" query="[[query]]" results="{{results}}" on-
google-map-search-results=cacher>
            </google-map-search>
            <google-map api-key="[[apiKey]]" map="{{map}}" on-google-map-ready="_onMapResolve"
fit-to-markers disable-zoom single-info-window>
                <template id="resultList" is="dom-repeat" items="[[cache]]">

                    <template is="dom-repeat" items="[[item]]" as="marker">

                        <google-map-marker latitude="{{marker.latitude}}"
longitude="{{marker.longitude}}">
                            <h2>{{marker.name}}</h2>
                            <span>{{marker.formatted_address}}</span>
                        </google-map-marker>
                    </template>
                </template>
            </google-map>
            <div class="search_form">
                <p>
```

```
                    <iron-icon icon=icons:search on-tap=search></iron-icon>
                    <input is=iron-input placeholder="Search" type=text name=start bind-
value={{searchQuery}}>
                </p>
            </div>
        </div>
    </template>
    <script>
        Polymer({
            is: "g-map",
            properties: {
                apiKey: {
                    type: String,
                    value: "AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"
                },
                map: {
                    type: Object
                },
                query: {
                    type: String,
                    value: ""
                },
                locations: {
                    type: Array,
                    value: []
                },
                cache: {
                    type: Array,
                    value: []
                },
                results: {
                    type: Array,
                    value: function() {
                            return [];
                        }
                    }
            },
            _onMapResolve: function() {
                var search = document.querySelector("google-map-search");
                this.locations.forEach(function(location) {
                    search.query = location;
                })
            },
            search: function() {
                this.query = this.searchQuery;
            },
            cacher: function() {
                this.push('cache', this.results);
            }
        })
    </script>
</dom-module>
```

Lea Marca de mapa de Google con construido en caché en línea:
https://riptutorial.com/es/polymer/topic/7487/marca-de-mapa-de-google-con-construido-en-cache

Capítulo 10: Modales reutilizables con polímero

Sintaxis

- Declaración del elemento: <tool-bar link2-share = "" link2-fork = "" modal-id = "" title = ""> </tool-bar>

Observaciones

Nota:

El código a través de este escrito, no es una copia de trabajo. Debe reemplazar los rellenos para hrefs, src y nombres de proyectos.

El código ilustra solo una prueba de concepto.

Para ver el elemento personalizado en acción,

ven aquí

Y, para navegar a través del uso y la estructura del elemento personalizado,

ven aquí

El ***uso*** está en "index.html"

El ***elemento*** está en "elements / tool-bar.html"

Examples

Modales

¿Desea agregar diseño de material a su cartera de negocios o de carrera? ¿No puedes resistirte a usar un modal? que aparece al hacer clic en las tarjetas nítidas que pretende diseñar, para cada uno de sus proyectos / productos!

Digamos que, para cada proyecto que haya realizado, o para cada producto que ha diseñado, necesita una tarjeta de material. Cada tarjeta debe tener

- Un icono, que muestra un cuadro de información, que enumera todas las características de
- Su producto o proyecto, en detalle. Un icono de "compartir esto", que comparte
- su producto o proyecto a través de las redes sociales Un ícono “bifurque esto”, que abre la página github para su producto / proyecto

Con HTML nativo, y suponiendo que use bootstrap o cualquier otro diseño de Flex Box, tendría que

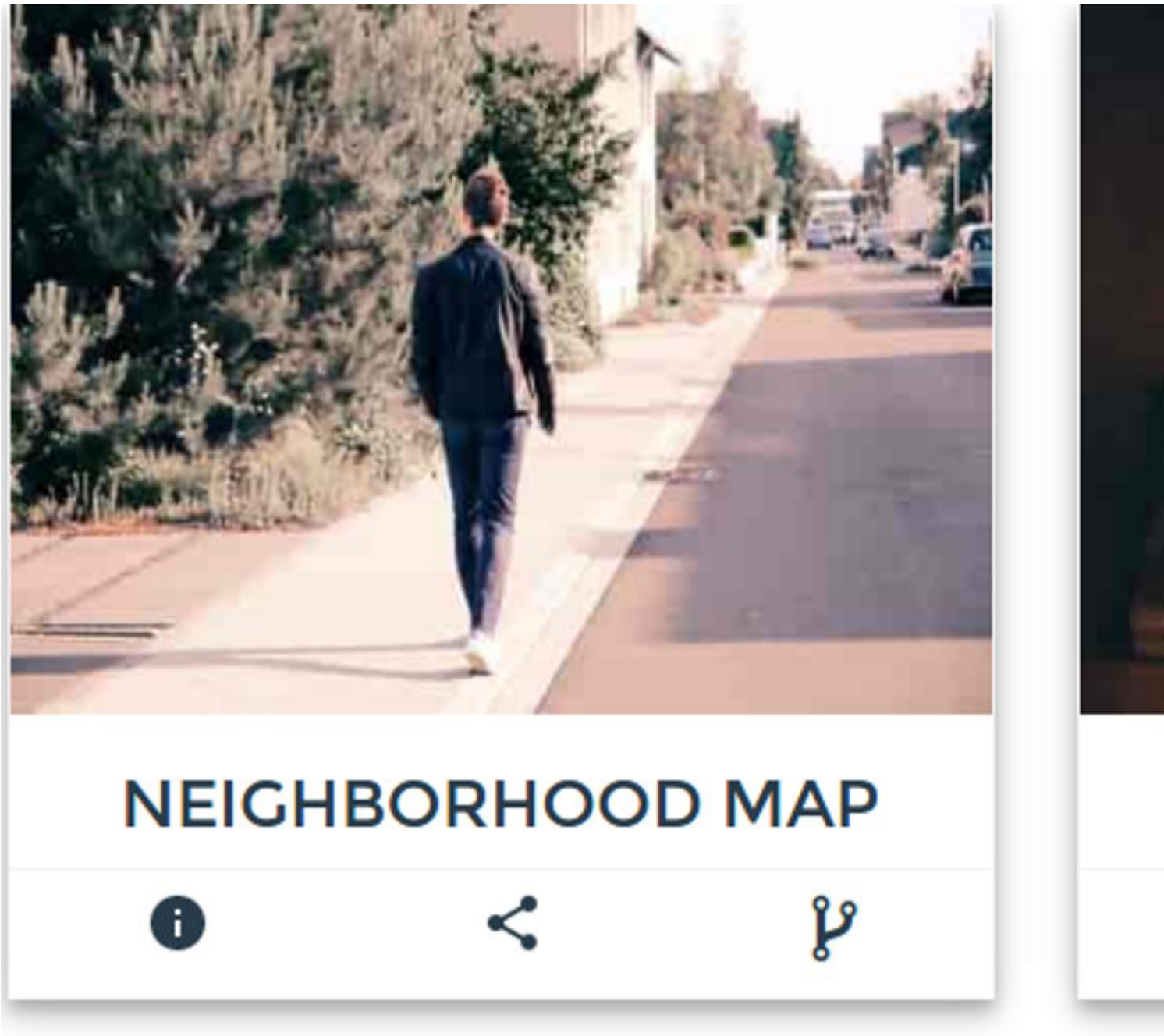
- Escribe un div de contenedor para cada nuevo proyecto con una fila de clase
- Envuelve tu imagen de héroe para cada proyecto en una columna de 12 cols de ancho
- Escribir otra fila
- Envuelva cada enlace (información | compartir | bifurcación) en una columna de 4 cols de ancho.
- Inyecte estas dos filas en un contenedor

Para principiantes. Una vez que haya hecho lo anterior para todos sus proyectos, debe trabajar en la creación de la ventana emergente requerida para cada proyecto.

- Descarga toneladas de javascript y css (al personalizar bootstrap para funcionalidad modal)
- Usted escribe un Modal con una identificación única, para diferenciar la información de cada proyecto (modal-P1 para el Proyecto 1, modal-P2 para el Proyecto 2 y así sucesivamente)
- Luego, componga el html requerido, para mostrar la información de cada proyecto, en su Modal correspondiente.

¿Cuáles son nuestras metas?

Solo para reiterar, visualmente, si tiene dos proyectos en su cartera, el objetivo es tener tarjetas como las siguientes:



Así que tenemos dos preocupaciones para cada proyecto

- La imagen del héroe
- La barra de herramientas con información (Pops up a Modal), compartir y bifurcar enlaces

Primero, abordemos lo que sucede cuando un usuario hace clic en el enlace de información en la barra debajo de cada proyecto. Necesitamos un Modal para que aparezca, con más información sobre el proyecto.

Entonces, ¿cómo exactamente configura un Modal con bootstrap?

- Escribes esto para activarlo, para cada proyecto.

```
<button type="button" class="btn btn-info btn-lg" data-toggle="modal" data-
target="#myProject1"><img src="path-to-info-icon"></img></button>
```

- Luego escribes el cuerpo del Modal así:

```
<div id="myProject1" class="modal fade" role="dialog">
     <div class="modal-dialog">
         <div class="modal-content">
             <div class="modal-header">
                 <button type="button" class="close" data-dismiss="modal"></button>
                 <h4 class="modal-title">Project Title</h4>
             </div>
             <div class="modal-body">
                 <p>Some text in the modal.</p>
             </div>
             <div class="modal-footer">
                 <button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
             </div>
         </div>
     </div>
</div>
```

y dependiendo del contenido y la duración de la Descripción del proyecto, ¡realmente se pierde un poco al monitorear la apertura y el cierre de una sopa div en el cuerpo del Modal!

Además, ese marcado es francamente tedioso para componer TODO EL MOMENTO que necesitas un modal.

¡Uf! ¿Y terminó ahí? No. Todavía necesitas escribir para cada proyecto,

- La imagen del héroe de la carta del proyecto, que se muestra sobre la barra de herramientas.

Entonces, ¿cómo agregamos la imagen del héroe para cada proyecto? MÁS html!

La imagen del héroe necesita.

- Una fila propia
- Una columna de 12 cols de ancho dentro de esa fila.

¿Cómo se vería el html completo para que mostremos un proyecto como una tarjeta que se muestra arriba?

```
<article id="project-neighbourhood">

    <div class="row">

        <div class="col-12">

            <img src="path-to-hero-image"></img>

        </div>

    </div>

    <div class="row">

        <div class="col-12">
            <div class="row">
                <div class="col-4">
```

```
                    <button type="button" class="btn btn-info btn-lg" data-toggle="modal"
data-target="#myProject1"><img src="path-to-info-icon"></img></button>
                </div>
                <div class="col-4">
                    <button class="btn btn-lg" id="share-on-g-plus"><img src="path-to-share-
icon"></img></button>
                </div>
                <div class="col-4">
                    <button class="btn btn-lg" id="fork"><img src="path-to-fork-
icon"></img></button>
                </div>
             </div>
        </div>
    </div>
</article>
```

¡Ahora digamos que tienes 10 proyectos para presumir! ¡Necesitas reescribir ese marcado desorden 10 veces! ¡Más 10 modos diferentes!

No tengo paciencia para reescribir ese html y 10 modos, ¡y sé que usted tampoco!

¿Modales, con polímero?

Nota

> Supongo que sabes
>
> Especificaciones de componentes web
>
> Polímero de google

¿Qué pasaría si pudiéramos tener un elemento personalizado, por ejemplo?

```
<tool-bar></tool-bar>
```

¿Y mágicamente hizo todo lo que el marcado Modal desordenado pudo?

Tentador eh!

¿Cómo se especifica qué Modal pertenece a qué proyecto?

¡Sencillo! solo escribe

```
<tool-bar modal-id="Project-cats"></tool-bar>
```

Así que se reserva el marcado con un ID de "Proyecto-gatos" para el proyecto en gatos, por ejemplo.

¿Cómo escribes lo que entra en el modal? ¡sencillo! Simplemente escriba su marcado normal, envuelto, dentro de la etiqueta personalizada ""

Al igual que:

```
<tool-bar modal-id="Project-cats">

<div class="col-12 modal-desc">
    <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.</p>
</div>

</tool-bar>
```

¿No es eso lo suficientemente simple?

Y si te estás preguntando cómo se ve el marcado completo? Incluyendo los enlaces compartir y la bifurcación, ver más abajo.

```
<div class=row>

    <div class="col-12"> <img src="path-to-hero-image"></img></div>

</div>

<div class="row">

    <tool-bar

    modal-id="Project-cats"

    link2-share="http://www.project-cats.com/kitten"

    link2-fork="https://github.com/myusername/my-project-cats">

        <div class="col-12 modal-desc">

            <p> yadda yadda blah blah</p>

        </div>

    </tool-bar>

</div>
```

Mucho mejor que reescribir la sopa div completa eh!

Por supuesto, puede acortarlo aún más y hacerlo completamente composible al abstraer la implementación de la imagen del héroe dentro del elemento personalizado, pero permítanos suspenderlo, como una preocupación posterior.

Barra de herramientas reutilizable con modal | Compartir | Iconos de horquilla - Codificando el elemento

Hasta ahora, hemos definido lo fácil que es escribir un elemento personalizado que oculta el html desordenado detrás de él y le da al usuario, una escritura fácil y un breve formato.

¡Es hora de codificarlo!

Nuestro elemento personalizado, para mostrar la barra debajo de la imagen del héroe debe

- Aceptar un enlace para ser compartido
- Aceptar un enlace al Repo para ser bifurcado
- Acepte un modalId para diferenciar entre otros Modales.
- Aceptar un título para el modal

Registrar el elemento personalizado de la barra de herramientas con Modal | Compartir | Iconos de horquilla

Registremos nuestro elemento personalizado, que acepta la lista de propiedades anterior.

```
<script>
Polymer({
    is:"tool-bar",
    properties:{
        link2Share:{
            type:String,
            value:""
        },
        link2Fork:{
            type:String,
            value:""
        },
        modalId:{
            type:String,
            value:""
        },
       title:{
            type:String,
            value:"myModal"
        }
    }
});
</script>
```

¡Muy agradable! A continuación, agregue el HTML que necesitamos.

En primer lugar, es necesario agregar los íconos de Modal / Share / Fork. Una simple ul será suficiente. Además, usaremos íconos de hierro de polímeros para mostrar íconos de Modal / Share / Fork.

Nota: Iconos de hierro de polímero

> Instale los conjuntos de iconos como tal, en la raíz de su proyecto.
>
> ```
> bash $ bower install --save PolymerElements/iron-icon
>
> bash $ bower install --save PolymerElements/iron-icons
> ```

A continuación, incluya los elementos de polímero instalados en la declaración del elemento

personalizado, como así

Nota:

A todos los efectos prácticos, nuestro elemento personalizado, se alojará en el interior.

Proyecto-raíz / elementos

```
<link rel="import" href="../bower_components/iron-icon/iron-icon.html">
<link rel="import" href="../bower_components/iron-icons/iron-icons.html">
<link rel="import" href="../bower_components/iron-icons/social-icons.html">
```

¡Muy agradable! Si se está preguntando cómo usar íconos de hierro, vea la nota a continuación.

Nota: Uso del icono del hierro y gramática.

Si necesitamos usar el ícono de información para nuestra ventana emergente modal, escribimos el marcado de la siguiente manera:

```
<iron-icon icon="icons:info"></iron-icon>
```

Si queremos que el icono de compartir en las redes sociales, entonces el atributo del icono de arriba, se convierte en

icon = " **social: compartir** "

Así que el significado gramatical, necesito el conjunto de iconos de "SOCIAL" y quiero el icono COMPARTIR de ese conjunto.

Alternativamente, puede usar font-awesome, en cuyo caso, puede usar,

Elige lo que mejor te parezca.

Una vez que las inclusiones están listas, escribimos la plantilla para nuestro elemento personalizado. Utilizaremos los enlaces de una manera para los datos en el marcado de nuestro elemento personalizado. Enlazaremos lo que sea pasado como atributo por el usuario, a sus propiedades correspondientes de nuestro elemento, mientras nos embarcamos en la llamada de registro de nuestro elemento.

Así que el modelo html para nuestro elemento personalizado es:

```
<dom-module id="tool-bar">

    <template id="tool-box">

        <ul class="flex-wrap toolbar">
            <li>
                <iron-icon icon="icons:info" id="anchor-for-[[modalId]]"
onclick="clickHandler(event)"></iron-icon>
            </li>
            <li>
                <a href="https://plus.google.com/share?url=[[link2Share]]" target="_blank"
```

```
><iron-icon icon="social:share"></iron-icon></a>
            </li>
            <li>
                <a href="[[link2Fork]]" target="_blank"><i class="fa fa-2x fa-code-fork" aria-
hidden=true></i></a>
            </li>
        </ul>

    </template>

</dom-module>
```

¡Guay! Así que la barra de herramientas tiene su plantilla ul, que enumera los tres enlaces.

- icono de información - ventana emergente modal
- comparte este ícono - comparte en google
- Bifurca este icono - Bifurca el repositorio

Además, hemos vinculado los atributos que el usuario escribe, a las propiedades respectivas.

Nota: hacer que el ID modal sea único

Hemos abordado la singularidad del icono de información, especificando el atributo id como tal

```
id="anchor-for-[[modalId]]" onclick="clickHandler(event)"></iron-icon>
```

Así que cada proyecto, tendrá un ID de anclaje único.

es decir,

Si el usuario pasa "project-cats" como modal-id, en la llamada a nuestro elemento personalizado, el id de ancla sería "anchor-for-project-cats"

Si el usuario agrega otro proyecto y pasa "project-puppy" como el atributo modal-id, la identificación del ancla sería anchor-for-project-puppy

y así.

Pero, ¿cómo nos aseguramos de que cada ícono de información se asigne a su propio Modal?

Al " **icono de hierro** " para **información** , se le debe asignar un atributo de " ***diálogo de datos*** ", que también debe ser único.

El atributo data-dialog, es similar al atributo data-target en el modal de arranque. Debe ser único para cada proyecto nuevo.

Se lo dejamos al usuario, para mantenerlo único. Por lo tanto, el usuario no puede tener el mismo atributo de ID de modal para diferentes llamadas a `<tool-bar>` .

Entonces, como se indica en la nota anterior, necesitamos mapear el ícono de información de cada Proyecto, a un diálogo de datos.

Vamos a añadir esa funcionalidad.

Utilizamos el método ready. Es muy similar al de Jquery.

```
$.ready();
```

.Se invoca, cuando el elemento personalizado, se termina de cargar.

Agregue esta función, después del objeto de propiedades, en la llamada de polímero para el registro.

```
ready: function() {
    document.querySelector("#anchor-for-" + this.modalId).setAttribute('data-dialog',
this.modalId);
}
```

¡Muy genial! Así que ahora tenemos

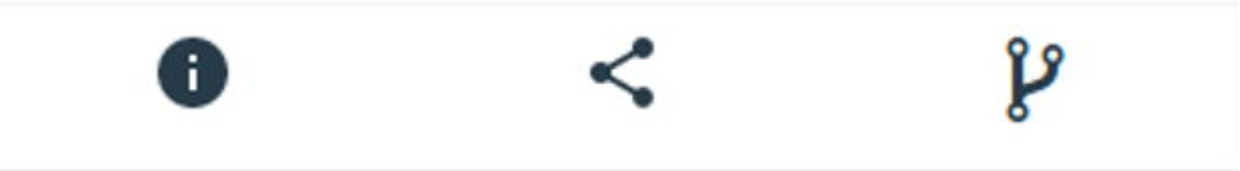
- Un elemento personalizado registrado como un elemento de polímero.
- Modal | Compartir | Iconos de horquilla añadidos al elemento personalizado.

Se adjunta un modal al ícono de hierro de información, con el id "ancla para [lo que sea que se pase como modal-id]". Eso es cortesía del método listo de nuestro elemento.

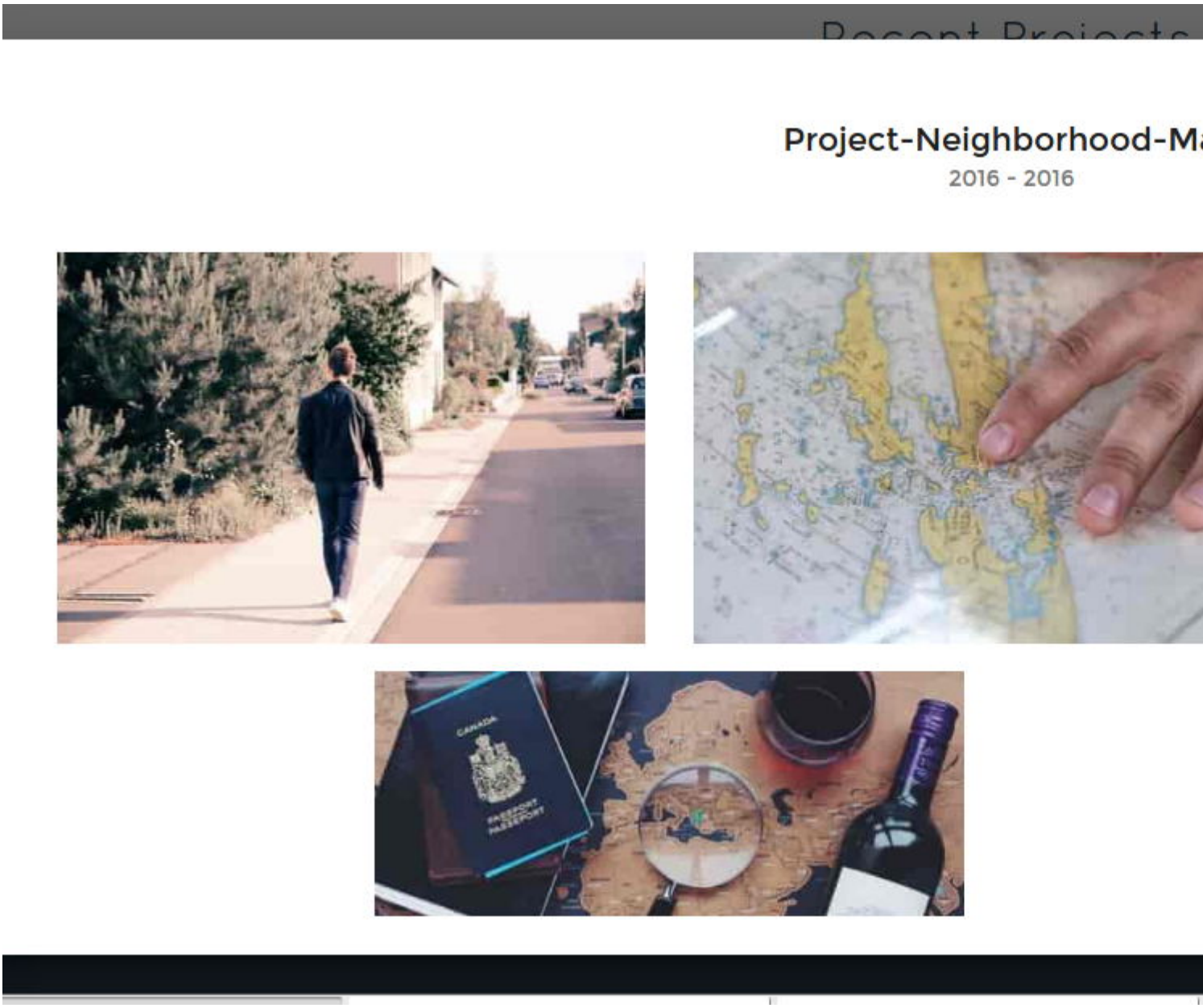
Así que, hasta ahora, escribiendo.

```
<tool-bar
link2-share="http://www.myportfolio.com/project-neighbourhood-map"
link2-fork="https://github.com/me/project-neighbourhood-map"
modal-id="project-neighbourhood-map">
</tool-bar>
```

Produce esto:



Pero cuando el usuario hace clic en ese ícono de información, algo así debería suceder.



y nuestro objetivo era envolver esta descripción completa del proyecto, dentro de la etiqueta `<tool-bar>` en la página de usuarios.

Recuerde Shadow DOM? Necesitamos tener un contenedor en nuestro elemento personalizado, que proporcione el DOM de sombra para los contenidos de nuestro Modal. El contenido en sí mismo, está envuelto dentro de una etiqueta `<content>` dentro del shadowDOM del elemento personalizado.

Para el contenedor que aloja la descripción del Proyecto, usamos el elemento Polímero " **diálogo de papel** ". Paper-dialog, se asigna como id, lo que se pasa como el mod-id en la declaración en la página del usuario.

Entonces, agregue este marcado, después de ***ul*** en el DOM del elemento personalizado de la barra de herramientas

```
<paper-dialog id="[[modalId]]" modal>
    <div class="text-right modal-close">
        <iron-icon icon="icons:power-settings-new" dialog-dismiss></iron-icon>
```

```
    </div>
    <h2 class="text-center text-capitalize">[[title]]</h2>
    <paper-dialog-scrollable>
        <div class="container-fluid">
            <div class="row flex-wrap info">
                <content></content>
            </div>
        </div>
    </paper-dialog-scrollable>
</paper-dialog>
```

Nota: Papel-diálogo-desplazable

> Usamos ***Paper-dialog-scroll-roll*** , para envolver la descripción de nuestro Proyecto, de modo que, Cualquier descripción larga, no se desborda.

El último bit es agregar la funcionalidad onclick para cuando se abra el modal, al hacer clic en el icono de información.

Cuando ocurre un clic, como es normal, se desencadena un evento, que pasamos a nuestra función ClickHandler así:

```
<iron-icon icon="icons:info" id="anchor-for-[[modalId]]" onclick="clickHandler(event)"></iron-
icon>
```

Así que una vez que la función clickHandler consigue que se le pase el evento,

- Necesita reunir, qué elemento activó el clic,
- Necesita reunir, el atributo de diálogo de datos de ese elemento, para determinar qué modal abrir (¿Recuerdas que cada proyecto tiene su propia identificación modal?)
- Entonces necesita llamar al método abierto, exactamente para ese modal

Entonces, la función es así:

Agregue esto, antes de la llamada de registro del elemento. Esto no es una parte del elemento, pero está dentro del elemento.

```
function clickHandler(e) {
    var button = e.target;
    while (!button.hasAttribute('data-dialog') && button !== document.body) {
        button = button.parentElement;
    }

    if (!button.hasAttribute('data-dialog')) {
        return;
    }

    var id = button.getAttribute('data-dialog');
    var dialog = document.getElementById(id);
    if (dialog) {
        dialog.open();
    }
}
```

Entonces, con eso, hemos completado la codificación del elemento personalizado, `<tool-bar>` .

La imagen completa. Elemento personalizado

```
<link rel="import" href="../bower_components/iron-icon/iron-icon.html">
<link rel="import" href="../bower_components/iron-icons/iron-icons.html">
<link rel="import" href="../bower_components/iron-icons/social-icons.html">
<link rel="import" href="../bower_components/paper-dialog/paper-dialog.html">
<link rel="import" href="../bower_components/paper-button/paper-button.html">
<link rel="import" href="../bower_components/paper-dialog-scrollable/paper-dialog-
scrollable.html">
<dom-module id="tool-bar">
        <template id="tool-set">
            <ul class="flex-wrap toolbar">
                <li>
                    <iron-icon icon="icons:info" id="anchor-for-[[modalId]]"
onclick="clickHandler(event)"></iron-icon>
                </li>
                <li>
                    <a href="https://plus.google.com/share?url=[[link2Share]]"
target="_blank">
                        <iron-icon icon="social:share"></iron-icon>
                    </a>
                </li>
                <li><a href="[[link2Fork]]" target="_blank"><i class="fa fa-2x fa-code-fork"
aria-hidden=true></i></a>
                </li>
            </ul>
            <paper-dialog id="[[modalId]]" modal>
                <div class="text-right modal-close">
                    <iron-icon icon="icons:power-settings-new" dialog-dismiss>
</iron-icon>
                </div>
                <h2 class="text-center text-capitalize">[[title]]</h2>
                <paper-dialog-scrollable>
                    <div class="container-fluid">
                        <div class="row flex-wrap info">
                            <content></content>
                        </div>
                    </div>
            </paper-dialog-scrollable>
        </paper-dialog>
    </template>

    <script>
            function clickHandler(e)
            {
                 var button = e.target;
                 while (!button.hasAttribute('data-dialog') && button !== document.body) {
                    button = button.parentElement;
                 }

                 if (!button.hasAttribute('data-dialog'))
                 {
                   return;
                 }

                var id = button.getAttribute('data-dialog');
                var dialog = document.getElementById(id);
                if (dialog) {
```

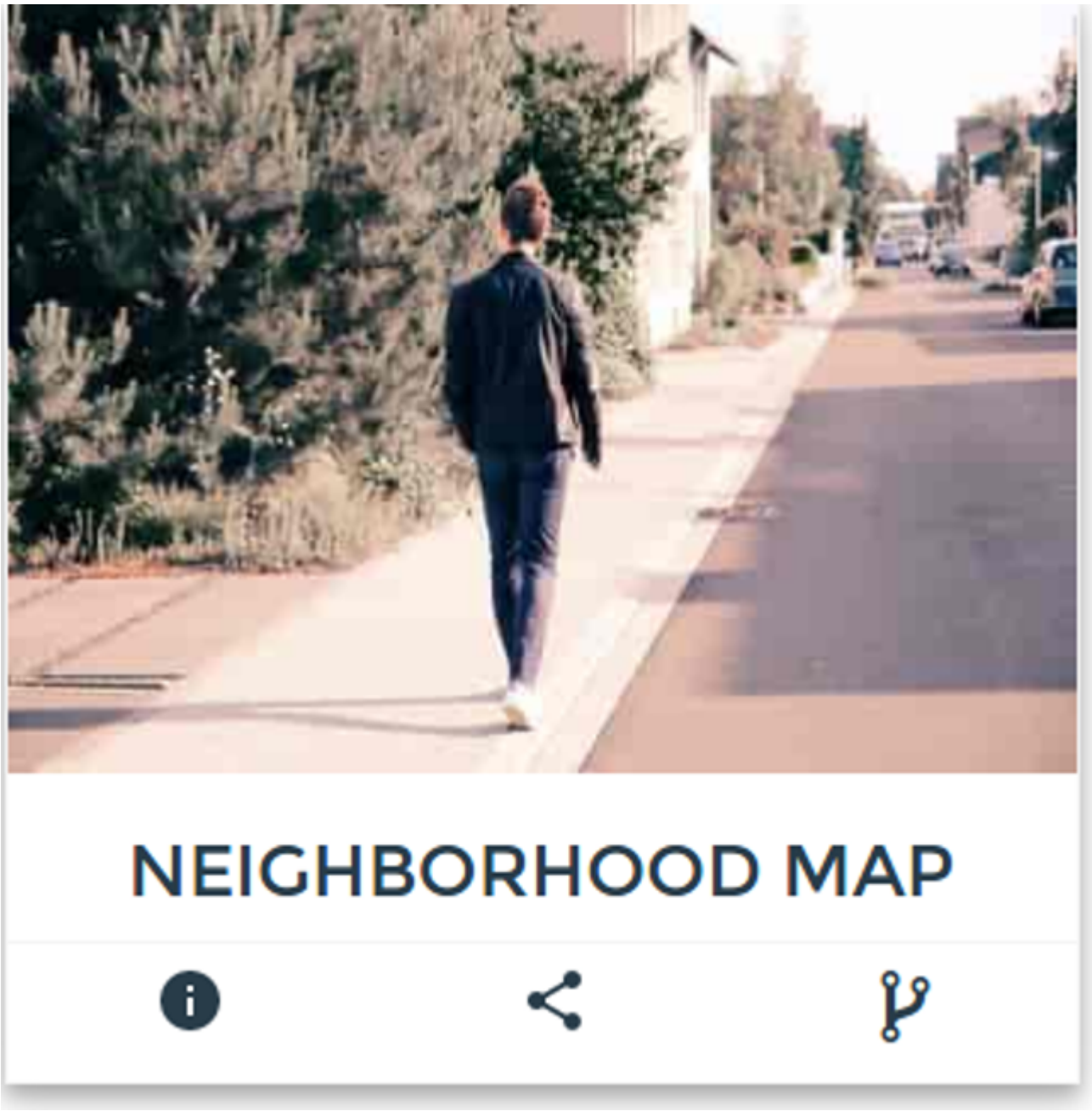
```
                dialog.open();
              }
           }
      Polymer({
             is: "tool-bar",
            properties: {
                  link2Fork:{
                      type:String,
                      value:""
                  },
                 link2Share: {
                       type: String,
                       value: ""
                  },
                 title: {
                       type: String,
                       value: null
                 },
                modalId:{
                       type:String,
                       value:""
              }
             },
            ready: function() {
                  document.querySelector("#anchor-for-" + this.modalId).setAttribute('data-
dialog', this.modalId);
            }
         });
    </script>
</dom-module>
```

Usando el elemento personalizado con Modal | Compartir | Iconos de horquilla

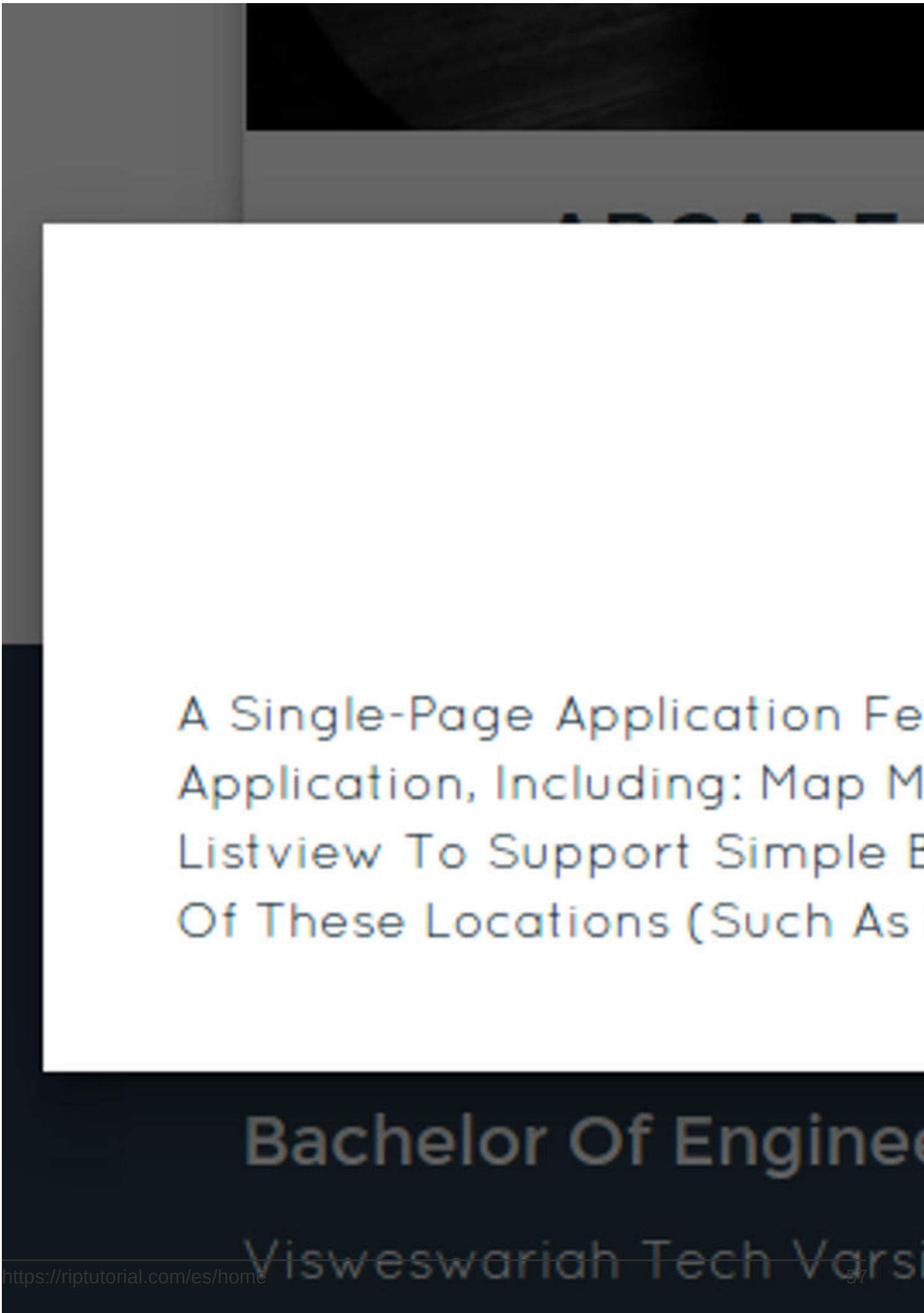
En la página que desee que muestre su cartera de productos / proyectos, invoque el elemento personalizado así:

```
<article id="project-neighbourhood">
    <div class="row">
        <div class="col-12 hero">
            <img src="path-to-hero-image">
        </div>
    </div>
    <tool-bar link2-share="http://www.mywebsite.com/neighbourhood" link2-
fork="https://github.com/myusername/neighbourhood" modal-id="project-neighbourhood-map"
title="project-neighbourhood-map">
                <div class="col-12">
                      <p> a single-page application featuring a map of my neighborhood or a
neighborhood I would like to visit. I will then add additional functionality to this
application, including: map markers to identify popular locations or places you'd like to
visit, a search function to easily discover these locations, and a listview to support simple
browsing of all locations. I will then research and implement third-party APIs that provide
additional information about each of these locations (such as Street View images, Wikipedia
articles, Yelp reviews, etc).</p>
               </div>
    </tool-bar>
</article>
```

y obtienes esto como vista previa



Y cuando haces clic en información, obtienes esto.


A Single-Page Application Fe
Application, Including: Map M
Listview To Support Simple B
Of These Locations (Such As
Bachelor Of Engine
Visweswariah Tech Vars

https://riptutorial.com/es/polymer/topic/7244/modales-reutilizables-con-polimero

Capítulo 11: SUPER-Optimización para la producción.

Introducción

Una vez que haya finalizado su proyecto, se preguntará cómo cargará esas cargas de 100 importaciones HTML en su servidor web e incluso si lo hace, cuántas horas tomará cargar su sitio para un solo cliente. En este tema, verá cómo convertir el desastre de desarrollo en archivos html y js únicos y refinados.

Sintaxis

- npm install: guarda paquetes utilizando Node.js Package Manager
- npm install -g: guarda los paquetes de npm como globales (útil para los paquetes de interfaz de línea de comandos)
- cd: establece el foco de la línea de comandos en una biblioteca específica en la que se pueden realizar los procesos
- vulcanizar: Resume todas las importaciones de HTML a un solo archivo
- Crisper: Convierte Inline JS en archivo HTML a archivo JS externo

Examples

Instalando todas las herramientas necesarias.

Ejecute el siguiente comando uno por uno:

```
npm install -g vulcanize crisper
```

Utilizar

- Vulcanizar: tritura todos los archivos de importación HTML en un solo archivo
- Crisper: Extrae inline js a su propio archivo

Nota : Es posible que los usuarios de Ubuntu deban prefijar el comando anterior con `sudo` .

Poniendolo todo junto

Coloque todas las importaciones de HTML en sus archivos en un solo archivo `elements.html` . No se preocupe si un archivo se importa más de una vez, se reducirá a una sola importación.

Correr vulcanizar y crujir

en su archivo `elements.html` , ejecute los siguientes comandos:

```
cd PATH/TO/IMPORTFILE/
vulcanize elements.html -o elements.vulc.html --strip-comments --inline-css --inline-js
crisper --source elements.vulc.html --html build.html --js build.js
```

Vulcanize el código fuente recuperado de todas las importaciones, luego reemplazó las importaciones por su código fuente.
Crisper sacó todos los js del archivo elements.vulc.html, lo puso en un solo archivo build.js, estableció una etiqueta de `script` refiriera al archivo `build.js` en el archivo `build.html`

Minificar los archivos

- Abrir minificador de HTML
- Abrir `build.html`
- Copia todo su código.
- En el primer área de texto del minfier HTML, pegue el código que copió de `build.html`
- Haga **clic en el** botón **Minify**
- En el segundo textarea, aparecerá el código minificado. Entendido
- Cree un archivo `build.min.html` y pegue todo su código copiado en él
- Abrir JSCompress
- Seleccione la segunda pestaña, es decir, **suba archivos de JavaScript**
- Haga clic en `Choose Files`
- Seleccione el archivo `build.js`
- Haga clic en el botón *Comprimir*
- Descargue el archivo como `build.js` en el mismo directorio que build.min.html

importando build.min.html

Elimine todas las importaciones anteriores de los archivos HTmL desde los cuales copió y pegó las importaciones. Reemplazar las importaciones con

```
<link rel="import" href="PATH/TO/IMPORTFILE/build.min.html">
```

Lea SUPER-Optimización para la producción. en línea:
https://riptutorial.com/es/polymer/topic/9336/super-optimizacion-para-la-produccion-

Capítulo 12: tabla de datos de hierro

Examples

Hola Mundo

Punto de partida inicial para `iron-data-table` de `iron-data-table` .

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
  </head>
  <body>
    <dom-module id="x-foo">
      <template>
        <style>
        </style>
        <paper-button on-tap="msg">Click Me</paper-button>

        <iron-ajax auto
          url="https://saulis.github.io/iron-data-table/demo/users.json"
          last-response="{{users}}"
          >
        </iron-ajax>
        <iron-data-table selection-enabled items="[[users.results]]">
          <data-table-column name="Picture" width="50px" flex="0">
            <template>
              <img src="[[item.user.picture.thumbnail]]">
            </template>
          </data-table-column>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
          <data-table-column name="Email">
            <template>[[item.user.email]]</template>
          </data-table-column>
        </iron-data-table>

      </template>
      <script>
        (function(){
```

```
          'use strict';
          Polymer({
            is: 'x-foo',
            msg: function() {
              console.log('This proves Polymer is working!');
            },
          });
        })();
      </script>
    </dom-module>
    <x-foo></x-foo>
  </body>
</html>
```

Importación CSS

Importar hoja de estilo externa.

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
    <link rel="import" href="iron-data-table/default-styles.html">
  </head>
  <body>
    <dom-module id="x-foo">
      <template>
        <style>
        </style>
        <paper-button on-tap="msg">Click Me</paper-button>

        <iron-ajax auto
          url="https://saulis.github.io/iron-data-table/demo/users.json"
          last-response="{{users}}"
          >
        </iron-ajax>
        <iron-data-table selection-enabled items="[[users.results]]">
          <data-table-column name="Picture" width="50px" flex="0">
            <template>
              <img src="[[item.user.picture.thumbnail]]">
            </template>
          </data-table-column>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
```

```
          <data-table-column name="Email">
            <template>[[item.user.email]]</template>
          </data-table-column>
        </iron-data-table>

      </template>
      <script>
        (function(){
          'use strict';
          Polymer({
            is: 'x-foo',
            msg: function() {
              console.log('This proves Polymer is working!');
            },
          });
        })();
      </script>
    </dom-module>
    <x-foo></x-foo>
  </body>
</html>
```

Detalles de la fila

Expanda los detalles de la fila para mostrar datos adicionales.

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
    <link rel="import" href="iron-data-table/default-styles.html">
  </head>
  <body>
    <dom-module id="x-foo">
      <template>
        <style>
          #grid1 data-table-row-detail {
            height: 100px;
          }
          #grid1 .detail {
            width: 100%;
            display: flex;
            justify-content: space-around;
            align-items: center;
            border: 2px solid #aaa;
          }
        </style>
        <paper-button on-tap="msg">Click Me</paper-button>
```

```
        <iron-ajax auto
          url="https://saulis.github.io/iron-data-table/demo/users.json"
          last-response="{{users}}"
          >
        </iron-ajax>
        <iron-data-table id="grid1" details-enabled items="[[users.results]]">
          <template is="row-detail">
            <div class="detail">
              <img src="[[item.user.picture.medium]]">
              <p>[[item.user.username]]</p>
              <p>[[item.user.email]]</p>
            </div>
          </template>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
        </iron-data-table>
      </template>
      <script>
        (function(){
          'use strict';
          Polymer({
            is: 'x-foo',
            msg: function() {
              console.log('This proves Polymer is working!');
            },
          });
        })();
      </script>
    </dom-module>
    <x-foo></x-foo>
  </body>
</html>
```

Editar detalles de la fila

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">
    <link rel="import" href="paper-input/paper-input.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
    <link rel="import" href="iron-data-table/default-styles.html">
  </head>
  <body>
```

```
    <dom-module id="x-foo">
      <template>
        <style>
          #grid1 data-table-row-detail {
            height: 100px;
          }
          #grid1 .detail {
            width: 100%;
            display: flex;
            justify-content: space-around;
            align-items: center;
            border: 2px solid #aaa;
          }
        </style>
        <paper-button on-tap="msg">Click Me</paper-button>

        <iron-ajax auto
          url="https://saulis.github.io/iron-data-table/demo/users.json"
          last-response="{{users}}"
          >
        </iron-ajax>
        <iron-data-table id="grid1" details-enabled items="[[users.results]]">
          <template is="row-detail">
            <div class="detail">
              <img src="[[item.user.picture.medium]]">
              <p>[[item.user.username]]</p>
              <p>[[item.user.email]]</p>
              <paper-input></paper-input>
            </div>
          </template>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
        </iron-data-table>
      </template>
      <script>
        (function(){
          'use strict';
          Polymer({
            is: 'x-foo',
            msg: function() {
              console.log('This proves Polymer is working!');
            },
          });
        })();
      </script>
    </dom-module>
    <x-foo></x-foo>
  </body>
</html>
```

Editar detalles de la fila usando sub-elemento

Este ejemplo utiliza un elemento separado para editar los datos vinculados a la plantilla de `row-detail` .

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">
    <link rel="import" href="paper-input/paper-input.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
    <link rel="import" href="iron-data-table/default-styles.html">
  </head>
  <body>
    <dom-module id="row-detail">
      <template>
        <img src="[[item.user.picture.medium]]">
        <span>[[item.user.username]]</span>
        <span>[[item.user.email]]</span>
        <paper-input></paper-input>
      </template>
      <script>
        (function() {
          'use strict';
          Polymer({
            is: 'row-detail',
            properties: {
              item: Object,
            },
          });
        })();
      </script>
    </dom-module>
    <dom-module id="x-foo">
      <template>
        <style>
          #grid1 data-table-row-detail {
            height: 150px;
          }
          #grid1 .detail {
            width: 100%;
            display: flex;
            justify-content: space-around;
            align-items: center;
            border: 2px solid #aaa;
          }
        </style>
        <paper-button on-tap="msg">Click Me</paper-button>

        <iron-ajax auto
          url="https://saulis.github.io/iron-data-table/demo/users.json"
          last-response="{{users}}"
          >
        </iron-ajax>
        <iron-data-table id="grid1" details-enabled items="[[users.results]]">
          <template is="row-detail">
```

```
            <div class="detail">
              <row-detail item="{{item}}"></row-detail>
            </div>
          </template>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
        </iron-data-table>
      </template>
      <script>
        (function(){
          'use strict';
          Polymer({
            is: 'x-foo',
            msg: function() {
              console.log('This proves Polymer is working!');
            },
          });
        })();
      </script>
    </dom-module>
    <x-foo></x-foo>
  </body>
</html>
```

Nota

Actualmente hay un problema descrito aquí que hace que la sección de detalles de la fila se contraiga si contiene un subelemento. El parche debe incluir `tabindex="0"` siguiente manera. (Ver esta respuesta de desbordamiento de pila).

x-foo.html

```
<template is="row-detail">
  <div class="detail">
    <row-detail item="{{item}}" tabindex="0"></row-detail>
  </div>
</template>
```

Seleccionar fila, evitar la deselección

El comportamiento predeterminado es anular la selección de la fila cuando se hace doble clic. En algunos casos de uso, es posible que desee desactivar este comportamiento de deselección.

Nota

`table.deselectItem(item)` anula la selección imperativa de un item. Esto funciona tanto con el `item` como con el `index` (cuando se usa la matriz de elementos) como argumento.

Trabajando jsbin

```
<!DOCTYPE html>
<html>
  <head>
    <base href="https://polygit.org/polymer+:master/iron-data-
table+Saulis+:master/components/">
    <link rel="import" href="polymer/polymer.html">

    <script src="webcomponentsjs/webcomponents-lite.min.js"></script>

    <link rel="import" href="iron-ajax/iron-ajax.html">
    <link rel="import" href="paper-button/paper-button.html">

    <link rel="import" href="iron-data-table/iron-data-table.html">
    <link rel="import" href="iron-data-table/default-styles.html">
  </head>
  <body>
    <x-foo></x-foo>
    <dom-module id="x-foo">
      <template>
        <style>
        </style>
        [[_computeSelectedStr(selectedItem)]]

        <iron-ajax
            auto
            url="https://saulis.github.io/iron-data-table/demo/users.json"
            last-response="{{users}}"
            >
        </iron-ajax>
        <iron-data-table id="grid"
                         selection-enabled
                         on-deselecting-item="_deselecting"
                         items="[[users.results]]"
                         selected-item="{{selectedItem}}"
                         >
          <data-table-column name="Picture" width="50px" flex="0">
            <template>
              <img src="[[item.user.picture.thumbnail]]">
            </template>
          </data-table-column>
          <data-table-column name="First Name">
            <template>[[item.user.name.first]]</template>
          </data-table-column>
          <data-table-column name="Last Name">
            <template>[[item.user.name.last]]</template>
          </data-table-column>
          <data-table-column name="Email">
            <template>[[item.user.email]]</template>
          </data-table-column>
        </iron-data-table>

      </template>
      <script>
        (function(){
              'use strict';
          Polymer({
            is: 'x-foo',
            observers: [
              '_selectedItemChanged(selectedItem)' ,
            ],
            _selectedItemChanged: function(ob) {
```

```
            console.log('selectedItem', ob);
          },
          _computeSelectedStr: function(ob) {
            return JSON.stringify(ob);
          },
          _deselecting: function(e) {
            e.preventDefault();
          }
        });
      })();
    </script>
  </dom-module>
 </body>
</html>
```

Lea tabla de datos de hierro en línea: https://riptutorial.com/es/polymer/topic/6447/tabla-de-datos-de-hierro

Capítulo 13: Usando bibliotecas externas de Javascript con Polymer

Introducción

Dado que todos los componentes web deben ser autónomos, incluidas todas sus dependencias, la importación de dependencias duplicadas se convertirá rápidamente en un problema con los scripts incluidos. Por lo tanto, los componentes web (y, por extensión, Polymer) utilizan las importaciones de HTML del W3C para administrar las dependencias de los componentes. El navegador puede almacenar estos archivos HTML importados y solo se cargarán una vez.

La mayoría de las bibliotecas externas aún no están preparadas para las importaciones de HTML. Afortunadamente, crear el contenedor HTML necesario es rápido y directo.

Parámetros

Parámetro	Descripción
this.resolveUrl ('../raries / turf.js')	Resuelve la ubicación de tu biblioteca.
function () {this.doSomething (argumento, anotherArgument);}. bind (this));	Llamar de vuelta. Este ejemplo utiliza un cierre para repetir esto.

Observaciones

En este ejemplo, la biblioteca utilizada en el componente se instala con el gestor de paquetes bower. Esto permite una fácil distribución de un componente dependiente de la biblioteca. Si la biblioteca que desea usar no se distribuye a través de un administrador de paquetes, aún puede cargarse de la misma manera pero su componente requerirá más esfuerzo para ser utilizado por otros.

El ejemplo de carga perezosa utiliza una cadena simple para la ruta de la biblioteca. Si se desea evitar las constantes de cadena mágica, las rutas podrían cargarse usando iron-ajax desde un archivo JSON a un objeto y pasar entre componentes si es necesario.

Examples

Importar un archivo HTML estático

1. Cree un archivo HTML (en este ejemplo, `libraries/turf.html`) con la biblioteca que desea cargar:

```
<script src="../../bower_components/turf/turf.min.js"></script>
```

2. Importe el archivo HTML (`libraries/turf.html`) en su componente con el resto de sus importaciones:

```
<link rel="import" href="../../bower_components/polymer/polymer.html">
<link rel="import" href="../libraries/turf.html">
```

3. Invoca tu biblioteca (ejemplo de uso):

```
    var point = turf.point([42.123123, -23.83839]);
```

Carga lenta

1. Cree un archivo HTML (en este ejemplo, `libraries/turf.html`) con la biblioteca que desea cargar:

```
<script src="../../bower_components/turf/turf.min.js"></script>
```

2. Importe y use su biblioteca cuando sea necesario:

```
doSomething: function(argument, anotherArgument): {

    // If library has not been loaded, load it
    if(typeof turf == 'undefined') {
        this.importHref(this.resolveUrl('../libraries/turf.js'), function() {
            // Once the library is loaded, recursively call the function
            this.doSomething(argument, anotherArgument);
        }.bind(this));

        return;
    }

    // Example usage of a library method
    var point = turf.point([42.123123, -23.83839]);
}
```

Lea Usando bibliotecas externas de Javascript con Polymer en línea: https://riptutorial.com/es/polymer/topic/6034/usando-bibliotecas-externas-de-javascript-con-polymer

Creditos

S. No	Capítulos	Contributors
1	Empezando con el polímero	Community, Fuzzical Logic, fybw id, Keith, Marc, Randy Askin, tony19, Ümit
2	Bucle, la plantilla dom-repetir.	Jp_
3	Creando la aplicación usando el kit de inicio de polímero	fybw id, Puru Vijay
4	Depuración	willsquire
5	Ejemplo de Polymer toggleAttribute	a1626
6	Examen de la unidad	kashesandr, Marc
7	Hoja de trucos de polímero	Josef Ježek
8	Manejo de eventos	a1626, Ümit
9	Marca de mapa de Google con construido en caché	Schrodinger's cat
10	Modales reutilizables con polímero	Schrodinger's cat
11	SUPER-Optimización para la producción.	Puru Vijay
12	tabla de datos de hierro	Mowzer
13	Usando bibliotecas externas de Javascript con Polymer	antibrian, craPkit