EBook Gratuito

APPRENDIMENTO

polymer

Free unaffiliated eBook created from
Stack Overflow contributors.

#polymer

Sommario

Di

Capitolo 1: Iniziare con il polimero

Osservazioni

Il progetto Polymer è composto da:

- Libreria Polymer : Polymer è una libreria leggera che consente di sfruttare appieno i componenti Web. Con Web Components, puoi creare elementi personalizzati riutilizzabili che interagiscono perfettamente con gli elementi incorporati del browser o suddividere l'app in componenti di dimensioni corrette, rendendo il tuo codice più pulito e meno costoso da mantenere.
- WebComponents Polyfill : WebComponents Polyfill è una libreria futura mirata a soddisfare le specifiche dei componenti Web del W3C. I browser che implementano completamente le specifiche non hanno bisogno di webcomponents.js. Tuttavia, alla maggior parte dei browser manca ancora una parte delle specifiche, quindi questa sarà una dipendenza per un po 'di tempo.
- Polymer App Toolbox : Polymer App Toolbox ti aiuta a creare e distribuire applicazioni Web progressive all'avanguardia con un sovraccarico e un carico utile minimi, sfruttando potenti funzionalità della piattaforma Web come componenti Web, Service Worker e HTTP / 2. Toolbox fornisce un'architettura basata su componenti, layout reattivi, un router modulare, supporto per la localizzazione, supporto chiavi in mano per l'archiviazione locale e la memorizzazione nella cache offline e consegna efficiente delle risorse di app non collegate. Adottare queste funzionalità singolarmente o usarle insieme per creare un'applicazione Web progressiva completa. Polymer spruzza un po 'di zucchero sulle API standard di Web Components, rendendo più facile ottenere ottimi risultati. La libreria Polymer offre un set di funzionalità per la creazione di elementi personalizzati. Queste funzionalità sono progettate per rendere più semplice e veloce la creazione di elementi personalizzati che funzionano come elementi DOM standard. Simile agli elementi DOM standard, gli elementi Polymer possono essere:

Versioni

Versione	Data di rilascio
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

Versione	Data di rilascio
v1.0.0	2015/05/27

Examples

Ciao mondo

Questo esempio crea un elemento Polymer chiamato `x-foo` , il cui modello si lega a una proprietà stringa, denominata "message". L'HTML dell'elemento viene importato nel documento principale, che consente l'uso dei `<x-foo>` in `<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>
```

Vedi la demo in CodePen

Utilizzo degli elementi dal catalogo dei polimeri

Polymer produce molti elementi ben costruiti che puoi utilizzare nella tua app.

Naviga nel loro catalogo degli elementi .

Passiamo attraverso il flusso di lavoro dell'utilizzo di un elemento includendo l' `paper-input` (Documentazione)

Scarica l'elemento

Per scaricare un elemento ci sono due modi:

pergolato

Il modo conveniente è utilizzare la riga di comando usando il comando bower `install` :

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

Nota: `--save` aggiunge l'elemento come dipendenza a `bower.json` della tua app.

file zip

L'altro modo è di aggiungere l'elemento selezionato (`paper-input` in questo caso) alla tua raccolta (nel catalogo Polymer) usando "Aggiungi alla raccolta" nella navigazione e scaricare la tua raccolta usando l'icona a forma di stella nell'angolo in alto a destra.

Questo genererà un file .zip che contiene l'elemento e tutte le sue dipendenze. Puoi quindi copiare la cartella `bower_components` all'interno di .zip / components nella directory principale della tua app.

Importa l'elemento nella tua app

Per importare l'elemento che hai appena installato, importa il file `.html` corrispondente:

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

Usa l'elemento

Ora puoi utilizzare l' `paper-input` all'interno del documento che hai importato in:

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

Struttura elementale

Abbiamo ottenuto il seguente elemento di base elemento `my-element` salvato come `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>
```

- `<link>` include la libreria Polymer che utilizza un'importazione HTML.
- `<dom-module>` è il wrapper DOM locale per l'elemento (in questo caso `my-element`).
- `<template>` è la definizione DOM locale effettiva.
- Lo `<style>` all'interno del `<template>` consente di definire gli stili che hanno un ambito per questo elemento e il suo DOM locale e non influenzerà nessun'altra cosa nel documento.
- Il `<content>` conserverà tutto ciò che collochi all'interno del tuo elemento.
- La pseudo classe `:host` corrisponde all'elemento personalizzato (`my-element`).
- La chiamata `Polymer` registra l'elemento.
- Il `is` proprietà è il nome dell'elemento **(deve** corrispondere al `<dom-module>` s' `id`)

Puoi importarlo nella tua app usando:

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

E usalo come tag:

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

Impostazione della prima app polimerica da un modello

Costruiamoci per costruire la tua fantastica app Web progressiva con Polymer!

Prima di poter installare Polymer è necessario quanto segue:

- Node.js: controlla StackOverflow Installazione della documentazione di Node.js
- Bower: puoi installare Bower utilizzando Node Package Manager installato con Node.js:

```
npm install -g bower
```

Installazione dell'interfaccia della riga di comando del polimero

Polymer CLI ti fornisce tutti gli strumenti necessari per i progetti polimerici:

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

Inizializza la tua app da un modello di app

Utilizza `polymer init` per inizializzare la tua app da un modello di app .

Un modello interessante è il modello `--app-drawer-template` . Usiamo questo:

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

Servi la tua app

Non c'è bisogno di costruire la tua prima fantastica app Polymer. Basta `serve` :

```
polymer serve --open
```

Ciò aprirà l'applicazione nel browser predefinito su `http://localhost:8080` .

Leggi Iniziare con il polimero online: https://riptutorial.com/it/polymer/topic/949/iniziare-con-il-polimero

Capitolo 2: Cheat Sheet Polymer

introduzione

Questo è un cheat sheet per la libreria Polymer 2.x. Fork of post di Monica Dinculescu.

Examples

Definire un elemento

Documenti: 1.x -> 2.x guida all'upgrade , registrazione di un elemento , moduli di stile condivisi .

```
<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>
```

Per ottenere la definizione della classe per un particolare tag personalizzato, è possibile utilizzare `customElements.get('element-name')` .

Estendere un elemento

Documenti: estendere elementi , modelli ereditati .

Invece di `Polymer.Element` , un elemento personalizzato può estendere un elemento diverso:

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

Per cambiare o aggiungere al modello del genitore, sovrascrivere il getter del `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>
```

Se non conosci la classe genitore, puoi anche usare:

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

Definire un mixin

Documenti: mixin , elementi ibridi .

Definire un mixin di espressioni di classe per condividere l'implementazione tra diversi elementi:

```
<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 il mixin in una definizione di 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>
```

Utilizzo di comportamenti ibridi (definiti nella sintassi 1.x) come 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>
```

Metodi del ciclo di vita

Documenti: callback del ciclo di vita , pronti .

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

Associazione dati

Documenti: associazione dati , associazione attributi , associazione a elementi array , collegamenti calcolati .

Non dimenticare: proprietà dei cammelli polimerici, quindi se in JavaScript usi `myProperty` , in HTML `myProperty` `my-property` .

Binding **unidirezionale** : quando `myProperty` cambia, `theirProperty` viene aggiornata:

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

Binding a **due vie** : quando `myProperty` cambia, `theirProperty` viene aggiornata e viceversa:

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

Attribuzione : quando `myProperty` è `true` , l'elemento è nascosto; quando è `false` , l'elemento è visibile. La differenza tra attributo e legame di proprietà è che il legame di proprietà è equivalente a `someElement.someProp = value` , mentre il binding di attributo è equivalente a: `someElement.setAttribute(someProp, value)`

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

Associazione calcolata : il binding all'attributo `class` ricompilerà gli stili quando `myProperty` cambia:

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

Gli osservatori

Documenti: osservatori , osservatori multi-proprietà , osservando le mutazioni degli array , aggiungendo gli osservatori dinamicamente .

L'aggiunta di un `observer` nel blocco delle `properties` consente di osservare le modifiche nel valore di una proprietà:

```
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 */) { /* ... */ }
```

Nel blocco degli `observers` :

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

Aggiunta di un osservatore dinamicamente per una proprietà `otherProperty` :

```
// Define a method.
_otherPropertyChanged(value) { /* ... */ }
```

```
// Call it when `otherPropety` changes.
this._createPropertyObserver('otherProperty', '_otherPropertyChanged', true);
```

Leggi Cheat Sheet Polymer online: https://riptutorial.com/it/polymer/topic/10710/cheat-sheet-polymer

Capitolo 3: Creazione di app con Polymer Starter Kit

introduzione

Polymer Starter Kit è un'ottima soluzione per iniziare a costruire applicazioni Web progressive . Offre un design del materiale reattivo Interfaccia utente, Page Routing, servizio Offline-first.

Sintassi

- Polimero: comando per l'avvio del polimero cli
- Polymer init: presenta un elenco di modelli da scegliere per creare il tuo elemento / app
- npm: avvia l'interfaccia CLI del gestore pacchetti Node.js
- npm install: installa un pacchetto
- npm install -g: installa un pacchetto globalmente sul tuo sistema. Utile per gli strumenti CLI

Examples

Installazione del Polymer Starter Kit

1. Assicurarsi di aver installato Polymer CLI nel sistema a livello globale. In caso contrario, apri la tua interfaccia riga di comando / terminale ed esegui questo comando: `npm install -g polymer-cli`

   Nota: se sei un utente di Ubuntu, potresti dover prefisso il codice precedente con la parola chiave `sudo` .

2. Dopo l'installazione di Polymer CLI, eseguire questo comando

   cd [LA TUA DIRECTORY IN CUI VUOI INIZIARE IL TUO PROGETTO]

```
polymer init
```

3. Ti verranno date 4 opzioni. Usa i tasti freccia per andare al 4 ° esempio **kit di partenza** . Premi il tasto `Enter` .

   Tutti i file richiesti verranno scaricati nella directory selezionata.

Leggi Creazione di app con Polymer Starter Kit online: https://riptutorial.com/it/polymer/topic/9330/creazione-di-app-con-polymer-starter-kit

Capitolo 4: Debug

Examples

Disabilitazione della memorizzazione nella cache HTML

Il caching di importazione HTML a volte significa che le modifiche apportate ai file HTML importati non si riflettono sull'aggiornamento del browser. Prendi la seguente importazione come esempio:

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

Se viene effettuato un cambiamento a `my-element.html` dopo aver precedentemente caricato la pagina, il file modificato potrebbe non essere scaricato e utilizzato nel documento corrente quando viene aggiornato (come è stato precedentemente importato e memorizzato nella cache). Questo può essere ottimo per una produzione, ma potrebbe ostacolare lo sviluppo.

Per disabilitare questo in Google Chrome:

- Apri DevTools di Google Chrome
- Seleziona il menu principale > Impostazioni
- Vai alla sezione Rete
- Seleziona "Disattiva cache (mentre DevTools è aperto)"

Ciò eviterà la memorizzazione nella cache delle importazioni HTML, ma solo quando DevTools è aperto.

Leggi Debug online: https://riptutorial.com/it/polymer/topic/5864/debug

Capitolo 5: Esempio di Polymer toggleAttribute

Sintassi

1. toggleAttribute (nome, bool, nodo)

Parametri

Nome	Dettagli
nome	Stringa: nome dell'attributo HTML che deve essere attivato
bool	booleano: booleano per forzare l'attributo on o off. Se non specificato, lo stato dell'attributo verrà invertito.
nodo	HTMLElement: nome del nodo che contiene l'attributo HTML. Predefinito a questo

Osservazioni

Un buon esempio di questo sarà la forma, in cui il pulsante di invio dovrebbe essere attivo solo se tutti i campi obbligatori sono stati inseriti.

Examples

Esempio di base

```
<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>
```

Ecco un plunker in esecuzione

Leggi Esempio di Polymer toggleAttribute online:
https://riptutorial.com/it/polymer/topic/5978/esempio-di-polymer-toggleattribute

Capitolo 6: ferro-dati-tavolo

Examples

Ciao mondo

Punto di partenza iniziale per `iron-data-table` .

JsBin funzionante

```
<!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>
```

Importazione CSS

Importa foglio di stile esterno.

JsBin funzionante

```
<!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>
```

Dettagli di riga

Espandi i dettagli delle righe per visualizzare dati aggiuntivi.

JsBin funzionante

```
<!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>
```

Modifica i dettagli della riga

JsBin funzionante

```
<!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>
```

Modifica i dettagli delle righe usando l'elemento secondario

Questo esempio utilizza un elemento separato per modificare i dati associati al modello `row-detail`
.

JsBin funzionante

```
<!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

Al momento è stato descritto un problema che causa la compressione della sezione dei dettagli della riga se contiene un elemento secondario. La patch deve includere `tabindex="0"` come segue. (Vedi la risposta Stack Overflow .)

x-foo.html

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

Seleziona una riga, evita la deselezione

Il comportamento predefinito è deselezionare la riga quando si fa clic due volte. In alcuni casi d'uso, potresti voler disabilitare questo comportamento di deselezione.

Nota

`table.deselectItem(item)` metodo `table.deselectItem(item)` imperativamente un oggetto. Funziona sia con l' `item` che con l' `index` (quando si utilizza l'array di elementi) come argomento.

JsBin funzionante

```
<!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>
```

Leggi ferro-dati-tavolo online: https://riptutorial.com/it/polymer/topic/6447/ferro-dati-tavolo

Capitolo 7: Gestione degli eventi

Osservazioni

- Ecco un plunker per tutti gli esempi
- Il nome `mylistener` maiuscole e minuscole e sarà sempre convertito in `mylistener` minuscole, ad esempio se si dispone di un attributo `on-myListener` listener `mylistener` `on-myListener` impostato sull'evento `mylistener` .
- Simile ad `listen` puoi anche usare un metodo non `unlisten` per rimuovere qualsiasi listener.
- La modifica di Poperty genera un evento con il nome di `property-changed` ad esempio se la proprietà è `myProperty` event sarà `my-propert-changed` (camel casing su '-'). Puoi ascoltarli usando `on-event` attributo `on-event` dell'oggetto `listener` .
- Il nuovo valore è sempre memorizzato in `e.detail.value` per gli eventi di modifica delle proprietà.

Examples

Ascolto di eventi usando l'oggetto listener

```
<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>
```

Ascoltatore annotato

Un altro modo per aggiungere un listener di eventi consiste nell'utilizzare annotazioni `on-event` in DOM. Di seguito è riportato un esempio che utilizza `up` evento `up` , che viene attivato quando il dito / il mouse sale

```
<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>
```

Ascoltatore imperativo

Puoi anche aggiungere / rimuovere listener imperativamente usando il metodo listen e unlisten di 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>
```

Eventi personalizzati

Puoi anche licenziare i tuoi eventi e poi ascoltarli dall'elemento Polymer della pagina HTML

Questo elemento attiva l'evento personalizzato

```
    <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>
```

Ecco un elemento che sta ascoltando quell'evento personalizzato

```
  <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>
```

Ascolto da 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 di modifica della proprietà

Proprietà con `notify:true` attiva anche 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 di eventi

È possibile retargetare un evento in Polymer, ovvero è possibile modificare i dettagli dell'evento come il `path` nascondendo così i dettagli reali di un evento / elemento dall'utente.

Per esempio, se un `div` all'interno `event-retargeting` elemento è sparare l'evento, ma lo sviluppatore non vuole l'utente a sapere che egli può retarget l'evento per `event-retargeting` elemento utilizzando il seguente codice.

```
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'));
```

Per vedere l'esempio di lavoro si prega di fare riferimento alla sezione `plunker` nella sezione `remarks` .

Leggi Gestione degli eventi online: https://riptutorial.com/it/polymer/topic/4778/gestione-degli-eventi

Capitolo 8: Google Map Mark With Built in Cache

Sintassi

- <G-map

```
api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

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

Parametri

API-chiave	tasto javascript API di google
posizioni	Elenco delle località che desideri contrassegnare sulla mappa di google
----------	----------

Osservazioni

Per l'intero codice, vai al repository

Per vedere la mappa di google in azione, guarda qui

Examples

A - Iniziare

In poche parole, il nostro elemento personalizzato dovrebbe

- Avere funzionalità di ricerca incorporata, che ricerca e contrassegna i luoghi sulla mappa.
- Accetta Un attributo chiamato location-array, che è un elenco di luoghi
- Avere un ascoltatore, per ascoltare un evento chiamato "google-map-ready" .Questo evento è attivato quando l'elemento è caricato.
    - Questo listener dovrebbe eseguire il looping della matrice di posizione e assegnare l'elemento successivo nel proprio array di posizione come "query di ricerca" corrente
- Hanno un metodo chiamato cache
    - Questo metodo memorizza nella cache la latitudine e la longitudine di ogni posizione nell'array di posizione, non appena la ricerca restituisce un risultato
- Avere un modello di ripetizione del Dom, che scorre attraverso i risultati memorizzati nella cache e rilascia un marcatore in ogni posizione

Quindi, lo scopo del nostro elemento personalizzato in sostanza significa, se passi una serie di posizioni in questo modo:

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

L'elemento personalizzato

- Non dovrebbe solo segnare Norvegia e Svezia sulla mappa, ma,
- Dovrebbe anche segnare eventuali ricerche successive sulla mappa - cioè, se cerchi Boston, dopo che la mappa ha contrassegnato Norvegia e Svezia, Boston dovrebbe anche avere un segnaposto sulla mappa

Importazione delle dipendenze

Innanzitutto, importiamo tutto ciò di cui abbiamo bisogno nel nostro elemento. Questo elemento è sotto

Elementi / 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

Alcune delle importazioni di cui sopra, non sono necessarie, ma buone da avere, nel caso in cui si desideri aggiungere funzionalità ai propri marcatori

Abbiamo necessariamente bisogno,

- Google Map
- google-map-marcatore
- google-map-search

Inoltre, si presume, sai come installare i singoli Polymer Elements tramite Bower

Registrazione del nostro elemento con polimero

Successivamente, registriamo il nostro elemento personalizzato come "g-map"

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

Così! il nostro elemento personalizzato è registrato con Polymer.

Aggiungeremo in seguito proprietà specifiche per la ricerca su google map e qualsiasi altra

proprietà di cui abbiamo bisogno.

Aggiunta di un modello al nostro elemento personalizzato - tramite modelli

Successivamente, aggiungiamo il modello per il nostro elemento personalizzato

Apri elementi / g-map.html e aggiungi un modello che associa gli elementi personalizzati DOM

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

Ora, avvolgi l'intero html e javascript che hai scritto per il nostro elemento personalizzato, all'interno di un modulo dom.

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

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

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

Molto bella! Abbiamo un progetto base

B - Rendering della mappa google di base

Prendi pronto per il browser - Polyfill

Crea una nuova pagina di destinazione per il nostro elemento, all'indirizzo index.html.

Includi polyfill per far funzionare un elemento personalizzato. Importa anche l'elemento personalizzato g-map, che abbiamo registrato con Polymer.

Nota:

- I webcomponents sono il polyfill necessario per il supporto del browser.
- Polymer, è incluso, in modo che possiamo usare la libreria per creare il nostro elemento personalizzato

Quindi Apri index.html e includi Polifere e Polimeri necessari. Inoltre, **importa** l'elemento personalizzato che abbiamo registrato

```
<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>
```

Quindi con il browser Polyfilled,

Invochiamo l'elemento polimero di google-map all'interno del nostro elemento personalizzato.

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

    <google-map></google-map>

</template>
```

e quindi, richiama l'elemento personalizzato all'interno di index.html, la nostra pagina di destinazione.

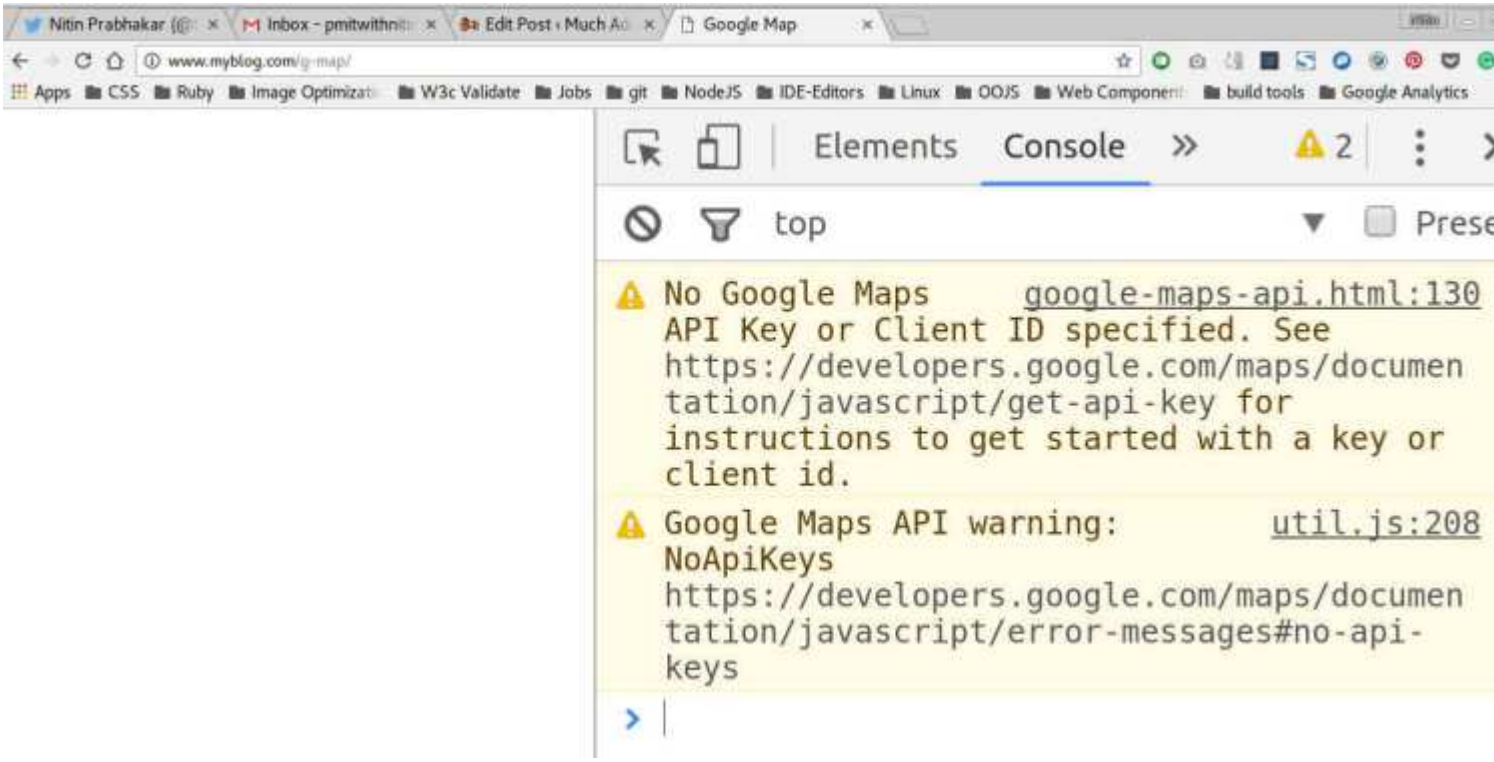
Quindi questo va dentro index.html

```
<body>

    <div class="container">

        <div class="row">

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

        </div>

    </div>

</body>
```

Ho provato a rendere la mia pagina con l'elemento personalizzato g-map a questo punto, e ho trovato una pagina EMPTY!

PERCHÉ??

Quando vedo il log della console, vedo questo



Ah!

Quindi, abbiamo bisogno di una chiave API per rendere una mappa da google.

Nota:

Ottenere una chiave API dall'API javascript di google è piuttosto semplice.

Basta seguire il link sottostante per generare la tua chiave personale per progetto.

Chiave API di Google

Ora, dal momento che la chiave API deve essere utilizzata, abbiamo due scelte.

- Consenti all'utente di passarlo come un attributo
- Avere una chiave API predefinita configurata

Ora, vorrei andare per il primo, semplicemente perché ogni chiave API ha i suoi limiti imposti. cioè può essere usato solo tante volte. Quindi, una chiave configurata e spedita con l'elemento che sviluppiamo, potrebbe presto esaurirsi.

Chiunque voglia utilizzare l'elemento personalizzato, può generare una nuova chiave e passarla come attributo.

così:

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

Quindi stiamo passando la chiave api come un attributo e all'interno del nostro elemento

personalizzato, lo associamo all'attributo chiave-chiave dell'elemento Polymer.

così:

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

Quindi, una volta passata la chiave API e aggiornata, non vediamo ancora la mappa!

Nota:

Le mappe di Google, hanno bisogno di qualche css per renderizzare. Abbiamo bisogno dell'altezza della mappa impostata in modo esplicito.

Quindi, crea un file css all'interno della directory css, ad esempio app.css e aggiungi queste righe

```
google-map{

    min-height:30vmax;
}
```

E ora, una volta aggiornato, lo vediamo

Sìì! Abbiamo appena reso una mappa con un markup minimo! Abbiamo appena scritto questo!

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

C - Aggiunta della capacità di ricerca al nostro elemento personalizzato

Per cercare un luogo, utilizziamo il potente elemento che Polymer spedisce, chiamato google-map-search.

Tutto quello che devi fare è passare

- un oggetto mappa e
- una stringa di query per l'elemento in questo modo:

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

Come passiamo l'oggetto della mappa? Da dove lo prendiamo?

Semplice!

Quando invochi l'elemento google-map, associa la mappa delle proprietà dell'elemento personalizzato alla mappa delle proprietà dell'elemento.

Quindi, invochiamo l'elemento google-map in questo modo:

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

Nota:

Facciamo un bind di dati bidirezionale per la mappa sopra, quando invochiamo il polimero di google map.

È rappresentato dalle parentesi graffe {{}} al posto di una rilegatura unidirezionale indicata da parentesi quadre [[]].

Quando facciamo un legame a due vie,

- Ogni volta che la proprietà della mappa di g-map dell'elemento personalizzato cambia, viene segnalato l'elemento google map

e,

- Ogni volta che cambiano le proprietà della mappa di google map, riceviamo una notifica nel nostro elemento personalizzato g-map.

Quindi, con l'associazione bidirezionale dei dati, ogni volta che l'oggetto della mappa cambia in google-map, il nostro elemento personalizzato viene aggiornato con le modifiche.

e passiamo l'oggetto della mappa come attributo all'elemento google-map-search, in modo che, qualsiasi risultato di ricerca sia segnato nella mappa attuale di rendering.

Come appare il DOM del nostro elemento personalizzato in questo 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>
```

Non abbiamo mai aggiunto la mappa e le proprietà di query per il nostro elemento personalizzato!

Aggiungi le proprietà nella chiamata di registrazione a Polymer

```
Polymer({

    is:"g-map",

    properties:{

        map:{

            type: Object

        },

        query:{

            type:String,

            value:""

        }
    }

});
```

Ora che abbiamo registrato le proprietà, notiamo che noi

- Stanno ottenendo la mappa attualmente renderizzata come un oggetto - tramite associazione dati e memorizzandola come una proprietà locale chiamata anche mappa,
- Avere una query di proprietà, passata a google-map-search

Accettazione dell'input dell'utente come query di ricerca

Ma chi ci fornisce la query da cercare? Err! proprio adesso? NESSUNO.

Questo è esattamente quello che facciamo dopo.

Aggiungiamo un modulo di ricerca, che l'utente che esegue il rendering della mappa, userebbe per inserire una query. Abbiamo bisogno di input per la casella di ricerca e usiamo l'input di ferro di Polymer.

Aggiungere un modulo di ricerca

Includere gli elementi di ferro necessari all'inizio del file 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>
```

Vogliamo il modulo di ricerca, nell'angolo in alto a sinistra, quindi aggiungiamo un input di ferro con la posizione assoluta, e alcuni css per fissarlo lì.

Quindi, avvolgi il DOM per il nostro elemento personalizzato, all'interno di un div chiamato "map-container", e aggiungi un div child chiamato "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>
```

Aggiungi alcuni css di base per appuntare il modulo di ricerca che abbiamo scritto in alto a destra della mappa

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

E poi, quando ci aggiorniamo, abbiamo questo. Il modulo di ricerca in alto a destra

Tutto ciò che resta da fare è, per noi, fornire un input per l'elemento google-map-search, tramite la proprietà query del nostro elemento personalizzato.

Quindi leghiamo la nostra "query" di proprietà all'input del ferro nel modulo di ricerca. così:

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

Nota:

Leghiamo la query di proprietà del nostro elemento personalizzato, all'input di ricerca utilizzando,

Attributo "bind-value" di un input di ferro.

Quindi, bind-value = {{query}} nell'input ferro precedente, assicura che qualsiasi modifica al testo all'interno dell'ingresso ferro venga notificata alla proprietà della query.

Con la nostra configurazione attuale,

Per ogni alfabeto che un utente digita nella casella di ricerca, si verifica una ricerca, che rallenta la mappa e consuma il limite sulla nostra chiave API.

Perché?

Poiché la query di proprietà del nostro elemento personalizzato è associata all'input di ferro e ogni lettera digitata nell'input di ferro, modifica il valore di "query", che quindi influenza l'html in

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

causando una ricerca per accadere

La nostra soluzione è semplice! È sufficiente associare l'attributo "query" dell'elemento di ricerca di google-map a una proprietà separata.

Diciamo, creiamo una proprietà "searchQuery" per il nostro elemento personalizzato.

Aggiungi questo, sotto la proprietà "query" nella chiamata Polymer

```
searchQuery :{

    type:String

}
```

Successivamente, modifica il binding nella chiamata dell'elemento di ricerca google-map nel DOM dell'elemento personalizzato

così:

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

Infine, tieni presente che abbiamo aggiunto un'icona di ferro (icona di ricerca) al nostro modulo di ricerca?

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

Aggiungiamo la funzione di ricerca al nostro elemento personalizzato. Si chiama toccando l'icona di ricerca.

Nel nostro metodo di ricerca, assegniamo qualsiasi cosa ci sia nella casella di ricerca, all'attributo "query" di google-map-search!

Aggiungi questa funzione, dopo l'oggetto "Proprietà" nella chiamata del polimero per la registrazione.

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

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

Molto bella! Quindi, ora, una ricerca avviene solo alla pressione dell'icona di ricerca.

Mostra i risultati della ricerca

abbiamo

- Ha reso una mappa
- Appuntato un modulo di ricerca sulla mappa
- Ha scritto la funzionalità per cercare su tocco dell'icona di ricerca

Ora dobbiamo mostrare i risultati.

Per questo, abbiamo bisogno di legare i risultati dell'elemento di ricerca di google-map, su una proprietà locale.

Quindi, cerchiamo di creare una proprietà per il nostro elemento personalizzato g-map, e denominarlo risultati duh!

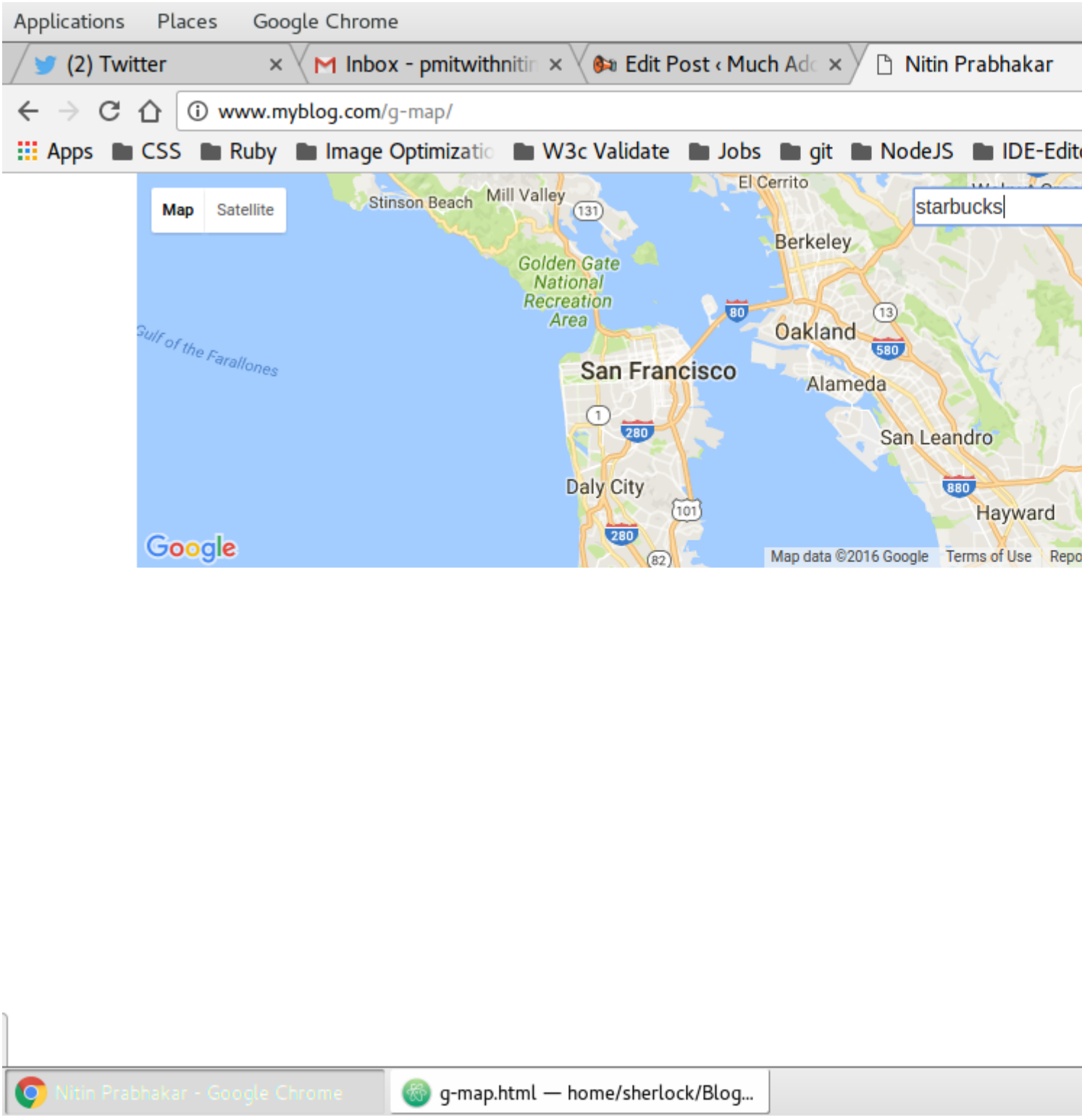
Nella chiamata di registrazione del polimero, aggiungi i risultati della proprietà di tipo oggetto nella dichiarazione delle proprietà

```
results:{

type:Object

}
```

E lega la proprietà dei risultati di google-map-search alla proprietà dei risultati locale che hai definito sopra.

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

Con quello sul posto, cerchiamo di aggiornare la pagina e cercare gli **starbucks** sulla mappa.



Non è successo niente!

Bene Duh! abbiamo solo associato i risultati di google-map-search alla proprietà results. Abbiamo scritto qualcosa per visualizzarli sulla mappa? NO!

Abbiamo bisogno di.

Quindi, sappiamo, che l'elemento Polymer google-map-search restituisce una serie di marker come risultati.

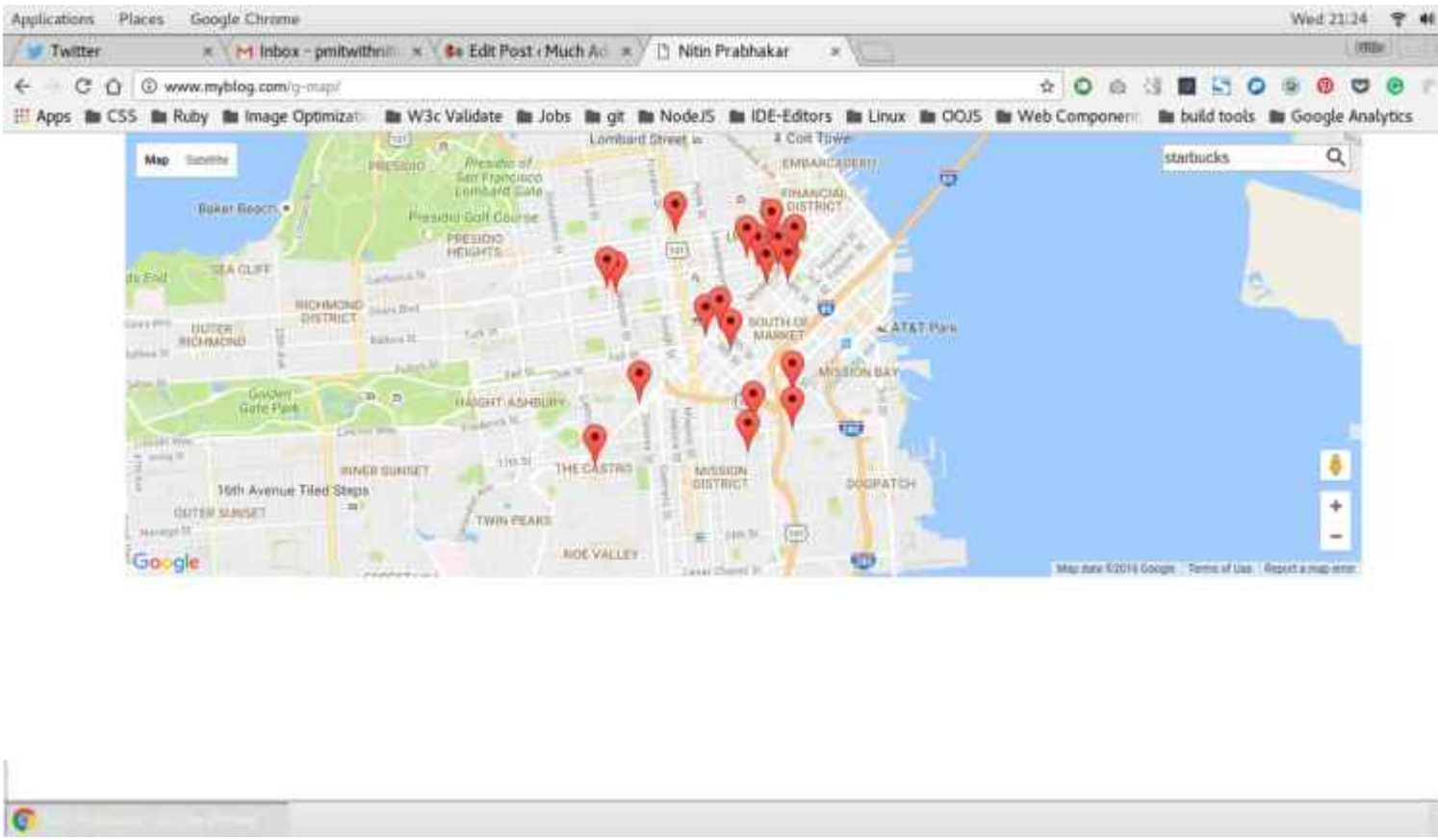
Dobbiamo scorrere l'array e posizionare i marker sulla mappa.

Quindi, scriviamo un modello dom-repeat, all'interno del nostro elemento, per visualizzare i marcatori per ogni risultato

così:

```
<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>
```

E ora su aggiornamento, vediamo questo



Molto bella!

Abbiamo aggiunto con successo la funzionalità di ricerca al nostro elemento personalizzato.

Tutto quello che dobbiamo fare è,

scrivi questo nel nostro index.html

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

E ottieni una mappa su cui puoi cercare!

D - Memorizzazione nella cache dei risultati di ricerca

Il nostro obiettivo, tuttavia, era quello di

- fai una lista di luoghi
- segnare tutti sulla mappa e,
- contrassegnare qualsiasi altro luogo di nostra scelta, utilizzando la casella di ricerca

Abbiamo una casella di ricerca, in cui l'utente può cercare una query alla volta. Per soddisfare più query, è necessario memorizzare nella cache i risultati per ogni query, prima di visualizzare i marcatori e contrassegnarli sulla mappa

Quindi, aggiungiamo

- Una proprietà dice luoghi che accettano una serie di posizioni
- Una cache di proprietà che memorizza i risultati di ogni query nell'array di ubicazione passata al nostro elemento personalizzato

```
cache:{

type:Object

},

locations:{

type:Object

}
```

Successivamente, scriviamo una funzione per memorizzare nella cache ogni risultato di ricerca, risultante da una ricerca per ogni posizione nell'array di posizione.

È piuttosto semplice!

Stiamo già raccogliendo i risultati in una proprietà chiamata "risultati", che quindi eseguiremo in loop per visualizzare i marcatori.

Tutto ciò che dobbiamo fare ora è spingere ogni risultato di ricerca in un array e la nostra cache è pronta!

Dobbiamo utilizzare un evento per attivare il caching. Tale evento viene chiamato "on-google-map-search-result" e viene attivato, dopo che l'elemento google-map-search termina una ricerca.

Quindi "on-google-map-search-result", chiamiamo una funzione dire "cacher", per memorizzare nella cache i risultati della ricerca corrente in una proprietà chiamata cache.

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

Nota:

Normalmente, per spingere un oggetto in un array, usiamo il metodo push vanilla

ad esempio: ho un nuovo posto da aggiungere alla mia lista di luoghi che voglio segnare, ad esempio NewYork.

Se la matrice che voglio aggiungere a si chiama posizioni, vorrei scrivere

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

Tuttavia, nel nostro caso, abbiamo bisogno di spingere qualsiasi risultato da google-map-search in un array chiamato cache. Poi abbiamo bisogno di scorrere la cache con gli ultimi risultati inclusi. Non è la cache esistente.

Ora, utilizziamo il modello dom-repeat per scorrere i risultati e posizionare i marcatori sulla mappa.

Tuttavia, se passiamo un array di località in questo modo:

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

Quindi, la proprietà dei risultati del nostro elemento personalizzato verrà sovrascritta per ogni elemento di quell'array.

Quindi, chiamiamo il metodo cacher on-google-map-search-results e spingiamo il risultato corrente nella proprietà cache.

Questa non può essere una spinta alla vaniglia come nella nota sopra.

Dobbiamo far sapere al dom-repeater che la nostra cache è stata sovrascritta con ogni nuova ricerca.

Quindi, usiamo una spinta speciale che Polymer spedisce, in questo modo:

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

Aggiungi quella funzione, dopo la funzione di ricerca che abbiamo scritto in precedenza.

La sintassi implica che la cache delle proprietà debba essere mutata, aggiungendo il valore corrente dei risultati della proprietà e qualsiasi dom-repeater che utilizza la proprietà cache per eseguire il loop, devono essere notificati della mutazione.

Infine, cambiamo il modello di ripetizione dom per scorrere la cache, anziché i risultati

Nota:

la cache è una matrice di matrici.

ogni elemento dell'array della cache è una matrice di risultati

Quindi il nostro ripetitore dom sarà così:

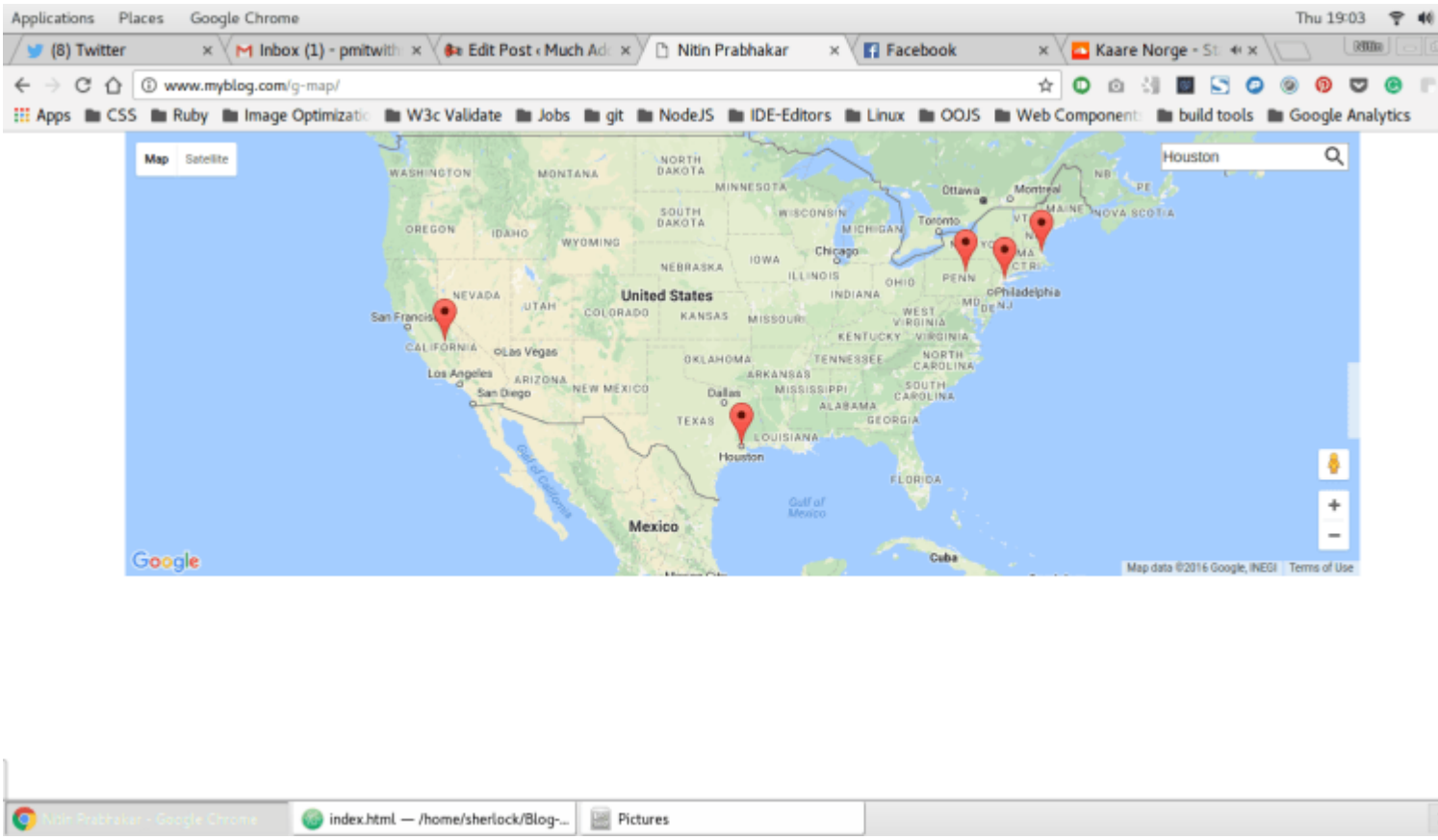
```
<template id="resultList" is="dom-repeat" items="[[cache]]">

    <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>
```

Molto bella! Abbiamo l'elemento personalizzato tutto codificato e pronto per essere utilizzato!

Cerchiamo di testarlo passando una serie di posizioni da index.html dovremmo?

```
<g-map

api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

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

e eccoci!



Il polimero completo di Google Map Polymer con built-in cache

```
<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>
```

Leggi Google Map Mark With Built in Cache online:
https://riptutorial.com/it/polymer/topic/7487/google-map-mark-with-built-in-cache

Capitolo 9: Looping, il modello dom-repeat.

Examples

Una lista di base

Questo è un elemento polimerico di base che mostra un elenco di nomi.

```
<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>
```

Leggi Looping, il modello dom-repeat. online: https://riptutorial.com/it/polymer/topic/6160/looping--il-modello-dom-repeat-

Capitolo 10: Moduli riutilizzabili con polimero

Sintassi

- Dichiarazione degli elementi: <barra degli strumenti link2-share = "" link2-fork = "" modal-id = "" title = ""> </ tool-bar>

Osservazioni

Nota:

Il codice attraverso questa scrittura, non è una copia funzionante. È necessario sostituire i riempitivi per hrefs, src e nomi di progetto.

Il codice illustra solo una prova di concetto.

Per vedere l'elemento personalizzato in azione,

andare qui

E, per esplorare l'utilizzo e la struttura dell'elemento personalizzato,

andare qui

L' ***utilizzo*** è in "index.html"

L' ***elemento*** è in "elements / tool-bar.html"

Examples

Modals

Quindi vuoi aggiungere design di materiali alla tua attività o alla tua carriera portfolio.huh? Non riesci a resistere all'utilizzo di un modale? che si apre con un clic su quelle carte nitide che intendi progettare, per ciascuno dei tuoi progetti / prodotti!

Diciamo, per ogni progetto, che hai fatto, o per ogni prodotto che hai progettato, hai bisogno di una carta materiale. Ogni carta deve avere

- Un'icona, che fa apparire una casella informativa, che elenca tutte le funzionalità di
- il tuo prodotto o progetto, in dettaglio. Un'icona "condividi questa", che condivide
- il tuo prodotto o progetto sui social media Un'icona "fork this", che apre la pagina github per il tuo prodotto / progetto

Con l'html nativo e assumendo che si usi il bootstrap o qualsiasi altro layout di flex box, è necessario

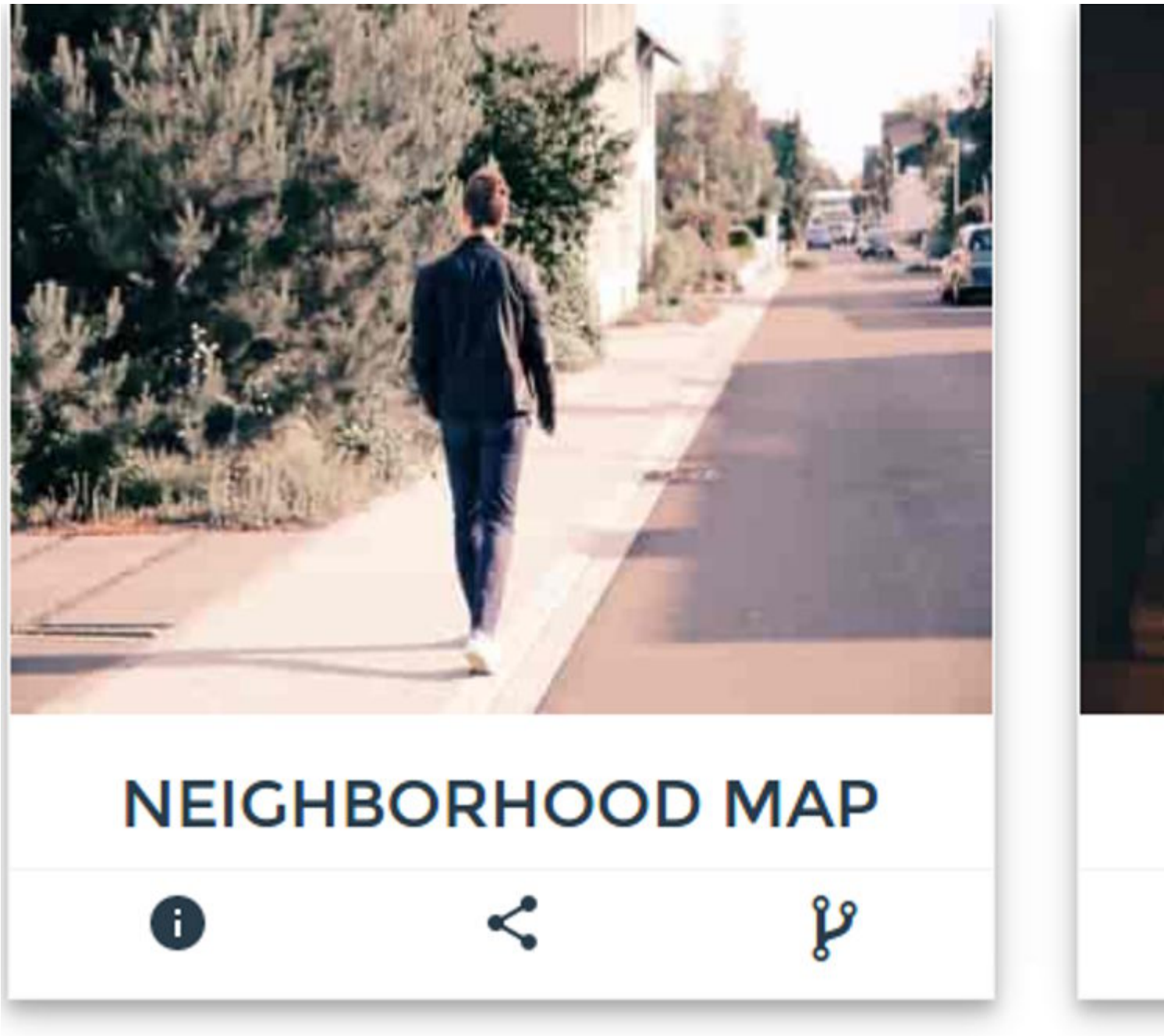
- Scrivi un div wrapper per ogni nuovo progetto con una riga di classe
- Avvolgi l'immagine dell'eroe per ogni progetto in una colonna di 12 colonne
- Scrivi un'altra riga
- Avvolgi ogni link (informazioni | condividi | forchetta) in una colonna di 4 colonne a parte
- Inietti queste due righe in un contenitore

Per i principianti. Una volta che hai fatto quanto sopra per tutti i tuoi progetti, devi lavorare sulla creazione del pop-up richiesto per ogni progetto. Quindi,

- Scarica tonnellate di javascript e css (personalizzando il bootstrap per la funzionalità modale)
- Scrivi un Modal con un ID univoco, per differenziare le informazioni di ogni progetto (modale-P1 per Progetto 1, modal-P2 per Progetto 2 e così via)
- Quindi componi l'html richiesto, per visualizzare le informazioni per ogni progetto, nel suo Modal corrispondente.

Quali sono i nostri obiettivi?

Solo per reiterare, visivamente, se hai due progetti nel tuo portfolio, l'obiettivo è quello di avere schede in questo modo:



Quindi abbiamo due preoccupazioni per OGNI progetto

- L'immagine dell'eroe
- La barra degli strumenti con informazioni (Pops up a Modal), condivisione e collegamenti a forcella

Innanzitutto, affrontiamo ciò che accade quando un utente fa clic sul collegamento di informazioni nella barra sotto ciascun progetto. Abbiamo bisogno di una modale da visualizzare, con maggiori informazioni sul progetto.

Quindi, come esattamente si configura un modal con bootstrap?

- Scrivi questo per attivarlo, per ogni progetto

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

- Quindi scrivi il corpo di Modal in questo modo:

```
<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>
```

e a seconda del contenuto e della lunghezza della descrizione del progetto, in realtà si perde un po 'di monitoraggio dell'apertura e della chiusura di una zuppa div in alto nel corpo di Modal!

Inoltre, questo markup è decisamente noioso da comporre OGNI MOMENTO hai bisogno di un modal.

Accidenti! E finì lì? No. Hai ancora bisogno di scrivere per ogni progetto,

- L'immagine dell'eroe della carta Progetto, che viene visualizzata sopra la barra degli strumenti.

Quindi, come aggiungiamo l'immagine dell'eroe per ogni progetto? ALTRO html!

L'immagine dell'eroe ha bisogno

- Una fila a sé stante
- Una colonna di 12 centimetri di larghezza propria all'interno di quella fila

In che modo l'html completo cercherà di visualizzare un progetto come una scheda mostrata sopra?

```
<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>
```

Ora dì che hai 10 progetti da mostrare! Devi riscrivere il markup incasinato 10 volte! Più 10 diversi modelli!

Non ho la pazienza di riscrivere l'html e 10 modali, e so che non lo farai!

Modali, con polimero?

Nota

> Presumo tu lo sappia
>
> Specifiche del componente Web
>
> Polimero di Google

Cosa succede se, potremmo avere solo un elemento personalizzato, ad esempio

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

e ha fatto magicamente tutto ciò che potrebbe significare modale Markup?

Allettante eh!

Come si specifica quale modale appartiene a quale progetto?

Semplice! Scrivi e basta

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

Così, per esempio, riserva il markup con un id di "Project-cats" per il progetto sui gatti.

Come scrivi cosa va nel modale? semplice! Scrivi il tuo normale markup, incapsulato, all'interno del tag personalizzato ""

Così:

```
<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>
```

Non è abbastanza semplice?

E se ti stai chiedendo come appare il markup completo? Compresa la condivisione e i collegamenti a forcella, vedi sotto.

```
<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>
```

Molto meglio di riscrivere la div div completa eh!

Ovviamente potresti accorciarlo ulteriormente e renderlo completamente componibile estraendo l'implementazione dell'immagine dell'eroe all'interno dell'elemento personalizzato, ma cerchiamo di tenerlo fuori, come una preoccupazione successiva.

Barra degli strumenti riutilizzabile con modale | Condividi | Icone della forcella - Codifica dell'elemento

Finora, abbiamo definito quanto sia facile scrivere un elemento personalizzato che nasconde il html disordinato dietro di esso e fornisce all'utente un markup facile da scrivere e breve.

È ora di programmarlo!

Il nostro elemento personalizzato, per visualizzare la barra sotto l'immagine dell'eroe dovrebbe

- Accetta un link per essere condiviso
- Accetta un collegamento al Repo per essere biforcato
- Accetta un modalId per differenziare tra altri Modals.
- Accetta un titolo per il modale

Registra l'elemento personalizzato della barra degli strumenti con Modal | Condividi | Icone della forcella

Cerchiamo di registrare il nostro elemento personalizzato, che accetta l'elenco sopra di proprietà.

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

Molto bella! Successivamente, aggiungi l'Html di cui abbiamo bisogno.

Per prima cosa, è necessario aggiungere le icone per Modal / Share / Fork. Sarà sufficiente un semplice ul. Inoltre, utilizzeremo le icone di ferro di Polymer per visualizzare le icone Modal / Share / Fork.

Nota: icone di ferro da polimero

> Installa i set di icone in questo modo, nella radice del tuo progetto.
>
> ```
> bash $ bower install --save PolymerElements/iron-icon
>
> bash $ bower install --save PolymerElements/iron-icons
> ```

Successivamente, Includi gli elementi Polymer installati nella dichiarazione dell'elemento

personalizzato, in questo modo

> Nota:
>
> Per tutti gli scopi pratici, il nostro elemento personalizzato sarà alloggiato all'interno
>
> ***Project-root / elementi***

```
<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">
```

Molto bella! Se ti stai chiedendo come usare le icone di ferro, vedi la nota qui sotto.

Nota: utilizzo dell'icona di ferro e grammatica

> Se abbiamo bisogno di usare l'icona delle informazioni per il nostro Popal modale, scriviamo il markup in questo modo:
>
> `<iron-icon icon="icons:info"></iron-icon>`
>
> Se vogliamo che l'icona di condivisione di social network, quindi l'attributo di icona sopra, diventa
>
> icon = " **social: share** "
>
> Quindi il significato della grammatica, ho bisogno del set di icone da "SOCIAL" e voglio l'icona SHARE da quel set.
>
> In alternativa puoi usare font-awesome, nel qual caso puoi usare,
>
> scegli ciò che ritieni più adatto.

Una volta che gli include sono fatti, scriviamo il modello per il nostro elemento personalizzato. Utilizzeremo i binder a una via per i dati nel markup dell'elemento personalizzato. Leghiamo tutto ciò che viene passato come attributi dall'utente, alle proprietà corrispondenti del nostro elemento, mentre ci imbarchiamo nella registrazione della nostra voce.

Quindi il progetto html per il nostro elemento personalizzato è:

```
<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>
```

Freddo! Quindi la barra degli strumenti ha il suo modello ul, che elenca i tre collegamenti

- icona informazioni - Popup modale
- condividi questa icona - condividi su google
- Forcella questa icona - Forchetta il repository

Inoltre, abbiamo associato gli attributi che l'utente scrive, alle rispettive proprietà.

Nota: rendere l'ID modale univoco

> Abbiamo affrontato l'univocità dell'icona delle informazioni, specificando l'attributo id in questo modo
>
> ```
> id="anchor-for-[[modalId]]" onclick="clickHandler(event)"></iron-icon>
> ```
>
> Quindi ogni progetto avrà un id di ancoraggio unico.
>
> vale a dire,
>
> Se l'utente passa "project-cats" come id-modale, nella chiamata al nostro elemento personalizzato, l'id di ancoraggio sarebbe "anchor-for-project-cats"
>
> Se l'utente aggiunge un altro progetto e passa "project-puppy" come attributo modal-id, l'id di ancoraggio sarebbe anchor-for-project-puppy
>
> e così via.
>
> Ma come facciamo a garantire che ogni icona delle informazioni sia mappata sul proprio modale?
>
> Alla " **icona di ferro** " per **informazioni** , dovrebbe essere assegnato un attributo " ***data-dialog*** ", che deve essere anche unico.
>
> L'attributo data-dialog, è simile all'attributo data-target nel modal bootstrap. Deve essere univoco per ogni nuovo progetto.
>
> Lasciamo all'utente, per mantenere l'unicità. Quindi l'utente non può avere lo stesso attributo modal-id per diverse chiamate `<tool-bar>` .

Quindi, come afferma la nota precedente, dobbiamo mappare l'icona delle informazioni di ogni

progetto, in una finestra di dialogo dei dati.

Aggiungiamo questa funzionalità.

Usiamo il metodo ready. È molto simile a quello di Jquery

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

. Viene richiamato quando l'elemento personalizzato ha terminato il caricamento.

Aggiungi questa funzione, dopo l'oggetto properties, nella chiamata Polymer per la registrazione.

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

Molto bello! Quindi ora abbiamo

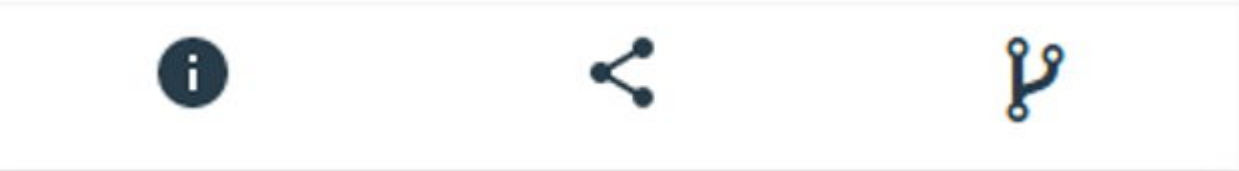
- Un elemento personalizzato registrato come elemento Polymer
- Modale | Condividi | Icone della forcella aggiunte all'elemento personalizzato

Un Modal è collegato all'icona di ferro delle informazioni, con l'id "anchor-for- [Qualunque sia passato come modal-id]". Questa è la cortesia del metodo pronto del nostro elemento.

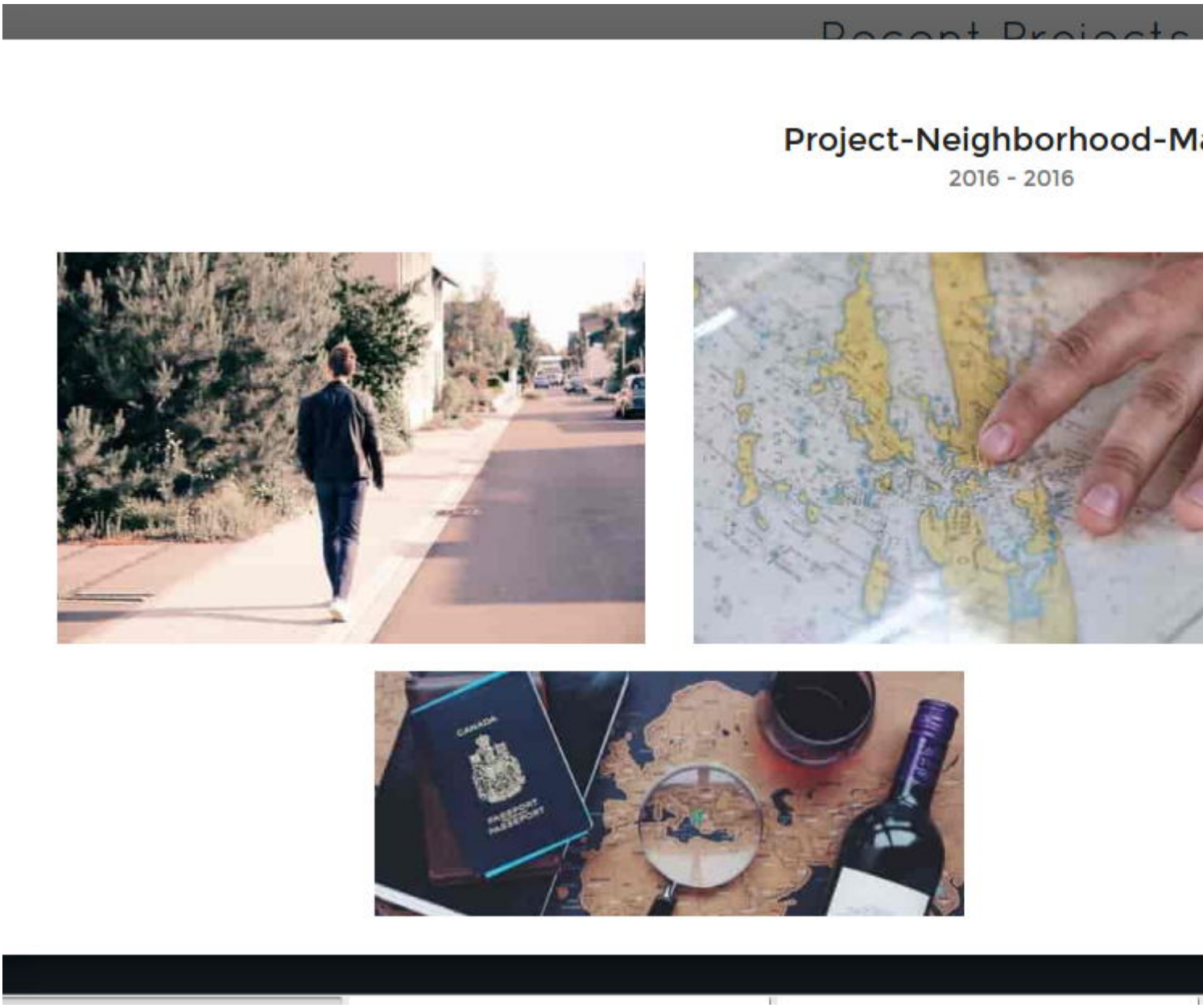
Quindi, finora, scrivendo

```
<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 questo:



Ma quando l'utente fa clic su quell'icona di informazioni, qualcosa come Questo dovrebbe accadere



e il nostro obiettivo era quello di avvolgere questa descrizione completa del progetto, all'interno del `<tool-bar>` nella pagina degli utenti.

Ricorda Shadow DOM? Abbiamo bisogno di un contenitore nel nostro elemento personalizzato, che fornisce il DOM ombra per i contenuti del nostro modal. Il contenuto stesso è racchiuso all'interno di un tag `<content>` all'interno dell'ombra personalizzata dell'elemento personalizzato.

Per il contenitore che ospita la descrizione del progetto, utilizziamo l'elemento Polimero " **carta-dialogo** ". Paper-dialog, è assegnato come id, qualunque sia passato come id-modale nella dichiarazione sulla pagina dell'utente.

Quindi, aggiungi questo markup, dopo l' ***ul*** nel DOM dell'elemento personalizzato della barra degli strumenti

```
<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: carta-dialogo scorrevole

> Usiamo ***Paper-dialog-scrollable*** , per racchiudere la descrizione del nostro Progetto, in modo che, Qualsiasi descrizione lunga, non trabocchi.

L'ultimo bit, è quello di aggiungere la funzionalità onclick per quando la modale dovrebbe aprirsi, al clic sull'icona delle informazioni.

Quando accade un clic, come è normale, viene attivato un evento, che passiamo alla nostra funzione clickHandler in questo modo:

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

Quindi una volta che la funzione clickHandler ottiene l'evento passato ad esso,

- Ha bisogno di raccogliere, quale elemento ha attivato il clic,
- Ha bisogno di raccogliere, l'attributo data-dialog di quell'elemento, per determinare quale modale deve essere aperto (Ricorda che ogni progetto ha il proprio ID modale?)
- Quindi ha bisogno di chiamare il metodo aperto, esattamente per quel modale

Quindi, la funzione è così:

Aggiungi questo, prima della chiamata di registrazione dell'elemento. Questa non è una parte dell'elemento, ma è inclusa all'interno dell'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();
    }
}
```

Quindi, con questo, abbiamo completato la codifica dell'elemento personalizzato, `<tool-bar>` .

Il quadro completo Elemento personalizzato

```
<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) {
```

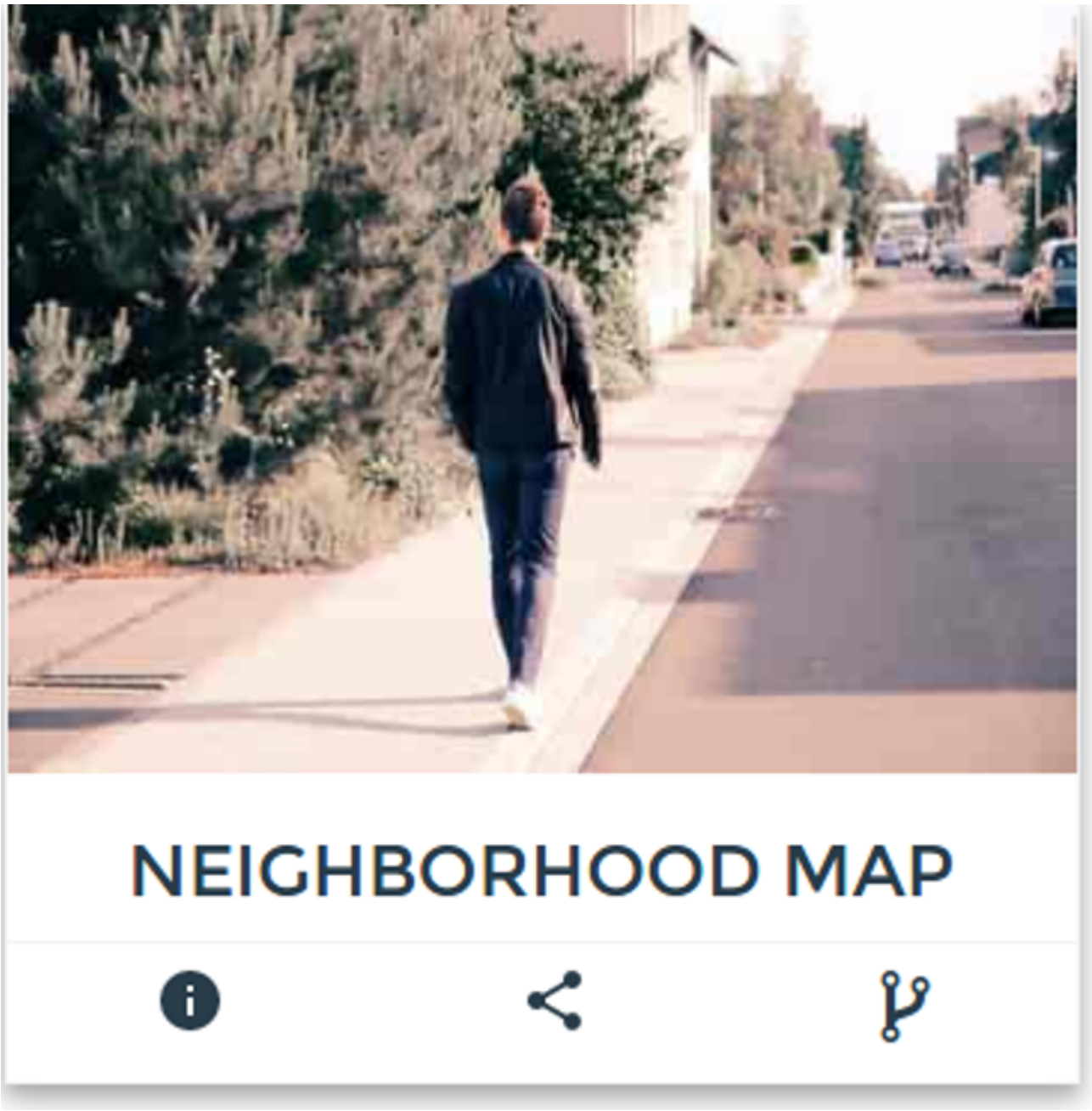
```
                dialog.open();
              }
           }
      Polymer({
             is: "tool-bar",
            properties: {
                  link2Fork:{
                      type:String,
                      value:""
                  },
                 link2Share: {
                       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>
```

Utilizzando l'elemento personalizzato con Modal | Condividi | Icone della forcella

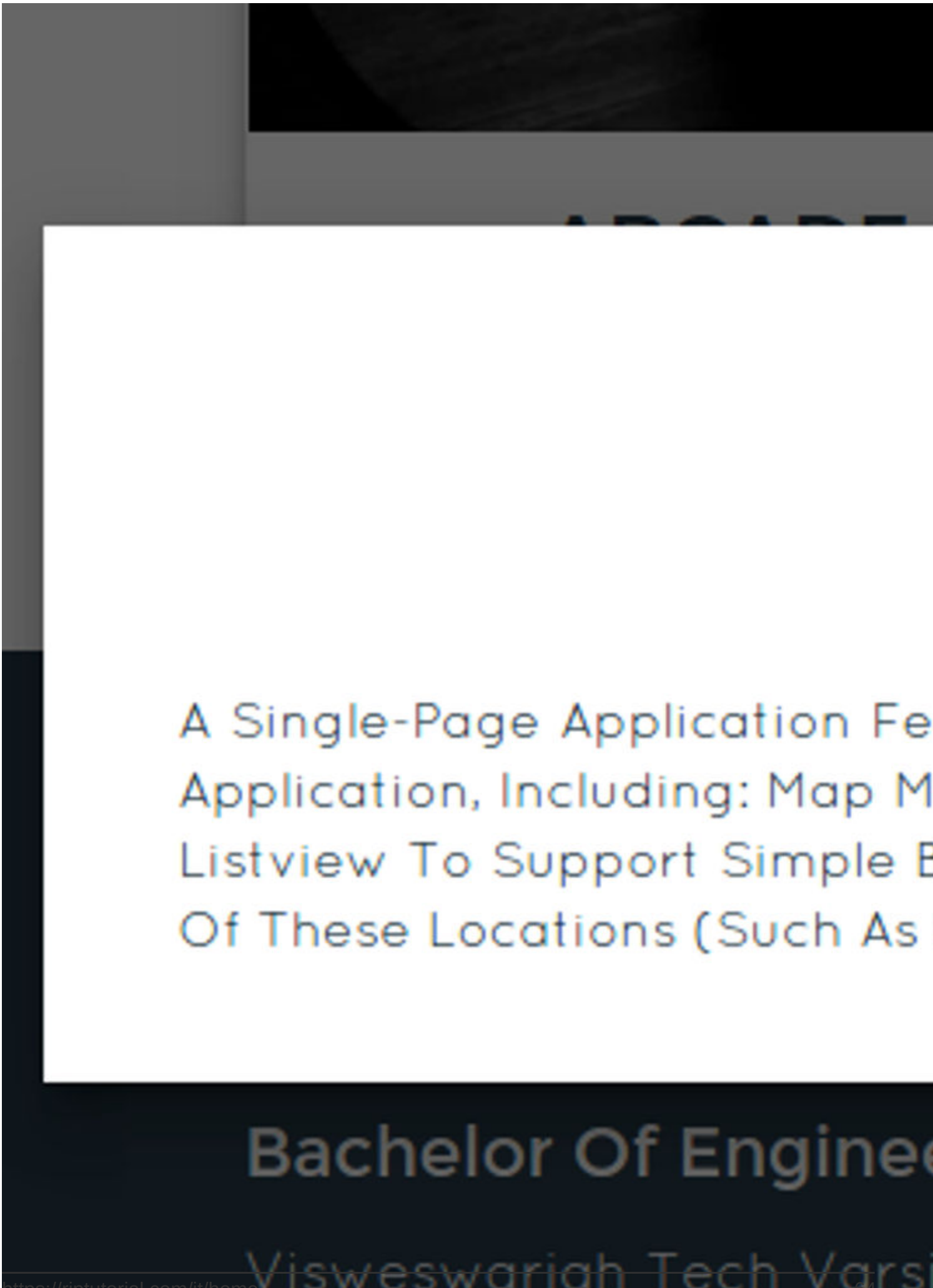
In qualsiasi pagina desideri visualizzare il tuo portafoglio di prodotti / progetti, invoca l'elemento personalizzato in questo modo:

```
<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>
```

e ottieni questo come anteprima



E quando fai clic su Info, ottieni questo


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/it/polymer/topic/7244/moduli-riutilizzabili-con-polimero

Capitolo 11: SUPER-Ottimizzazione per la produzione

introduzione

Una volta che il tuo progetto è terminato, non ti resta che chiedervi come caricherete il carico di 100 importazioni HTML sul vostro server web e anche se lo farete, quante ore il vostro sito dovrà caricare per un singolo cliente. In questo argomento, vedrai come convertire il pasticcio di sviluppo in singoli file html e js raffinati.

Sintassi

- npm install: salva i pacchetti usando Node.js Package Manager
- npm install -g: salva i pacchetti da npm come globali (utile per i pacchetti di interfaccia della riga di comando)
- cd: imposta lo stato attivo della riga di comando su una libreria specifica in cui è possibile eseguire i processi
- vulcanize: scricchiola tutte le importazioni HTML in un singolo file
- Crisp: converte JS in linea nel file HTML in un file JS esterno

Examples

Installare tutti gli strumenti richiesti

Esegui il seguente comando uno per uno:

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

Uso

- Vulcanizza: esegue il crunch di tutti i file di importazione HTML in un singolo file
- Crisper: estrae js inline sul proprio file

Nota : gli utenti di Ubuntu potrebbero aver bisogno di prefixare il comando precedente con `sudo` .

Mettere tutto insieme

Metti tutte le importazioni html nei tuoi file in un singolo file `elements.html` . Non preoccuparti di un file importato più di una volta, verrà ridotto a una singola importazione.

Esecuzione vulcanizzata e più nitida

sul tuo file `elements.html` , esegui i seguenti comandi:

```
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
```

Vulcanizza il codice sorgente recuperato di tutte le importazioni, quindi sostituisce le importazioni con il loro codice sorgente.
Crisper ha preso tutto il js dal file elements.vulc.html, lo ha inserito nel singolo file build.js, ha impostato un tag `script` fa riferimento al file `build.js` nel file `build.html`

Minimizzando i file

- Apri minifier HTML
- Apri `build.html`
- Copia tutto il suo codice
- Nella prima textarea di HTML minfier, incolla il codice copiato da `build.html`
- Fai clic **sul** pulsante **Minify**
- Nella seconda textarea verrà visualizzato il codice minificato. Copia questo
- Creare un file `build.min.html` e incollare tutto il codice copiato al suo interno
- Apri JSCompress
- Seleziona la seconda scheda, ad esempio **carica i file JavaScript**
- Clicca su `Choose Files`
- Seleziona il file `build.js`
- Clicca sul pulsante *Comprimi*
- Scarica il file come `build.js` nella stessa directory di build.min.html

importare build.min.html

Rimuovi tutte le importazioni precedenti da quei file HTmL dai quali hai copiato le importazioni. Sostituire le importazioni con

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

Leggi SUPER-Ottimizzazione per la produzione online:
https://riptutorial.com/it/polymer/topic/9336/super-ottimizzazione-per-la-produzione

Capitolo 12: Test unitario

Osservazioni

Web Component Tester : lo strumento per le app di test delle unità realizzate con Polymer. Si ottiene un ambiente di test basato su browser, configurato immediatamente con mocha , chai , async , lodash , sinon e sinon-chai , test-fixture , accessibility-developer-tools . WCT eseguirà i test su qualsiasi browser installato localmente o in remoto tramite Sauce Labs.

Examples

Semplice esempio con web-component-tester

installazione

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

impostare

wct.conf.js

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

in esecuzione

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

test / 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);
    }

});
```

Leggi Test unitario online: https://riptutorial.com/it/polymer/topic/3898/test-unitario

Capitolo 13: Utilizzo di librerie Javascript esterne con Polymer

introduzione

Poiché tutti i componenti Web devono essere autonomi, incluse tutte le loro dipendenze, l'importazione delle dipendenze duplicate diventerebbe rapidamente un problema con gli script inclusi. Pertanto, Web Components (e, per estensione, Polymer) utilizzano le importazioni HTML W3C per la gestione delle dipendenze dei componenti. Questi file HTML importati possono essere memorizzati nella cache dal browser e verranno caricati solo una volta.

La maggior parte delle librerie esterne non sono ancora preparate per le importazioni HTML. Fortunatamente la creazione del wrapper HTML necessario è veloce e diretta.

Parametri

Parametro	Descrizione
this.resolveUrl (' ../ biblioteche / turf.js')	Risolve la posizione della libreria
function () {this.doSomething (argomento, anotherArgument);}. bind (this));	Richiama. Questo esempio utilizza una chiusura per recurse this.doSomething ()

Osservazioni

In questo esempio, la libreria utilizzata nel componente viene installata con il gestore dei pacchetti bower. Ciò consente una facile distribuzione di un componente dipendente dalla libreria. Se la libreria che desideri utilizzare non è distribuita tramite un gestore di pacchetti, può comunque essere caricata allo stesso modo, ma il tuo componente richiederà uno sforzo maggiore per essere utilizzato da altri.

L'esempio di caricamento lazy usa una stringa semplice per il percorso della libreria. Se si desiderava evitare le costanti di stringa magiche, i percorsi potevano essere caricati usando iron-ajax da un file JSON in un oggetto e passati tra i componenti se necessario.

Examples

Importa un file HTML statico

1. Crea un file HTML (in questo esempio `libraries/turf.html`) con la libreria che vuoi caricare:

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

2. Importa il file HTML (`libraries/turf.html`) nel tuo componente con il resto delle tue importazioni:

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

3. Invoca la tua libreria (esempio di utilizzo):

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

Caricamento pigro

1. Crea un file HTML (in questo esempio `libraries/turf.html`) con la libreria che vuoi caricare:

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

2. Importa e usa la tua libreria quando necessario:

```
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]);
}
```

Leggi Utilizzo di librerie Javascript esterne con Polymer online:
https://riptutorial.com/it/polymer/topic/6034/utilizzo-di-librerie-javascript-esterne-con-polymer

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con il polimero	Community, Fuzzical Logic, fybw id, Keith, Marc, Randy Askin, tony19, Ümit
2	Cheat Sheet Polymer	Josef Ježek
3	Creazione di app con Polymer Starter Kit	fybw id, Puru Vijay
4	Debug	willsquire
5	Esempio di Polymer toggleAttribute	a1626
6	ferro-dati-tavolo	Mowzer
7	Gestione degli eventi	a1626, Ümit
8	Google Map Mark With Built in Cache	Schrodinger's cat
9	Looping, il modello dom-repeat.	Jp_
10	Moduli riutilizzabili con polimero	Schrodinger's cat
11	SUPER-Ottimizzazione per la produzione	Puru Vijay
12	Test unitario	kashesandr, Marc
13	Utilizzo di librerie Javascript esterne con Polymer	antibrian, craPkit