FREE eBook

LEARNING polymer

Free unaffiliated eBook created from
Stack Overflow contributors.

#polymer

Table of Contents

About

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

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

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

Chapter 1: Getting started with polymer

Remarks

The Polymer project consists of:

- Polymer library: Polymer is a lightweight library that helps you take full advantage of Web Components. With Web Components, you can create reusable custom elements that interoperate seamlessly with the browser's built-in elements, or break your app up into right-sized components, making your code cleaner and less expensive to maintain.
- WebComponents Polyfill: WebComponents Polyfill is a future targeted library aimed at fulfilling the W3C web components specifications. Browsers that fully implement the specification do not need webcomponents.js. However, most browsers are still missing some part of the spec, so this will be a dependency for quite some time.
- Polymer App Toolbox: Polymer App Toolbox helps you build and deliver cutting-edge Progressive Web Apps with minimal overhead and payload, by leveraging powerful web platform features like Web Components, Service Worker and HTTP/2. The Toolbox provides a component-based architecture, responsive layouts, a modular router, localization support, turnkey support for local storage and offline caching, and efficient delivery of unbundled app resources. Adopt these features individually, or use them together to build a full-featured Progressive Web App. Polymer sprinkles a bit of sugar over the standard Web Components APIs, making it easier for you to get great results. The Polymer library provides a set of features for creating custom elements. These features are designed to make it easier and faster to make custom elements that work like standard DOM elements. Similar to standard DOM elements, Polymer elements can be:

Versions

Version	Release Date
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
v1.0.0	2015-05-27

Examples

Hello World

This example creates a Polymer element named `x-foo`, whose template binds to a string property, named "message". The element's HTML is imported into the main document, which allows usage of `<x-foo>` tags 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>
```

See demo in CodePen

Using Elements from the Polymer Catalog

Polymer prodives a lot of well built elements for you to use in your app.

Browse them in their Element Catalog.

Let's go through the workflow of using an element by including `paper-input` (Documentation)

Download the Element

To Download an element there are two ways:

Bower

The convinient way is to use the command line using the bower `install` command:

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

Note: `--save` adds the element as a dependency to the `bower.json` of your app.

ZIP file

The other way is to add the selected element (`paper-input` in this case) to your collection (in the Polymer Catalog) using "Add to Collection" in the navigation and download your collection using the star icon in the upper right corner.

This will generate a .zip file that contains the element and all of its dependencies. You can then copy the `bower_components` folder inside the .zip/components to the root directory of your app.

Import the Element in your app

To import the element you've just installed, import the corresponding `.html` file:

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

Use the Element

Now you can use `paper-input` inside the document you imported it to:

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

Basic Element Structure

We got the following very basic element `my-element` saved as `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>
```

- The `<link>` includes the Polymer library using an HTML import.
- The `<dom-module>` is the local DOM wrapper for the element (in this case, `my-element`).
- The `<template>` is the actual local DOM definition.
- The `<style>` inside the `<template>` lets you define styles that are scoped to this element and its local DOM and will not affect anything else in the document.
- The `<content>` will hold anything you place inside your element.
- The `:host` pseudo class matches the custom element (`my-element`).
- The `Polymer` call registers the element.
- The `is` Property is the element's name (it **has** to match the `<dom-module>`'s `id`)

You can import it in your app using:

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

And use it as a tag:

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

Setting up your first polymer app from a Template

Let's set yourself up to build your own awesome Progressive Web App with Polymer!

Before you can start installing Polymer you require the following:

- Node.js - check out the StackOverflow Installing Node.js Documentation
- Bower - you can install Bower using the Node Package Manager installed with Node.js:

  ```
  npm install -g bower
  ```

Installing the Polymer Command Line

Interface

The Polymer CLI provides you with all tools needed for Polymer Projects:

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

Initialize your app from an app template

Use `polymer init` to initialize your app from an app template.

A cool template is the `--app-drawer-template`. Let's use that:

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

Serve your app

There is no building needed to serve your first awesome Polymer app. Just `serve` it:

```
polymer serve --open
```

This will open the app in your default browser on `http://localhost:8080`.

Read Getting started with polymer online: https://riptutorial.com/polymer/topic/949/getting-started-with-polymer

Chapter 2: Creating app using Polymer Starter Kit

Introduction

Polymer Starter Kit is an excellent solution to start building Progressive Web Apps. It offers a responsive material design User Interface, Page Routing, Offline-first service.

Syntax

- Polymer: Command for initiating polymer cli
- Polymer init: Presents a list of templates to choose to build your element/app
- npm: Initiates the Node.js Package Manager CLI interface
- npm install: Installs a package
- npm install -g: Installs a package globally on your system. Useful for CLI tools

Examples

Installing Polymer Starter Kit

1. Ensure that you have Polymer CLI installed in your system globally. If not, open your command line/terminal interface and run this command: `npm install -g polymer-cli`

   Note: If you're an Ubuntu user, you may have to prefix the above code with `sudo` keyword.

2. After Polymer CLI installs, run this command

   cd [YOUR DIRECTORY IN WHICH YOU WANT TO START YOUR PROJECT]

```
polymer init
```

3. You will be given 4 options. Use arrows keys to go to the 4th one i.e. **starter-kit**. Hit `Enter` key.

   All the required files will be downloaded in your selected directory.

Read Creating app using Polymer Starter Kit online:
https://riptutorial.com/polymer/topic/9330/creating-app-using-polymer-starter-kit

Chapter 3: Debugging

Examples

Disabling HTML Import caching

HTML Import caching will sometimes mean that changes made to HTML files that get imported do not get reflected upon browser refresh. Take the following import as an example:

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

If a change is done to `my-element.html` after previously loading the page, then the changed file may not be downloaded and used in the current document when it is refreshed (as it was previously imported and cached). This can be great for a production, but might hinder development.

To disable this in Google Chrome:

- Open up Google Chrome's DevTools
- Select the Main Menu > Settings
- Go to the Network section
- Select "Disable cache (while DevTools is open)"

This will avoid caching HTML Imports, but only when DevTools is open.

Read Debugging online: https://riptutorial.com/polymer/topic/5864/debugging

Chapter 4: Event Handling

Remarks

- Here's a plunker for all the examples
- Attribute's name are case-insensitive and will always be converted to lowercase, e.g if you have an attribute `on-myListener` listener will be set on `mylistener` event.
- Similar to `listen` you can also use `unlisten` method remove any listener.
- Poperty change fires an event by the name of `property-changed` e.g if property is `myProperty` event will be `my-propert-changed`(camel casing to '-'). You can listen to them either by using `on-event` attribute of `listener` Object.
- New value is always stored in `e.detail.value` for property change events.

Examples

Event listening using listener Object

```
<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>
```

Annotated Listener

Another way to add an event listener is to use `on-event` annotation in DOM. Below is an example using `up` event, which is fired when finger/mouse goes up

```
<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>
```

Imperative listener

You can also add/remove listener imperatively using listen and unlisten method of 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>
```

Custom Events

You can also fire your own events and then listen to them from either Polymer element of HTML page

This Element fires the custom event

```
    <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>
```

Here's an element which is listening to that custom event

```
  <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>
```

Listening from 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>
```

Property change event

Properties with `notify:true` also fires an event

```
<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>
```

Event Retargeting

It is possible to retarget an event in Polymer ie you can change event details like `path` thus hiding the actual details of an event/element from user.

For e.g if a `div` inside `event-retargeting` element is firing the event but developer does not want the user to know that he can retarget the event to `event-retargeting` element by using the following code.

```
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'));
```

To see the working example please refer to `plunker` in `remarks` section.

Read Event Handling online: https://riptutorial.com/polymer/topic/4778/event-handling

Chapter 5: Example of Polymer toggleAttribute

Syntax

1. toggleAttribute(name, bool, node)

Parameters

Name	Details
name	String: name of the HTML attribute which needs to be toggled
bool	boolean: Boolean to force the attribute on or off. When unspecified, the state of the attribute will be reversed.
node	HTMLElement: name of the node which contains the HTML attribute. Defaults to this

Remarks

A good example of this will be form, where submit button should only be active if all the mandatory fields have input.

Examples

Basic example

```
<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>
```

Here's a running plunker

Read Example of Polymer toggleAttribute online:
https://riptutorial.com/polymer/topic/5978/example-of-polymer-toggleattribute

Chapter 6: Google Map Mark With Built in Cache

Syntax

- <g-map

```
api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

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

Parameters

api-key	google's javascript api key
locations	List of locations you want to mark on the google map
----------	----------

Remarks

For the entire code, Navigate to the Repository

To see the google map in action, Look Here

Examples

A - Getting Started

In a nutshell, Our Custom Element Should

- Have a Built in Search Functionality, that searches and marks places on the map.
- Accept An attribute called location-array , which is a list of places
- Have a Listener , for listening to an event called “google-map-ready“.This event is fired, when the element has loaded.
    - This listener, should loop through the location-array, and assign the next element in your location-array as the current “search query”
- Have A method called cache
    - This method, caches the latitude and longitude of each place in the location-array, as soon as the search returns a result
- Have a Dom-repeat template, that loops through the cached results, and drops a marker at each location

So, the purpose of our custom element in essence means, If you pass an array of locations like so:

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

The custom element

- Should Not only mark Norway and Sweden on the map, but,
- Should also mark any subsequent searches on the map – i.e, If you search for Boston, after the map has marked Norway and Sweden, Boston should also have a pin on the map

Importing Dependencies

First, Let us import whatever we need into our element.This element is under

elements/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">
```

Note

Some of the above imports, aren't necessary, but good to have, in case you fancy add on functionality on your markers

We necessarily need,

- google-map
- google-map-marker
- google-map-search

Also, It is presumed, you know how to install the individual Polymer Elements Via Bower

Registering Our Element With Polymer

Next, we register our custom element as "g-map"

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

So! our custom element is registered with Polymer.

We will add google map search specific properties and any other properties we need, later.

Adding a Blueprint to our custom element - via templates

Next, we add the template for our custom element

Open up elements/g-map.html, and add a template that binds the custom elements DOM

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

Now, wrap the entire html and javascript you wrote for our custom element, inside of a dom-module.

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

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

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

Very Nice! We have a basic blueprint

B - Rendering the basic google map

Getting You Browser Ready - Polyfill it

Create a new landing page for our element, at index.html.

Include polyfills for a custom element to work.Also import the custom element g-map, which we registered with Polymer.

Note:

- Webcomponents, are the necessary polyfill for browser support.
- Polymer, is included, so that we can use the library to create our custom element

So Open up index.html, and include the necessary Polyfills and Polymer. Also, **import** the custom element we registered

```
<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>
```

So With the Browser Polyfilled,

Let us invoke the google-map polymer element inside our custom element.

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

    <google-map></google-map>

</template>
```

and then, invoke the custom element inside index.html, our landing page.

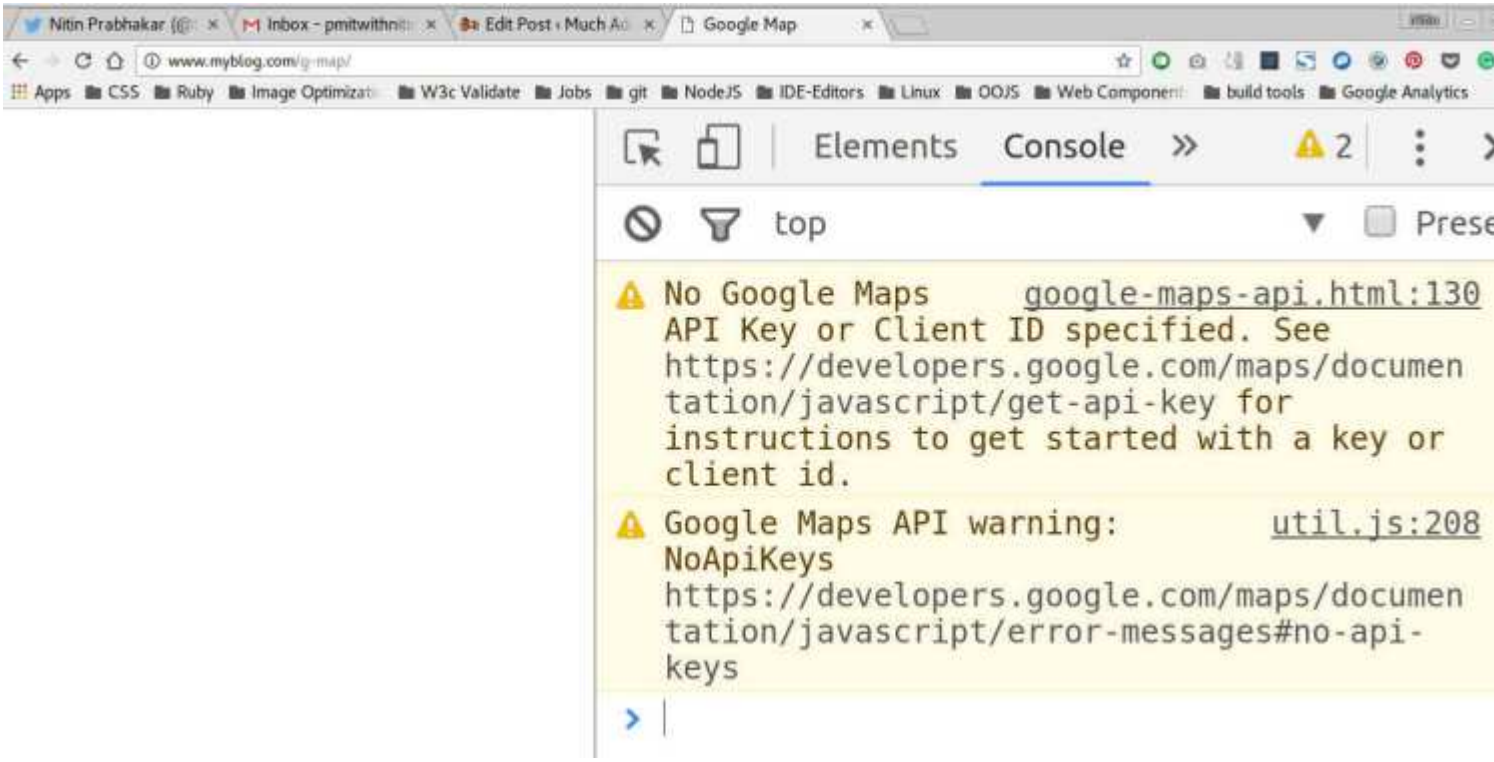
So this goes inside the index.html

```
<body>

    <div class="container">

        <div class="row">

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

        </div>

    </div>

</body>
```

I tried rendering my page with the custom element g-map at this point, and found an EMPTY page!

WHY??

When I see the console log, I see this



Ah!

So, We need an api key to render a map from google.

Note:

Getting an API key from google's javascript API, is pretty simple.

Just follow the link below, to generate your personal key per project.

Google API Key

Now, Since the API Key is to be used, we have two choices.

- Let the user pass it as an attribute
- Have a default API Key configured

Now, I would go for the first, simply because, any API Key has its imposed limits. i.e it can only be used so many times.So, a key configured and shipped with the element we develop, could soon run out.

Whoever wants to use the custom element, can generate a new key, and pass it as an attribute.

like so:

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

So We are passing the api key as an attribute and within our custom element, we bind it on to the Polymer element's api-key attribute.

like so:

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

So Once we pass the API Key, and refresh, we Still do not see the map!

Note:

Google maps, need some css to render.We need the height of the map set explicitly.

So, make a css file inside the css directory, say app.css and, add these lines

```
google-map{

    min-height:30vmax;
}
```

And now upon refresh, we see this

Yay! We just rendered a map with just minimal markup! We just wrote this!

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

C - Adding Search Ability to our custom element

For Searching a place, we use the powerful element that Polymer ships, called google-map-search .

All you need to do, is pass

- a map object and
- a query string to the element like so:

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

How do we pass the map object? Where do we get that from?

Simple!

When you invoke the google-map element, bind your custom element 's property map, to the element's property map.

So, we invoke the google-map element like so:

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

Note:

We do a two way data bind for map above, when invoking the google-map polymer.

It is represented by the curly braces {{}} instead of a one way binding indicated by square braces[[]].

When we do a two way bind,

- Whenever our custom element g-map's map property changes, google-map element is notified

and,

- Whenever google-map's map property changes, we are notified in our custom element g-map.

So, with the two way data binding in place, anytime the map object changes in google-map, our custom element is updated with the changes.

and we pass the map object as an attribute to the element google-map-search, so that , any search results are marked in the current map rendered.

How does our custom element's DOM look at this moment?

```
<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>
```

We never added the map and query properties for our custom element!

Add the properties in the registration call to Polymer

```
Polymer({
```

```
    is:"g-map",

    properties:{

        map:{

            type: Object

        },

        query:{

            type:String,

            value:""

        }
    }

});
```

Now that we have registered the properties, we note, that we

- Are getting the currently rendered map as an object – via data binding and storing it as a local property also called map,
- Have a property query, passed to google-map-search

Accepting User Input as Search Query

But who gives us the query to search for? Err! right now? NOBODY.

Thats precisely what we do next.

Let us add a search form, which the user who renders the map, would use to input a query. We need inputs for the search box, and we use Polymer's iron input.

Adding a Search Form

Include the necessary iron elements at the beginning of the 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>
```

We want the search form, at the top left corner, so we add an iron input with position absolute, and some css to pin it there.

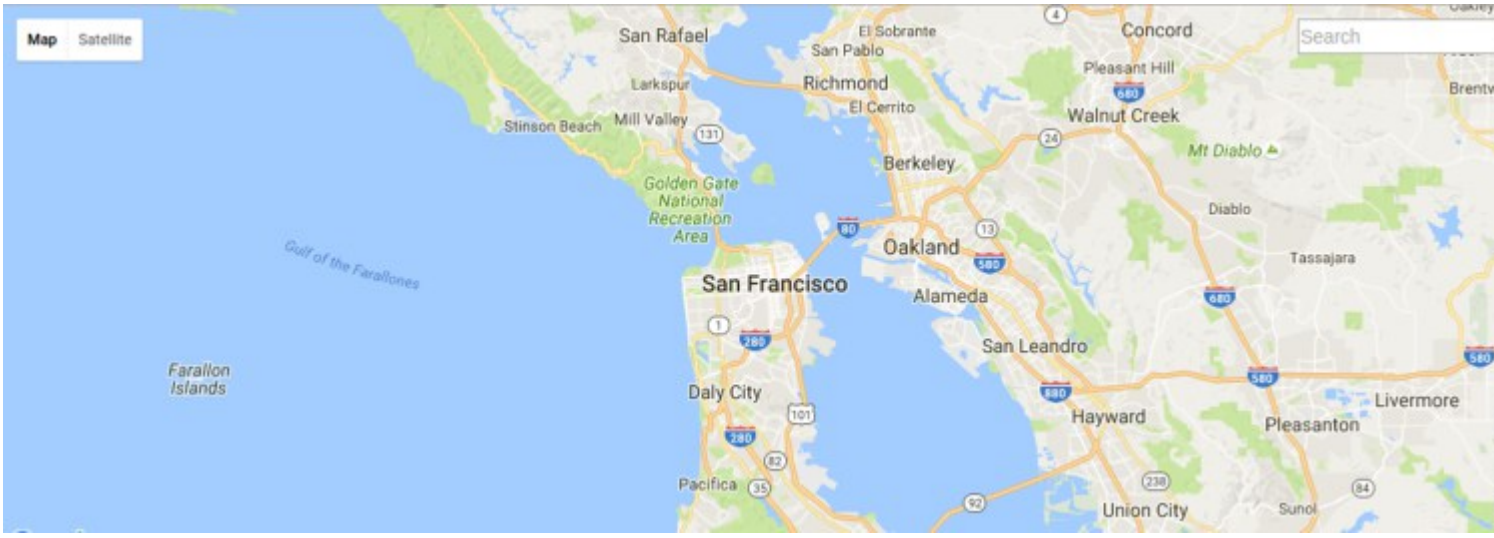
So, wrap the DOM for our custom element, inside a div called "map-container", and add a child div called "search_form".

```
<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>
```

Add some basic css to pin the search form we wrote to the top right of the map

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

And , then when we refresh, we have this. The search form at the top right



All that is left to do is, for us to provide an input to google-map-search element, via the query property of our custom element.

So, we bind our property “query”, to the iron input in the search form. like so:

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

Note:

We bind our custom element’s property query, to the search input using,

“bind-value” attribute of an iron input.

So, bind-value={{query}} in the above iron input, makes sure, any changes to the text inside the iron input, is notified back to the query property.

With our current setup,

For every alphabet a user types into the search box, a search happens, which slows down the map, and Consumes the limit on our API Key.

Why?

Because, our custom element’s property query, is bound to the iron input, and every letter typed into the iron input, changes the value of “query“, which then affects the html in

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

causing a search to happen

Our solution is simple! Just bind the “query” attribute of google-map-search element, to a separate property.

Let us say, we create a property “searchQuery” for our custom element.

Add this, below the “query” property in the Polymer call

```
searchQuery :{

    type:String

}
```

Next, change the bind in google-map-search element call in our custom element’s DOM

like so:

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

Lastly, Note that we added an iron icon (search icon) to our search form?

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

Let us add that search function to our custom element. It is called on tapping the search icon.

In our search method, we assign whatever is there in the search box, to the “query” attribute of google-map-search!

Add this function, after the “properties” object in the polymer call for registration.

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

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

Very Nice! So, Now, A Search happens only on tap of the search icon.

Display The Search Results

We have

- Rendered a Map
- Pinned a search form to the Map
- Wrote the functionality to search on tap of the search icon

Now we need to display the results.

For that, we need to bind the results from the google-map-search element, onto a local property.

So, Let us create a property for our custom element g-map, and name it results duh!

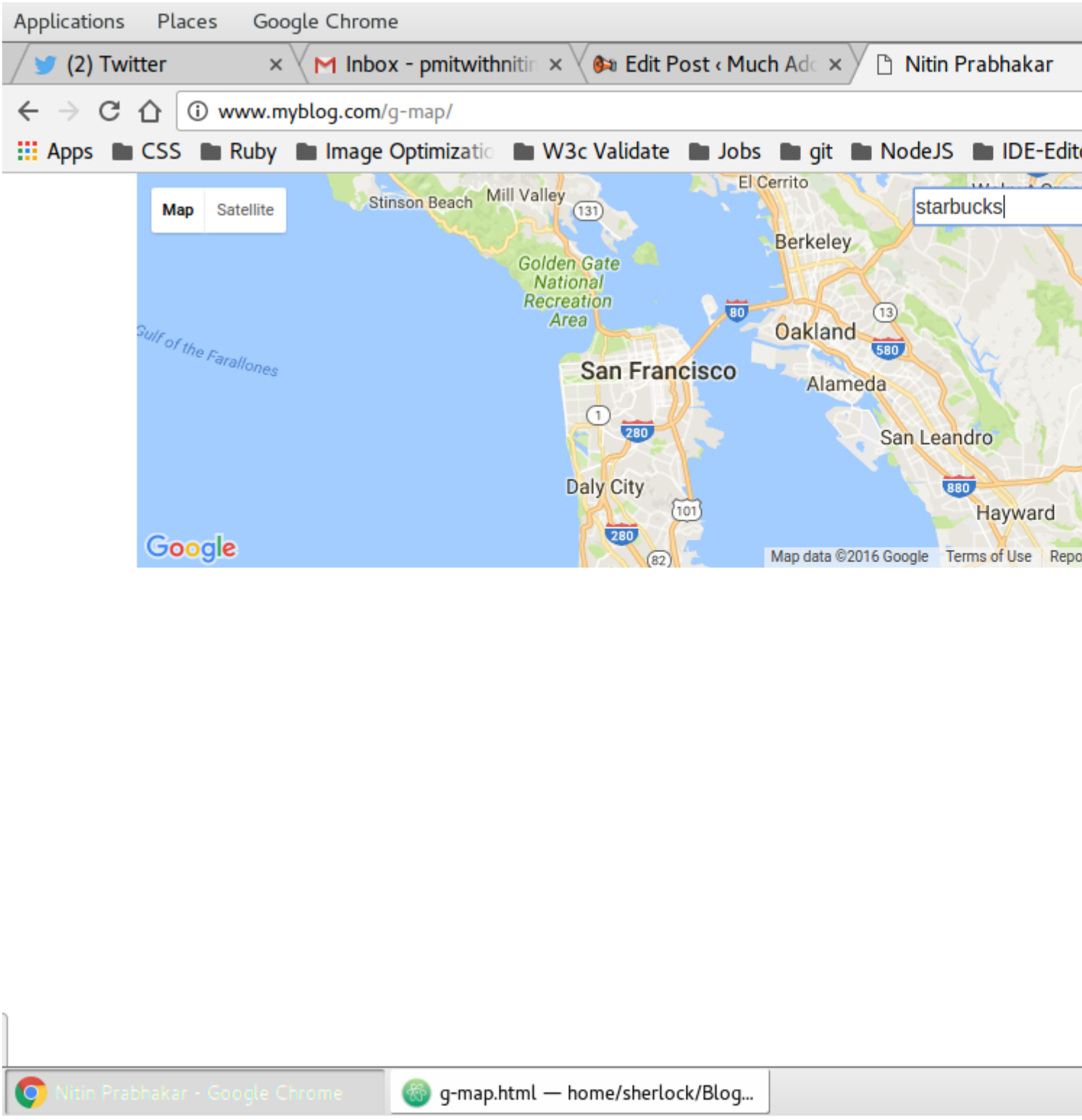
In the Polymer registration call, add the property results of type object in the properties declaration

```
results:{

type:Object

}
```

And bind the results property of google-map-search to the local results property you defined above.

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

With that in place, let us refresh the page, and search for **starbucks** on the map.



Nothing happened!

Well Duh! we just bound the results from google-map-search, to the results property. Did we write anything to display them on the map? NO!

We Need to.

So, we know, that the Polymer element google-map-search, returns an array of markers as results.

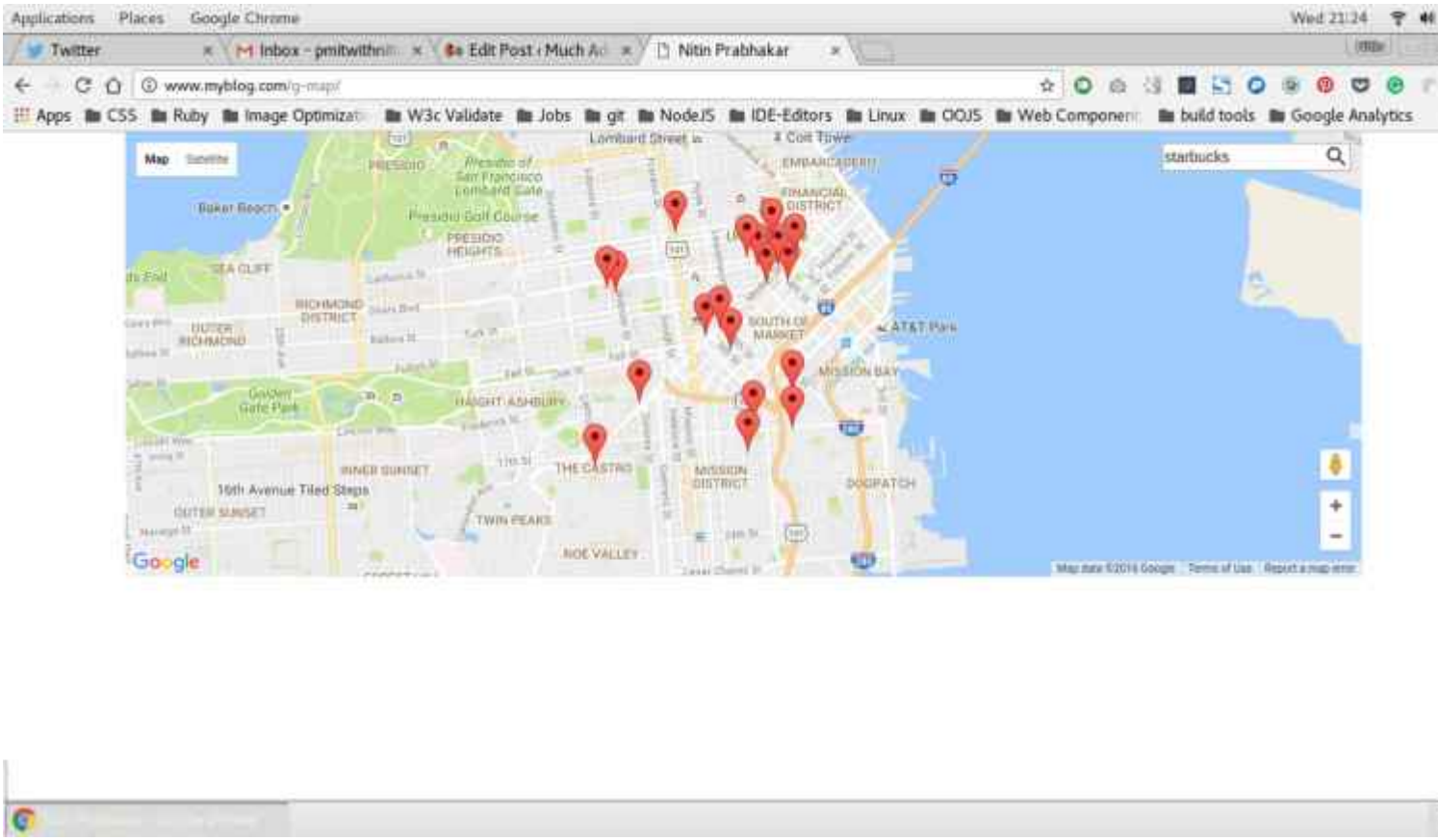
We need to loop through the array, and place markers on the map.

So, we write a dom-repeat template, within our element, to display the markers for each result

like so:

```
<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>
```

And Now on refresh, we see this



Very Nice!

We have successfully added the search functionality to our custom element.

All we need to do is,

write this in our index.html

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

And you get a map on which you can search!

D - Caching Search Results

Our goal however, was to

- make a list of places
- mark all of them on the map and,
- mark any other place of our choice, by using the search box

We have a search box, where the user can search for one query at a time.In order to accommodate multiple queries, we need to cache the results for each query, before we display the markers and pin them on the map

So, Let us add

- A Property say locations which accepts an array of locations
- A Property cache which stores the results of each query in the location array passed to our custom element

```
cache:{

type:Object

},

locations:{

type:Object

}
```

Next, we write a function to cache each search result , resulting from a search for every place in the location array.

It is pretty simple!

We already are gathering the results into a property called “results”, which we then are looping through to display markers.

All we now need to do, is push each search result into an array, and our cache is done!

We need to use an event to trigger caching. That event, is called, “on-google-map-search-result” and it is fired, after the element google-map-search finishes a search.

So “on-google-map-search-result” , we call a function say “cacher” , to cache the current search results into a property called cache.

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

Note:

Normally, to push an item into an array, we use the vanilla push method

e.g: I have a new place to add to my list of places I want to mark, say NewYork.

If the array I want to add it to is called locations, I would write

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

However, in our case, we need to push any result from google-map-search into an array called cache.Then we need to loop through the cache with the latest results included.Not the existing cache.

Now, we use the dom-repeat template to loop through the results and place markers on the map.

However, If we pass a location array like so:

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

Then, the results property of our custom element , will be overwritten for every item in that array.

Hence, we call the cacher method on-google-map-search-results event, and push the current result into the cache property.

This can not be a vanilla push like in the note above.

We Need to let the dom-repeater know, that our cache has been overwritten with each new search.

So, we use a special push that Polymer ships, like so:

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

Add that function, after the search function we wrote earlier.

The syntax implies, that the property cache is to be mutated, by addition of the current value of the property results and any dom-repeaters using the cache property to loop through, are to be notified of the mutation.

Lastly, we change the dom-repeat template to loop through cache, instead of results

Note:

cache is an array of arrays.

each element of the cache array, is a results array

So our dom-repeater, will be like so:

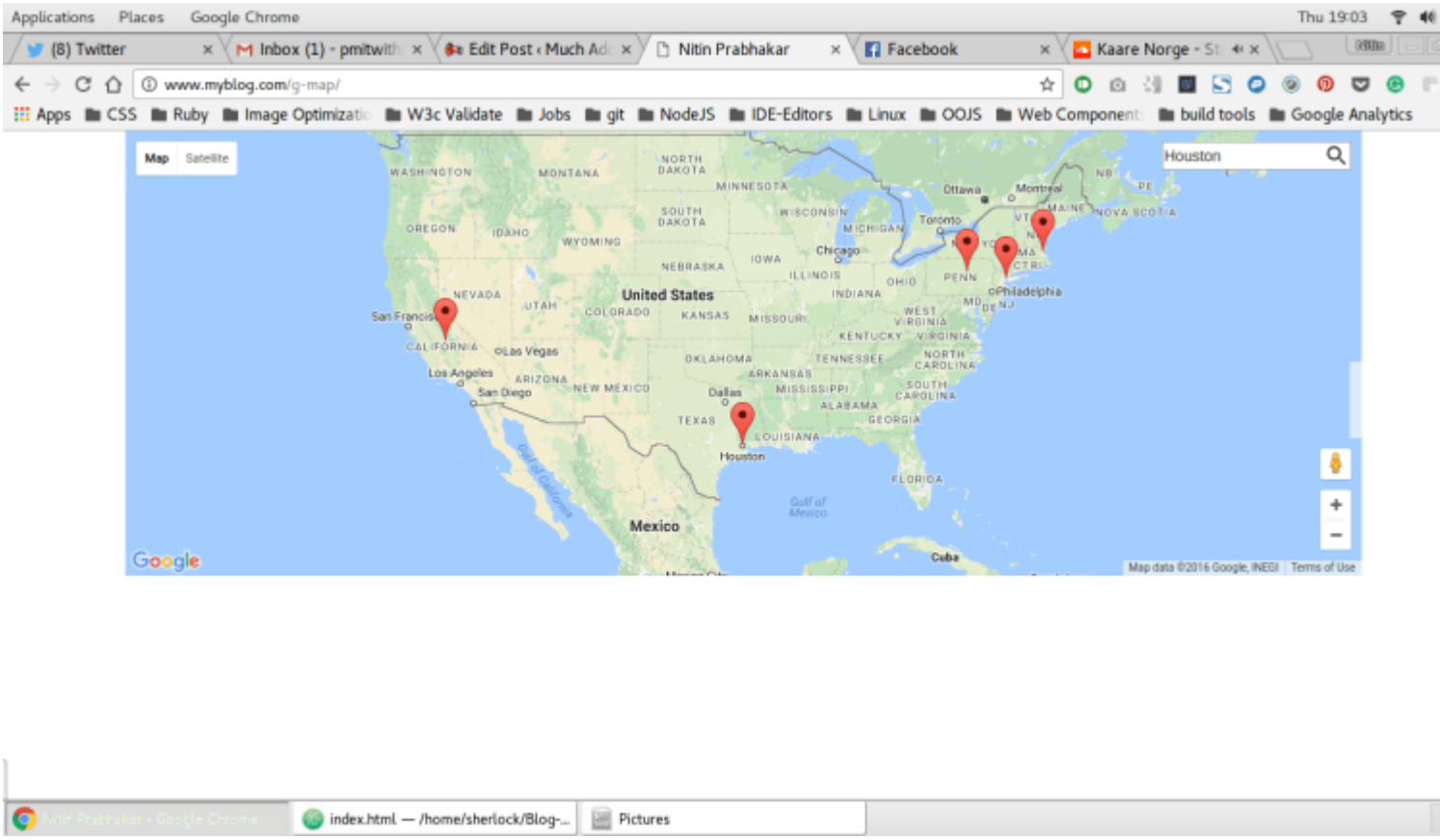
```
<template id="resultList" is="dom-repeat" items="[[cache]]">

    <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>
```

Very Nice! We have the custom element all coded and ready to be used!

Let us test it out by passing an array of locations from index.html shall we?

```
<g-map

api-key="AIzaSyBLc_T17p91u6ujSpThe2H3nh_8nG2p6FQ"

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

and there we go!



The Complete Custom Google Map Polymer With In Built 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>
```

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

Chapter 7: iron-data-table

Examples

Hello world

Initial starting point for `iron-data-table`.

Working jsBin

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

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

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

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

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

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

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

CSS import

Import external style sheet.

Working jsBin

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

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

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

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

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

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

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

Row details

Expand row details to display additional data.

Working jsBin

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

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

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

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

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

Edit row details

Working jsBin

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

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

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

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

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

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

Edit row details using sub-element

This example uses a separate element to edit bound data to the `row-detail` template.

Working jsBin

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

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

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

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

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

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

Note

There is currently an issue described here that causes row details section to collapse if it contains a sub-element. The patch is to include `tabindex="0"` as follows. (See this Stack Overflow answer.)

x-foo.html

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

Select row, prevent deselection

Default behavior is to de-select row when clicked twice. In some use cases, you might want to disable this de-selecting behavior.

Note

`table.deselectItem(item)` method will imperatively deselect an item. This works with both `item` or `index` (when using items array) as an argument.

Working jsBin

```
<!DOCTYPE html>
<html>
```

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

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

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

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

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

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

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

Read iron-data-table online: https://riptutorial.com/polymer/topic/6447/iron-data-table

Chapter 8: Looping, the template dom-repeat.

Examples

A basic list

This is a basic polymer element that show a list of names.

```
<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>
```

Read Looping, the template dom-repeat. online: https://riptutorial.com/polymer/topic/6160/looping--the-template-dom-repeat-

Chapter 9: Polymer Cheat Sheet

Introduction

This is a cheat sheet for the Polymer 2.x library. Fork of post from Monica Dinculescu.

Examples

Defining an element

Docs: 1.x -> 2.x upgrade guide, registering an element, shared style modules.

```
<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>
```

To get the class definition for a particular custom tag, you can use `customElements.get('element-name')`.

Extending an element

Docs: extending elements, inherited templates.

Instead of `Polymer.Element`, a custom element can extend a different element:

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

To change or add to the parent's template, override the `template` getter:

```
<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>
```

If you don't know the parent class, you can also use:

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

Defining a mixin

Docs: mixins, hybrid elements.

Defining a class expression mixin to share implementation between different elements:

```
<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>
```

Using the mixin in an element definition:

```
<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>
```

Using hybrid behaviors (defined in the 1.x syntax) as 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>
```

Lifecycle methods

Docs: lifecycle callbacks, ready.

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

Data binding

Docs: data binding, attribute binding, binding to array items, computed bindings.

Don't forget: Polymer camel-cases properties, so if in JavaScript you use `myProperty`, in HTML you would use `my-property`.

One way binding: when `myProperty` changes, `theirProperty` gets updated:

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

Two way binding: when `myProperty` changes, `theirProperty` gets updated, and vice versa:

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

Attribute binding: when `myProperty` is `true`, the element is hidden; when it's `false`, the element is visible. The difference between attribute and property binding is that property binding is equivalent to `someElement.someProp = value`, whereas attribute binding is equivalent to: `someElement.setAttribute(someProp, value)`

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

Computed binding: binding to the `class` attribute will recompile styles when `myProperty` changes:

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

Observers

Docs: observers, multi-property observers, observing array mutations, adding observers dynamically.

Adding an `observer` in the `properties` block lets you observe changes in the value of a property:

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

In the `observers` block:

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

Adding an observer dynamically for a property `otherProperty`:

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

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

Chapter 10: Reusable Modals with Polymer

Syntax

- Element Declaration: <tool-bar link2-share=""link2-fork=""modal-id=""title=""></tool-bar>

Remarks

Note:

The code through this writeup, is not a working copy. You need to replace the fillers for hrefs,src's and project names.

The code illustrates only a Proof of concept.

To see the custom element in action,

go here

And, to browse through the usage and structure of the custom element,

go here

The ***Usage*** is in "index.html"

The ***element*** is in, "elements/tool-bar.html"

Examples

Modals

So you want to add material design to your business or career portfolio.huh? You Just can not resist using a Modal? which pops upon a click on those crisp cards you intend to design, for each of your projects/products!

Let us say, for each project, you have done, or for each product you have designed, you need a material card.Each such card should have

- An icon, that pops up an info box, that lists all the features of
- your product or project, in detail. A "share this" icon, that shares
- your product or project across social media A "fork this" icon, that opens up the github page for your product/project

With native html, and assuming you use bootstrap or any other flex box layout, you would need to

- Write a wrapper div for each new project with a class row
- Wrap your hero image for each project in a column 12 cols wide

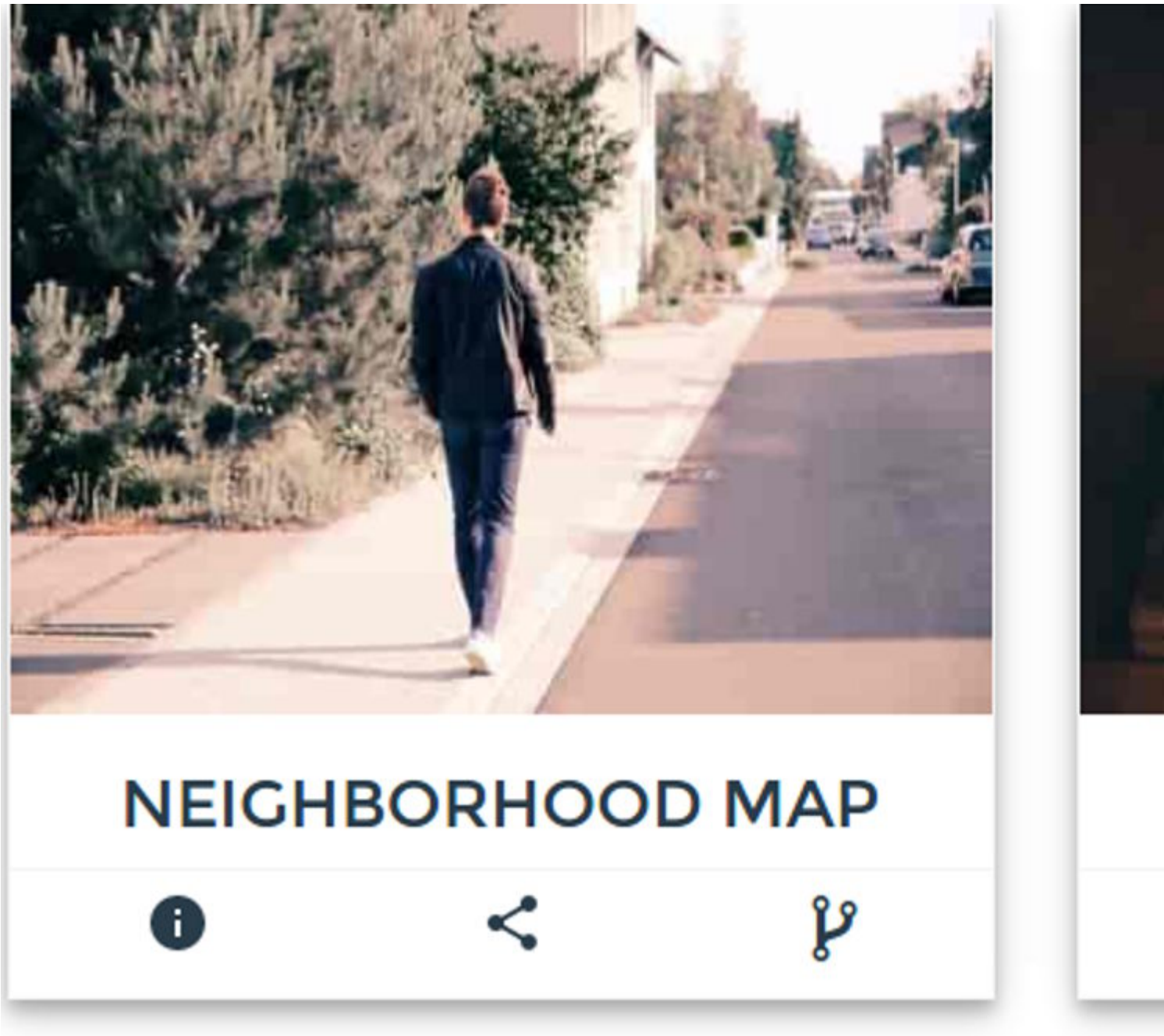
- Write Another row
- Wrap each link(info | share | fork) in a column 4 cols wide of its own
- Inject these two rows in to a container

For starters. Once you have done the above for all your projects, you need to work on creating the required pop up for each project.So,

- You download tons of javascript and css (by customizing bootstrap for modal functionality)
- You write a Modal with a unique id, to differentiate each project's info(modal- P1 for Project 1, modal-P2 for Project 2 and so on)
- You then compose the html required, to display information for each project, in its corresponding Modal.

What are our goals?

Just to reiterate, visually, If you have two projects in your portfolio, the goal is to have cards like so:



So we have two concerns for EACH Project

- The hero image
- The tool bar with info(Pops up a Modal),share and fork links

First, Let us tackle what happens when a user clicks on the info link in the bar below each project.We need a Modal to pop up, with more info about the project.

So How exactly do you configure a Modal with bootstrap?

- You write this to trigger it, for each project

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

- You then write the body of the Modal like so:

```
<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>
```

and depending on the content and the length of your Project Description, You actually get a little lost monitoring the opening and closing of a div soup above in the body of the Modal!

Plus, That markup is downright tedious to compose EVERYTIME you need a modal.

Phew! And did it end there? No. You still need to write for each project,

- The Project card’s hero image,that is displayed above the tool bar.

So How do we add the hero image for each project? MORE html!

The hero image needs

- A row of its own
- A column 12 cols wide of its own inside that row

How would the complete html look for us to display one project as a card shown above?

```
<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>
```

Now say you have 10 projects to show off! You need to rewrite that messy markup 10 times! Plus 10 different Modals!

I do not have patience to rewrite that html, and 10 Modals, and I know you dont too!

Modals, With Polymer?

Note

> I assume you know
>
> Webcomponent Specifications
>
> Polymer from Google

What if, we could just have one custom element, say

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

and it magically did everything that messy Modal markup could?

Tempting eh!

How do you specify which Modal belongs to which project?

Simple! just write

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

So that reserves the markup with an id of “Project-cats” for the project on cats for example.

How do you write what goes into the modal? simple! Just write your normal markup, wrapped, within the custom tag “”

Like so:

```
<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>
```

Ain't that simple enough?

And If you are wondering How the complete markup look? Including the share and the fork links, see below.

```
<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>
```

Much better than rewriting the complete div soup eh!

Of course you could shorten it further, and make it completely composible by abstracting the hero image implementation inside the custom element, but let us hold off on that, as a later concern.

Reusable Toolbar with Modal | Share | Fork icons - Coding the element

So far, we defined how easy it is to write a custom element that hides the messy html behind it and gives the user, an easy to write and brief markup.

Time to code it!

Our custom element, to display the bar below the hero image should

- Accept a Link to be shared

- Accept a Link to the Repo to be forked
- Accept a modalId to differentiate between other Modals.
- Accept a Title for the Modal

Register the tool bar custom element with Modal | Share | Fork icons

Let us register our custom element, that accepts the above list of properties.

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

Very nice! Next, add the Html we need.

First up, The icons for Modal/Share/Fork need to be added.A simple ul will do.Also,we'll be using, Polymer's iron icons for displaying Modal/Share/Fork icons.

Note: Iron Icons from Polymer

> Install the icon sets like so, in the root of your project.
>
> ```
> bash $ bower install --save PolymerElements/iron-icon
>
> bash $ bower install --save PolymerElements/iron-icons
> ```

Next, Include the installed Polymer elements in the custom element declaration, like so

> Note:
>
> For all practical purposes, our custom element, will be lodged inside
>
> ***Project-root/elements***

```
<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">
```

Very Nice! If you are wondering how to use iron icons, see the note below.

Note: Iron Icon usage and Grammar

> If we need to use the info icon for our Modal Popup, we write the markup like so:
>
> `<iron-icon icon="icons:info"></iron-icon>`
>
> If we want the social networking share icon, then the icon attribute above, becomes
>
> icon="**social:share**"
>
> So the grammar meaning, I need the iconset from "SOCIAL" and I want the SHARE icon from that set.
>
> You can alternatively, use font-awesome , in which case, you can use,
>
> choose what you deem best fit.

Once the includes are done,we write the template for our custom element. We will be using one way binders for data in our custom element's markup.We bind whatever is passed as attributes by the user, to their corresponding properties of our element, while embarking on our element's registration call.

So the html blueprint for our custom element is:

```
<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>
```

Cool! So the tool bar has its templated ul , which lists the three links

- info icon – Modal Popup
- share this icon – share on google
- Fork this icon – Fork the repo

Also, we have bound the attributes the user writes, to the respective properties.

Note: Making the Modal ID unique

> We have addressed uniqueness of the info icon, by specifying the id attribute like so
>
> ```
> id="anchor-for-[[modalId]]" onclick="clickHandler(event)"></iron-icon>
> ```
>
> So each Project, will have a unique anchor id.
>
> i.e,
>
> If the user passes “project-cats” as modal-id, in the call to our custom element, the anchor id would be “anchor-for-project-cats”
>
> If the user adds another project, and passes “project-puppy” as the modal-id attribute, the anchor id would be anchor-for-project-puppy
>
> and so on.
>
> But How do we ensure, each info icon is mapped to its own Modal?
>
> The “**iron icon**“ for **info**, should be assigned a “***data-dialog***” attribute, which needs to be unique as well.
>
> The data-dialog attribute, is similar to the data-target attribute in the bootstrap modal.It needs to be unique for every new Project.
>
> We leave it to the user, to maintain uniqueness. So the user can not have the same modal-id attribute for different `<tool-bar>` calls.

So as the note above states, we need to map each Project’s info icon, to a data-dialog.

Let us add that functionality.

We use the ready method.It is very similar to Jquery’s

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

.It is invoked, when the custom element , is finished loading.

Add this function, after the properties object,in the Polymer call for registration.

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

Very Cool! So now we have

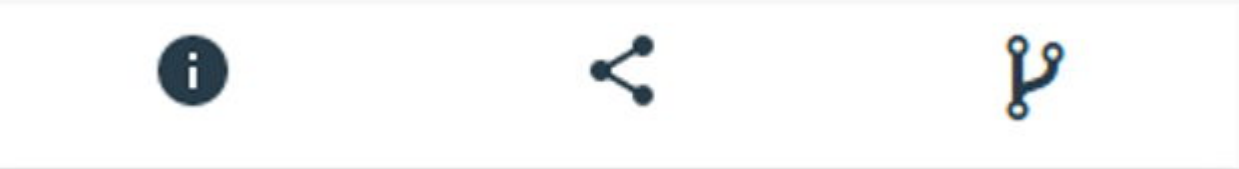
- A custom element registered as a Polymer element
- Modal | Share | Fork icons added to the custom element

A Modal is attached to the info iron-icon,with the id “anchor-for-[Whatever is passed as modal-id]”. That is courtesy the ready method of our element.

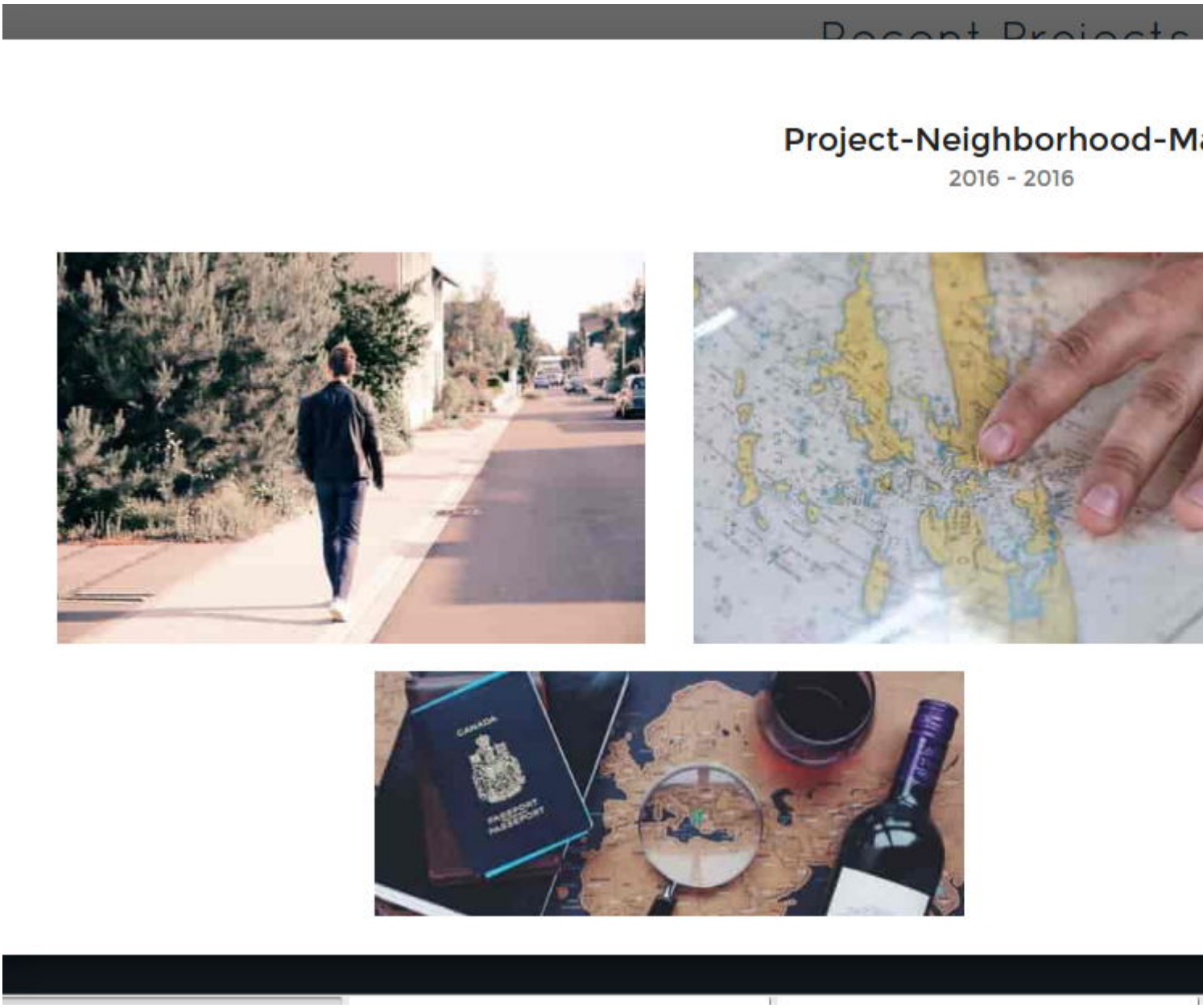
So, so far, writing

```
<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>
```

Produces this:



But when the user clicks on that info icon, something like This should happen



and our goal was to wrap this complete description of the project, within the `<tool-bar>` tag on the users page.

Remember Shadow DOM? We need to have a container in our custom element, that provides the shadow DOM for our Modal's contents.The content itself, is wrapped inside a `<content>` tag inside the custom element's shadowDOM.

For the container that hosts the Project's description, we use the "**paper-dialog**" Polymer element. Paper-dialog, is assigned as id, whatever is passed as the modal-id in the declaration on the user's page.

So, add this markup, after the ***ul*** in the tool-bar custom element's DOM

```
<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>
```

Note: Paper-dialog-scrollable

> We use ***Paper-dialog-scrollable***, to wrap our Project's description, so that, Any lengthy descriptions, do not overflow.

The last bit, is to add the onclick functionality for when the modal should open, on click of the info icon.

When a click happens, as is normal, an event is triggered, which we pass to our clickHandler function like so:

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

So Once the clickHandler function gets the event passed to it,

- It needs to gather, which element triggered the click,
- It needs to gather, the data-dialog attribute of that element, to ascertain which modal to open(Remember each project has its own modal id?)
- It then needs to call the open method, for exactly that modal

So, the function is like so:

Add this, before the element's registration call.This is not a part of the element, but is scoped to within the element.

```
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();
    }
}
```

So, with that, we have completed coding the custom element, `<tool-bar>`.

The complete picture. Custom Element

```
<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();
               }
```

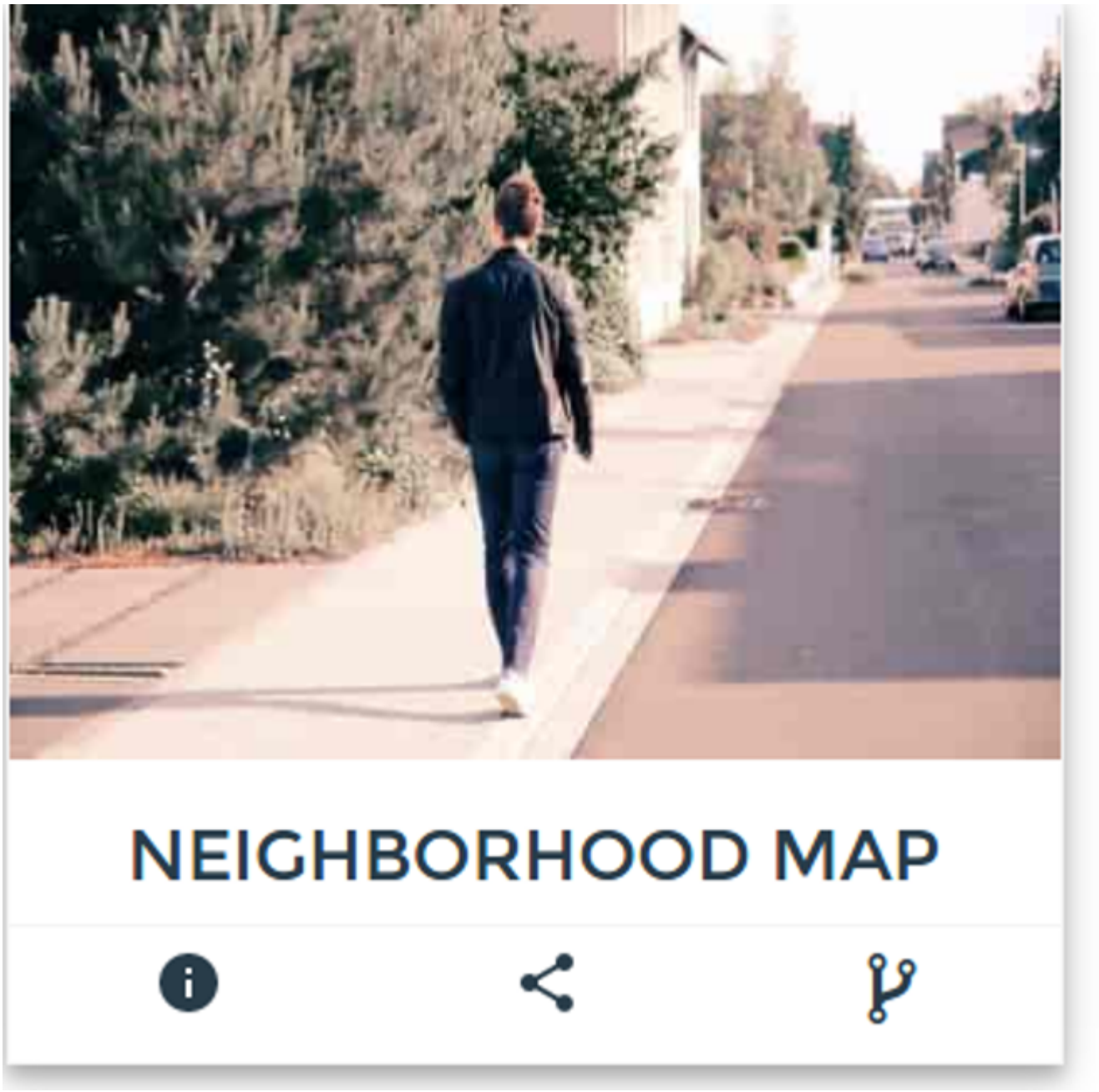
```
          }
      Polymer({
             is: "tool-bar",
            properties: {
                  link2Fork:{
                      type:String,
                      value:""
                   },
                 link2Share: {
                       type: String,
                       value: ""
                   },
                 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>
```

Using the custom element with Modal | Share | Fork icons

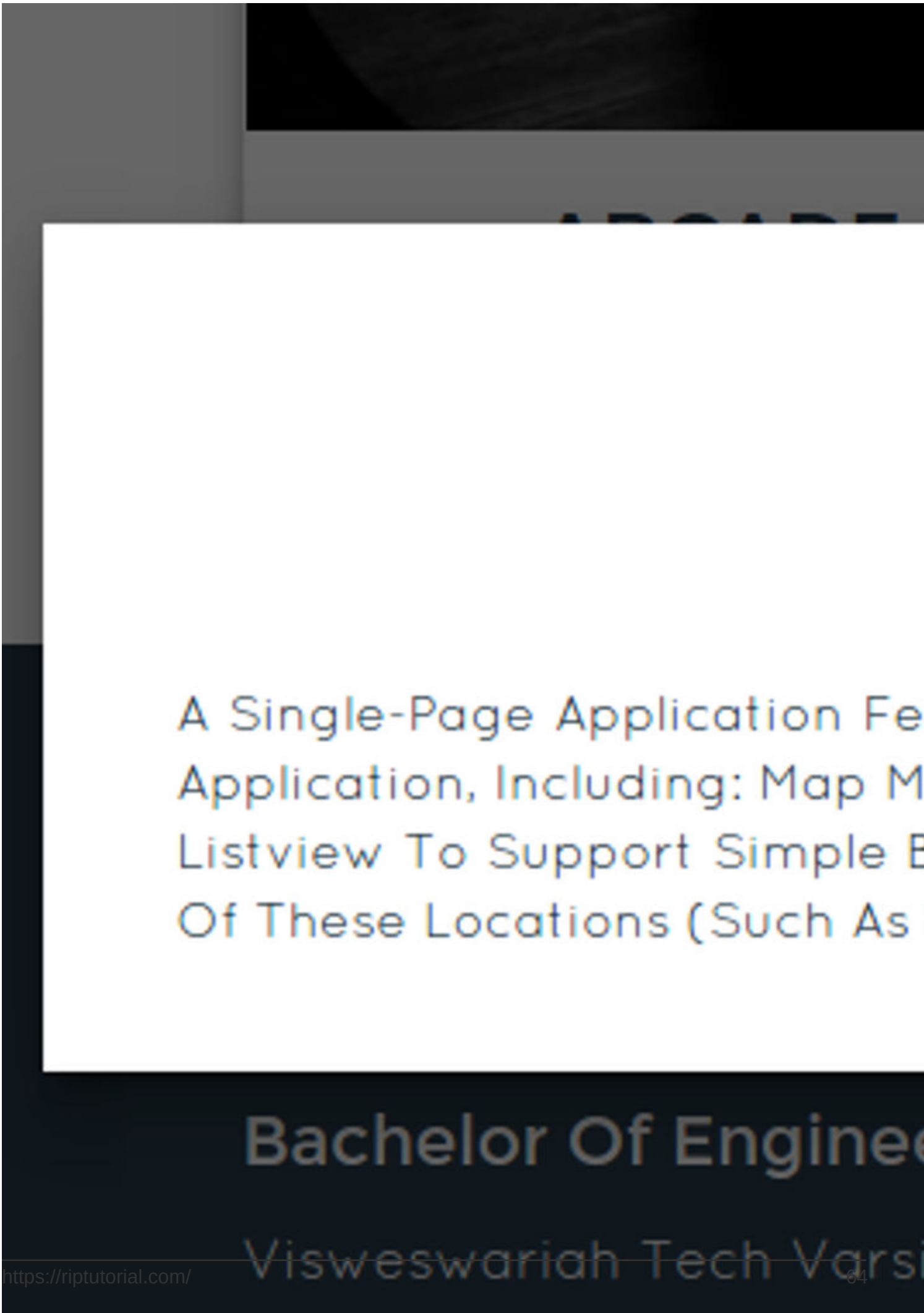
On whichever page you want to display your product / project portfolio, invoke the custom element like so:

```
<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>
```

and you get this as preview



And when you click on info, you get this


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/polymer/topic/7244/reusable-modals-with-polymer

Chapter 11: SUPER-Optimising for production

Introduction

Once your project is done, you are left to wonder how will you upload those load of a 100 HTML imports on your web server and even if you do that, how much hours your site is going to take to load for a single client. In this topic, you'll see how to convert the development mess into refined single html and js files.

Syntax

- npm install: saves packages using Node.js Package Manager
- npm install -g: Saves packages from npm as global (Useful for Command Line interface packages)
- cd: Sets the focus of Command Line to a specific library in which processes can be done
- vulcanize: Crunches down all the HTML imports to a single file
- Crisper: Converts Inline JS in HTML file to external JS file

Examples

Installing all the required tools

Run the following command one by one:

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

Use

- Vulcanize: Crunches all the HTML import files into a single file
- Crisper: Extracts inline js to its own file

Note: Ubuntu users may need to prefix the above command with `sudo`.

Putting it all together

Put all the html imports in your files in a single file `elements.html`. Don't worry about a file being imported more than once, it'll be crunched down to a single import.

Running vulcanize and crisper

on your `elements.html` file, run the following commands:

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

Vulcanize retrieved source code of all the imports, then replaced imports by their source code. Crisper took all the js out of the elements.vulc.html file, put it in single build.js file, set a `script` tag referring the `build.js` file in the `build.html` file

Minifying the files

- Open HTML minifier
- Open `build.html`
- Copy all its code
- In the HTML minfier's first textarea, paste the code you copied from `build.html`
- Click **Minify** button
- In the second textarea, minified code will appear. Copy that
- Create a `build.min.html` file and paste all your copied code in it
- Open JSCompress
- Select 2nd tab i.e. **Upload JavaScript Files**
- Click on `Choose Files`
- Select `build.js` file
- Click on *Compress* button
- Download the file as `build.js` in the same directory as of build.min.html

importing build.min.html

Remove all previous imports from those HTmL files from which you copy-pasted the imports. Replace imports with

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

Read SUPER-Optimising for production online: https://riptutorial.com/polymer/topic/9336/super-optimising-for-production

Chapter 12: Unit Testing

Remarks

Web Component Tester - the tool for unit testing apps built with Polymer. You get a browser-based testing environment, configured out of the box with mocha, chai, async, lodash, sinon & sinon-chai, test-fixture, accessibility-developer-tools. WCT will run your tests against whatever browsers you have installed locally, or remotely via Sauce Labs.

Examples

Simple example using web-component-tester

installing

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

setting up

wct.conf.js

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

running

```
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);
    }

});
```

Read Unit Testing online: https://riptutorial.com/polymer/topic/3898/unit-testing

Chapter 13: Using external Javascript Libraries with Polymer

Introduction

Since all Web Components must be self-contained, including all their dependencies, duplicated dependency import would quickly become an issue with script includes. Thus, Web Components (and, by extension, Polymer) use W3C HTML imports for managing component dependencies. These imported HTML files can be cached by the browser and will only be loaded once.

Most external libraries are not yet prepared for HTML imports. Fortunately creating the necessary HTML wrapper is quick and straight forward.

Parameters

Parameter	Description
this.resolveUrl('../libraries/turf.js')	Resolves the location of your library
function() {this.doSomething(argument, anotherArgument);}.bind(this));	Callback. This example uses a closure to recurse this.doSomething()

Remarks

In this example, the library used in the component is installed with the package manager bower. This allows for easy distribution of a library-dependent component. If the library you wish to use is not distributed via a package manager, it can still be loaded the same way but your component will be require more effort to be used by others.

The lazy loading example uses a simple string for the library path. If one wished to avoid magic string constants, paths could be loaded using iron-ajax from a JSON file into an object and passed between components if needed.

Examples

Import a static HTML file

1. Create an HTML file (in this example `libraries/turf.html`) with the library you want to load:

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

2. Import the HTML file (`libraries/turf.html`) in your component with the rest of your imports:

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

3. Invoke your library (example usage):

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

Lazy Loading

1. Create an HTML file (in this example `libraries/turf.html`) with the library you want to load:

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

2. Import and use your library when needed:

```
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]);
}
```

Read Using external Javascript Libraries with Polymer online:
https://riptutorial.com/polymer/topic/6034/using-external-javascript-libraries-with-polymer

Credits

S. No	Chapters	Contributors
1	Getting started with polymer	Community, Fuzzical Logic, fybw id, Keith, Marc, Randy Askin, tony19, Ümit
2	Creating app using Polymer Starter Kit	fybw id, Puru Vijay
3	Debugging	willsquire
4	Event Handling	a1626, Ümit
5	Example of Polymer toggleAttribute	a1626
6	Google Map Mark With Built in Cache	Schrodinger's cat
7	iron-data-table	Mowzer
8	Looping, the template dom-repeat.	Jp_
9	Polymer Cheat Sheet	Josef Ježek
10	Reusable Modals with Polymer	Schrodinger's cat
11	SUPER-Optimising for production	Puru Vijay
12	Unit Testing	kashesandr, Marc
13	Using external Javascript Libraries with Polymer	antibrian, craPkit