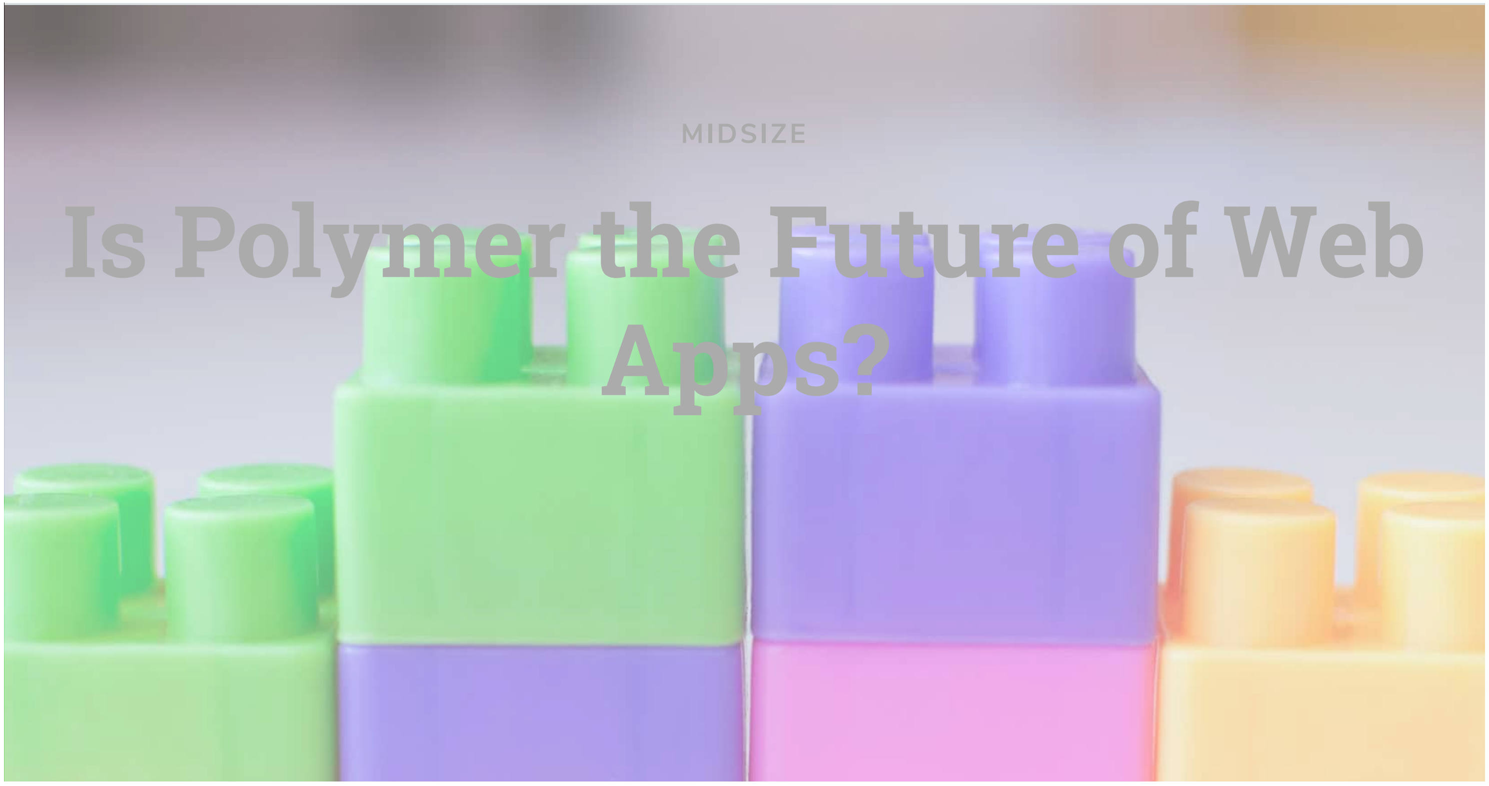


Polymer vs Angular vs React: Using Web Components to Evolve the Experience

October 10, 2016 by Dimitar Danailov

DEVELOPMENT WEB

Most developers and users won't argue building on and maximizing the experiential capabilities of the modern web has been a slow process. When scoping a development project, clients have historically asked for native solutions citing better, more reliable performance. All that is changing, and Polymer, the open-source JavaScript library created by front-end developers working with Chrome, is making it possible.

Progressive Web Apps: Next Generation Development

Download Our Guide

What Is Polymer, and Why Do Web Components Matter?

Polymer, or the Polymer project, is the first library that gives developers the necessary tools and definitions to build an application via web components. Polymer benefits users and businesses. Due to the ease-of-use web components offer, developers can increase their productivity while building applications that boast better support and increased longevity.

Why Developers Like Polymer

Polymer uses open web technologies and new web standards.

Polymer supports shadow and shady Document Object Model (DOM).

The Polymer DOM layer is closest to the Native JavaScript layer. (It's closest to native DOM APIs, meaning, there is no barrier for developers who already know them.)

Onboarding is much faster.

Connection with third party libraries is very easy. (For our project, establishing a connection between Polymer and d3js / Mathjax was very important.

Arguably, it's the best JavaScript library when it comes to Progressive Web Apps and lazy loading.

Important Note: Angular Version 2 Has a Similar Functionality

Java

```
import {RouterModule} from '@angular/router'
import {NgModule} from '@angular/core'

@NgModule({
  declarations: [ MyComponent, MyHomeRoute ],
  bootstrap: [ MyComponent ],
  imports: [
    RouterModule.forRoot([
      { path: 'home', component: MyHomeRoute },
      { path: 'lazy', loadChildren: './my-lazy-module' }
```

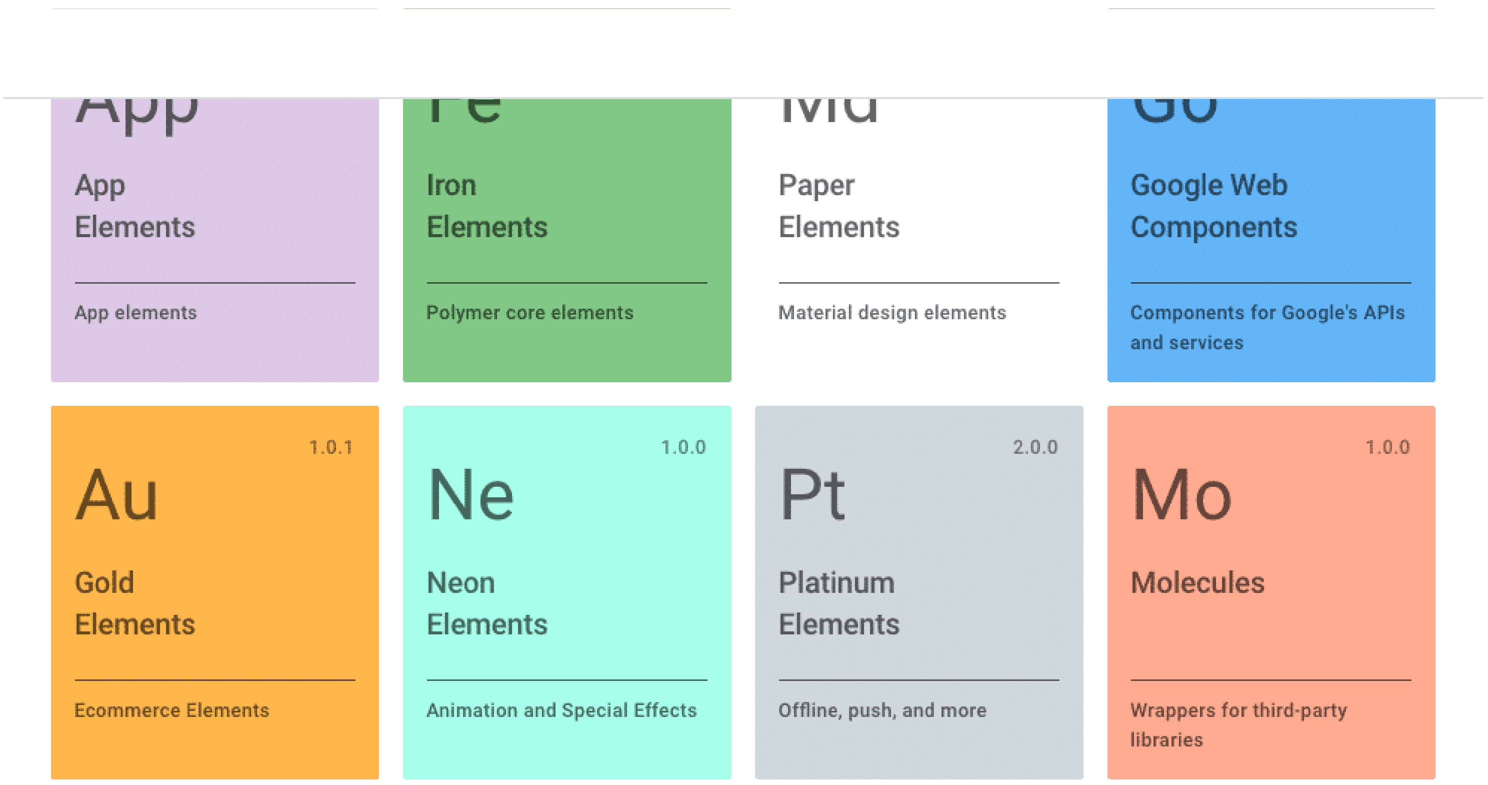
Article: Angular 2 RC5 – NgModules, Lazy Loading and AoT compilation

Understanding the Polymer Catalog

When I was little, I loved to play with Legos. I remember it was very easy to build different houses, vehicles, farms, etc. with the same “components” / pieces. “Web components” and “Lego components” have similar behavior: Every component could live on its own and is reusable. One big component has different small subcomponents, just like every Lego house has windows, doors, rooms, etc.

Like Lego, Polymer has a vast supply of ready to use web components. The Polymer catalog has the following sections:

- App elements (App elements)
- Iron elements (Core elements)
- Paper elements (Material design elements)
- Google web components (Components for Google’s Apis and services)
- Gold elements (e-commerce elements)
- Neon elements (Animation and special effects)
- Platinum elements (Offline, push)
- Molecules (Wrapper for third – party libraries)



Source: https://elements.polymer-project.org

Polymer Sample Build

Let's start with something very simple, a blog with a few subcomponents:

XHTML

```xhtml
<blog-app>
  <!-- Sub component -->
  <toolbar>
    Categories:
    <nav>
        <li>
            <a data-page="art" href="#/art/list">Art</a>
        <a data-page="film" href="#/film/list">Film</a>
         <a data-page="photo" href="#/photo/list">Photo</a>
        </li>
    </nav>
  </toolbar>
  <!-- Sub Component -->
  <blog-pages>
    <!-- First blog page -->
    <blog-page></blog-page>
    <!-- Second blog page -->
    <blog-page></blog-page>
    <!-- Third blog page -->
    <blog-page></blog-page>
  </blog-pages>
</blog-app>
```

library (webcomponentsjs) and several polymer elements.

JavaScript

```
{
 "name": "blog-page",
 "dependencies": {
   "webcomponentsjs": "^0.7.22",
   "app-route": "PolymerElements/app-route#^0.9.2",
   "iron-selector": "PolymerElements/iron-selector#^1.5.2",
   "iron-ajax": "PolymerElements/iron-ajax#^1.2.0",
   "paper-spinner": "PolymerElements/paper-spinner#^1.2.0",
   "iron-image": "PolymerElements/iron-image#^1.2.3"
 }
}
```

Why Polymer Worked: Modernizing a Legacy System

Several years ago, we began work on a complex legacy system with an architecture built from .Net web forms, jQuery and MSSQL, Server 2012.

The System Had Several Challenges:

Difficulty supporting and debugging

Painful process to fix and add new features

Insufficient documentation and convoluted source code

Difficulty using with mobile devices, tablets and chromebooks

Memory leaks

Poor performance in offline mode

We needed a mobile-first system with a reliable offline mode.

Additional Needs:

The documentation should be improved.

Onboarding of the new members should be much easier.

After several meetings with product owners, the main skeleton of the application was ready:

The architecture should follow a microservices pattern.

Rest API framework for each services will be: ASP.NET MVC Core.

A good single page application framework was required.

We considered several good single page frameworks including: Ember.js, Angular, React and Polymer.

Ember.js

Ember.js is a very good framework if the REST API framework is Ruby on Rails and Ember doesn't support Shadow DOM.

Angular

Angular does not support shadow and shady DOM was a major stumbling block. Additionally, migration between Angular 1+ and Angular 2.0 would have been painful requiring additional resources and time.

Deep Dive: React vs Polymer

Considering the two remaining options, both had distinct advantages. React boasted a much larger community of developers than Polymer. After several meetings, our team chose Polymer. The reasons were:

1. The Polymer Project has better support for web components compared to React. (For more about React and web components, please to visit: React and Web Components.)

2. Version 1.0+ is stable version, and the community is growing very fast.

3. The integration of the new modules is not a painful task.

4. Polymer has better offline modules than React.

5. React license
React is an open source library under BSD, but has an additional patent clause: Additional Grant of Patent Rights Version 2 "**If you are using or considering using React in a project, you might want to consult a**

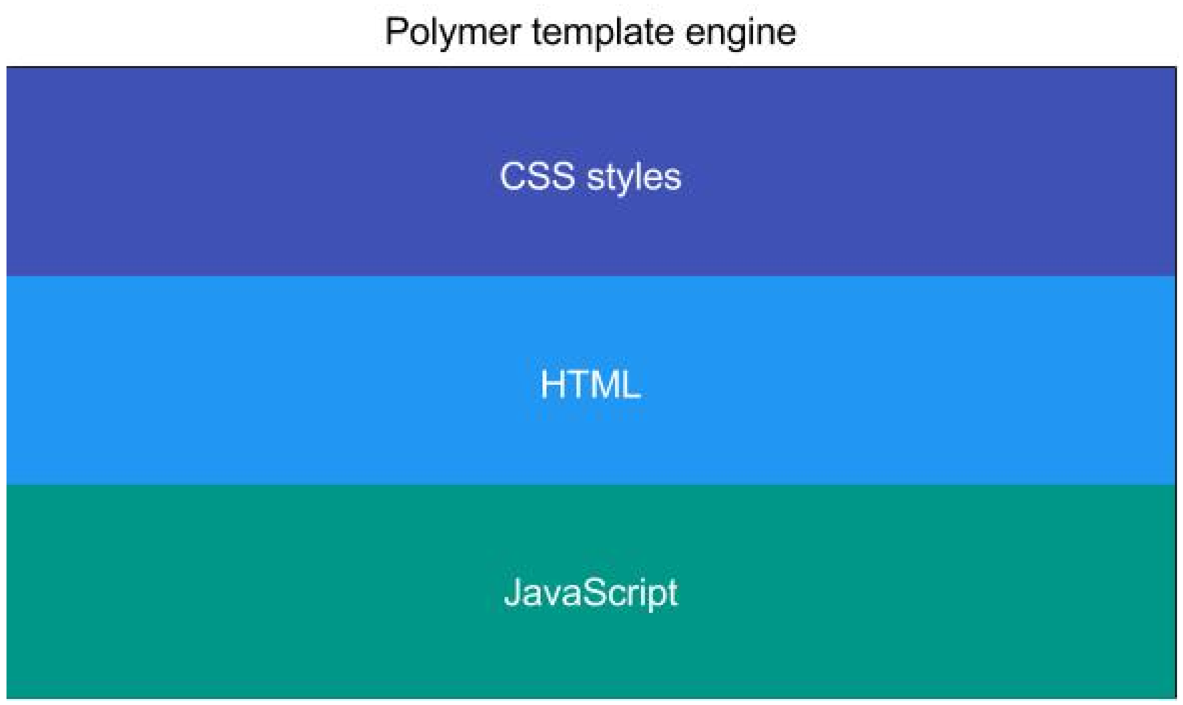
Facebook. If you do take legal actions or in other ways challenge Facebook, your license to use React is immediately revoked. Your license is also revoked if you have any legal disputes if you have legal disputes with any other company using React. **This is the reason why both Google and Microsoft employees are not allowed to use React.js in their work** — according to Rob Eisenberg, creator of the Aurelia framework and a former member of the Angular 2 development team." — http://react-etc.net

6. Our system will use different third-party JavaScript libraries and service workers. We think this is much easier with Polymer.

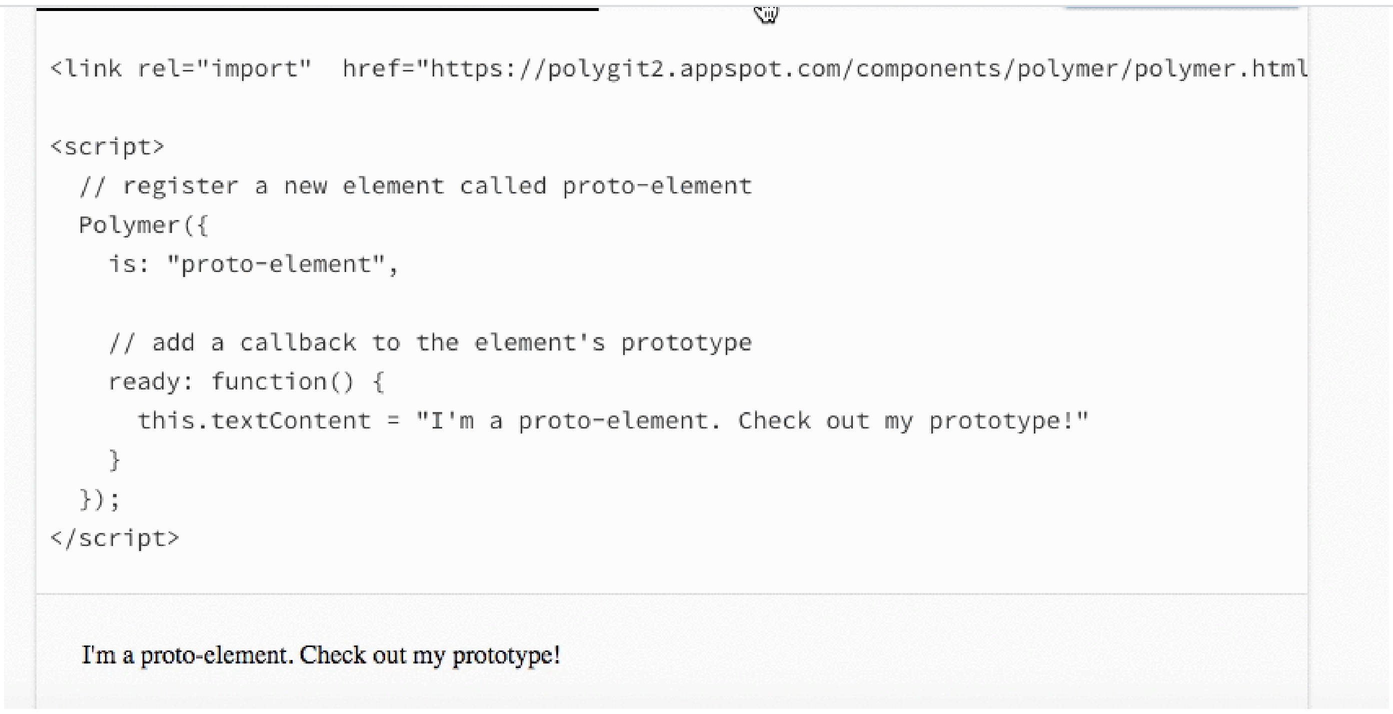
8 Advantages of Using Polymer vs Angular / React

Advantage 1: Template Engine

Using Polymer, developers can create a custom element in a single html file which is more intuitive. It is easy to understand the component without opening several files at once:



Click to see an example like the below.



```
<link rel="import"  href="https://polygit2.appspot.com/components/polymer/polymer.html

<script>
  // register a new element called proto-element
  Polymer({
    is: "proto-element",

    // add a callback to the element's prototype
    ready: function() {
      this.textContent = "I'm a proto-element. Check out my prototype!"
    }
  });
</script>
```

```
I'm a proto-element. Check out my prototype!
```

Source: Polymer Project

Is This Template Engine Better Rather than Angular (version 1) and React?

Code is much cleaner, and the developer has all definition in one place.

Angular 1 allows inline structure (definition in a single file.) But this structure would be strange for a large application.

In React, the HTML template is declared in component definition. (https://github.com/dimitardanailov/build-a-wiki-with-react-and-firebase/blob/master/src/components/App.js)

My opinion about the template engine is:

React has a better template engine than Angular version 1. (I prefer mixing JavaScript with html rather than html JavaScript.)

natural for every developer.)

Note: Angular version 2 has a similar template but the necessary definitions are:

Controller

Service

HTML file with template definition

External CSS file

Example: https://plnkr.co/edit/ngN888w5uzpjMSu0ceh1?p=preview

```
import { VehicleService } from './vehicle.service';

@Component({
  'selector': 'my-vehicles',
  'templateUrl': 'app/vehicles.component.html',
  'styleUrls': ['./app.characters.component.css'],
  'providers': [VehicleService]
})
export class VehiclesComponent {
  constructor (private _vehicleService: VehicleService) { // Injection
  }
  vehicles = this._vehicleService.getVehicles();
}
```

Polymer Summit 2016 shares the diagram about the using of the HTML, CSS and JavaScript between Polymer, React, Angular and Ember.

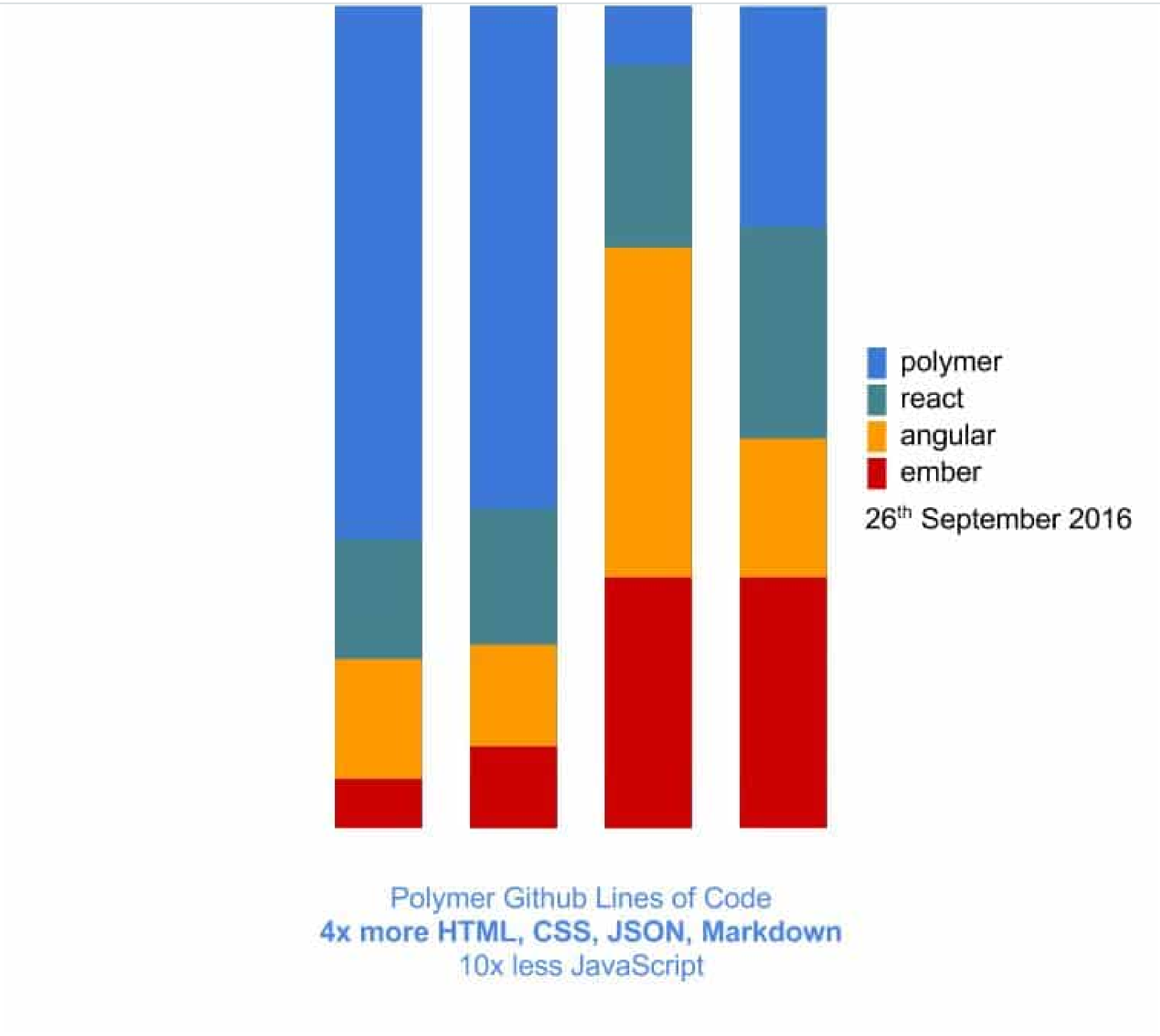


Image Source: Polymer Summit 2016

Advantage 2: Behaviors

Polymer uses behaviors to share code between different components. A Polymer behavior can define:

- Lifecycle callbacks
- Declared properties
- Default attributes

- Event listeners

This concept is similar to:

- Angular 1 Services / Factories
- Angular 1 Directives
- Angular 2 Providers / Classes

Angular version 1 introduces complexities not beneficial for the developers or product owners. For example, Angular 1 doesn't have clear a definition between Factory and Service: AngularJS: Service vs. provider vs. factory. The question is a part from the Stack Overflow wiki.)

When the concepts are not clear, this is a painful for the development team and critical for **the application and deadlines.**

Polymer has a large set of helpful behaviors:

- Polymer.IronA11yKeysBehavior — provides a normalized interface for processing keyboard commands that pertain to WAI-ARIA best practices.
- Polymer.IronCheckedElementBehavior — Use to implement a custom element that has a checked property, which can be used for validation if the element is also required.
- Polymer.IronFitBehavior — fits an element in another element using max-height and max-width, and optionally centers it in the window or another element.
- Polymer.IronMenuBehavior — implements accessible menu behavior.
- Use Polymer.IronOverlayBehavior to implement an element that can be hidden or shown, and displays on top of other content. It includes an optional backdrop, and can be used to implement a variety of UI controls including dialogs and drop downs. Multiple overlays may be displayed at once.
- Polymer.IronRangeBehavior provides the behavior for something with a minimum to maximum range.
- Polymer.IronResizableBehavior is a behavior that can be used in Polymer elements to coordinate the flow of resize events between "resizers" (elements that control the size or hidden state of their children) and

Polymer.IronScrollTargetBehavior allows an element to respond to scroll events from a designated scroll target.

Use Polymer.IronValidatableBehavior to implement an element that validates user input. Use the related Polymer.IronValidatorBehavior to add custom validation logic to an iron-input.

This sets of the behaviors was a very helpful for our team and **reduce** the development time.

Advantage 3: Styling

Polymer supports:

- Mixin declaration (Custom CSS mixin)
- CSS variables

Polymer vs Angular: These features are not supported from Angular or React:

- React: Add support for CSS variables in style attributes
- Angular: Updating a CSS style globally with variable in scope

Our project required different themes with different contrast. If our choice had been Angular or React, this would have opened the door for other issues.

Polymer CSS variables:

XHTML

```
<link rel="import"  href="https://polygit2.appspot.com/components/polymer/polymer.html">
<dom-module id="custom-button">
   <template>
      <!-- scoped CSS for this element -->
      <style>
         :host {
         position: relative;
         display: inline-block;
         width: 32px;
         height: 32px;
         background-color: var(--my-custom-button-color, #ccc);
         cursor: pointer;
         }
      </style>
```

```
        is: "custom-button",
        properties: {
          isActive: {
            type: Boolean,
            value: false
          }
        },

        listeners: {
          'tap': 'changeTheme'
        },

        changeTheme: function() {
          this.isActive = !(this.isActive);

          if (this.isActive) {
            this.customStyle['--my-custom-button-color'] = '#000';
          } else {
            this.customStyle['--my-custom-button-color'] = '#ccc';
          }

          this.updateStyles();
        }
      });
   </script>
</dom-module>
```

Example: https://plnkr.co/edit/eShHm4IukPwZ9Hoc7elJ?p=preview

Advantage 4: Working with Native JavaScript Objects Is Much Easier

The polymer DOM layer is very different from jQuery DOM layer or Angular (version 1) DOM layer. (jQuery has a jquery element, and Angular has an angular element.)

The polymer DOM layer is closest to the DOM API layer.

Example with querySelectorAll() (querySelectorAll: Returns a list of the elements within the document (using depth-first pre-order traversal of the document's nodes) that match the specified group of selectors. The object returned is a NodeList.)

```
var childnodes = Polymer.dom(element).querySelectorAll('my-custom-element');
```

— And polymer has several additional methods for selecting elements. For convenience, several utility methods are available on the Polymer element prototype:

getEffectiveChildren(). Returns a list of effective child elements for this element.

queryEffectiveChildren(selector). Returns the first effective child that matches selector.

queryAllEffectiveChildren(selector). Returns a list of effective children that match selector.

Source: https://www.polymer-project.org/1.0/docs/devguide/local-dom

Example:

```
var effectiveChildren = Polymer.dom(element).getEffectiveChildNodes();
```

Data binding is not a problem for Angular or React, but the story is different for DOM manipulation and element selection.

Polymer builds a map / collection with “quick” nodes (if an element has id attribute). A node specified in the element’s template with an id is stored on the this.$ hash by id.

```
Hello World from <span id="name"></span>!

<script>
Polymer({

  is: 'x-custom',

  ready: function() {
    this.$.name.textContent = this.tagName;
  }

});

</script>
```

This is huge advantage if you are a JS / DOM developer. Angular strives to provide another abstraction and dictates the view from the model / state and thus implicitly makes it harder to work directly with the DOM

I’d like to share a thought from my colleague:

“Polymer DOM manipulations are much more intuitive for frontend developers as they resemble the long ago established practices by not abstracting the DOM but instead embracing it.”

Angular / React or jQuery has more guides and examples compared to the Polymer Project. My opinion for the Polymer documentation: Polymer has a much better structure of the documentation and examples from Angular (version 1) or React. I had conversations with junior and middle-level developers about Polymer vs Angular as it related to onboarding. The overall impression is:

1. Polymer documentation has a better structure and examples than Angular or React.

2. Polycasts | Web Shows – Google Developers is a great developer show. Angular and React don't have a developer cast.

3. The Polymer catalog helps with: Which are the most common scenarios for each web component.

Example: Our project doesn't use Material design (Paper Elements) or payments (Gold elements). It uses only Iron elements(Polymer Core elements). Developers only need to read documentation about core elements.

Advantage 6: Connecting with Third Party Libraries Is Very Easy

For us, it was incredibly important for the application being to have a good bridge / connection between d3js, Mathjax, text to speech (external modules) and other internal components. I mentioned early, Polymer doesn't have an extra layer and sharing data between components, and d3 or Mathjax is very easy.

Example 1 application with MathJax and Polymer.

Advantage 7: The Best JavaScript Library When It Comes to Progressive Web Apps and Lazy Loading

"Three cutting-edge new features of the web platform—Web Components, HTTP/2 + Server Push, and Service Worker—all work seamlessly together to provide a totally new and amazingly efficient way to deliver applications to users."

Apps.

The Polymer App Toolbox and Polymer CLI make it easy to build progressive application. This technique has a name: “PRPL Pattern“:

- **Push** components critical for initial route
- **Render** the initial route ASAP
- **Pre-cache** components for remaining routes
- **Lazy-load** and create next routes on-demand

The blog app I created is built using the toolbox and served using the PRPL pattern and multi-view progressive web app.

Advantage 8: Very Large Set of Custom Elements for Free

The Polymer Project has a large set of custom elements:

- https://vaadin.com/elements
- https://customelements.io/
- https://beta.webcomponents.org presented at Polymer Summit 2016

Of course Angular, jQuery and React have a lot of third-party libraries / widgets but when we talk about the data grids, custom input elements, uploaders, icons, etc. in many cases these widgets are not free or don’t have a commercial license.

This is why many companies invest in interoperability layer: https://vaadin.com/blog/-/blogs/using-polymer-components-in-angular-2

Polymer Disadvantages

isn't an easy task. React and Angular (1 and 2) have a better community, and it is much easier hiring an engineer with an Angular or React background. This is a huge advantage.

For us, the main Polymer disadvantages are:

- Some components don't have good documentation and examples.
- Browser supports (Custom Elements v0 is supported only by Safari and Google) — all browser vendors are working on web components supports (Native support).
- If time is limited and your development team hasn't have experience with web components.

Looking Ahead: The Future of Polymer

The Google team is already working on the next version (Polymer version 2). The current version (Polymer version 1) is using v0 Custom Elements. v0 is supported now by Google and Safari only. Custom Elements v1 will be supported by a variety of other browsers including: Microsoft, Opera, Firefox. Developers and businesses can look forward to better performance and the reduced need for Polyfills. This step is huge for the Polymer community.

Example: Custom Elements

```
// https://w3c.github.io/webcomponents/spec/custom/
// https://developer.mozilla.org/en-US/docs/Web/Web_Components/Custom_Elements/Custom_Elemen
// https://www.html5rocks.com/en/tutorials/webcomponents/customelements/

  class RadioButton extends HTMLElement {
  constructor() {
    super();
  }

  connectedCallback() {
    this.setAttribute('role', 'radio');
    this.setAttribute('tabindex', 0);
    this.setAttribute('aria-checked', false);
  }
}

// define method has three properties: 1. tag name, 2. Class Name, 3. options
window.customElements.define('radio-button', RadioButton, { 'extends': 'radio' });
```

components v1 should be as smooth as possible.

“In order to make it as easy as possible for existing Polymer components to take advantage of the v1 Web Component API’s with minimal pain, we want the 1.x => 2.0 transition to be as smooth as possible.” – Polymer team

Best Use Cases for Polymer

Of course, a silver bullet doesn’t exist. However, I would recommend the use of Polymer in the following cases:

- Big single page applications
- Completed projects where communication is required between parent – child component and child – parent component
- Applications with many different widgets / components
- Your application should support shadow DOM
- Your application should contain different third party libraries
- Keyboard accessibility

Other Notes

Increased blog app documentation can be found here.

View the work.

Demo.

DIMITAR DANAILOV

Software Architect & Technology Futurist, Dimitar Danailov isn't content to just build and architect software, he constantly seeks to scale and support it more effectively. Specializing in enterprise software development, Dimitar has developed technologies and platforms for the healthcare and education sectors. Industry interests include big data (Hadoop), microservices, cloud computing, .Net Core and Java, devops, and web components(the Polymer Project).

When he isn't deep in the architecture and management of data, Dimitar keeps active traveling and playing a variety of sports from football to badminton. Dimitar studied Hadoop platform and application framework along with big data analytics at the University San Diego. He has received additional training in MongoDB and has volunteered at Open Fest 2016, Rails Girls Sofia 7, Startup Weekend Varna, Varna Cloud Conf Google Developer Group Varna, HackConf and Azure Bootcamp.

You might also like:

BLOG

STARTUP

Exploring ASP.NET Core 2 and Angular 5 Applications with Docker

READ POST

CONTACT US

Subscribe

3036 Hennepin Ave.

Minneapolis, MN 55408

+1.612.823.4000

BULGARIA

67 Prof. Tsvetan Lazarov Blvd.

Sofia 1592, Bulgaria

+359.2.862.2632

SWEDEN

Pepparedsleden 1

431 50 Mölndal, Sweden

+46.70.646.6067

Cookie Policy Privacy Policy Terms of Service