## Overview

This spike is to determine if we can use the new *Element* construct of Angular 6 to build up applications using a Web Component (an HTML Standard) that is written in Angular 6.  
The element is compiled down to an HTML standard Web Component usable pretty much anywhere HTML is used.

## What is a MicroApp ?

MicroApps : Vertical Services – have a Frontend and a Backend.

This spike focuses on the Frontend.

The Facilitator platform can be a collection of vertical services, or *MicroApps*. The ability to build an instance of Facilitator by defining the specific set of services a client needs, maximizes the functionality, configurability and maintenance of the Facilitator offering.

How to configure is another spike and design decision (possibilities, of NPM, Build configuration, “Control Panel” has been talked about, config file, database, … possibly others).

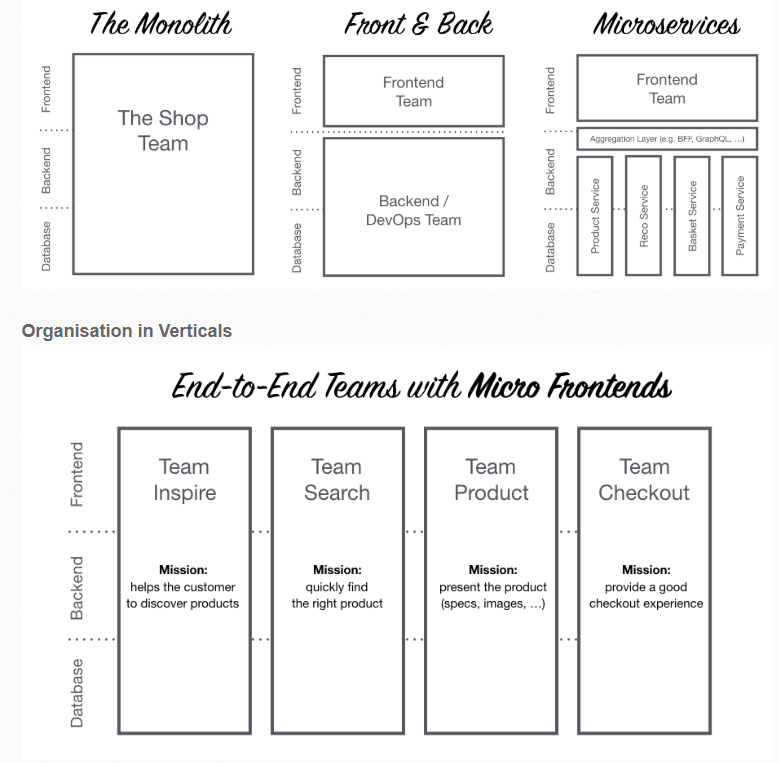
Vertical services (MicroApps) can be worked on by different teams.  
In order to fully support a MicroApps we want to drive towards the bottom half of this diagram (End-to-End Teams with Micro Frontends)

Figure 1 from website: https://micro-frontends.org/

**Thesis:**  Angular 6 (ng6) is the correct framework for the above Frontend components work because:

1. The ng6 construct, called element, is suited to prescriptively build the pluggable Frontend layer
2. The building and maintenance of the MicroApp (both Frontend and Backend) within a Repository will simplify work on a particular vertical
3. Web standards are good for interconnect ability of the vertical service, it can be put into an HTML page, or even a React page.
4. Team competencies mean a quicker refactor of existing codebase (from AngularJS 1.5) to Angular 6 as many concepts of the framework are similar, which simple to transition to syntactic differences only.
5. The Angular payload can be reduced, by the Ivy compiler so components don’t impact page load times
6. Design considerations can be understood for getting Guidelines put together for architecting MicroApps

**Findings:**

Modern SPA (Single Page Apps) have become monolithic over time[[1]](#footnote-1). This makes them more difficult to maintain. We have experienced this with Facilitator Core.

## Elements

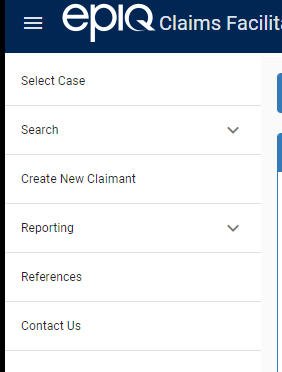
The ng6 Element is a new construct introduced by the Google team in Angular 6. It addresses the issues developers are facing with increasingly complex SPA’s that tend towards monolithic (single code base) design which evolves into greater and greater complexity. The Google Angular team recognized they were doing 90% of the componentization already needed, but not enough to expose it outside of Angular.

An element is an abstraction layer, and a good one for encapsulating what this paper calls a MicroApp’s UI (or Frontend). The Element is designed it to be a compiled component, that can be a) self-bootstrapping, and b) dropped into any HTML page, another Angular “main App”, or even a React or Vue.js project. Other important points about ng6 elements:

* They are Self Bootstrapping.
* They actually host the Angular Component inside a Custom Element.
* It bridges between the DOM APIs and Angular Components.
* Anyone can use it without having the knowledge of how Angular works.

*Angular elements are ordinary Angular components packaged as custom elements, a web standard for defining new HTML elements in a framework-agnostic way*

## Testing it out



If I write a simple Menu and treat “Search” like a vertical MicroApp, it will draw into focus the intent of the spike. I will NOT implement any functionality of “Search”, just show in a simple HTML file the left nav menu. Then I will create an ng6 element to contain the Search MicroApp.

The spike is all about integration and possibilities and answering the thesis (prove/disprove) points. I won’t spend time on anything but the new ng6 element code, its syntax, and interaction (not any functional implementation of the MicroApp itself related to “search”)

### Setup ng CLI environment to play with Narrative

1. From cmd line > npm install –g @angular/cli  
   401 Unauthorized: @angular/cli@latest
   1. And getting a 401, at this line:     http fetch GET 401 <https://epiqsystems.pkgs.visualstudio.com/_packaging/Epiq.npm/npm/registry/@angular%2fcli>
   2. Reached out to George Alexander, answers back quickly:
   3. Edited %UserProfile%\.npmrc (delete all lines, so an empty file). Fixed it for command in # 1)

The feed its trying to use is the old VSTS Feed.

There may be a .npmrc file stored with the source.

There may be a .npmrc file stored in “%UserProfile%\.npmrc”

Then I think it maybe the global is  “%programFiles%\nodejs\etc\.npmrc”

The source come from the npmrc file.

You should pull from the npm master feed in artifactory. If the npmrc file is in source control, then it should be removed.

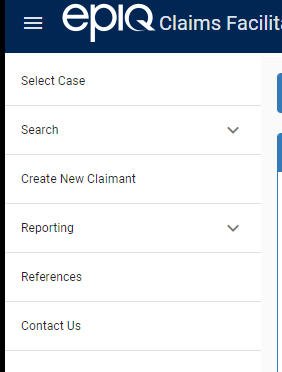
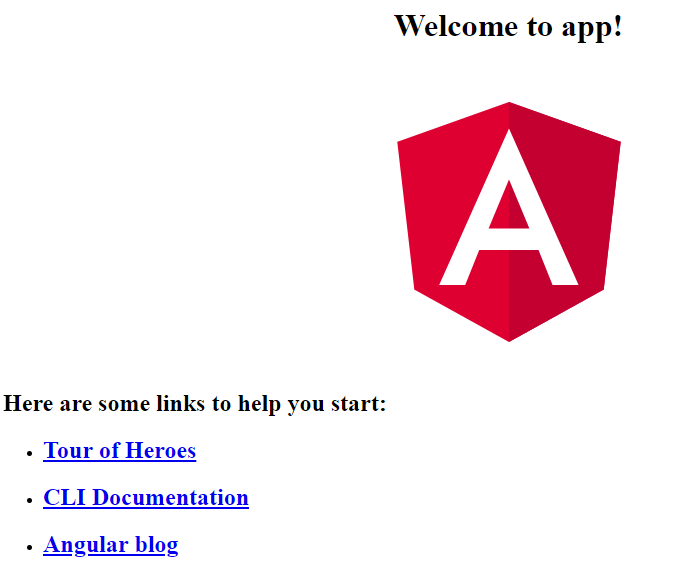
I think there is a way to override the file using the command: <https://docs.npmjs.com/cli/install>

It looks limited to pulling from source out of a public git repo.

* 1. Installed the Angular CLI in global npm repo on my system. Location %appdata%/npm

1. Confirm install: > ng - -version yields Angular CLI 6.0.8, Node 8.9.4
2. Create the ng cli application
   1. > ng new myapp
3. > cd myapp
4. > ng serve
   1. Navigate in browser to http://localhost:4200/

## How to construct spike home page

Not wanting to spend much time on “visual” stylistics, I will covert to something like this:

The only active left nav button will be ‘Search’, and it dynamically load the <search-app></search-app> in the main area of the page (to the right of the left nav).

## Adding Element package to project dir

1. > ng add @angular/elements (from myapp dir)  
   This command adds *polyfill* ( document-register-element.js ) and @angular/elements pkg
2. > npm install –save @angular/material @angular/cdk  
    > npm install –save @angular/animations

## Code modifications to emulate SideNav

1. Glommed code from Facilitator Core (to get terminology and layout of LeftNav)
2. Stripped it of any .ts things (vm.) for now
3. Figured out that AngularJS Material and Angular (ng6) Material – are different libs AND different syntax (example md-list, is now mat-list, etc.)
4. Newer Material for ng docs: <https://material.angular.io/components/sidenav/api>   
   uses the ‘mat’ prefix
5. edited app.module.ts to add:  
   import {

MatSidenavModule,

MatToolbarModule,

MatListModule,

MatCardModule

} from '@angular/material';

import { CUSTOM\_ELEMENTS\_SCHEMA } from '@angular/compiler/src/core';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

MatSidenavModule,

MatToolbarModule,

MatListModule,

MatCardModule

],

providers: [],

bootstrap: [AppComponent],

schemas: [CUSTOM\_ELEMENTS\_SCHEMA]

})  
.  
.  
.  
  
Error in No NgModule metadata found for ‘AppModule’.

1. >npm remove webpack
2. >npm install –save-dev @angular/cli@latest
3. > rmdir node\_modules /s, npm cache clear –force
4. > npm cache verify
5. then rerun > npm install
6. ng build –prod –no-build-optimizer  
   ERROR in ./src/main.ts  
   Module build failed: Error: main.ts is missing from TypeScript compilation. Please make sure it is in your tsconfig via the ‘files’ or ‘include’ property.
7. Included “main.ts” in tsconfig.app.json “files” section

ERROR in src\app\app.module.ts(24,13): Error during template compile of 'AppModule'

Only initialized variables and constants can be referenced in decorators because the value of this variable is needed by the template compiler in 'CUSTOM\_ELEMENTS\_SCHEMA'

'CUSTOM\_ELEMENTS\_SCHEMA' is not initialized at @angular\compiler\src\core.ts(119,22).

1. Fixed, combiled (use #11 always to get proper syntax and other error messages !!!)
2. Add back material design > ng add @angular/material
3. Get latest Mat Design for NG:   
   >ng add @angular/material@6.2.1
4. Ng generate @angular/material:materialNav –-name mat-sidenav
5. OK. 1st Phase complete. Result: ng6 application with LeftNav (simulated) to get the point across. Next, understand completely “element” design and bootstrapping, AND how to launch (load) dynamically on the click event from the LeftNav for the Search. What will come up will be a dummied up “Quick Search” (simulated), proving the componentization works ng6 app launching ng6 microApp.

## Element research

Definition of element *noun*1 Elements are sugar around Angular components, allowing packaging, to use Angular components outside of Angular.   
They conform to a web standard, called *custom elements.*  They are useful for defining new HTML elements in a *framework agnostic way* using Angular 6 (or later).

Because Angular components can contain other angular components, you can build up a subsystem’s UI, called a *microApp*. The microApp is a UI frontend to one, or more, microService backend services.

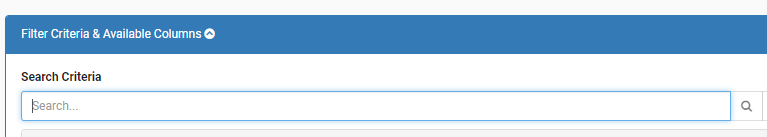
Elements are a Web Platform, browser supported, construct which allows the extend HTML by defining essentially a new HTML tag.   
(Browsers: Chrome, Opera, Safari, IE 11? ).   
Elements are also sometimes referred to as *Web Components* from the HTML side of things.

The browser keeps a registry of Web Components, or Angular Elements, in this spike.

## createCustomElement()

The @angular/elements package exports this API which provides a bridge from Angular’s component interface and change detection to the built-in DOM API.

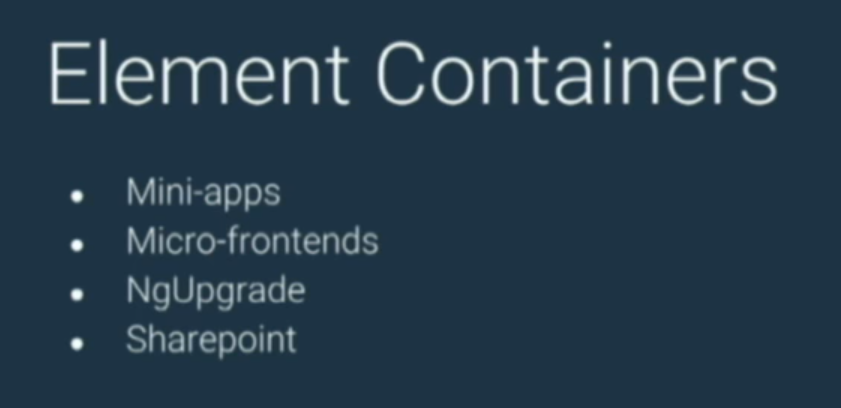
## Simulation of search ui



I’ll construct in a separate directory (at same level as myapp) a microApp called quickSearchApp.  
It will look like the above (minimal components, no functionality tied to back end microService(s) ).  
The point of the spike is the inclusion of the angular element.   
  
This brings up a good point : any microApp should have its own solution. (its own root git).  
Later we will do a spike on packaging them up (likely in npm).

We’ll use a material design look-n-feel of the above.

Education: Watched this video on Elements (:25 min) <https://www.youtube.com/watch?v=Z1gLFPLVJjY&t=4s> 17:34 in talks about our use case #2 microApp use.



Also, use case #3

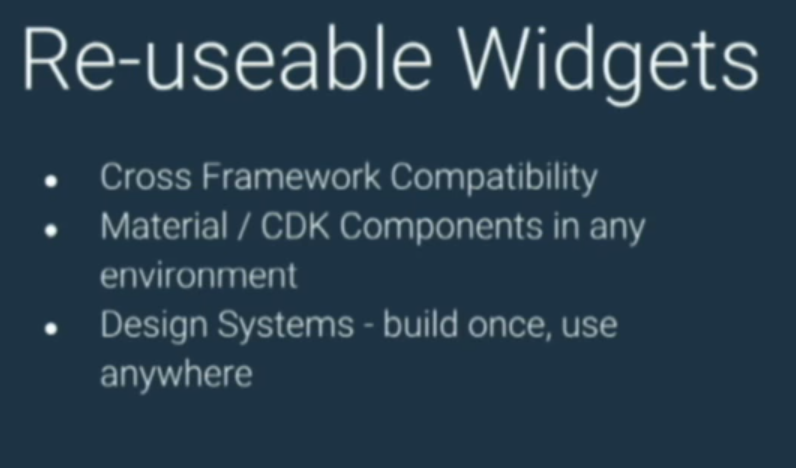


Figure 2 Reusable widgets - example DatePickers

### Narrative

1. > ng generate component cuickSearchApp  
   This command can only be run inside of a CLI project.
2. Create new CLI app
   1. > ng new quickSearchApp
3. > Ng add @angular/elements
4. Add a component (that will be made into an element)  
   > ng g component quickSearchApp --inline-style --inline-template -v Native
5. I then in the quickSearchApp\src\app directory  
   purposefully rename the pattern quickSearchApp.component.ts, etc. and remove .component. so it becomes quickSearchApp.ts  
   Why? Even though an Element IS an ng component that is wrapped to be a Standard, ‘custom HTML Element’ (Web Component) we want to treat the microApp as the shell of possibly may other components. (those will have the .component. in the name when created). This allows us, along with a naming convention   
     
   Naming convention then: the outer SPA app that host the microApps is simply ‘app’ (as is the default in ng), and the microApp name follws the above example.

**Pattern:**

**{microApp-name}App** (example: quickSearchApp)

1. Adjust other files like the microApp.module.ts, ..\main.ts to reflect this subtle but important naming convention change.
2. See source files for correlations, but this is the gist:



1. > nb build
2. Then in .\dist, >source-map-explorer vendor.js to see files in the
3. Bundling – building source files into a bundle is done using broccoli and webpack
   1. Webpack converts commonjs modules to something that browsers understand
   2. Proccoli.js is responsible for running all build tasks even webpack transformations are part of broccoli tasks

## Design Considerations

* Deep initial evaluation is necessary to find logical breaks of verticals. I have no doubt that the Alpha team (Scott, Ron, Richard, Nina, Petar) can help in defining logical breaks of vertical functionality of the Facilitator platform.
* Messaging contract between. I think Pub/Sub messaging as discussed as a necessary cross-cutting concern (globally available service) allows for the messaging between verticals.
* Test strategy: I think more automation of interface testing, for individual vertical’s solutions is in order, the more automated the better.
* Self-contained: any features should be self-contained (not rely on other MicroApps)

## Alternatives

Some other packaging platforms or libraries

1. Ionic (heavy weight, multi-faceted) framework. Larger company buy-in, likely to get package.
2. Svelte. <https://svelte.technology/guide> more singular focus of combining things,  
   seems like Component focused, and tries to eliminate framework bloat at compile time,  
   like a javascript reduction (redux) by compile philosophy. Very new, however.
3. Stencil. Compiler for Custom Web Components. Async rendering by React Fiber.  
   Created by Ionic team, to build faster, more capable components that worked across all major frameworks.
4. SkateJS – a wrapper around Web Components for cross platform use

What they all have in common mostly is the leveraging of Custom Elements, as the direction that HTML and Web standards have gone to combine elements and get away from JS Framework restrictiveness or allegiances.

I think leveraging Angular 6 is fine, for microApps except for the bloat (currently) solvable in ng7 (by Ivy compilation) and potentially the cutting edge, not being IE ready (need to test it out).

## Risks

1. Elements + Ivy is not due until ng7 it sounds like, while melding the Elements further into the Angular syntax and reducing footprint of element (we all know Angular is a big framework download). These things are not yet ready for prime time it seems.
2. Lack of IE direct support for elements; Edge and IE will be there, not yet; however apparently a workaround is available using “polyfills”, so 95% of functionality is currently supportable back to IE9.



1. <https://itnext.io/building-micro-frontend-applications-with-angular-elements-34483da08bcb> [↑](#footnote-ref-1)