Introduction to AngularJS 2

By Jeffry Houser

Updated 11/2016

A DotComIt Whitepaper

Angular 2 has officially been released, but there is not a lot of information available about how to best develop applications with it. Angular is built to be used by TypeScript, JavaScript, or Dart. However, only the TypeScript tutorials are fleshed out. Coming from an Angular 1 background with JavaScript, I wanted to start my Angular 2 adventures using JavaScript.

This guide will show you how I created my first Hello World Angular 2 application using JavaScript.

Import the Angular 2 libraries

One difference I noticed between Angular 1 and Angular 2 is that Angular 2 code base is split up among many different libraries. I believe this was intentional so you could easily remove components of Angular that you do not need; thus making the overall load time for your application smaller.

Let's look at a bunch of these:

```
<!-- IE required polyfill -->
<script src="https://npmcdn.com/core-js/client/shim.min.js"></script>
```

The first is a polyfill library for Internet Explorerer. Polyfill is a term for implementing an API not natively supported. Internet Explorer requires this to support promises and dynamic modules loading, as defined as part of the ECMAScript 2015 specification. Other browsers support this by default. You can leave this out if IE is not required for your application.

```
<script src="https://unpkg.com/zone.js@0.6.25?main=browser"></script>
<script src="https://unpkg.com/reflect-metadata@0.1.8"></script>
```

Zone and Reflect are the Polyfills used by AngularJS.

```
<script src="https://unpkg.com/rxjs@5.0.0-beta.12/bundles/Rx.js"></script>
```

Reactive Extensions RXJS is a library for composing asynchronous and event based programs using Observable collections. Observable collections are an implementation of the Observer design pattern. It is a way for one object to notify others of state changes. Data binding is an example of this. RXJS is used in many Angular 2 applications.

Finally, import the main Angular 2 libraries:

```
<script src="https://unpkg.com/@angular/core/bundles/core.umd.js"></script>
<script src="https://unpkg.com/@angular/common/bundles/common.umd.js"></script>
<script src="https://unpkg.com/@angular/compiler/bundles/compiler.umd.js"></script>
<script src="https://unpkg.com/@angular/platform-browser/bundles/platform-
browser.umd.js"></script>
<script src="https://unpkg.com/@angular/platform-browser-dynamic/bundles/platform-
browser-dynamic.umd.js"></script>
```

The official API reference contains detailed information of what each library is, but for the purposes of this article I decided not to go into details.

Create Your First Angular 2 Component

The first thing we'll do is create an Angular 2 component. The default Angular 2 method is to wrap the component around an Immediately Invoked Function Expression (IIFE) function, like this:

```
(function(app) {
})(window.app || (window.app = {}));
```

There is no Angular code in this snippet yet, but I want to point out that a variable is passed into the function. This is window.app, which is an object. Some shorthand trickery is used to define window.app as an empty object. The code could be rewritten like this:

```
window.app = {};
(function(app) {
})(window.app);
```

I find the approach of storing a pointer to the main application object curious. Global variable are sometimes considered bad practice and when building Angular 1 applications, I went out of my way to avoid creating new ones. Here we create one for the main Angular component.

Now, let's flesh out the component:

```
app.AppComponent = ng.core.Component({
       selector: 'my-app',
       template: '<h1>My First Angular 2 App</h1>'
   })
   .Class({
       constructor: function() {}
   });
```

Inside the empty app object, a property named AppComponent is created. This creates an Angular component using the ng.core.Component() function. In Angular 2 a component seems to be synonymous with a directive in Angular 1. The Component() function accepts an object as its argument. The object has two properties. The first is the selector, which is the component name and how it will be referenced in HTML code. The second is the template, which is the HTML Text. For the purposes of this sample, I defined the template inline.

After the Component() function, you'll see the dot notation used to call a second method named Class(). This is where we define the JavaScript code behind the component, sort of like a Controller in Angular 1. For the moment this component uses no JavaScript. It is an entity that includes some markup and some JavaScript code. Angular 2 purposely separates the JavaScript and markup portions of the template.

Create an Angular Module

The next step to create a running AngularJS application is to create the Angular module. After the component is defined add another IIFE:

```
(function(app) {
})(window.app || (window.app = {}));
```

Then create the module:

```
app.AppModule = ng.core.NgModule({
       imports: [ ng.platformBrowser.BrowserModule ],
       declarations: [ app.AppComponent ],
       bootstrap: [ app.AppComponent ]
   })
   .Class({
       constructor: function() {}
   });
```

The NgModule() function is called on the ng.core library. It operates similar to the Component() function; but this time creates an Angular app instead of an Angular component. For the purposes of this article, think of an app as a collection of components. A single array argument is passed into the NgModule() function; which is a list of configuration options. Chaining off the module creation, a class is create with a constructor. We are not putting anything in the class for the purposes of this sample.

The argument list in the NgModule() contains imports, declarations, and bootstrap. The imports define the other, external, modules that are required for this app. The declaration and bootstrap both refer to our custom components made for this application. In this case, they reference the app.AppComponent variable.

This is a switch from Angular 1 development, where in most cases you would define the module first, and all components, directives, and services later.

Bootstrap the Component

In Angular 2, you initialize an Angular module by bootstrapping it. This tells Angular 2 to examine the HTML; find any tags it recognizes, and replace them with the relevant HTML and JavaScript interaction. This statement is, once again, wrapped in an IIFE:

```
(function(app) {
})(window.app || (window.app = {}));
```

The IIFE's function accepts the window.app variable, and we use the same short-hand approach to define that variable if it is not yet defined. Next, add an event listener to the DOMContentLoaded event. DOMContentLoaded is a browser event that launches whenever the page is finished loading:

```
document.addEventListener('DOMContentLoaded', function() {
});
```

Inside the DOMContentLoaded event handler function, call the bootstrap() method on the platformBrowserDynamic object:

```
ng.platformBrowserDynamic.platformBrowserDynamic()
                         .bootstrapModule(app.AppModule);
```

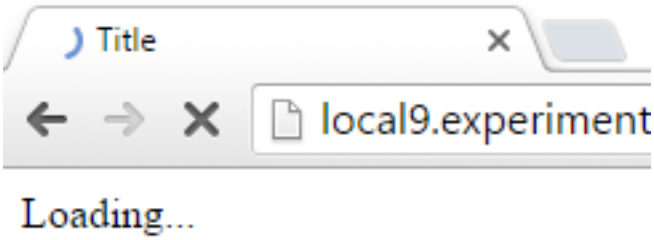
Now, the JavaScript behind your first Angular 2 app is ready to go. The last step is to add some HTML.

Use Component in HTML

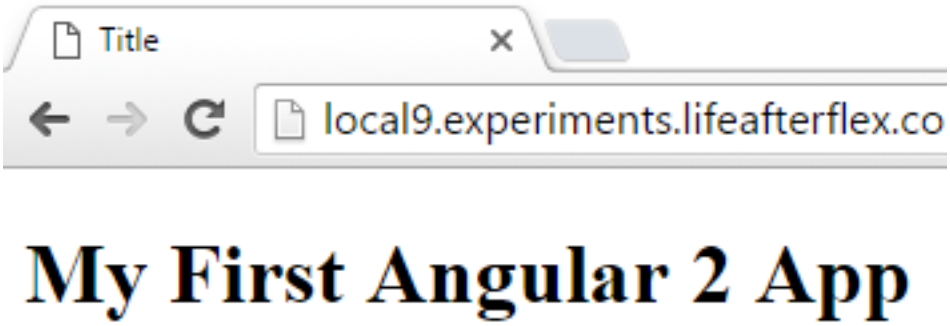
Inside the body of the HTML application, create an instance of the my-app component:

```
<my-app>Loading...</my-app>
```

Then you're app should be ready. Try to run it; you should see the loading message:



This message should go away quickly, to be replaced by the actual app:



You can play with this code here.

Binding in Angular 2

I wanted to take the initial sample a step further. This next sample will demonstrate how to bind user input to an Angular 2 display. To do that we only need to make a few architecture changes, and then a bunch of changes to the AppComponent.

First, we'll need to import Angular forms library:

```
<script src="https://unpkg.com/@angular/forms@2.2.0"></script>
```

Angular 2 is purposely split up into multiple libraries, so you only have to import the items that you need. Next, tell your module to use the forms library. To do this, modify the imports command on the ngModule:

```
imports: [ ng.platformBrowser.BrowserModule, ng.forms.FormsModule ],
```

The imports value is an array, so adding the new import is just a matter of adding it to the end of the array. All this is so we can use the ngModel directive inside our template.

First, make changes to the AppComponent's Class. Add a helloTo variable inside the constructor:

```
.Class({
    constructor: function() {
        this.helloTo = "World"
    }
})
```

The class parallels a controller inside of an AngularJS 1 application.

Next, we need to modify the template inside the component. First, review the existing component:

```
ng.core.Component({
    selector: 'my-app',
    template: '<h1>My First Angular 2 App</h1>'
})
```

The selector does not need to change, but we can easily change the template. First, change the header to say hello world:

```
template: '<h1>Hello {{helloTo}}</h1>' +
```

Angular 2 uses the same view binding syntax as Angular 1 does. The plus at the end of the template is the JavaScript concatenation operator. It means we're going to add more lines. Let's start with an input:

```
'<input type="text" [value]="helloTo"
                    (input)="helloTo=$event.target.value" /><br/>' +
```

The input is a standard HTML input, with the type of text. The value and the input parameters are both related to Angular. The value is enclosed in square brackets. I interpret this to mean that whenever the helloTo variable changes it will also change the value of the input. This is not bidirectional. The input is surrounded by parenthesis. This is like calling a function. Whenever the input changes, the function will be executed; and the function changes the helloTo variable. The function body is defined in-line; but we could split it out into a separate function inside the component's class.

I couldn't find solid definition for the bracket or parenthesis syntax in relation to the input and AngularJS. But, my interpretation is that the parenthesis always means a method is called; and the brackets always mean a value is set.

Angular 2 also includes a parallel the ngModel directive from Angular 1:

```
'<input type="text" [(ngModel)]="helloTo" /><br/>'
```

The ngModel text is enclosed in both brackets and parenthesis. This represents two way binding in Angular 2. Behind the scenes Angular 2 changes the helloTo class variable whenever the input changes; and the text input whenever the helloTo variable changes.

Try to run this:

As you type in one of the inputs, you'll see the other input changes as well as the hello header.

Play with the code here.

There are a few other ways you could set up the input to achieve the exact same purpose, but for the purposes of this article I decided to stop at two.

Add a Button Click

Another common element I need to do when developing applications is to respond to button clicks. In Angular 1 I used the ngClick event. Angular 2 uses a similar concept, but is slightly different implementation. Here is the component's template as a reminder:

```
template: '<h1>Hello {{helloTo}}</h1>' +
          '<input type="text" [value]="helloTo"
                  (input)="helloTo=$event.target.value" /><br/>' +
          '<input type="text" [(ngModel)]="helloTo" /><br/>'
```

The template consists of a header which binds to the helloTo variable. It has two inputs, both representing different methods to change the helloTo variable.

We'll add one more item to the template. Here is a button:

```
'<button (click)="onReset()">reset</button><br/>' +
```

I put the button after the header, but before the inputs. Instead of using an ngClick directive, the directive is named click, and is enclosed by parenthesis. This will call a method inside the component's class. A quick refresher on the class:

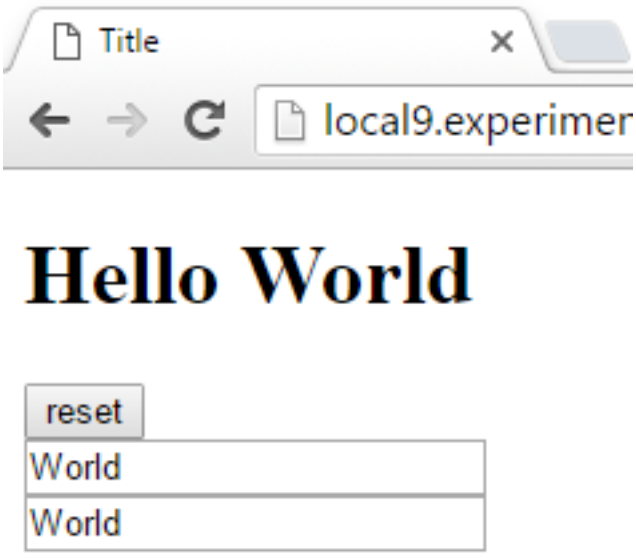
```
.Class({
    constructor: function() {
        this.helloTo = "World"
```

```
    }
});
```

We want to the button to reset the helloTo variable to its default value, World. After the helloTo variable is created, add a function named onReset():

```
this.onReset = function(){
   this.helloTo = "World";
}
```

Run the code, and you should see something like this:



Change one of the inputs, and click the reset button. The app should go back to its default state.

[Play with the code here.]

Final Thoughts

I enjoyed my experiments with Angular 2 and JavaScript. Based on available documentation, I'd be cautious about jumping into Angular 2 deep at the time of this writing. If I were going to; I'd focus on TypeScript. The full documentation for JavaScript is not there yet and most of the community questions revolve around TypeScript answers, which are not always easily portable to JavaScript. Despite all this; I'm cautiously optimistic about our Angular 2 future.