Angular Dependencies

By: Mosh Hamedani

In **package.json,** you find these dependencies in our Angular app:

systemjs

This is the module loader we use in our application. With a module loader, we no longer have to add tens or hundreds of script tags (in the right order) in our index.html. Instead, we import the main or starting module of our application into a module loader (like system.js), and based on the "import" statements we use in our code (to bring other modules), it'll automatically downloading those modules in the right order. In production applications, we can also have our module loader create a bundle of all our application modules. There are many module loaders out there, and a few populate ones include: **browserify**, **webpack**, and **system.js**. System.js is the one that Angular team prefers.

es6-promise

The next version of Javascript (es6) introduces a nice and easy API to work with promises. We use promises to handle the results and errors of asynchronous operations (eg AJAX calls, web workers, etc). This library (es6-promise) is a polyfill for es6 promises, meaning it brings es6 promises into es5.

es6-shim

Apart from promises, es6 brings many other useful features, like modules, classes, etc. This library provides compatibility shims so current JS engines behave as close as possible to es6.

Angular Dependencies

By: Mosh Hamedani

reflect-metadata

With es7, we will be able to add metadata (also called annotation or decorator) to a class or function. This library is a polyfill for metadata API in es7.

```
@Component({ selector: “app” })
class AppComponent { }
```

rxjs

This is Reactive Extensions for Javascript library, which introduces an elegant way to work with asynchronous operations. We have a complete section on Reactive Extensions and observables later in the course. Angular uses a concept called “observables” in the implementation of its Http and Jsonp classes. Angular 1, we used promises, but Angular 2 embraces observables. You’ll find out why later in the course.

zone.js

One of the complexities of Angular 1 is its digest loop, which helps Angular 1 figure out the changes in the model and refresh the view. In Angular 2 we don’t have this concept anymore. In Angular 2, all browser events are “monkey-patched”, so even if you an event handler outside your Angular app, it will still be notified and can detect changes in objects. Angular team have extracted this part from the core Angular script and open-sourced it as a separate library that can be re-used by others. This library is called zone.js.