Overview

* What is Angular 2
  + Much like AngularJS, framework to create SPAs (Single Page Applications).
  + Makes use of TypeScript, a superset of JavaScript.
    - Can also be used with traditional JavaScript and Dart (Another superset of JavaScript).
    - Benefits of using TypeScript in Angular 2
      * Static Compilation
        + Allows a developer to see parameters of functions, classes, and variables.
        + Easy to detect errors.
        + IntelliSense
      * Enforcing data types (goes back to easily detecting errors)
      * Shared likeness with new ES6 syntax (decorators, import, export, let statements)
      * Heavily documented and preferred method of developing in Angular 2
    - Grievances
      * Type definitions must be available in order to use libraries that were not made to be used directly with Angular 2 (e.g. JQuery, Velocity, etc.)
  + Framework following the Web Components Spec.
  + Many Core concepts from AngularJS have carried over such as filters (now referred to as pipes), Data Binding, 2 way data binding.
  + Angular CLI is a command like tool to aid in starting new projects or handles following development best practices.

npm install -g angular-cli

* Useful Commands
  + ng new app-name
    - Create new app boilerplate with necessary dependencies
  + ng serve
    - Serves up app on localhost:4200
  + ng generate blueprint (g for shorthand)
    - Creates a new component, pipe, interface, class, etc.
* Components
  + Unlike AngularJS, controllers no longer house application logic.
  + Components encapsulate logic and let you divide views into widget like pieces that can be reused and repurposed to your liking.
  + Components encapsulate styles that are only applied to the component.
  + Components are routable.
    - This means that an Angular 2 application is essentially an amalgamation of components, each housing additional components to make said application function.
  + Components handle their own dependencies and also inherit dependencies.
  + Components can take input data from components that house them or push actions upward (data down actions up).
  + Component Anatomy:
    - Imports
      * Dependencies from Angular 2 or from any Angular 2 libraries
    - Decorators
      * Used to define component Metadata
        + Metadata configures component tributes such as its HTML selector, location of its stylesheet or template sheet, class bindings, and included dependencies such as pipes and components.
      * Informs Angular 2 to transform the exported class to whatever the decorator is defining.
    - Export
      * Exports logic for the component that the template will be referencing.
        + Variables, functions, decorators, constructors, and more can all be used inside the component class.
      * Lifecycle Hooks – Events triggered that can be used to execute logic in accordance to a variety of different scenarios than a component may undergo. Below are a few of the many available hooks.
        + ngOnInit

Called once after the first ngOnChanges.

* + - * + ngOnChanges

Called whenever there is a change in the input properties of the component. Including once before ngOnInit.

* + - * + ngOnDestroy

Called once when a component is destroyed. Usually used as a means to cleanup, i.e. Unsubscribing from observables to avoid memory leaks.

* Routing
  + Angular 2 router has been deprecated on multiple occasions through its development cycle.
  + Previous iteration of router had routes defined within each component through a decorator. Routes are now defined in the app.module TS file.
  + Routes can redirect to other defined routes, and each route has a component associated with it.
  + Routes can have child routes allowing for routes such as /cookbooks/:id/recipe/:id
  + The above route definition would look like:

[

{

path: '',

redirectTo: '/cookbooks',

pathMatch: 'full',

},

{

path: 'cookbooks',

children: [

{

path: '',

component: CookbooksComponent

},

{

path: ':id',

children: [

{

path: '',

component: CookbookDetailComponent

},

{

path: 'recipe/:id',

component: RecipeComponent

}

]

}

]

}

]

* + To allow routing to occur, route definitions must be made in app.module and <router-outlet></router-outlet> must be included in the root level component.
  + Angular CLI currently does not support the creation of routes as Angular 2’s router has been changing up until release. Regardless, routing is still fully supported.
* Services
  + Services in Angular 2, specifically data services can be used to store a number of different operations that are to be used throughout an app.
  + Using the @injectable decorator allows for the service to be injected as a provider into a component.
  + Functions that make requests to an api and return data can be defined in the TS class.
  + Data coming back from APIs can be mapped to a strongly typed class or interface, a handy feature for security concerns.
  + These requests make can be mapped to promises and handle errors.
  + The inclusion of the RXJS library allows for the use of observables, which components can subscribe and unsubscribe to receive data asynchronously. Observables also have more functionality than a promise, with access to map, reduce, filter, etc.
* Pipes
  + Pipes are Angular 2’s implementation of filters.
  + Pipes take in data and transform said data into a desired format i.e. currency, splicing a list, searching a list, text formatting, etc.
  + Available pipes in Angular 2
    - DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, SlicePipe, and PercentPipe.
    - JSON pipe for debugging
    - And more…
  + FilterBy and GroupBy pipes have been removed for performance reasons.
  + 2 Types of Pipes Impure and Pure Pipes
    - Pure pipes only check for changes in primitive input values, not changes within objects.
    - Impure pipes will be called on each component change detection cycle.
      * Must be implemented carefully as they can drastically impact performance.
* Interpolation and Data Binding
  + Data can be binded to HTML using square bracket notation or interpolation.
    - <Input placeholder=”{{placeholderText}}”/>
    - <Input [placeholder]=”placeholderText” />
  + Interpolation should work everywhere in the template. It’s a matter of taste of how it should be used.
  + To bind actions from component logic to the template, parenthesis notation is used.
    - <button (click)=”onButtonClick()”></button>
  + Output from a component is captured in the same manner.
    - <app-search (searchRecords)=”search($event)”><app-search>
      * The data emitted from the component is then captured by the paired function.
  + To Initiate two way data binding between the template and the component logic, square parenthesis notation is used
    - <input [(ngModel)]=”inputText” />
  + This does wonders for performance as data flow is always unidirectional unless otherwise stated.
* Directives
  + Directives that alter the composition of the template is another concept carried over from AngularJS

n git push -u origin master

* + g-repeat has been replaced with \*ngFor
    - <span \*ngFor=”let apple of apples”></span>
  + ng-if has been renamed \*ngIf
  + ng-show and hide have been replaced with \*ngIf but can be replicated by using the ngClass directive.
  + ng-switch is now [ngSwitch]
  + All other directives such as ng-click have been replaced by action biding in tags such as (click), (keyup), (mouseover), etc.