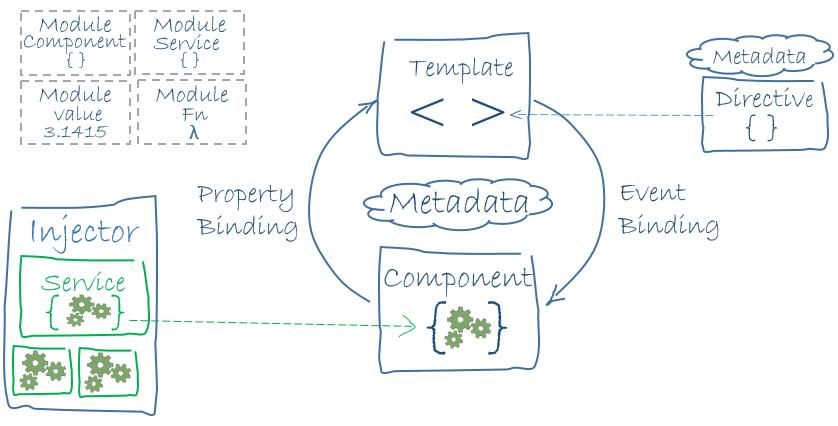
Angular 2

Style Guide: <https://github.com/johnpapa/angular-styleguide/blob/master/a2/README.md>



Bootstrap Root Component - AppComponent

* Main: app/main.ts

import {bootstrap} from 'angular2/platform/browser';

import {AppComponent} from './app.component'; // Root Component in file: app/app.component.ts

bootstrap(AppComponent);

* Root Component: app/app.component.ts

import {Component} from 'angular2/core';

@Component({

selector: 'my-app',

template: '<h1>My First Angular 2 App</h1>'

})

export class AppComponent { }

Module

* Create: export class MyComponent { } // class/function/values
* Use: import {MyComponent} from “./my.component”;
* Library Module: 1+ modules. e.g. Angular Libraries: angular2/core, angular2/common, angular2/router
  + Use: import {Component} from “angular2/core”;

Component - Controller

export class HeroListComponent **implements OnInit** {

constructor(private \_service: HeroService){ }

heroes:Hero[]; // **property**

selectedHero: Hero;

**ngOnInit()** {

this.heroes = this.\_service.getHeroes();

}

selectHero(hero: Hero) { this.selectedHero = hero; } // **action**

}

Template – View

<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>

<div \*ngFor="#hero of heroes" (click)="selectHero(hero)">

{{hero.name}}

</div>

<hero-detail \*ngIf="selectedHero" [hero]="selectedHero"></hero-detail>

* Syntax
  + Everything on the RHS of = is a " **Property** ". Except when you need to assign an **literal string, use** " **‘ ’** "
  + Built-in **Directives**: \*ngIf/ngFor/ngSwitch/ngClass/ngStyle
  + DOM Property - **[]**
    - ***Element*** - <***img*** **[src]** = "heroImageUrl">
    - ***Component*** - <***hero-detail*** **[hero]**="currentHero"></hero-detail> // hero-details is a custom Component
    - ***Built-in Custom Directive*** - <***div*** **[ngClass]** = "{selected: isSelected}"></div> // ngClass is a Built-in directive
  + Interpolation: **{{** expression **}}** // the expression result will turn into a string.
    - Can be a substitute of DOM Property Binding
  + DOM Event - **()**
    - Element - <button **(click)** = "onSave()">Save</button>
    - Component - <hero-detail **(deleteRequest)**="deleteHero()"></hero-detail>
    - Directive - <div **(myClick)**="clicked=$event">click me</div> // custom directive: **myClickDirective**
    - $event: a DOM event object with useful properties like target/target.value

<input [value]="currentHero.firstName" (input)="currentHero.firstName=$event.target.value" >

* + 2-way DOM Property - **[(ngModel)]**
    - <input **[(ngModel)]**="heroName">
    - Equivalent: <input **[ngModel]**="currentHero.firstName" **(ngModelChange)**="setUpperCaseFirstName($event)">
  + Html Attribute – **[attr.htmlAttrib] // use Html Attribute when No DOM Property to bind**
    - <button **[attr.**aria-label**]**="help">help</button>
  + Html Class Attribute – **[class] /** **[class*.single-class-name*] // dot notation useful for single class assignment** 
    - Replacement: <div **[class]**="classnames">Special</div>
    - **[ngClass]** is [better](https://angular.io/docs/ts/latest/guide/template-syntax.html#!)! <div **[class.**special**]**="isSpecial">Special</div>
  + **Directive [ngClass] // useful for assigning multiple classes**
    - <div **[ngClass]**="*setClasses()*"> xyz </div>

*setClasses()* { // assign multiple classes

let classes = {

*saveable*: this.canSave, // true

*modified*: !this.isUnchanged, // false

}

return classes;

}

* + Html Style Attribute – **[style.cssStyle] // useful for single style property**
    - <button **[style.**color**]** = "isSpecial ? 'red' : 'green'">
  + **Directive [ngStyle] // useful for setting multiple style properties values**
    - <div **[ngStyle]**="*setStyles()*"> xyz </div>

*setStyles()* { // set multiple style properties values

let styles = {

'font-weight': !this.isUnchanged ? 'bold' : 'normal', // normal

'font-size': this.isSpecial ? '24px' : '8px', // 24px

}

return styles;

}

* + **Directive [ngSwitch]**

<span **[ngSwitch]** ="toeChoice">

<span **\*ngSwitchWhen**="'Eenie'">Eenie</span>

<span **\*ngSwitchDefault**>other</span>

</span>

* + **Directive [ngFor]**
    - <div **\*ngFor**="#hero of heroes">{{hero.fullName}}</div> // **# is like var for local variable**
    - With Index: <div \*ngFor="#hero of heroes; #i=**index**">{{i + 1}} - {{hero.fullName}}</div>
    - Tracking: <div \*ngFor="#hero of heroes" **\*ngForTrackBy**="*trackByHeroes*">({{hero.id}}) {{hero.fullName}}</div>
      * *trackByHeroes*(index: number, hero: Hero) { return hero.id; }
  + **Directive [ngIf]**
    - <div **\*ngIf**="currentHero">Hello, {{currentHero.firstName}}</div>

Decorator – metadata

**@Component**({

**selector**: 'hero-list',

**templateUrl**: 'app/hero-list.component.html',

**template**: ` some inline html, could be multiline using back-ticks `,

**styleUrls**: [‘app/hero-list.component.css’, ‘more.css’],

**directives**: [HeroDetailComponent], // an array of the Components or Directives that **this template use**

**providers**: [HeroService] // an array of DI services that **this component uses**

})

*export class HeroesComponent { ... }*

* ES6 Template String using **back-ticks** significantly simplify multiline strings. Whitespace inside will be considered part of the string.

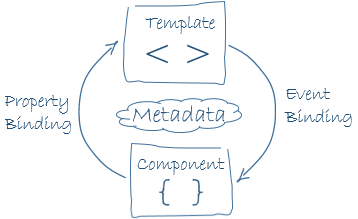
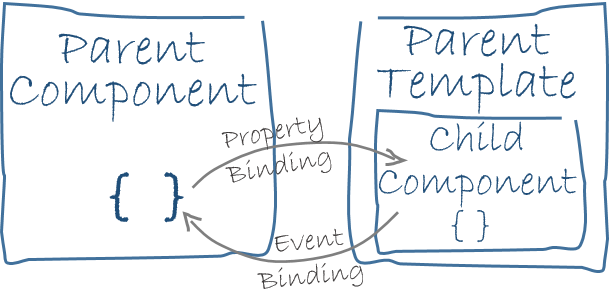
**`** <div>

<input … >

</div> **`**

Data Binding

* Communication between a template and its component and between parent and child components

Service

* Just an export POCO. e.g.

export class Logger {

log(msg: any) { console.log(msg); }

error(msg: any) { console.error(msg); }

warn(msg: any) { console.warn(msg); }

}

Directive

* A directive is a class with directive metadata. In TypeScript we'd apply the **@Directive** decorator to attach metadata to the class.
* There 3 kinds of directives "component", "structural" and "attribute" directives.
* **Component** directivies
* **Structural** directives alter layout by adding, removing, and replacing elements in DOM. e.g. \*ngFor, \*ngIf, \*ngSwitch
* **Attribute** directives alter the appearance or behavior of an existing element. e.g. [(ngModel)], [ngClass], [ngStyle]
  + e.g.

import {**Directive, ElementRef**, Input} from 'angular2/core';

**@Directive**({

**selector: '[**myAttribute**]'**, // Required

**host: {**

**'(mouseenter)': '**onMouseEnter()**'**,

'(mouseleave)': 'onMouseLeave()'

**}**

})

export class MyAttributeDirective { // **ElementRef** allows us to **access the DOM element**.

**@Input('myAttribute')** highlightColor: string; // ‘myAttribute’ is an **alias** for highlightColor property

private \_defaultColor = 'red';

**@Input() set** defaultColor(colorName:string){ // a 2nd property using setter AND **without requiring to use selector**. **Confusing!**

this.\_defaultColor = colorName || this.\_defaultColor;

}

constructor(**private** el: **ElementRef**) { }

onMouseEnter() { this.\_highlight(this.highlightColor || this.\_defaultColor); }

onMouseLeave() { this.\_highlight(null); }

private \_highlight(color: string) {

**this.el.nativeElement**.style.backgroundColor = color;

}

}

* Input/Output properties. Either with a decorator **inside a Component class** or in a **metadata array inside @Component**. Don't do both!
  + **@Input(‘optional alias’)** hero: Hero; // as decorator.
    - data flows **into** that property from a template binding expression.
  + **@Output(‘optional alias’)** ***myEventSource*** = new EventEmitter<Hero>(); // as decorator. For a **child** Component to **communicate** to its **parent** Component via **raising** a **Custom event**
    - events stream **out** of that property and toward the handler in a template binding statement

myFiringMethod(selectedHero: Hero) {

this.selectedHero = selectedHero;

**this.*myEventSource*.emit**(*selectedHero*); // *selectedHero* **payload** becomes the **$event in the parent** Component

}

* + <hero-detail [**hero**]="currentHero" (**deleteRequest**)="deleteHero**($event**)"></hero-detail> // parent Component template

**Parent Component *Template* interacts with child Component** ***via local variable*** [link](https://angular.io/docs/ts/latest/cookbook/component-communication.html#!)

* Example: **local variable** **#timer**

<button (click)="**timer.**start()">Start</button>

<button (click)="**timer.**stop()">Stop</button>

<div class="seconds">{{**timer.**seconds}}</div>

<countdown-timer **#timer**></countdown-timer> // **countdown-timer** is the **child** component

**Parent Component *Class* calls a ViewChild** [link](https://angular.io/docs/ts/latest/cookbook/component-communication.html#!)

* The **local variable** approach above is **limited** within the parent template. The parent component **class** has no access to the child.
* Solution use **@ViewChild** to ***inject*** the Child Component int Parent Component

export class ParentComponent {

**@ViewChild(ChildComponent) child: ChildComponent;**

this.child.**myMethod()**;

let p = this.child**.myProp**; // can support **both read and write**

}

**Dependency Injection**

* **Create** Injectable Service

import {**Injectable**} from 'angular2/core';

**@Injectable()**

export class HeroService { }

* **Register** service at **root level** to make it available everywhere
  + bootstrap(AppComponent, **[BackendService, HeroService, Logger]**);
* **Register** service at **Component level**. Get a new instance of the service with each new instance of that component.

**@Component**({

**providers: [HeroService]**

})

export class HeroesComponent { ... }

* **Using: constructor**(private \_service: **HeroService**){ }

Lifecycle Hook

* ngOnChanges/ngOnInit/ngDoCheck/ngAfterViewInit/ngOnChanges/ngOnDestroy
  + Implement **OnInit interface** which provides the **Function Types** of **ngOnInit** (duck typing?)
* ngOnChanges: called whenever property is changed. 1st time is Component -> View, 2nd time is View -> Component
  + Example

import {Component, **Input, OnChanges, SimpleChange**} from 'angular2/core';

@Component({

selector: 'version-child',

template: `

<h3>Version {{major}}.{{minor}}</h3>

<ul> <li \*ngFor="#change of changeLog">{{change}}</li> </ul>

`

})

export class VersionChildComponent **implements OnChanges** {

@Input() major: number;

changeLog: string[] = [];

**ngOnChanges(**changes: **{[propKey:string]: SimpleChange}**){

let log: string[] = [];

**for** (let propName **in changes**) {

let changedProp = **changes[propName]**;

let from = JSON.stringify(changedProp.previousValue);

let to = JSON.stringify(changedProp.currentValue);

log.push( `${propName} changed from ${from} to ${to}`);

}

this.changeLog.push(log.join(', '));

}

}

Expression Operators

* **Pipe ( | )**
  + <div>{{ title **| uppercase | lowercase** }}</div> // chaining
  + [Date](https://angular.io/docs/ts/latest/api/common/DatePipe-class.html)
    - <div>Birthdate: {{currentHero?.birthdate **| date:'longDate'**}}</div>
    - date:'**fullDate/longDate/medium/shortTime/Mdy**'
  + Numeric
    - <div>{{price | **currency**}}</div>
    - <div>{{value | **percent**:'1.1-1'}}</div> // minIntegerDigits.minFractionDigits-maxFractionDigits
    - <div>{{value | **number**:'1.1-3'}}</div> // minIntegerDigits.minFractionDigits-maxFractionDigits
  + **Async**
    - Auto subscribe to a Promise or Observable, returns the emitted value
  + **json**: <div>{{currentHero **| json**}}</div> // for **debugging**
  + **Custom Pipes - @Pipe**

import {Pipe, PipeTransform} from 'angular2/core';

**@Pipe**({name: 'exponentialStrength'})

export class ExponentialStrengthPipe implements PipeTransform {

transform(value:number, [exponent]) : number {

var exp = parseFloat(exponent);

return Math.pow(value, isNaN(exp) ? 1 : exp);

}

}

* Elvis ( ?. ): just like C#

[Http Client](https://angular.io/docs/ts/latest/guide/server-communication.html)

* External module:
  + <script src="node\_modules/angular2/bundles/**http.dev.js**"></script>
  + <script src="node\_modules/rxjs/bundles/**Rx.js**"></script>
* http.get method returns an Observable of HTTP Responses (Observable<Response>) from the RxJS library and map is one of the RxJS operators.
* **Hosting** in a Service

import {Injectable} from 'angular2/core';

**import {Http, Response} from 'angular2/http';**

import {Hero} from './hero';

**import {Observable} from 'rxjs/Observable';**

@Injectable()

export class HeroService {

constructor (private http: Http) {}

getHeroes () {

return **this.http.get(**'app/heroes')

// use **.map to transform** Observable<Response> to a JSON object

**.map**(response => **<Hero[]> response .json().***data*) // not always *.data*, it is backend specific

**.do**(data => console.log(data)) // debugging: eyeball results in the console

**.catch**(this.handleError);

}

addHero (name: string) : Observable<Hero> {

let body = JSON.stringify({ name });

let headers = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: headers });

return **this.http.post**('app/heroes', body, options)

.map(response => <Hero> **response** .json().data)

.catch(this.handleError)

}

private handleError (error: Response) {

log.error(error);

return **Observable.throw(error.json().error ||** 'Server error'**);**

}

}

* **Register** http module & service in the Root Component

**import {HTTP\_PROVIDERS} from 'angular2/http';**

import {HeroService} from './hero.service';

@Component({

providers: [

**HTTP\_PROVIDERS,**

*HeroService*,

]

})

export class AppComponent { }

* **Consuming** the Service using **RxJs**

export class HeroListComponent implements OnInit {

constructor (private \_heroService: **HeroService**) {}

errorMessage: string;

heroes:Hero[];

ngOnInit() { this.getHeroes(); }

getHeroes() {

this.\_heroService.getHeroes()

**.subscribe(**

**heroes =>** this.heroes **= heroes**,

**error => this.errorMessage = <any>error**);

}

addHero (name: string) {

if (!name) {return;}

this.\_heroService.addHero(name)

**.subscribe(**

**hero =>** this.heroes.push(**hero**),

error => this.errorMessage = <any>error);

}

**heroesAsyncPipe: Observable<Hero[]>;**

getHeroes**ForAsyncPipe**() {

**this.heroesAsyncPipe = this.\_heroService.getHeroes();**

}

}

* **Consuming** using **Async Pipe**

<ul>

<li \*ngFor="#item of **heroesAsyncPipe | async**">{{item}}</li>

</ul>

**Route**

* External Module: <script src="node\_modules/angular2/bundles/**router.dev.js**"></script>
* Add <base href="/"> at the top of the <head> section
* **Register Route Mappings** in **Root Component**

**@RouteConfig**([

{ **path:** '/dashboard', **name:** 'Dashboard', **component:** DashboardComponent, **useAsDefault: true**},

{ **path:** '/heroes', **name:** 'Heroes', **component:** HeroesComponent },

{ **path**: '/detail/**:id**', name: 'HeroDetail', **component**: HeroDetailComponent },

])

* **Use Router at Root Component**

**import { RouteConfig, ROUTER\_DIRECTIVES, ROUTER\_PROVIDERS } from 'angular2/router';**

@Component({

selector: 'my-app',

**template:** ` // provide **router links** based on **router names**

<nav>

<a **[routerLink]="['*Dashboard*']"**>Dashboard</a> // specify the ***route name***

<a **[routerLink]**="**['*Heroes*']**">Heroes</a>

</nav>

**<router-outlet></router-outlet> // Route Component Placeholder**

`,

directives: [**ROUTER\_DIRECTIVES**],

providers: [

**ROUTER\_PROVIDERS,**

*HeroService*,

]

})

export class AppComponent { }

**Parent Child Routes** – promote component-path compositions flexibility

* **Parent Route** Configuration – “**/…**”
  + That means this is an incomplete route. The finished route will be some combination of the parent /crisis-center/ route and a route from the child router that belongs to the designated component.

@RouteConfig([

{ **path:** '/crisis-center**/...**', name: 'CrisisCenter', component: **CrisisCenterComponent**},

])

* **Child Route** Configuration in app/crisis-center/**crisis-center.component**.ts

@RouteConfig([

{ **path:'/'**, name: 'CrisisList', component: CrisisListComponent, useAsDefault: true},

])

Cheatsheet - <https://angular.io/docs/ts/latest/guide/cheatsheet.html>

Snippets

* <http://codewithdan.com/2016/03/19/angular-2-typescript-and-html-snippets-for-vs-code/>
* <https://github.com/johnpapa/vscode-angular2-snippets>

Tutorials

* <https://angular.io/docs/ts/latest/tutorial/>
* <https://egghead.io/series/angular-2-fundamentals>
* <http://www.mithunvp.com/angular-2-in-asp-net-5-typescript-visual-studio-2015/>