Angular

Toby Dussek

Course times each day

- 9:30 start
- 11:00 coffee break
- 12:00 lunch break
- 3:00 tea break
- 4:30 end

framework training
business value through education

Welcome and introductions

- Two questions:
  - What you currently know
  - What you need to know

framework training
business value through education

The Modern Web Stack

- HTML5
- CSS
- JavaScript APIs
- See http://caniuse.com/

- NodeJS

ECMAScript and TypeScript

- https://kangax.github.io/compat-table/es6/

ES6/7 syntax and features

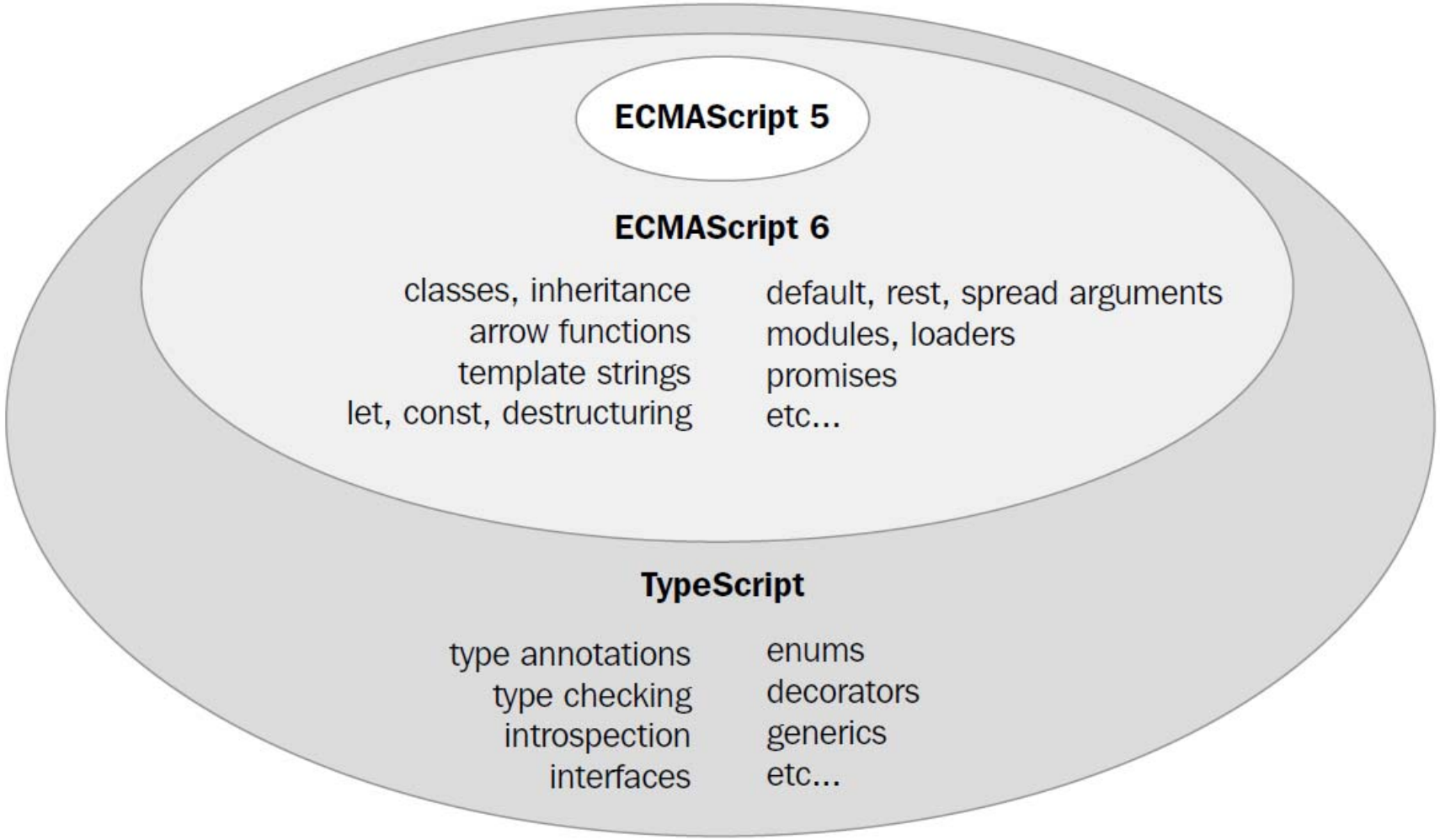
- Classes
  - ES6 (and TypeScript) do not provide support for multiple inheritance
- Arrow Functions
- let and const
- Back ticks and ${} interpolation
- Default and …rest parameters
- Modules
- Decorators
- Shadow DOM
- Promises

ECMAScript Versions

- ES3
  - The classic JavaScript everybody knows
- ES5
  - 2009 added some Object-like features
    - var o = Object.create(null)
- ES6 (now called ES2015)
  - Jun 2015
  - Sugar-syntax
- ES7 (now called ES2016+)
  - The ongoing additions to ECMAScript

framework training
business value through education

ES6, ES5 and TypeScript



Angular Command Line Interface (cli)

- Much easier to use NodeJS to create projects using Angular cli
- npm install –g @angular/cli
- ng new proj-name
- cd proj-name
- ng serve --open

- Create new content
  - ng generate component comp-name
  - ng g c comp-name

Scaffolding

- Component
  - ng g component my-new-component
- Directive
  - ng g directive my-new-directive
- Pipe
  - ng g pipe my-new-pipe
- Service
  - ng g service my-new-service
- Class
  - ng g class my-new-class
- Guard
  - ng g guard my-new-guard
- Interface
  - ng g interface my-new-interface
- Enum
  - ng g enum my-new-enum
- Module
  - ng g module my-module
- Module with routing
  - ng g module my-module --routing

CLI Modules

- ng generate module app-routing --flat --module=app
  - --flat puts the file in src/app instead of its own folder.
  - --module=app registers it in the imports array in AppModule

Configuring ng serve

- ng serve --host 0.0.0.0 --port 4201
- ng serve aot

- https://github.com/angular/angular-cli/wiki/serve

Testing

- ng test
- ng e2e

Best Practices and Patterns

- During early examples, will use common anti-patterns for simplicity
- Quickly move to better patterns

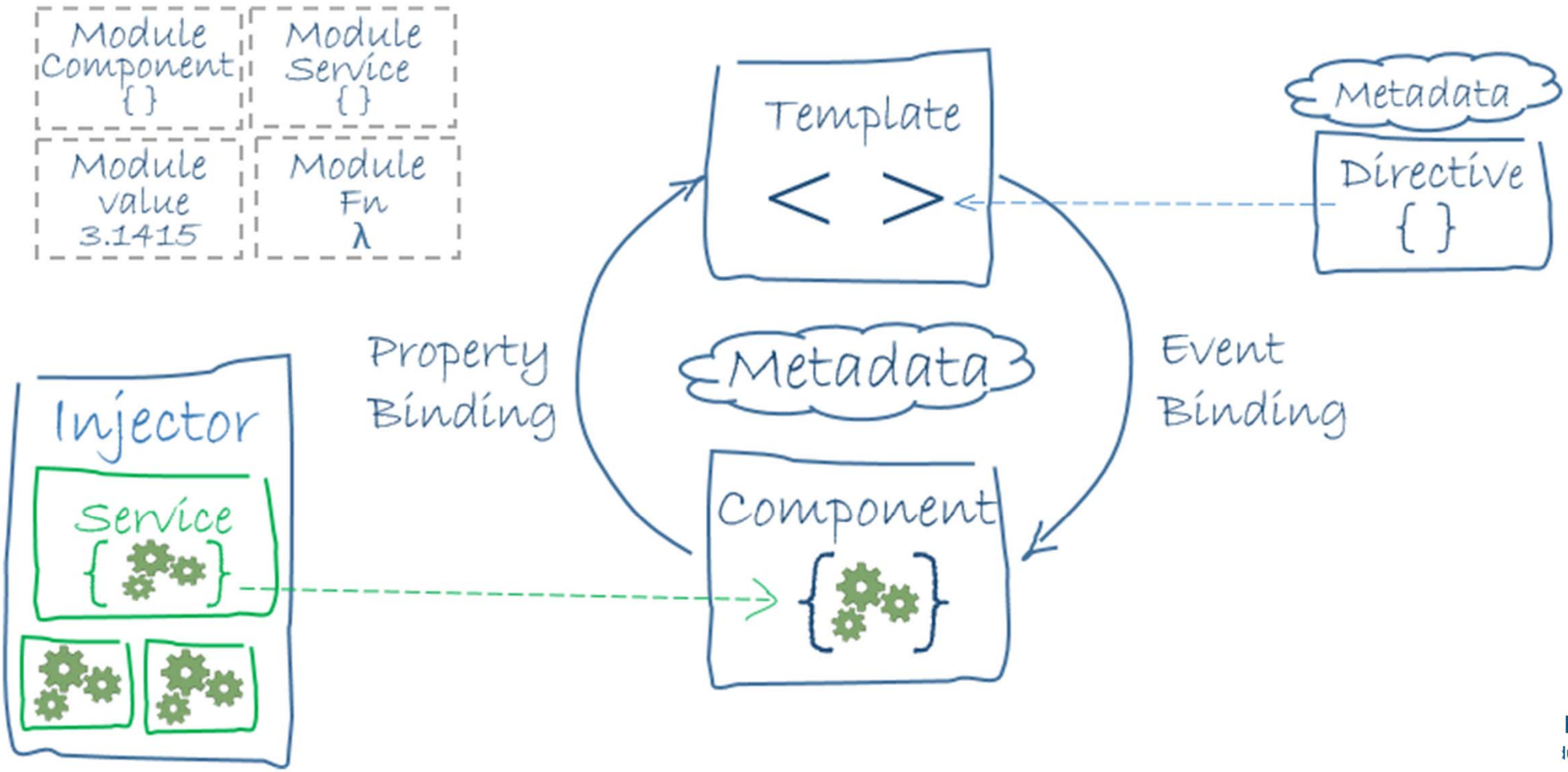
Online Resources

- https://angular.io/
- https://cli.angular.io/
- http://reactivex.io/
- https://material.angular.io/

Configure for Angular development

- Aiming for future-compatibility with ES6/ES7
- Meanwhile write code and transpile to current compatible browsers
- Online documentation only complete for Typescript
  - Pure JavaScript documentation is patchy
  - 'Most of the documentation has been written for TypeScript developers and has not yet been translated to JavaScript'
  - Refer regularly to the developer guide and API documentation
- Very useful summary
  - https://angular.io/guide/cheatsheet

Angular Architecture



Web components

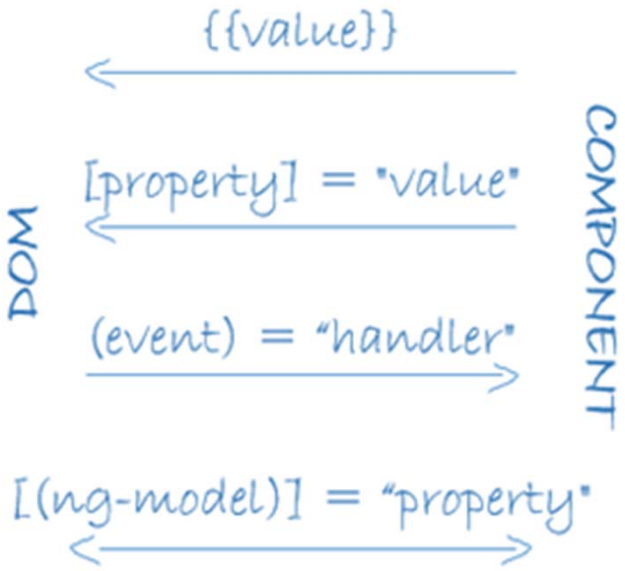
- Four technologies used to build elements with a high level of visual expressivity and reusability
- Leads to modular, consistent, and maintainable web
  - Templates are pieces of HTML that structure the content to be rendered
  - Custom Elements contain traditional HTML elements and also custom wrapper items that provide further presentation elements or API functionality
  - Shadow DOM This provides a sandbox to encapsulate the CSS layout rules and JavaScript behaviours of each custom element
  - HTML Imports to host other HTML document fragments
- An Angular component is a custom element that contains a template to host the HTML structure of its layout

TypeScript Compiler (tsc)

- Can combine modules together into a single output file
- tsc --outFile app.js module.ts
- Explore transpiling with
  - Traceur
  - Babel

Data Binding

- {{value}} interpolation
- [property] binding
- (event) binding
- [(two-way)] data binding
  - [()] BANANA IN A BOX



- Angular processes all data bindings once per JavaScript event cycle
  - from the root of the application component tree down to the leaves

framework training
business value through education

Binding Expressions

Data direction	Syntax	Binding type

One-way
from data source
to view target
	`{{expression}}`
	`[target] = "expression"`
	`bind-target = "expression"`
		Interpolation
		Property
		Attribute
		Class
		Style

One-way from view target to data source	`(target) = "statement"` `on-target = "statement"`	Event
Two-way	`[(target)] = "expression"` `bindon-target = "expression"`	Two-way

https://angular.io/docs/ts/latest/guide/template-syntax.html

HTML Attributes vs. DOM value properties

- The HTML attribute and the DOM property are not the same thing, even when they have the same name
- https://angular.io/docs/ts/latest/guide/template-syntax.html#!#template-expressions
- Angular Template binding works with properties and events, not attributes
  - The only role of attributes is to initialize element and directive state
  - When we data bind, we're dealing exclusively with element and directive properties and events

Data bindings with input properties

- <my-timer [seconds]="25"></my-timer>
- [seconds] is an input property
  - the my-timer component will contain a setter function for the seconds property
  - Useful when injecting data into custom properties of a component
- Can inject data from a variable
  - [seconds]="storedValueOfSeconds“
- Or a literal value
  - [seconds]=" 'some string value' "

- Use bracket syntax to make native HTML attributes reactive to values of fields in component
  - <h1 [hidden]=“hiddenFlag">
- Angular has syntactic sugar for binding expressions
  - <h1 [attr.hidden]=" hiddenFlag">
  - <input [class.is-valid]="isValid">
  - <div [style.width.px]="myWidth">
- Naming Convention
  - Custom input and output property names use camelCase
  - Built-in input and output property names tend to be lowercase

Mostly One-way Binding

- One of the main differences between Angular and AngularJS is it now favours one-way data binding as the core pattern of data management
  - Most of the data management processes in Angular are one way only
  - FormsModule provides two-way data binding via the NgModel directive

Naming Conventions

- Convention helps to identify which classes are components and which files contain components
  - E.g. AppComponent goes in app.component.ts
  - HeroDetailComponent is in hero-detail.component.ts
  - All component names end in Component
  - All service names end in Service
  - All component file names end in .component
  - All service file names end in .service
- Spell file names in lower-dash-case (kebab-case)
  - Se we don't need to worry about case sensitivity on server or source control

https://angular.io/docs/ts/latest/tutorial/toh-pt3.html

Target and Source Properties

-
  <button (click)="onSave()">Save</button>
- The binding target
  - to the left of the equals sign
  - is the property or event inside the binding punctuation: [], () or [()]
- The binding source
  - to the right of the equals sign
  - is either inside "quotes" or within an {{interpolation}}

Two Way Binding with NgModel Directive

- The syntax of ngModel gives a very good hint to what's going on
- A single attribute blends an event handler and a property binding
  - Hence the combination of brackets plus braces [()]
  - Can inject a value into the target control AND listen to changes made on the value at the same time
- This is two-way data binding
  - Usually has a non-trivial impact on performance

- To use ngModel, you must import the FormsModule and add it to the Angular module's imports list

framework training
business value through education

Property binding or interpolation?

- We often have a choice between interpolation and property binding
- The following binding pairs do the same thing:

```
<p><img src="{{heroImageUrl}}"> is the <i>interpolated</i> image.</p>
<p><img [src]="heroImageUrl"> is the <i>property bound</i> image.</p>

<p><span>"{{title}}" is the <i>interpolated</i> title.</span></p>
<p> <span [innerHTML]="title"></span>  is the <i>property bound</i> title.</p>
```

- Interpolation is a convenient alternative for property binding in many cases
  - Angular translates those interpolations into the corresponding property bindings before rendering the view
  - There is no technical reason to prefer one form to the other

framework training
business value through education

Content Security

- Angular data binding sanitizes values before displaying them
- It will not allow HTML with script tags to leak into the browser
  - neither with interpolation or property binding

Forms and NgForm

- Every form in any template is automatically an ngForm directive, unless we explicitly add a formGroup attrubute to the form
  - Can cancel this automatic binding by appending the ngNoForm attribute to any <form> tag you do not want to become an ngForm
  - Automatically creates a FormGroup called ngForm
  - Automatically adds (ngSubmit) as an output

NgModel Directive

- Specifies a selector of ngModel
- <input name='frmField' ngModel>
  - Creates a one-way data binding
  - Creates a FormControl called frmField, as a child of the FormGroup object

Reactive and Template-Driven Forms

- FormsModule provides
  - ngModel
  - NgForm
- ReactiveFormsModule provides
  - formControl
  - ngformGroup
- FormBuilder provides
  - control function
  - group function

Controls, ControlGroups class

- Angular provides a more efficient form model where everything flows in one direction only
- There are a lot of upsides for this, but probably the most relevant reason is the noticeable impact on performance that traditional two-way data-binding has on applications, in comparison to other patterns where information flows in one direction only

Event binding with Output properties

- Output property will trigger an event handler
  - <my-timer (countdownComplete)="onCountownCompleted()">
- Output properties can be mapped to interaction events such as
  - click, touchend
  - mouseover, mouseout
  - focus, blur
  - keyup
  - <button (click)="doSomething()">Click me</button>

EventEmitter

- EventEmitter is the built-in event bus of Angular
- Provides support for
  - Emitting Observable data
  - Subscribing Observer consumers to data changes

Communicating between components through custom events

- To create custom event bindings
  - Import Output and EventEmitter
    - import { Component, Input, Output, EventEmitter } from '@angular/core';
  - configure an output property in the component
    - @Output() complete: EventEmitter<any> = new EventEmitter();
    - this.complete.emit(null);
  - attach an event handler function to it
    - <countdown [seconds]="25“ (complete)="onCountdownCompleted()"></countdown>
    - onCountdownCompleted(): void {alert('Time up!');}

Canonical form (no need for brackets)

- [seconds] could be represented as bind-seconds
- (complete) could be represented as on-complete

Angular Coding Style Guide

- https://angular.io/guide/styleguide

Emitting data through custom events

- Can emit an event containing custom data
  - @Output() progress: EventEmitter<number> = new EventEmitter();
  - this.progress.emit(this.seconds);
- Can refer to the reserved variable $event
  - <countdown [seconds]="25“ (progress)="timeout = $event"
- $event is a pointer to the payload of the output property
  - Reflects the value we pass to the emit() function

Local references in templates

- Setting a local reference on the component itself will give access to its public façade
- Flag the instance in the component template with a local reference
  - Prefix with a hash
    - #counter
  - Or with ref-
    - ref-counter
- No need for an event emitter

External Templates

- External template names follow a convention
  - share the filename of the component they belong to
  - including any prefix or suffix of the component filename
- @Component({
  selector: ‘my-timer',
  directives: [CountdownComponent],
  templateUrl: './my-tasks.html‘
  })
- Often in a sub-folder
  - templateUrl: './app/my-component.html‘

Http, HttpClient and HttpModule

- HttpModule is the classic NgModule for using @angular/http
- Class constants
  - Http
  - Response
  - RequestOptions
  - Headers
- HttpClient is new since Angular 5

Sanitize for Security

- Angular data binding is on alert for dangerous HTML
- It sanitizes values before displaying them
- It will not allow HTML with script tags to leak into the browser, through interpolation or through property binding
- https://angular.io/docs/ts/latest/guide/template-syntax.html#!#template-expressions

HTTP Security

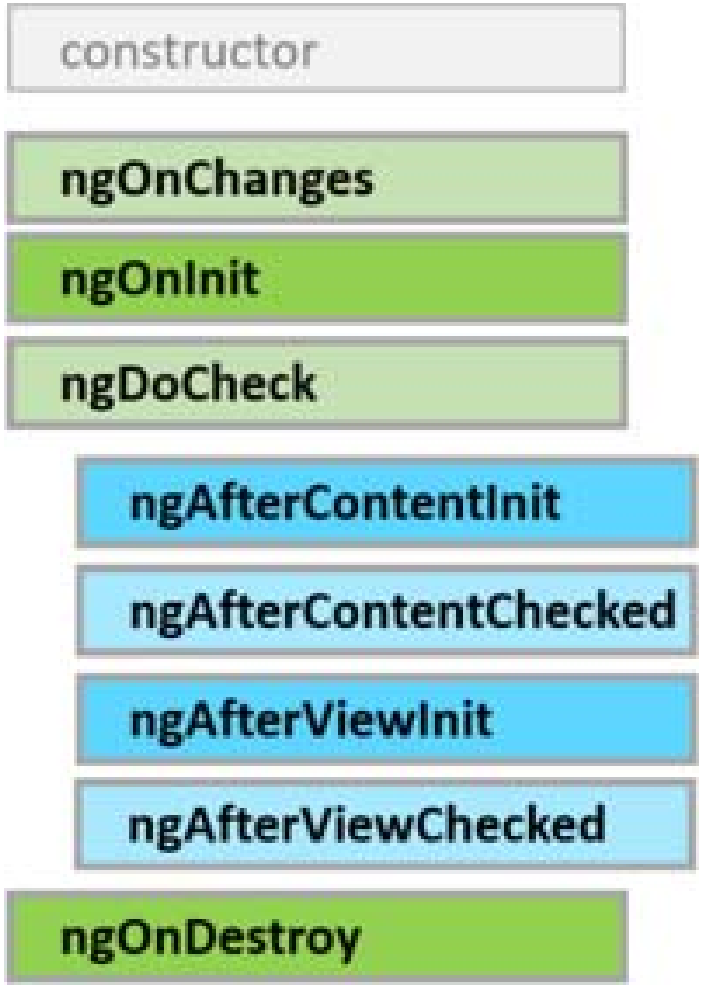
- Old-school JSONP is read-only
- More modern servers support Cross Origin Resource Sharing (CORS)
- JSON is wrapped in curly braces {[n,m,p]}
  - but ES6 treats that as a code block
  - which makes it susceptible to attack { alert(`malicious`) }
- Therefore we wrap our data in some property {data:[n,m,p]}
  - https://stackoverflow.com/questions/3503102/what-are-top-level-json-arrays-and-why-are-they-a-security-risk

Injectable Decorator

- @injectable() emits metadata about a service
- Angular may need to inject other dependencies into this service
- It’s NOT about making our service injectable into other components
- https://angular.io/docs/ts/latest/tutorial/toh-pt4.html
- -- try removing injectable from the hero service

Lifecycle Hooks

- Angular offers a number of interfaces for tapping into critical moments in the component lifecycle
  - after each change ngOnchanges
  - at creation ngOnInit
  - at its eventual destruction ngOnDestroy
- Each interface has a single method
- When the component implements that method, Angular calls it at the appropriate time



framework training
business value through education

Routing

- Set the base tag
  - Base href is essential for routing
  - <base href="/">
- Add <router-outlet> tags to the template
- RouterOutlet is one of the ROUTER_DIRECTIVES
  - The router displays each component in the <router-outlet> as we navigate through the application
- Add the [routerLink] binding to anchor tags
  - The RouterLink directive is another of the ROUTER_DIRECTIVES
  - Bound to an array to tell the router where to navigate when the user clicks the link
- https://angular.io/docs/ts/latest/tutorial/toh-pt5.html

Dependency Injection

- Absolutely at the heart of what Angular is trying to do
- Historically could have used requires
- TypeScript includes --module flag for this

Directives

- Directives are a prominent part of Angular core
- An Angular custom component is a directive with a template view
- Directives can affect the way HTML elements or custom elements behave and display their content
- Can build directives with no attached view
  - Apply to existing DOM elements
  - Existing HTML contents and behaviour is accessible to the directive

Core Directives

- Structural Directives
- NgIf
  - removes or recreates a portion of the DOM tree based on an expression
  - NOT the same as show/hide
- NgFor (NgForOf)
  - iterates through any iterable object
  - binds each of its items to a template
- NgSwitch, NgSwitchCase, and NgSwitchDefault
  - Classic switch-case construct
- Attribute Directives
- NgStyle and NgClass
  - For controlling CSS

Development and Production modes

- By default applications are bootstrapped and initialized in Development mode
  - the Angular runtime throws warning messages and assertions to the browser console
- Development mode can be disabled by Production mode
  - import { bootstrap } from '@angular/platform-browser-dynamic';
    import { enableProdMode } from 'angular/core';
    import AppComponent from './app.component';
    enableProdMode();
    bootstrap(AppComponent, []);
- Production Mode does not show errors

Simple cli for Production

- ng build --target=production base-href '/‘
  - build makes use of bundling and some tree-shaking
  - --prod build runs limited dead-code elimination via UglifyJS
  - --prod build defaults to –aot

- ng doc <keyword>
  - Opens online documentation for a keyword

CSS resources

- CSS images and fonts will be copied automatically as part of a build
  - If a resource is less than 10kb it will also be inlined

Pipes

- uppercase, lowercase
- number, percent, currency
  - :digitInfo {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
- slice
- Date
- json
- replace
- i18nPlural, i18nSelect
- async

Custom Pipes

- Import Pipe decorator from Angular core
- Create a new class decorated with this decorator
- Name class with custom selector choice
- Implement the PipeTransform interface
- Class implementation:
  - Transform method (required by PipeTransform interface)
  - Return a type and two parameters:
    - The input itself
    - Optional spread argument containing the configuration settings for this pipe
- Custom pipes must be explicitly declared in the pipes property of the decorator configuration of each component that uses them

Angular Lazy Loading

- To configure lazy loading in an application, three steps are required
- Create a module class for each bundle to be loaded
  - loaded modules are child modules and the loading module is a parent module
- In each child module, create the RouterModule by calling its forChild method with route definitions instead of forRoot
- In the parent module, configure a route definition whose loadChildren field identifies the resources to be loaded
- AVOID accidentally eager loading e.g. with an import

Unit and e2e Testing

- unit tests
  - the focus is on individual components such as controllers, services, factories, providers, directives etc.
- end-to-end (e2e) tests
  - the focus is on how the application or a module, as a whole, works, such as confirming the click of a button has certain outcomes

Protractor

- Protractor is an end-to-end testing framework
- Runs on NodeJS
- Tests are written in Jasmine
- Runs on any server
  - Usually use Selenium Server
- Automates testing every layer of a web app
  - UI
  - Client-side logic
  - Server-side services

Using npm to install Testing Tools (requires Java SDK)

- Choose where you want your web root to be
- At a command prompt install karma globally
  - `npm install karma –g`
  - `karma --version`
- At a command prompt install protractor globally
  - `npm install protractor –g`
  - `protractor --version`
  - some locations will require admin level access
- Then update the webdriver manager (which comes with protractor)
  - `webdriver-manager update`
- and when done...
  - `webdriver-manager start`
- then browse to http://localhost:4444/wd/hub
- https://juliemr.github.io/protractor-demo/

framework training
business value through education

End to End Test-Driven Development

- first write the test
- then write the code until it passes the test
- then improve the code
- then write another test and iterate

Assemble, Act, and Assert

```
describe('',function(){
    beforeEach(function(){
    });
    it('',function(){
    });
});
```

Actions on locators

- Nothing happens until an action is called on a locator
- Most common actions are
  - click
  - sendKeys
  - clear
  - getAttribute
- They are all async so they all return a promise
- This means we can chain them together

Finding Multiple Elements

- element.all returns an array
- all has helpers
  - .count
  - .get(index)
  - first
  - last
- Find sub elements by chaining selectors

Running in Headless Mode

- Both Karma and Protractor allow Headless testing
  - http://cvuorinen.net/2017/05/running-angular-tests-in-headless-chrome/