



WESTERNACHER
SOLUTIONS



ANGULAR INTRODUCTION

Created by **Michał Szczepaniak** / **twitter**

TIME SCHEDULE

9:15 - 11:00 Part One

11:00 - 11:15 Break

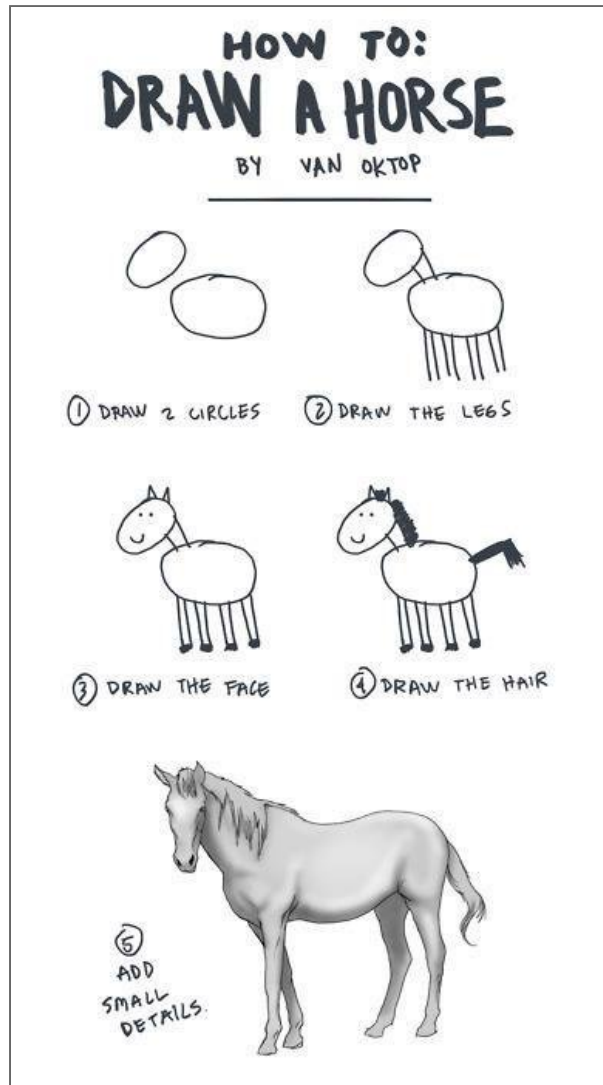
11:15 - 12:45 Part Two

AGENDA:

1. Creating your first application using Angular CLI
2. Angular big picture
3. Components
4. Modules
5. Template syntax
6. Services
7. Directives
8. Routing
9. Forms
10. Where to go next

can slightly change depending on the time left and questions

TOO FAST TOO SOON



APPROACH

Simple incremental changes

Next lesson will build on top of the previous one

STUCK?

Github Repository: **<https://github.com/mszczepaniak/mates>**

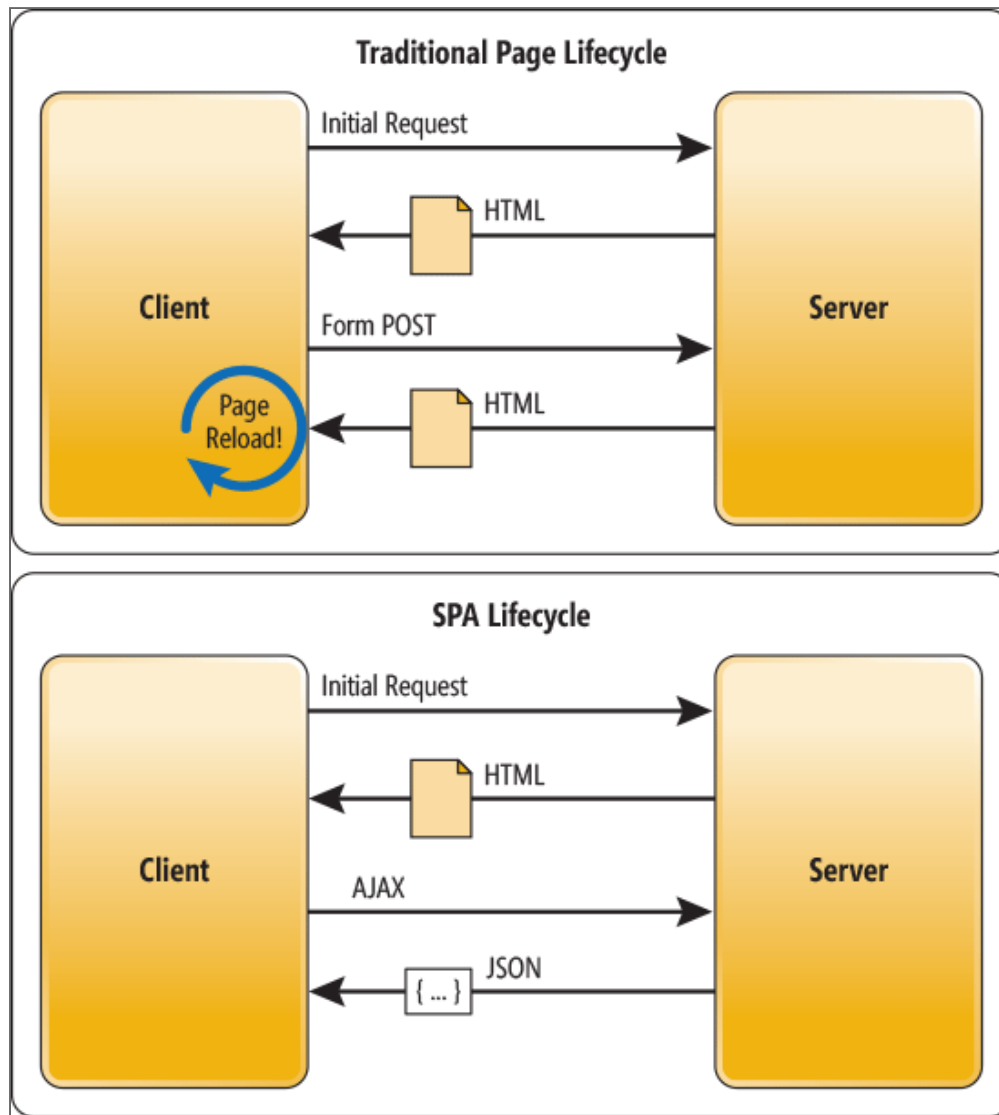
If Stuck -> check out the next commit

WHAT WILL WE BUILD? (MORE OR LESS)

LET'S RUN IT

```
ng new westie-mates
```

SPA AND MPA: HOW IT WORKS



SPA VS MPA

SPA advantages over MPA:

- faster navigation
- improved user experience
- decoupling of front-end and back-end development

SPA VS MPA 2

SPA disadvantages to MPA:

- The whole application has to be loaded before start.
- UI code is not compiled, so it's harder to debug and it's exposed to potential malicious user
- SEO (search engine optimization) implications; since your pages are built in the browser, the search engine crawler will see a different version of the page than that of your users

SPA VS MPA 3

But then you will also find detractors of many of these bullets:

- Lazy load reduces start up time
- by adding tree shacking you get slimmer code to load
- minified and uglified code is almost unreadable
- web crawlers (especially Google's) seem to not be so dumb after all and can load a SPA for proper index

You will also see that every answer to “when to use a SPA vs MPA / MVC / WhateverIAmDoingNow” is “it depends”, but no one will tell you a real recipe.

LET'S GET STARTED

```
cd westie-mates  
ng serve --open
```

The `ng serve` command launches the server, watches your files, and rebuilds the app as you make changes to those files

WHY ANGULAR?

Why Angular 2?



**Built for
Speed**



Modern



**Simplified
API**



**Enhances
Productivity**

BIG PICTURE

7

Data Binding

Modules

Structural Directives

Components

Services

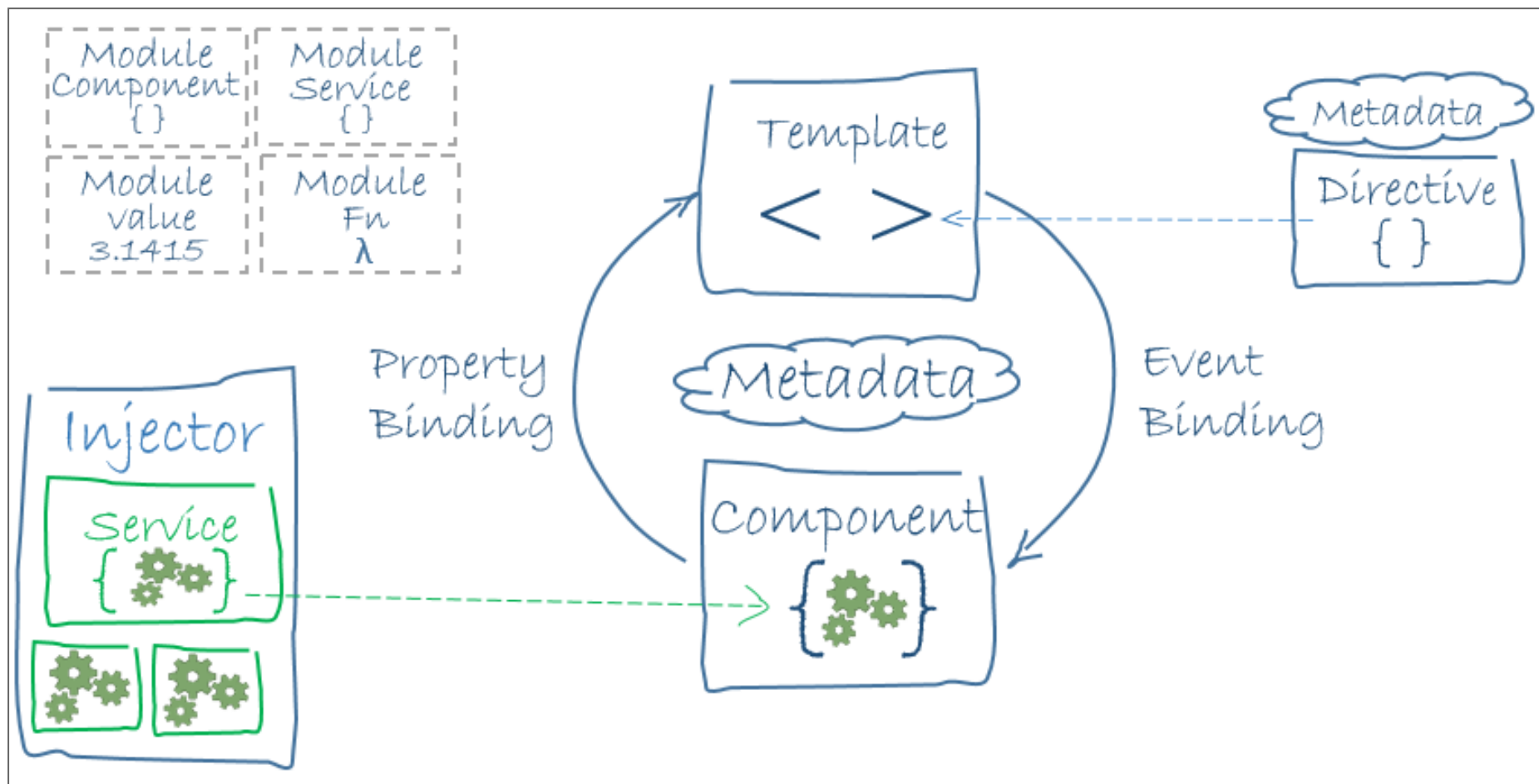
Templates

Dependency
Injection

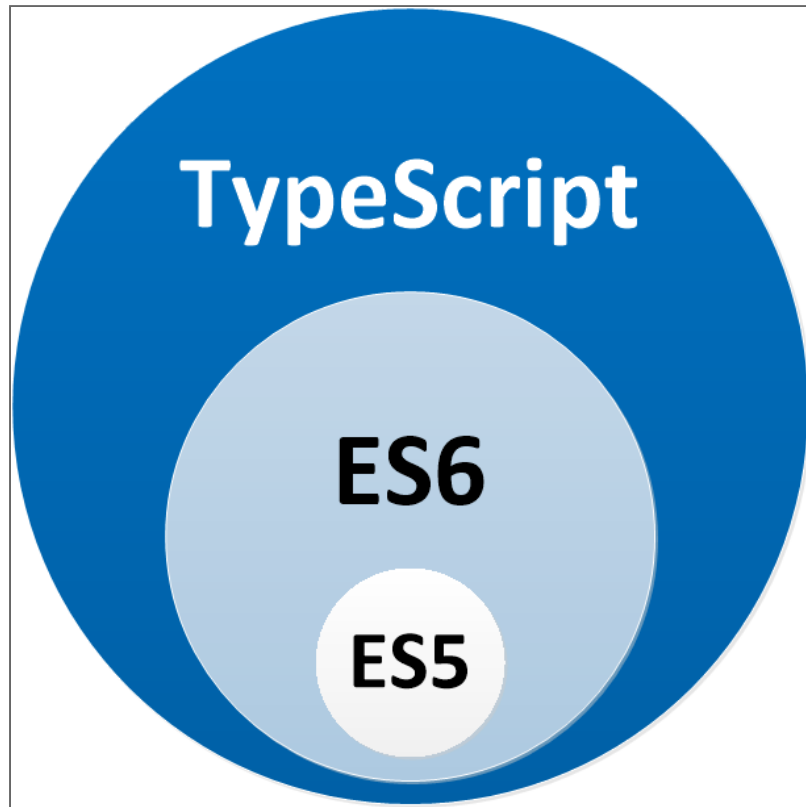
7 Keys to Angular 2



BIG PICTURE



TYPESCRIPT



TYPESCRIPT

Javascript that scales.

Typed superset of JavaScript that transpiles to plain JavaScript

Features (among others):

- types
- interfaces
- generics
- inheritance

TYPESCRIPT

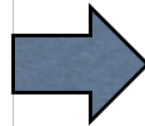
“We love TypeScript for many things... With TypeScript, several of our team members have said things like ‘I now actually understand most of our own code!’ because they can easily traverse it and understand relationships much better. And we’ve found several bugs via TypeScript’s checks.”

— Brad Green, Engineering Director—Angular

TRANSPILATION

TypeScript

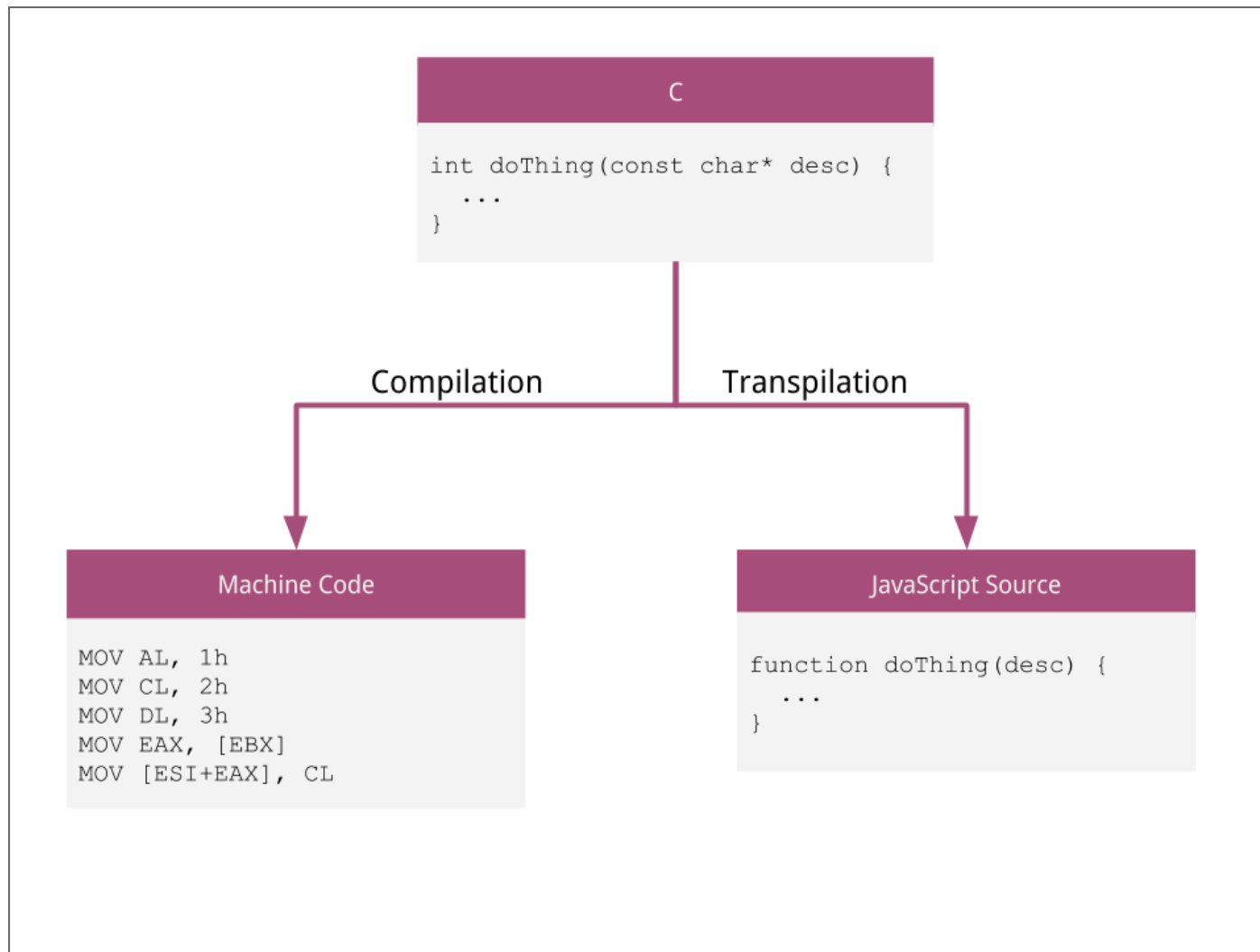
```
class Duck {  
  name: string = 'Bill';  
  quack(): void {  
    // Quack!  
  }  
}
```



ECMAScript 5

```
var Duck = (function () {  
  function Duck() {  
    this.name = 'Bill';  
  }  
  Duck.prototype.quack = function () {  
    // Quack!  
  };  
  return Duck;  
})();
```

WHY TRANSPILATION?



TYPESCRIPT PLAYGROUND

<http://www.typescriptlang.org/play/>

ANGULAR CLI

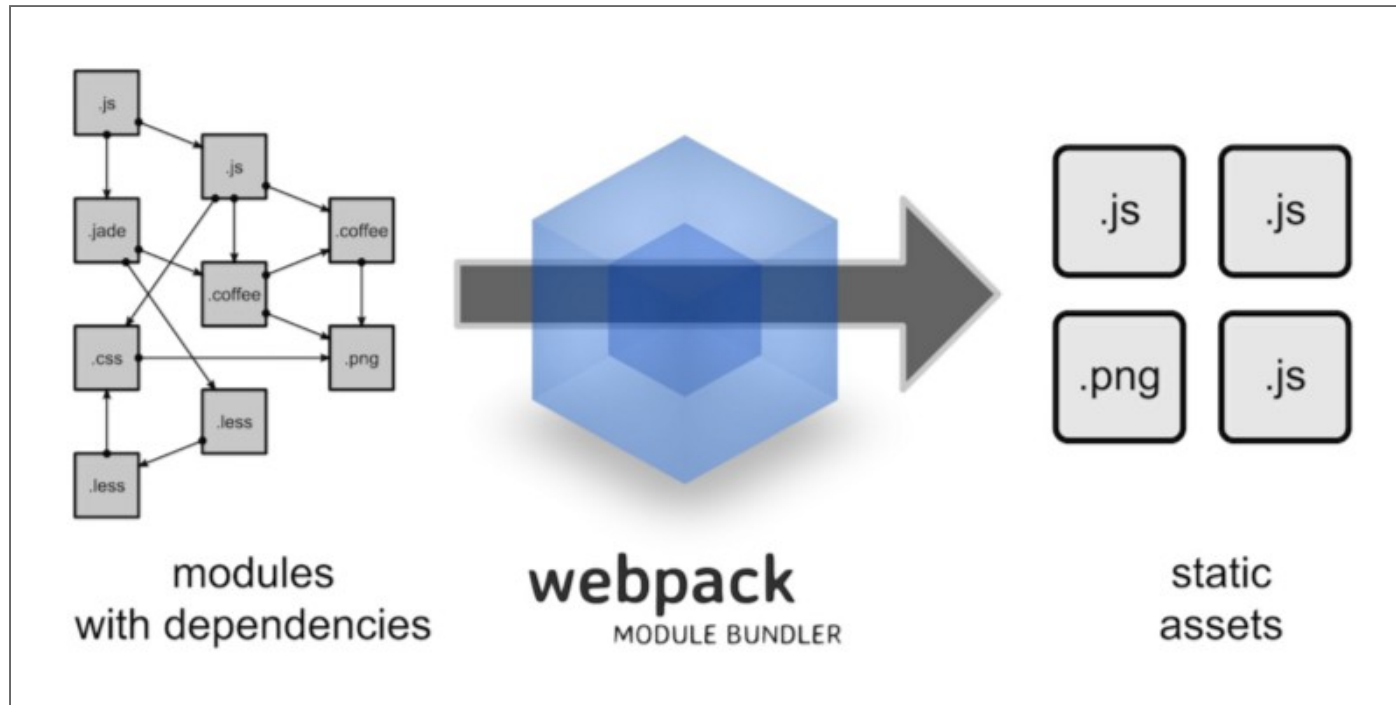
The Angular CLI is a command line interface tool that can create a project, add files, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

Webpack is a popular module bundler, a tool for bundling application source code in convenient chunks and for loading that code from a server into a browser.

ANGULAR CLI

| Component | ng g component my-new-component |
|------------------|--|
| Directive | ng g directive my-new-directive |
| Pipe | ng g pipe my-new-pipe |
| Pipe | ng g pipe my-new-pipe |
| Service | ng g service my-new-service |
| Interface | ng g interface my-new-interface |
| Module | ng g module my-module |
| Pipe | ng g pipe my-new-pipe |

WEBPACK



SIMPLE HTML INDEX PAGE

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0">
    <title>reveal.js</title>
    <link rel="stylesheet" href="css/reveal.css">
    <link rel="stylesheet" href="css/theme/solarized.css">
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>
<script src="lib/js/head.min.js"></script>
<script src="js/reveal.js"></script>
</body>
</html>
```

WHAT CLI DID

bootstrapped the app

set up testing, local server, root module and first component

set up typescript

set up dependencies

BROWSER PLATFORM?



DEMO

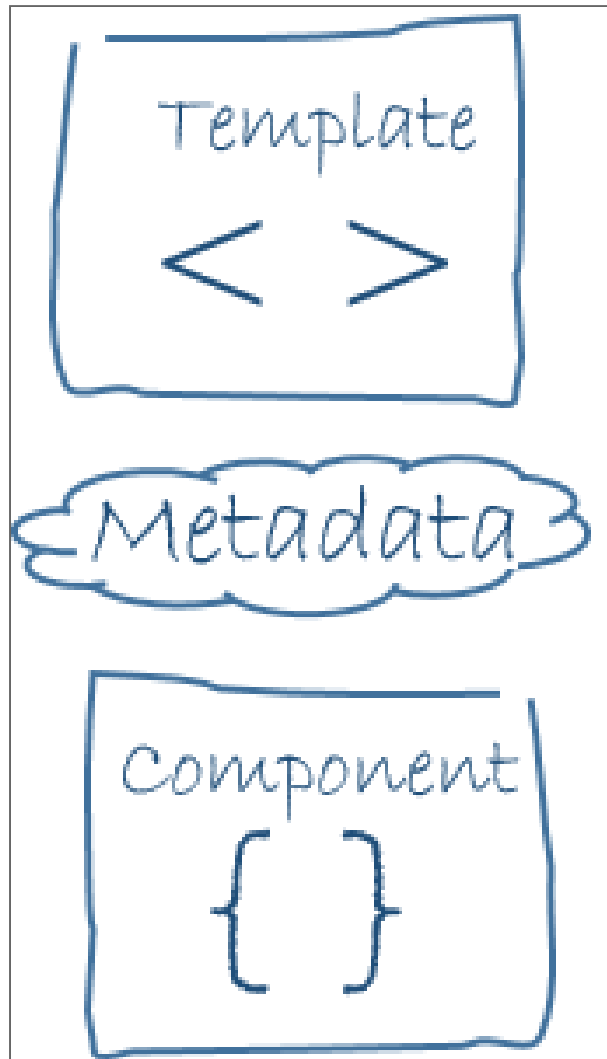
Checkpoint 0

COMPONENT

A component controls a part of screen called a view

The class interacts with the view through an API of properties and methods

COMPONENT



COMPONENT CODE

```
@Component({  
  selector: 'app-mate',  
  template: `<h1>Hello westernacher!</h1>`  
})  
export class MateComponent{  
  constructor() { }  
}
```

@@@ DECORATORS

A function that adds metadata to a class, its members (properties, methods) and function arguments.

Decorators are a JavaScript language feature, implemented in TypeScript and proposed for ES2016 (also known as ES7).

@Component() @NgModule() @Injectable()

@Input() @Output()

@Directive() @Pipe()

TEMPLATE

```
<div
  class="container"
>
  <h3 class="selectable">
    <i class="fa fa-users padme pull-left" aria-hidden="true"></i>
    Mate Number {{mateNo}}:
  </h3>

  <div class="row">
    <div class="col-sm-6">
      {{mate.name}}
    </div>

    <div class="col-sm-6">
      <button
        type="button"
        class="btn btn-danger"
        (click)="select(mate.name, mate.id)"
      >
        <i class="fa fa-address-book" aria-hidden="true"></i>
        Click me
      </button>
    </div>
  </div>
</div>
```

INTERPOLATION

```
{{ mate.name }}
```

```
{{ 2+2 }}
```

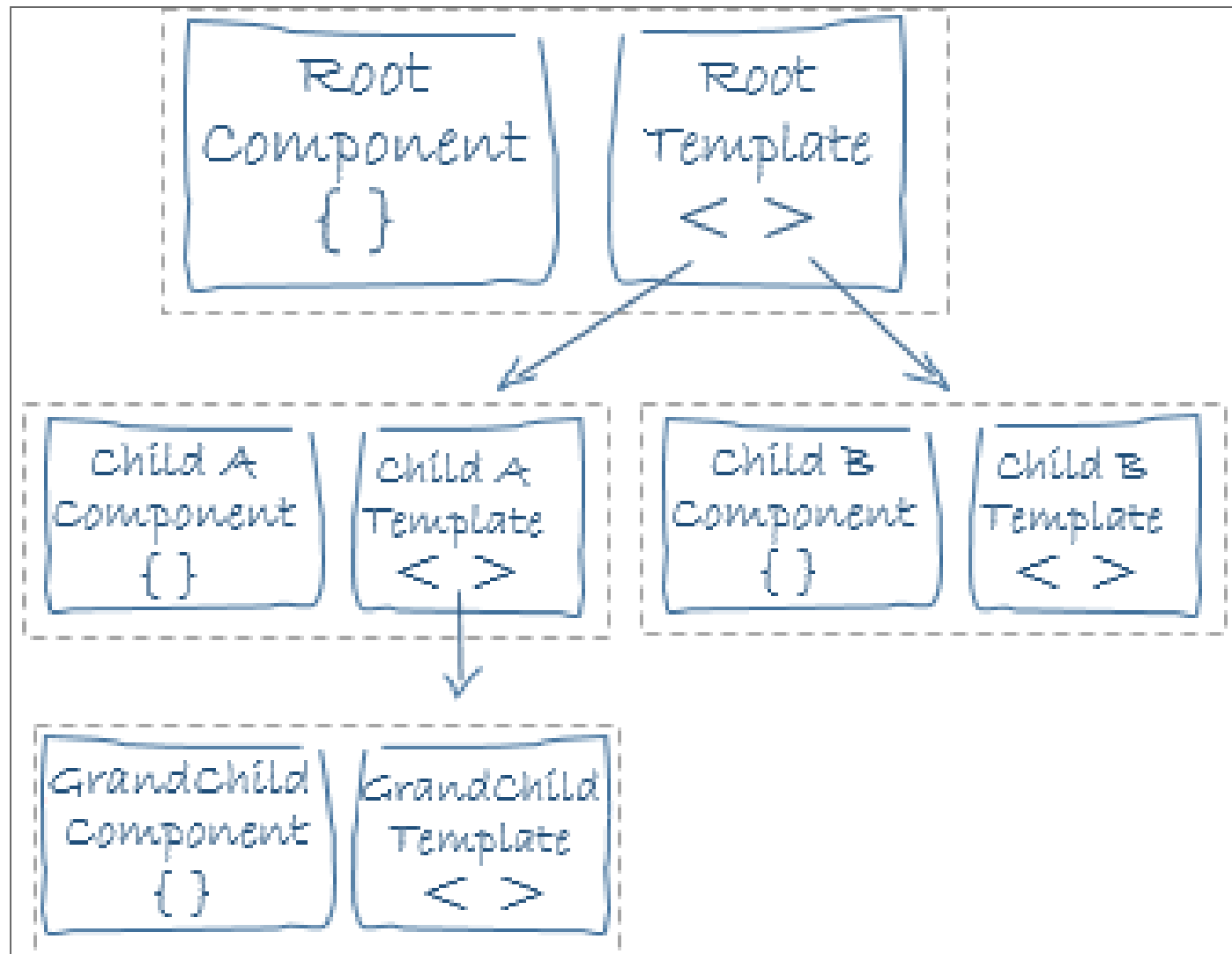
```
{{ getBeer() }}
```

You use interpolation to weave calculated strings into the text between HTML element tags and within attribute assignments

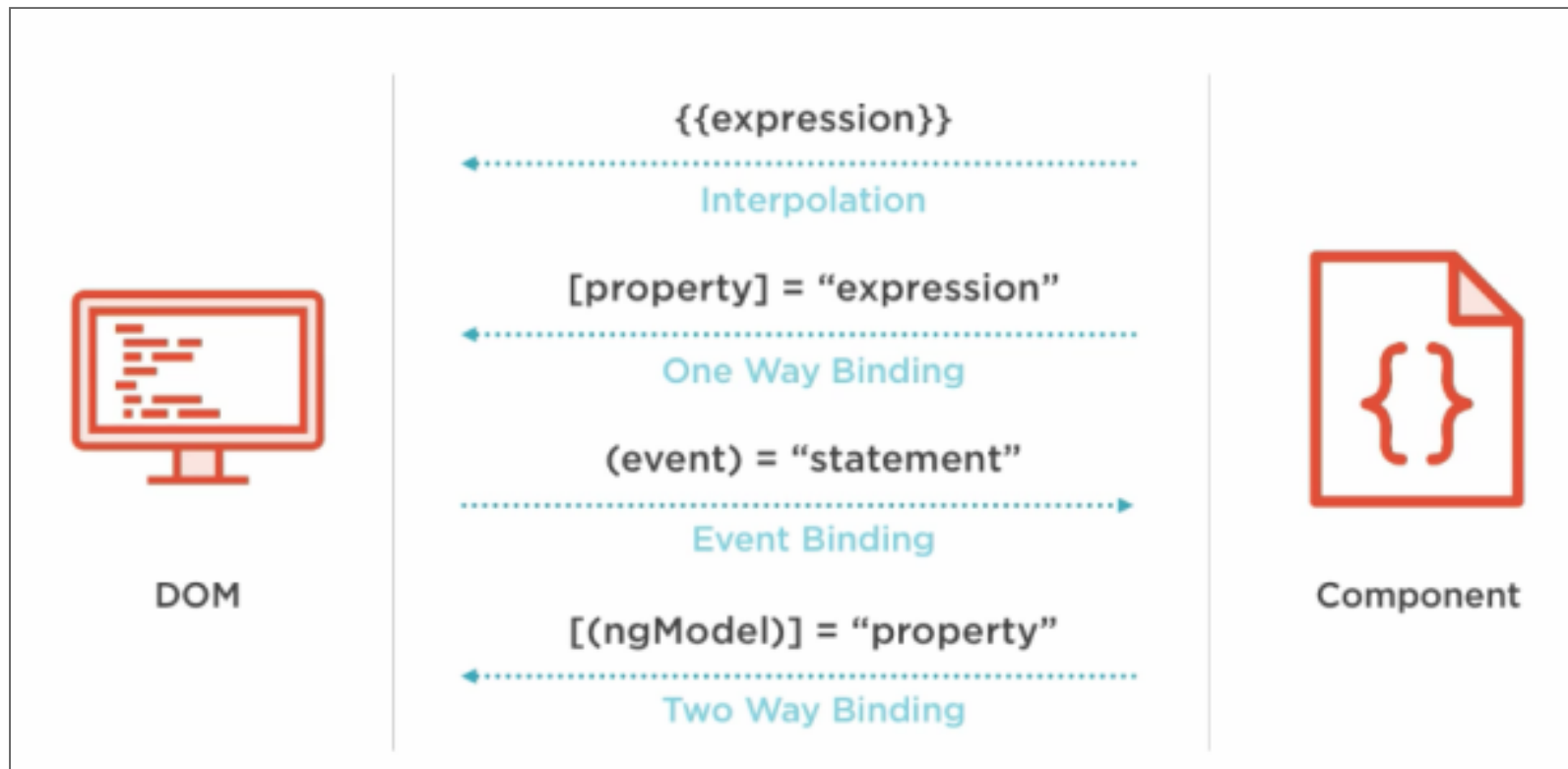
DEMO COMPONENT INTERPOLATION

Checkpoint 0

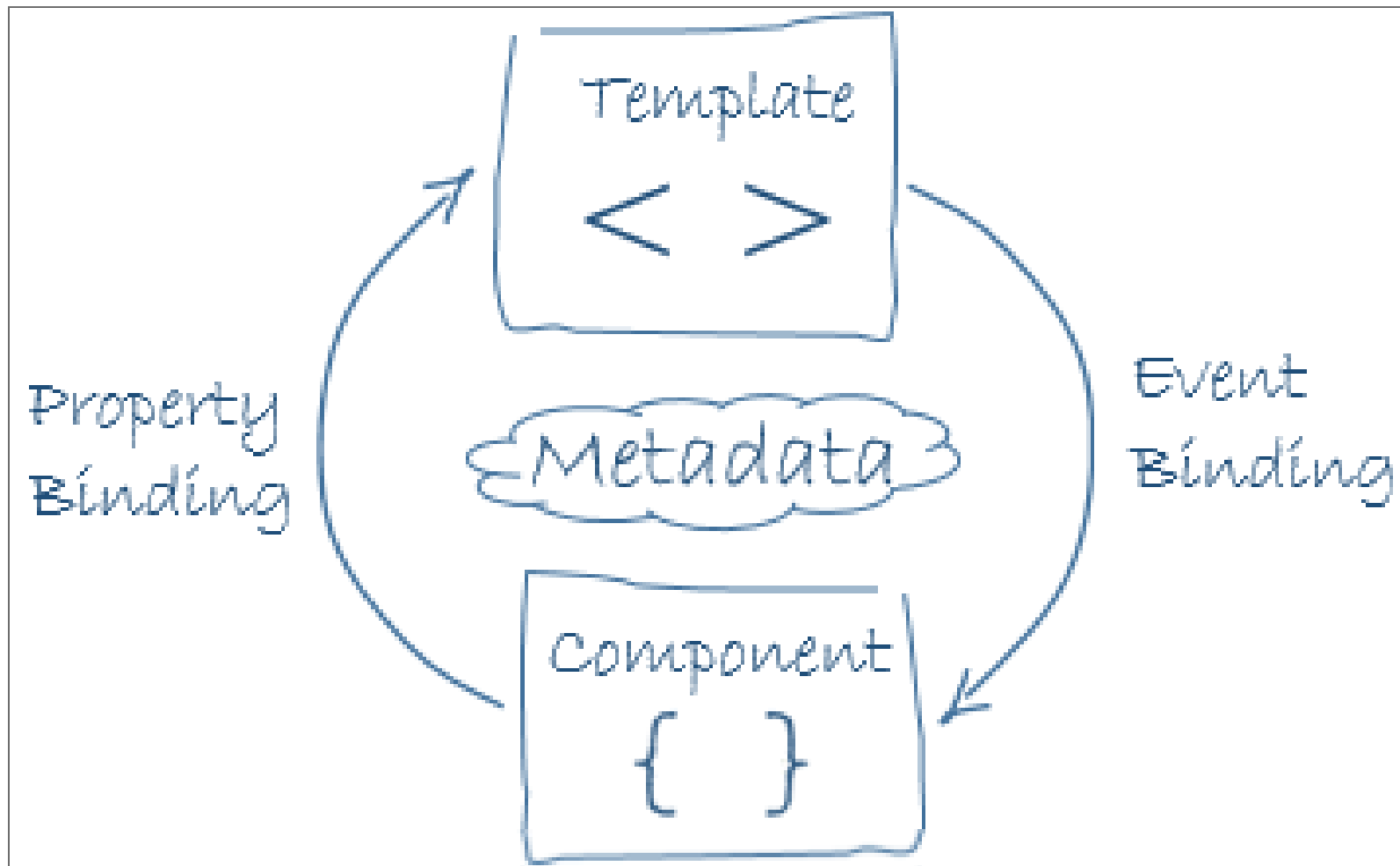
COMPONENTS ARCHITECTURE



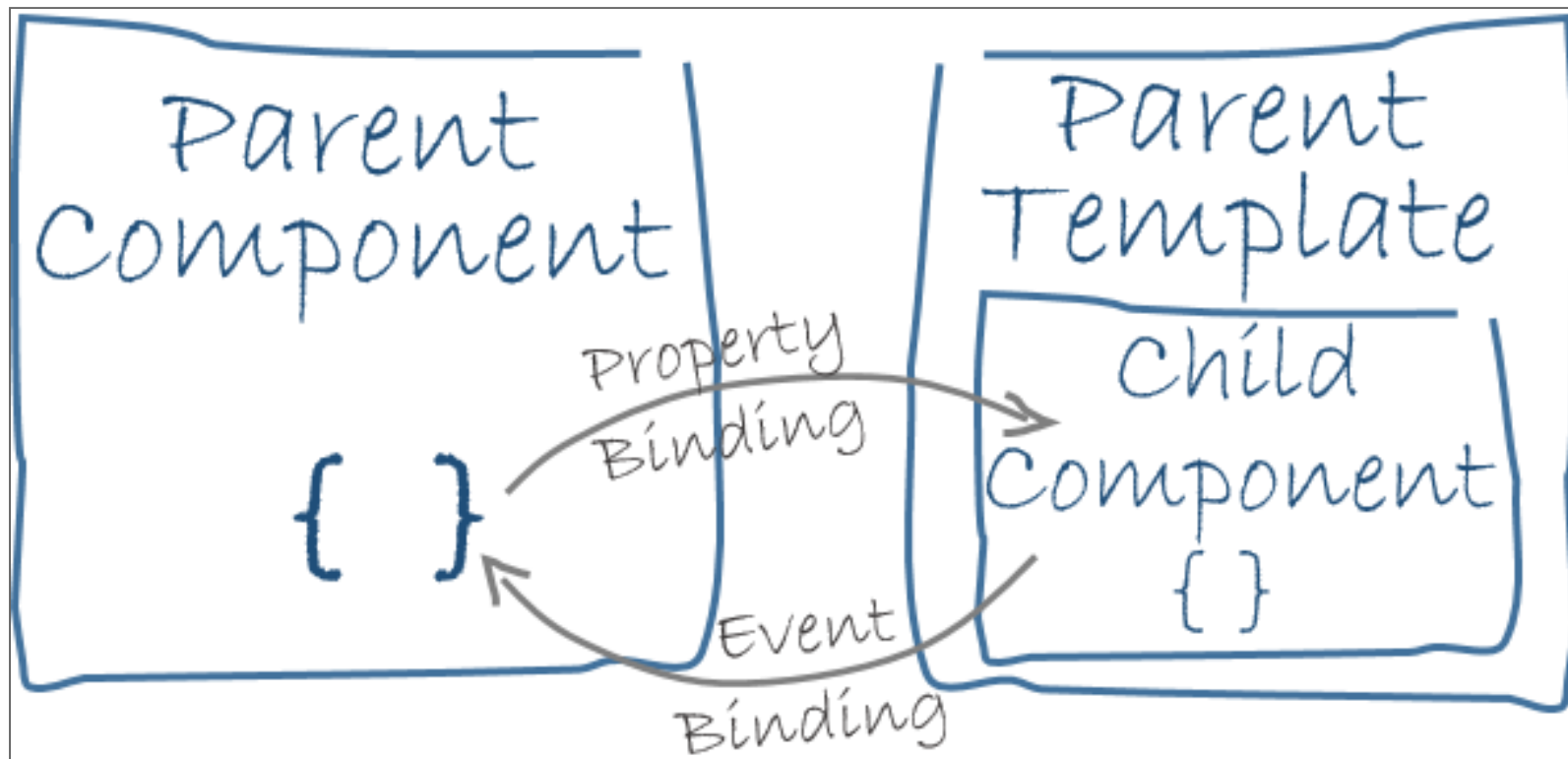
BINDINGS



BINDINGS IN PRACTICE



BINDINGS IN PRACTICE 2



COMPONENT WITH BINDINGS

```
@Component({
  selector: 'westie-mate',
  templateUrl: './mate.component.html',
  styleUrls: ['./mate.component.css']
})
export class MateComponent implements OnInit {
  @Input() mate: any;
  @Input() mateNo: number;
  @Output() mateSelected = new EventEmitter();
  constructor() { }

  ngOnInit() { }

  select(mateName, mateId) {
    this.mateSelected.emit({name: mateName, id: mateId});
  }
}
```

COMPONENT WITH BINDINGS

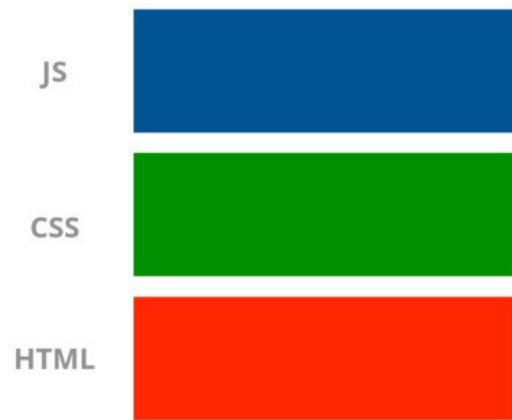
```
<westie-mate  
  [mate]="mate"  
  [mateNo]="i"  
  (mateSelected) = "handleMateSelected($event)"  
>
```

DEMO COMPONENTS BINDING

Checkpoint 1

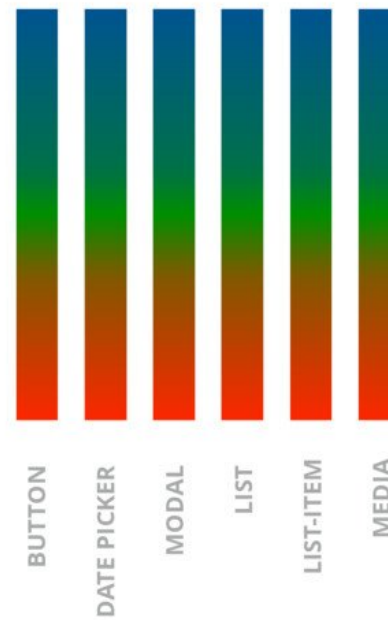
SEPARATION OF CONCERNS

Separation of Concerns



Separation of Concerns

(only, from a different point of view)



MODULE

Every Angular app has at least one Angular module class, the root module, conventionally named AppModule

While the root module may be the only module in a small application, most apps have many more feature modules, each a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.

An Angular module, whether a root or feature, is a class with an @NgModule decorator.

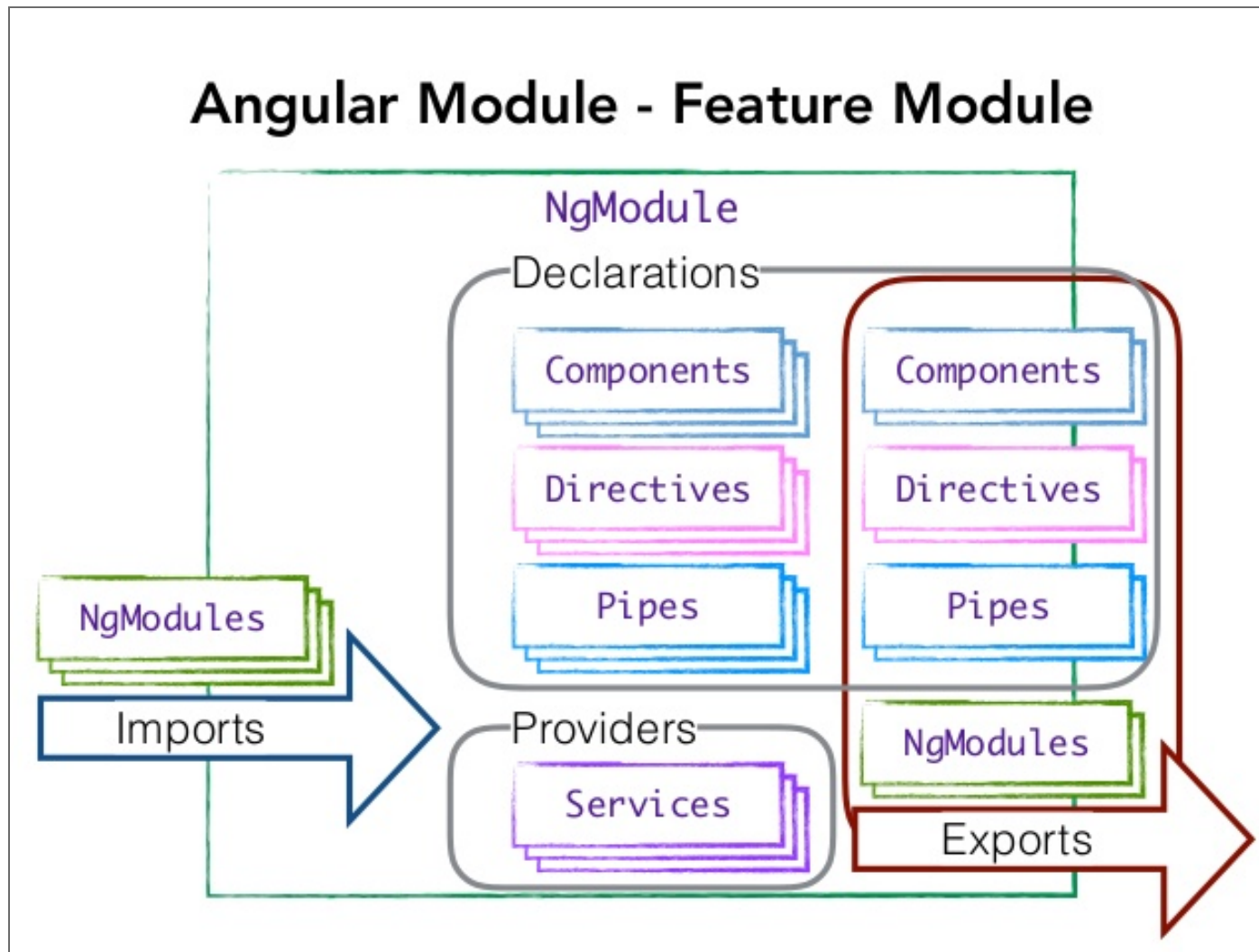
Angular modules are distinct from ES6 modules - they work together and complement each other

MODULE

An Angular module class describes how the application parts fit together. Every application has at least one Angular module, the root module that you bootstrap to launch the application. You can call it anything you want. The conventional name is AppModule.

imports, exports, declarations, bootstrap

MODULE



DECLARATIONS

the view classes that belong to this module. Angular has three kinds of view classes: components, directives, and pipes

EXPORTS

the subset of declarations that should be visible and usable in the component templates of other modules

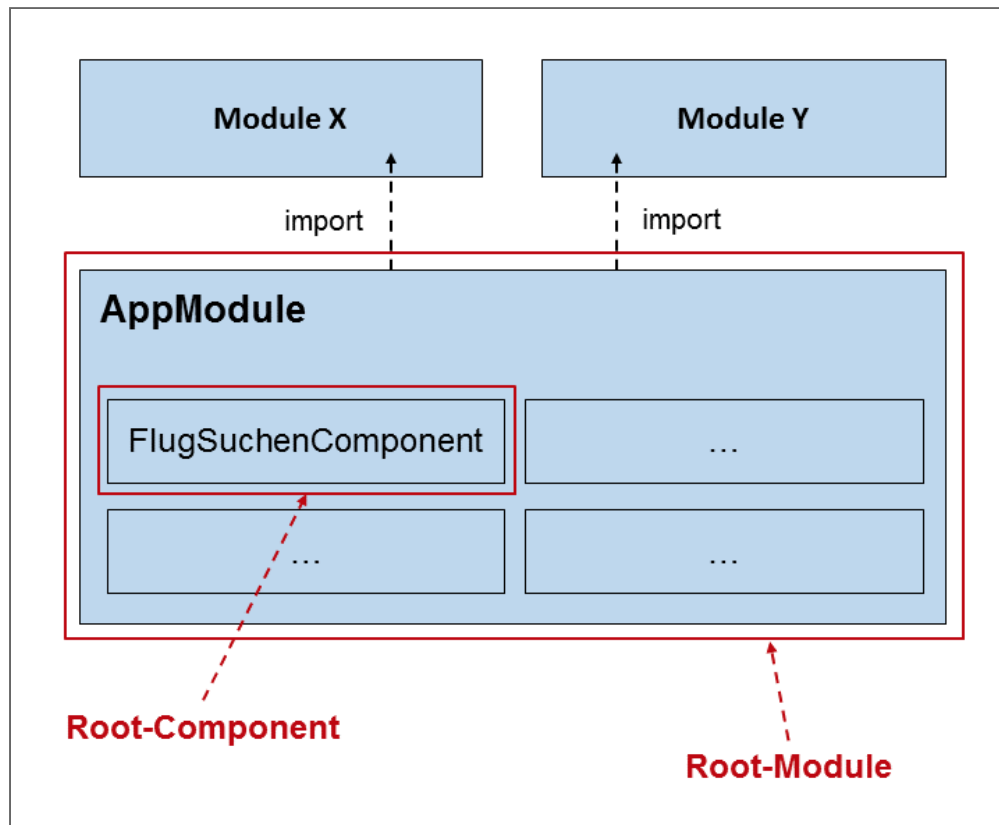
IMPORTS

other modules whose exported classes are needed by component templates
declared in this module

BOOTSTRAP

the main application view, called the root component, that hosts all other app views. Only the root module should set this bootstrap property

MODULE



APP MODULE CODE

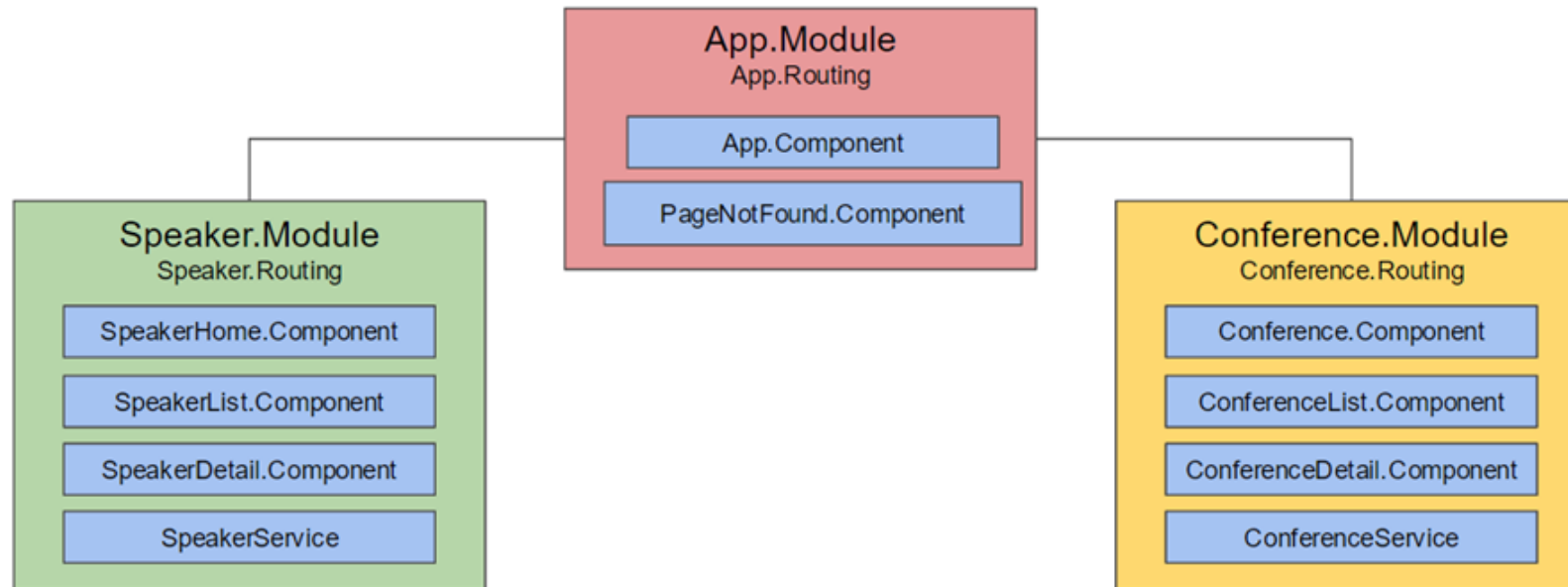
```
@NgModule({  
  declarations: [  
    AppComponent,  
    NotFoundComponent,  
    LoginComponent  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule,  
    MateModule,  
    RouterModule.forRoot(appRoutes)  
  ],  
  providers: [  
  ],  
  bootstrap: [AppComponent]  
})  
export class AppModule {
```

FEATURE MODULE CODE

```
@NgModule({  
  declarations: [  
    MateComponent,  
    MateListComponent,  
    MateDetailsComponent  
  ],  
  imports: [CommonModule],  
  exports: [MateListComponent],  
  providers: [  
    MateService,  
    ToasterService  
  ]  
})  
export class MateModule {}
```

SAMPLE MODULE ARCHITECTURE

Speaker Register Application Architecture



DEMO MODULES

Checkpoint 2

3RD PARTY STYLES

```
npm install bootstrap --save  
npm install font-awesome --save
```

WHERE TO PUT THEM?

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0">

    <title>reveal.js</title>

    <link rel="stylesheet" href="css/reveal.css">
    <link rel="stylesheet" href="css/theme/solarized.css">
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>
<script src="lib/js/head.min.js"></script>
<script src="js/reveal.js"></script>
</body>
</html>
```

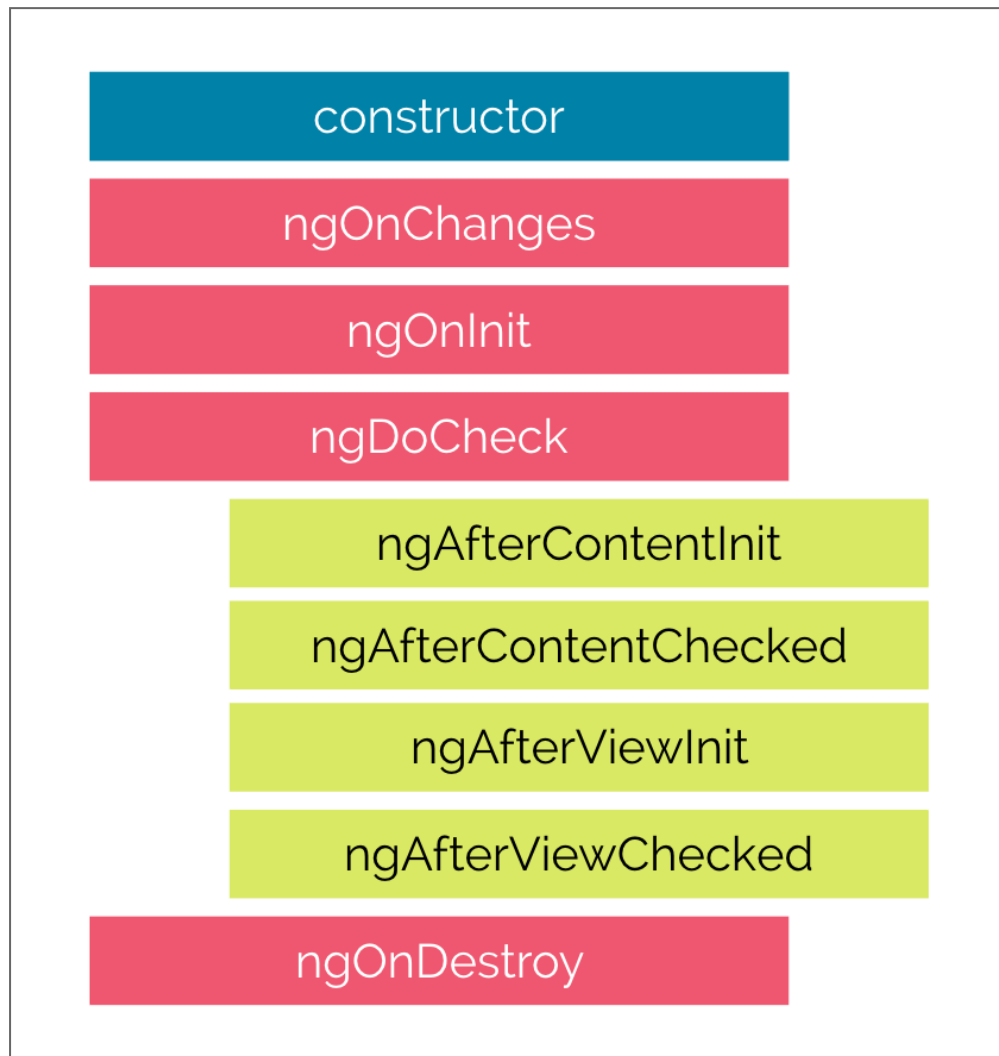
3RD PARTY STYLES

```
"styles": [  
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",  
  "../node_modules/font-awesome/css/font-awesome.min.css",  
  "styles.css"  
],
```

DEMO STYLES

Checkpoint 3

LIFECYCLE HOOKS



LIFECYCLE HOOKS

Angular creates, updates, and destroys components as the user moves through the application. Your app can take action at each moment in this lifecycle through optional lifecycle hooks, like `ngOnInit()`.

CONSTRUCTOR VS NGONINIT

The constructor method on an ES6 class (or TypeScript in this case) is a feature of a class itself, rather than an Angular feature. It's out of Angular's control when the constructor is invoked, which means that it's not a suitable hook to let you know when Angular has finished initialising the component.

By adding OnInit lifecycle hook, Angular can fire a method once it has finished setting the component up, and as the naming suggests, the hook is part of the component lifecycle

CONSTRUCTOR USAGE

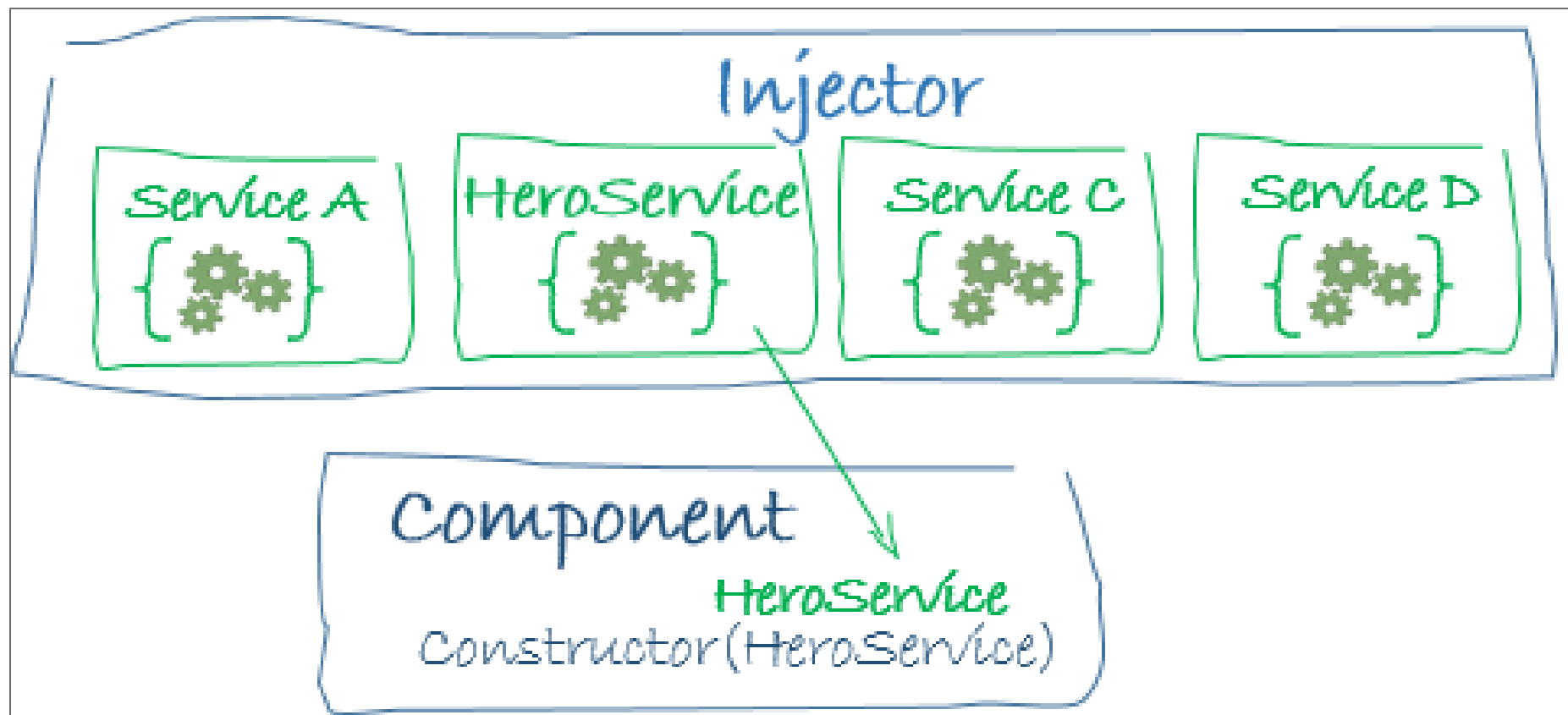
There is a suitable scenario for using the constructor. This is when we want to utilise dependency injection - essentially for “wiring up” dependencies into the component.

As the constructor is initialised by the JavaScript engine, and TypeScript allows us to tell Angular what dependencies we require to be mapped against a specific property

DEMO HOOKS

Checkpoint 4

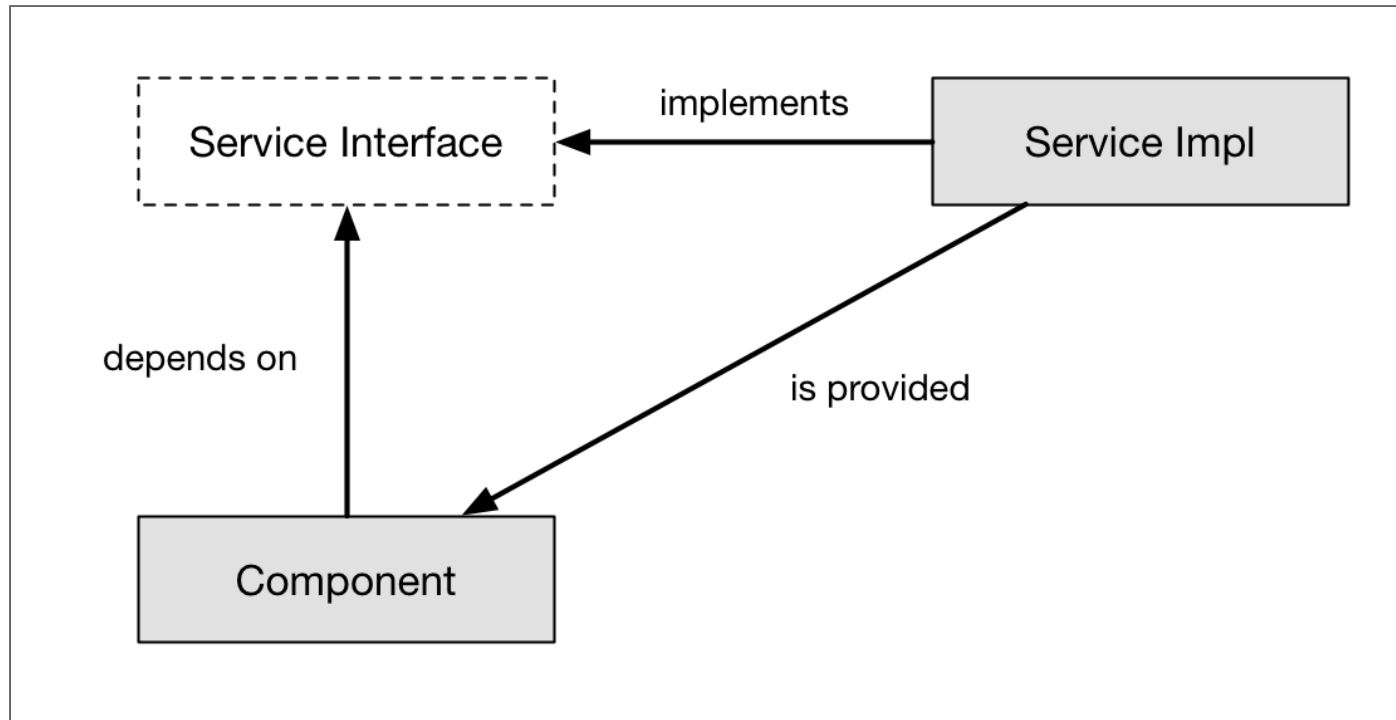
SERVICES



DEPENDENCY INJECTION

The idea behind dependency injection is very simple. If you have a component that depends on a service. You do not create that service yourself. Instead, you request one in the constructor, and the framework will provide you one. By doing so you can depend on interfaces rather than concrete types. This leads to more decoupled code, which enables testability, and other great things.

DEPENDENCY INJECTION



SERVICE CODE

```
@Injectable()
export class MateService {
  getUsers() {
    return [];
  }
}
```

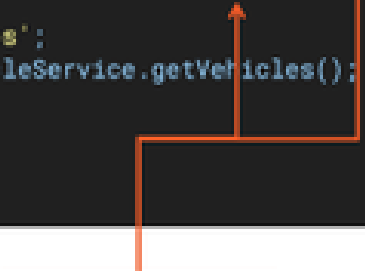
DEPENDENCY INJECTION

Dependency Injection

Angular 1

```
angular
  .module('app')
  .controller('VehiclesController', VehiclesController);

VehiclesController.$inject = ['VehicleService'];
function VehiclesController(VehicleService) {
  var vm = this;
  vm.title = 'Services';
  vm.vehicles = VehicleService.getVehicles();
}
```




Injection

Angular 2

```
import { VehicleService } from './vehicle.service';

@Component({
  selector: 'my-vehicles',
  templateUrl: 'app/vehicles.component.html',
  providers: [VehicleService]
})
export class VehiclesComponent {
  constructor(
    private _vehicleService: VehicleService) { }
  vehicles = this._vehicleService.getVehicles();
}
```



Injection

DI CODE

```
export class LoginComponent implements OnInit {  
  constructor(private router: Router) { }  
}
```


DEMO SERVICES

Checkpoint 5

3RD PARTY WRAPPER SERVICE CODE

```
npm install jquery --save  
npm install toastr --save
```

3RD PARTY WRAPPER SERVICE CODE

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.min.js",  
  "../node_modules/toastr/build/toastr.min.js"  
],
```

3RD PARTY WRAPPER SERVICE CODE

```
declare const toastr: any;

@Injectable()
export class ToasterService {
  success(message: string, title?: string) {
    toastr.success(message, title);
  }
}
```

HTTP SERVICE

```
getHeroes (): Promise {
    return this.http.get(this.heroesUrl)
                        .toPromise()
                        .then(this.extractData)
                        .catch(this.handleError);
}

addHero (name: string): Promise {
    let headers = new Headers({ 'Content-Type': 'application/json' });
    let options = new RequestOptions({ headers: headers });

    return this.http.post(this.heroesUrl, { name }, options)
                    .toPromise()
                    .then(this.extractData)
                    .catch(this.handleError);
}

private extractData(res: Response) {
    let body = res.json();
    return body.data || { };
}

private handleError (error: Response | any) {
    // In a real world app, we might use a remote logging infrastructure
    let errMsg: string;
```

DEMO MORE SERVICES

Checkpoint 6

DIRECTIVES

| Type | Structural | Attribute |
|-------------|---|--|
| Description | alter layout by adding, removing, and replacing elements in DOM | alter the appearance or behavior of an existing element. |
| Example | ngFor | ngModel |

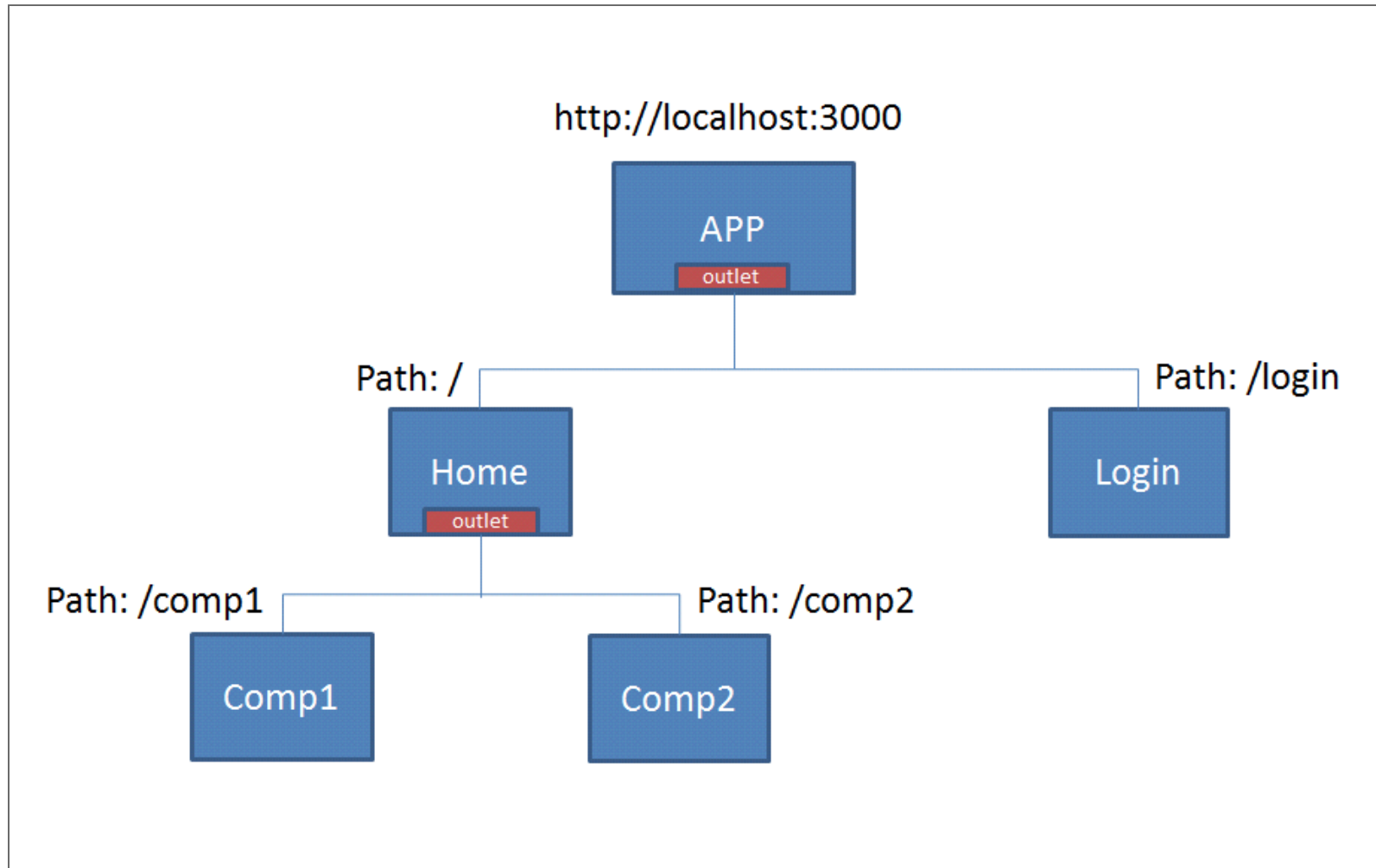
DIRECTIVES EXAMPLES

```
<div *ngFor="let mate of mates; let i=index"></div>  
<div *ngIf="mates"></div>  
[ (ngModel) ]=username
```


DEMO DIRECTIVES

Checkpoint 7

ROUTING



ROUTES

```
export const appRoutes: Routes = [  
  {path: 'login', component: LoginComponent},  
  {path: 'mates', component: MateListComponent},  
  {path: '', redirectTo: '/login', pathMatch: 'full'},  
];
```

WILD CARD

```
{path: '**', component: NotFoundComponent}
```

ROUTER OUTLET

```
<router-outlet></router-outlet>
```

ROUTER LINKS

```
<a  
class="btn btn-primary btn-lg"  
routerLink="/mates"  
>  
<span class="glyphicon glyphicon-home"></span>  
    Take Me Home  
</a>
```

STARTING POINT

```
<base href="/">
```

PARAMS

```
{path: 'mates/:username', component: MateListComponent},  
{path: 'mates/details/:id', component: MateDetailsComponent},
```


COMPONENT NAVIGATION

```
this.router.navigateByUrl(`/mates/details/${mate.id}`);
```

DEMO ROUTING

Checkpoint 8

FORMS CODE

```
<h1>Login</h1>
<div class="col-md-4">
  <form
    #loginForm="ngForm"
    autocomplete="off"
    (ngSubmit)="login(loginForm.value)"
  >
    <div
      class="form-group"
      novalidate
    >
      <label for="userName">User Name:</label>

      <em
        class="text-danger pull-right"
        *ngIf="loginForm.controls.username?.invalid &&loginForm.controls.username?.touched"
      >
        Required
      </em>

      <input
        id="username"
        required
        [(ngModel)] =username
        name="username"
```

FORM TEMPLATE VARIABLE

```
<form
  #loginForm="ngForm"
  (ngSubmit)="login(loginForm.value)"
>
```

FORM INPUT

```
<input
  id="username"
  [(ngModel)]="username"
  name="username"
  type="text"
  placeholder="User Name..."
/>
```

VALIDATION

```
<em
  class="text-danger pull-right"
  *ngIf="loginForm.controls.username?.invalid && loginForm.controls.username?.touched"
>
  Required
</em>
```

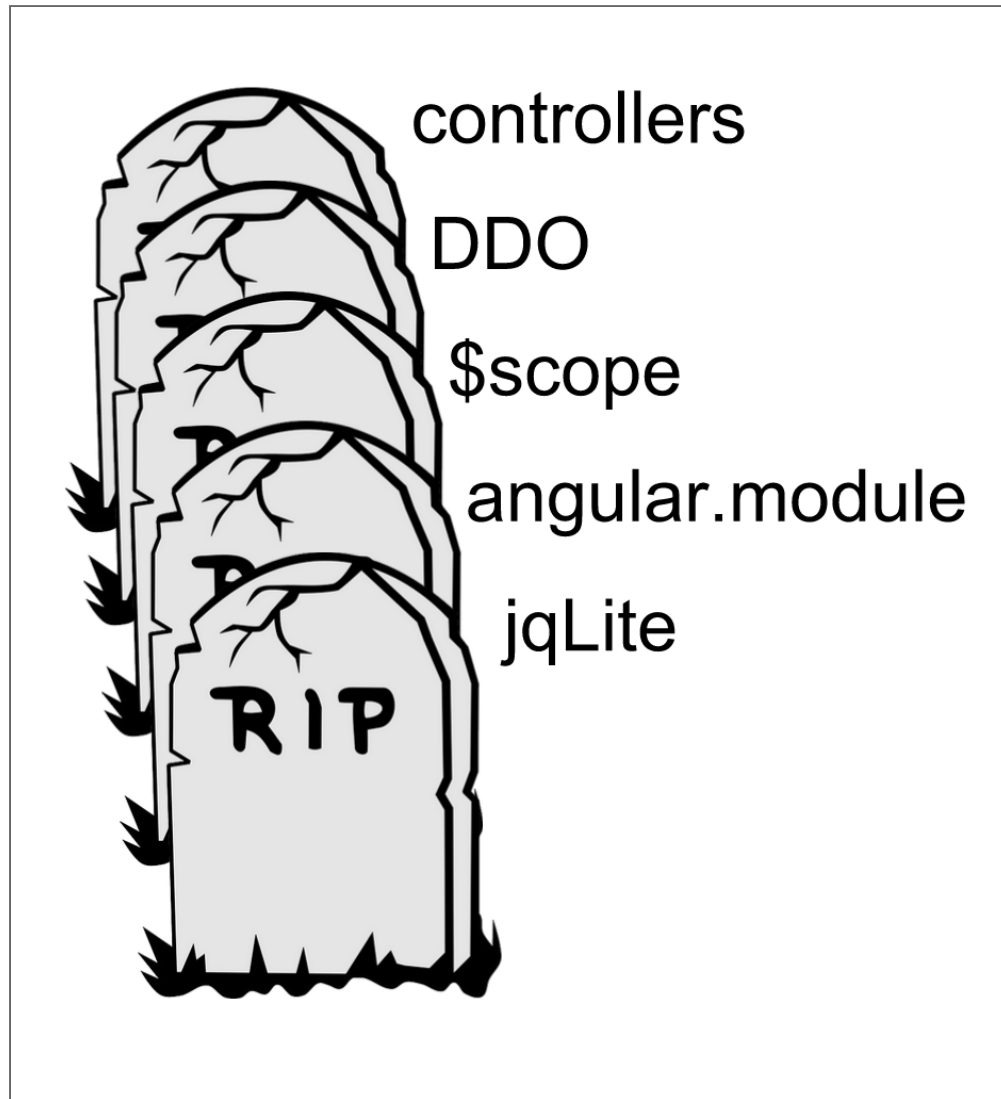
FORM STATE

| State | Class if true | Class if false |
|----------------------------------|----------------------|-----------------------|
| The control has been visited. | ng-touched | ng-untouched |
| The control's value has changed. | ng-dirty | ng-pristine |
| The control's value is valid. | ng-valid | ng-invalid |

DEMO FORMS

Checkpoint 9

CHANGES FROM ANGULAR 1



STYLEGUIDE

<https://angular.io/styleguide>

WHERE TO GO NEXT?

Awesome angular repo

Victor Savkin series

BLOGS TO FOLLOW

Todd Motto

Victor Savkin

John Papa

Thoughtram

TOPICS NOT COVERED

RxJS

Redux and ngrx

Pipes

Testing

Advanced routing with guards

Reactive forms

DEMO FORMS

Checkpoint 10

QUESTIONS?

* BUILT IN PIPES

Angular comes with a stock of pipes such as DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, and PercentPipe. They are all available for use in any template.

* BUILT IN PIPES - DATE

```
@Component({  
  selector: 'hero-birthday',  
  template: `
```

The hero's birthday is {{ birthday | date }}

```
`  
  })  
  export class HeroBirthdayComponent {  
    birthday = new Date(1988, 3, 15); // April 15, 1988  
  }  
}
```

* CUSTOM PIPES

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'michal'})
export class NamePipe implements PipeTransform {
  transform(value: any): any {
    if (value === 'westernacher') {
      return 'michal';
    }
    return value;
  }
}
```

* PIPES

```
{{ mate | michal }}
```