
DIRECTIVES: [MYDIRECTIVE, M
PROPERTY; DECLARES AN INPUT
MYEVENT = NEW EVENTEMITTER();
(MY-EVENT)="DOSOMETHING()">. @
CLICK) W
CHILDCOMPONENT; BINDS THE FIRST RESULT OF THE COMPONENT CONTENT QUERY
CONTENTCHILDREN(MYPREDICATE) MYCHILDCOMPONENTS; BINDS THE RESULTS
NENTS PROPERTY OF THE CLASS. @VIEWCHILD(MYPREDICATE) MYCHILDC
IEW QUERY (MYPREDICATE) TO THE MYCHILDCOMPONENT PROPE
N(MYPREDICATE) MYCHILDCOMPONENTS; BINDS THE RES
ASS. NOT AVAILABLE FOR DIRECTIVES. DIRECTIVE
R(MYSERVICE: MYSERVICE, ...) { ... } THE C
IOUS WORK HERE. NGONCHANGE
NGONINIT() { ... } CALLED
ME THAT THE INPU
CONTENTI
AL

Formation

Angular 1
Angular 1 Avancée
Angular 1 -> Angular 2

Angular 2
TypeScript

NightClazz

Sommaire

1. Changements et nouveaux concepts
2. Premier ‘Hello World’
3. Commencer un projet "production ready"

Pour faire les TP

- NodeJS / NPM
- Votre IDE préféré
  - Préconisation Visual Studio Code
- De la bonne humeur
- Du Wifi : Zenika-Guest / Guest4Zenika!2015
- Squelette de l'application disponible sur Github : https://github.com/Gillespie59/nightclazz-angular2
- Gitter disponible

Points négatifs d'Angular 1

- Différence entre directives et controller
- Performance du Two-way data-binding
- Hierarchie des scopes
- Pas de Server Side Rendering
- Plusieurs syntaxe pour les services
- API des directives trop complexe

Points positif d'Angular 2

- API plus simple qu'en Angular 1
- Amélioration des performances
- Trois types d'éléments manipulés : component, pipe et service
- Basé sur des standards : Web Component, Decorator, ES6, ES7
- Nouvelle syntaxe pour les templates
- Server Side Rendering
- Librairies pour migrer : ngUpgrade et ngForward

Points négatifs d'Angular 2

- Nouvelle phase d'apprentissage du framework
- Incompatibilité avec la première version
- De nouveau concepts à apprendre (Zone, Observable, SystemJS)

TypeScript

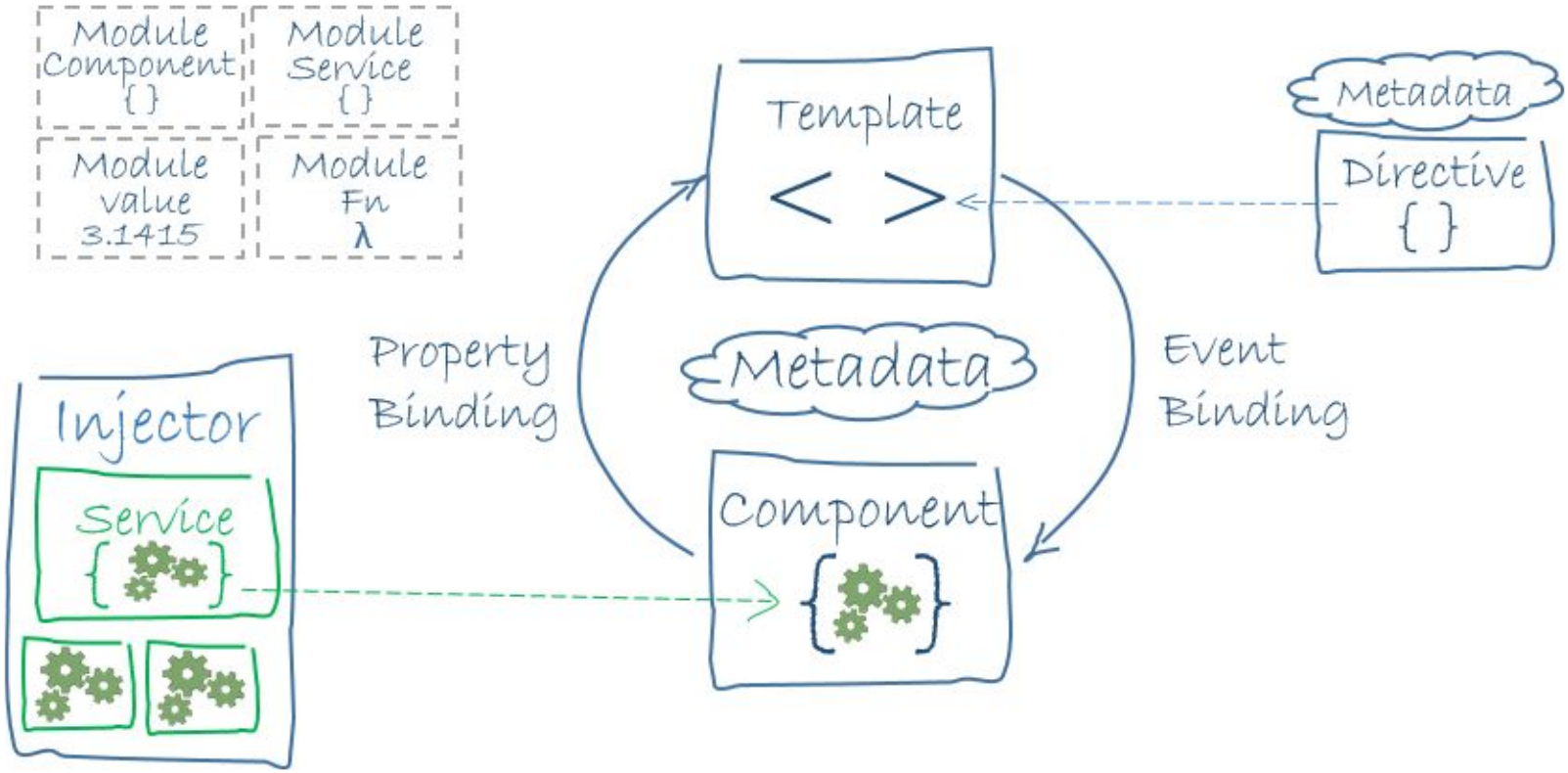
- Pas nécessaire pour Angular 2 mais vraiment mieux.
- Phase de compilation pour transformer en JavaScript.
- Le JavaScript est du TypeScript
- Typage
- Génériques
- Classes/Interfaces/Héritage
- Décorateurs
- ...

TypeScript

```
var id : number ;
let active : boolean ;
const powers : string[] ;

class Hero {
   constructor(public nickName : string){}
}
```

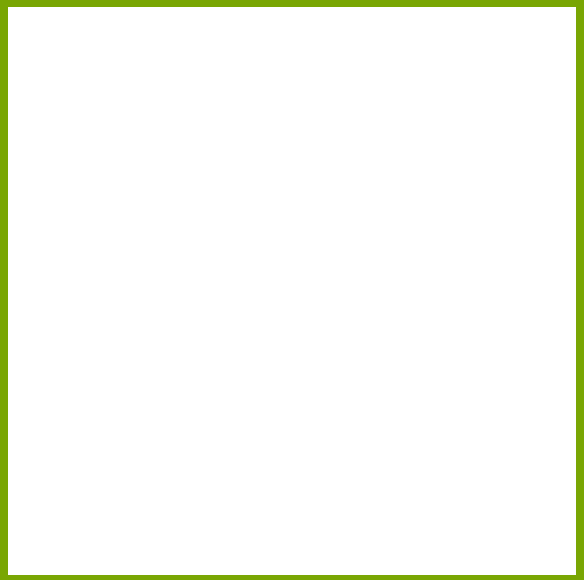
Architecture



Architecture

- Component
  - Classe qui représente un élément graphique dans lequel on définit un template
- Metadata
  - Moyen d'indiquer à Angular comment utiliser la classe
- Directives
  - Composants sans template (ngFor, ngIf, …)
- Pipes
  - Formatage d'une donnée (anciennement filter)
- Services
  - Code métier

Hello World



Démarrage du projet

```
npm install angular-cli -g

ng new Application
cd Application
./node_modules/typings/dist/bin install
// node .\node_modules\typings\dist\bin install
ng test
ng serve
```

Serveur sur :
http://localhost:4200

Architecture

- lite-server
  - Serveur locale de travail
- typescript
  - Compile les fichiers .ts en .js
- TSLint
- Karma, Protractor
- SystemJS
  - Chargeur de modules universel

Exercice 1

Dans le fichier app/app.component.ts

Ajouter une variable à la classe et la ‘binder’ dans le template.

app.component.ts Application/src/app

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     moduleId: module.id,
5     selector: 'app-root',
6     template: '<h1>{{projectName}}</h1>',
7     styleUrls: ['app.component.css']
8   })
9   export class AppComponent {
10    title = 'app works!';
11    projectName: string = "Hello World";
12  }
13
```

Exercice 2

Ajouter un champs de saisie pour le nom du projet.

Astuce : Banana in the box


```
template: `<h1>{{projectName}}</h1>
  <input  type="text" [(ngModel)]="projectName"/>
`,
```

Exercice 3

Ajouter un button et un listener sur le click

```
<button (click)="dislayProjectName()">Alert</button>
```

```
dislayProjectName(){
  alert(this.projectName)
}
```

Exercice 4

Afficher le button uniquement si projectName n'est pas vide

```
<button *ngIf="projectName">Hello</button>

<button template="ngIf:projectName">
  Hello</button>

<template [ngIf]="projectName">
  <button>Hello</button>
</template>
```

Pause

Exercice 5

Afficher la liste des héros

```
<h1>{{projectName}}</h1>
<ul>
  <li *ngFor="let hero for heroes">
    {{hero}}
  </li>
</ul>
```

```
heroes: string[] = ['Ironman', 'Le fauve']
```

Exercice 6

Créer un nouveau composant :

ng generate component hero --inline-template

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'hero',
  template: '{{name}}'
})
export class HeroComponent {
  @Input()
  name:string;
}
```

Exercice 7

Utiliser le composant

```
template: `
  {{projectName}}
  <ul>
    <li *ngFor="let hero in heroes">
      <hero [name]="hero">
    </li>
  </ul>
`,
directives: [Hero],
```

Exercice 8

Créer une classe de héro

ng generate class shared/hero

```
export class Hero{
  name: string;
}
```

Faire les modifications requises pour que l'input de hero.component.ts soit un **Hero**

Exercice 9

Créer un service de héros

ng generate service shared/hero service

```
import {Hero} from "./hero";
export class HeroService {
  findHeroes(): Hero[] {
    return [
      new Hero('Ironman'),
      new Hero('Le fauve')];
  }
}
```

Exercice 10

Injecter et utiliser le service de héros dans app/heroes/heroes.component.ts

```
@Component({
  providers: [HeroService]
})
export class ApplicationAppComponent {
  heroes: Hero[];

  constructor(private heroService:HeroService){
    this.heroes = heroService.findHeroes();
  }

}
```

Exercice 11

Passer le service en Observable

```
import { Injectable } from '@angular/core';
import {Hero} from './hero'
import {Observable} from 'rxjs/Observable';
import 'rxjs/Rx';

@Injectable()
export class HeroService {

  constructor() {}

  findHeroes():Observable<Hero[]>{
    return Observable.of([new Hero('Ironman'), new Hero('Le fauve')]);
  }
}
```

```
constructor(heroService: HeroService) {
  heroService.findHeroes().subscribe(
    heroes => this.heroes = heroes);
}
```

Exercice 12

Faire un appel HTTP

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

import {Hero} from './hero'
import {Observable} from 'rxjs/Observable';
import {map} from 'rxjs/operator/map';
import 'rxjs/Rx';

@Injectable()
export class HeroService {

  private _url:string = 'https://raw.githubusercontent.com/ggaulard/ggaulard.github.io/master/workshop/heroes';
  constructor(private http:Http) {}

  findHeroes():Observable<Hero[]>{
    return this.http.get(this._url)
      .map(data => data.json());
  }
}
```

Exercice 13

Filtrer les héros

Ajouter un champs de saisie et une variable de filtre.

Pour valider le fonctionnement, afficher simplement la variable à la suite du champs.

Exercice 14

Création d'un pipe
app/heroes/hero.pipe.ts

ng generate pipe shared/hero pipe

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

@Pipe({
  name: 'hero'
})
class HeroPipe implements PipeTransform {
  transform(value:Hero[], filter:string):Hero[]{
    return value.filter(h => {
      return true;
    });
  }
}
```

Exercice 15

Utilisation du pipe

```
template: `
  {{projectName}}
  <input [(ngModel)]="heroFilter" />
  <ul>
    <li *ngFor="let hero in (heroes | hero:heroFilter)">
      <hero [name]="hero">
    </li>
  </ul>
`,
directives: [Hero],
pipes: [HeroPipe]
```

Exercice 16

Afficher les images associées aux héros.

Exercice 17

Ajouter un nouveau chemin

Dans le fichier `app/app.component.ts`

```
<li router-active>
  <a [routerLink]=" ['Heroes'] ">Heroes</a>
</li>
```

```
@RouteConfig([
  { path: '/heroes',
    name: 'Heroes',
    component: Home },
  ...
```

Exercice 18

Créer un nouveau composant
app/heroes/heroes.component.ts

```
import {Component} from "angular2/core";
@Component({selector: 'heroes',
    template: `<h1>heroes</h1>`})
export class HeroesComponent {}
```

Le définir comme composant de notre route

```
@RouteConfig([
  { path: '/heroes',
    name: 'Heroes',
    component: HeroesComponent },
  ...
```

Merci à tous !