



zenika

AK



Guillaume Gaulard

UX Enthusiast

Senior Front End Developer

Flexible Scrum Master

ggaulard.github.io

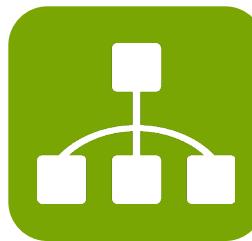


Conseil



Performance
Études
Audit
Prototypage
Veille technologique

Réalisation



Analyse métier et conception
Architecture applicative
Élaboration de FrameWorks
Développement et tests
Déploiement

Formation



Calendrier inter/
intra entreprise
Cycle personnalisé/
Sur-mesure
Coaching spécifique

Innovation



Veille technologique
R&D externalisée
POC
Ateliers de brainstorming
zStartup

NightClazz



#NightClazz
@ZenikALT



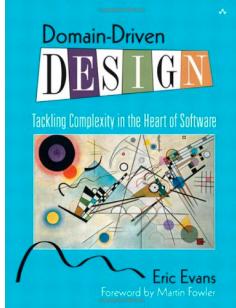
4 février
Golang



10 mars
Spark et ML



12 mai
Lagom



19 mai
Domain Driven Design



Internet des Objets

Formation



Angular 2

Angular 1 -> Angular 2



Angular 1

Angular 1 avancé



TypeScript

Recrutement



NightClazz

Sommaire

1

Changements et nouveaux concepts

2

Premier 'Hello World'

3

Commencer un projet "production ready"

A gagner 2 T-shirts

Au hasard parmi les inscrits





4K

Pour faire les TP

- git
- NodeJS
- Votre IDE préféré
 - Préconisation Atom (+ Packages : atom-typescript et angular2-snippets)
- Slides ggaulard.github.io/workshop/angular2.pdf



Points négatifs d'AngularJS

- Différence entre directives et contrôleur
- Performance du two-way data-binding
- Hiérarchie des scopes
- Pas de Server-Side Rendering
- Plusieurs syntaxes pour les services
- API des directives trop complexe



Points positif d'Angular 2

- API plus simple qu'avec AngularJS
- 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 nouveaux concepts à apprendre (Zone, Observable, SystemJS)



TypeScript

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

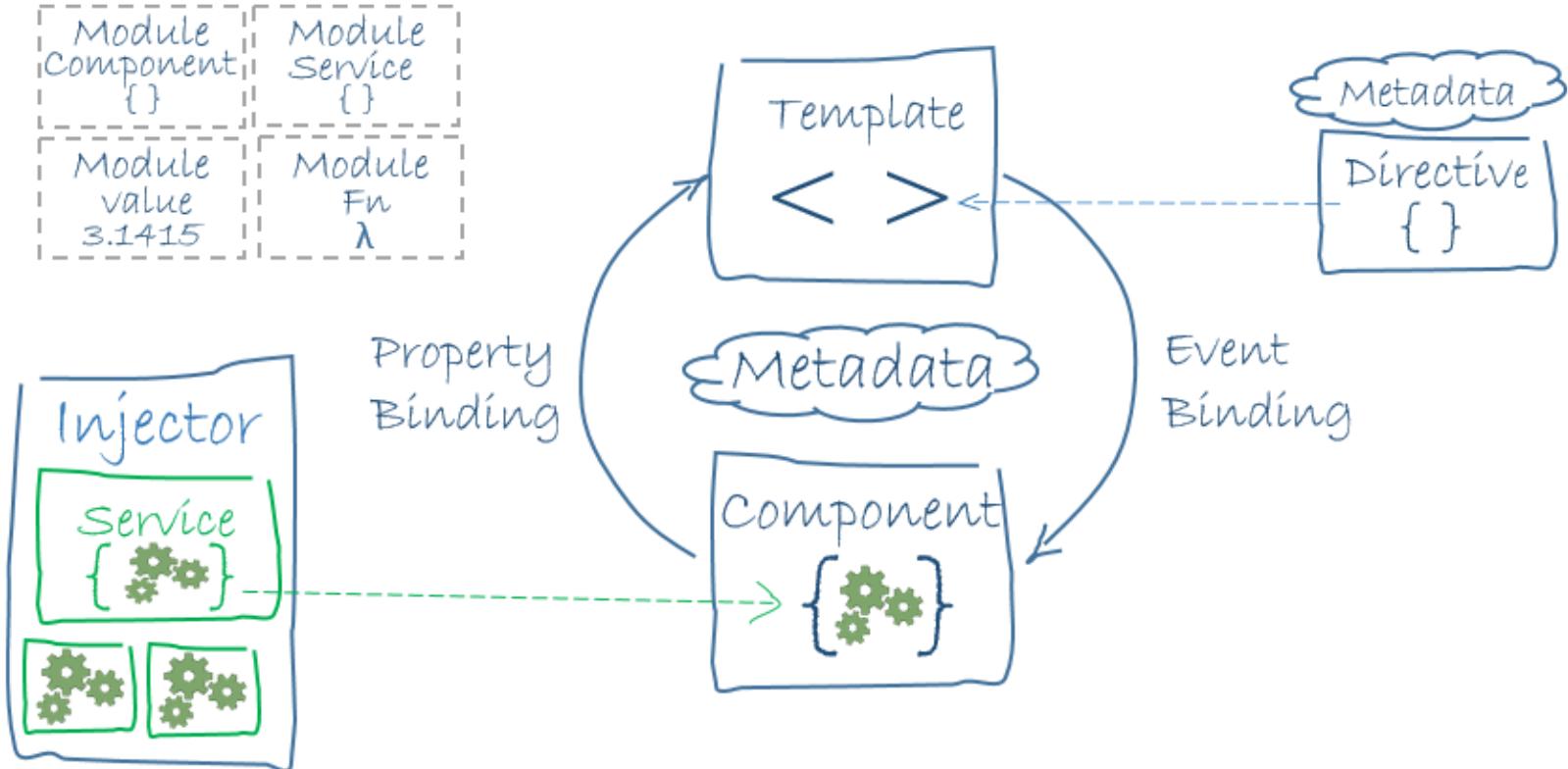


TypeScript

```
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, nglf, ...)
- Pipes
Formatage d'une donnée (anciennement filter)
- Services
Code métier

Hello World



Démarrage du projet

```
git clone
```

```
https://github.com/ggaulard/quickstart.git  
hero-workshop
```

```
cd hero-workshop
```

```
npm install
```

```
npm start
```

Serveur sur :
<http://localhost:8080>



Plunker [goo.g1/edjmoe](http://plnkr.co/goo.g1/edjmoe)

Architecture

- lite-server
Serveur local de travail
- typescript
Compile les fichiers .ts en .js
- 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.

```
@Component({
  selector: 'my-app',
  template: `<h1>My First {{projectName}}</h1>`
})
export class AppComponent {
  private projectName: string = "Hello World";
}
```

Exercice 2

Ajouter un champs de saisie pour le nom du projet.

Astuce : Banana in the box



```
template: `<h1>My First {{projectName}}</h1>
<input type="text" [(ngModel)]="projectName"/>
`<br/><br/><input [ngModel]="projectName"
(ngModelChange)="projectName=$event">
```

() evenement [] propriété

Exercice 3

Ajouter un button et un listener sur le click

```
<button (click)="sayHello()">Hello</button>
```

```
sayHello(): void {  
    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>
```



Starter Project

Démarrage du projet

```
git clone  
https://github.com/ggaulard/angular2-  
webpack-starter.git  
starter-workshop
```

```
cd starter-workshop
```

```
npm install  
npm start
```

Pizza Time !

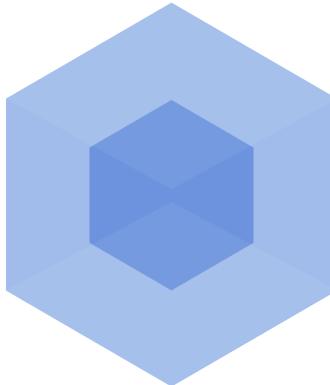


Architecture

- Webpack
Gestionnaire de modules
- Karma, Jasmine, Mocha, Protractor, Istanbul
Outils de test
- TSLint
Analyseur de code

Webpack

- Gestion des require('...')
- Définition d'un loader par type de fichier
- Serveur de développement
- Livrable pour la production



Tests

npm test

Observer la sortie console

Aller /coverage/Phantom.../app/index.html

Exercice 1

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 2

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 },
  ...]
```

Exercice 3

Afficher une liste de héros

```
template: `

<h1>heroes</h1>
<ul>
  <li *ngFor="#name of names">
    {{name}}
  </li>
</ul>`

names: string[] = ['Ironman', 'Le fauve'];
```

Exercice 4

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

```
import { Component, Input } from 'angular2/core';

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

Exercice 5

Utiliser le composant

```
import { HeroComponent } from './hero.component';

template: `
  <h1>heroes</h1>
  <ul>
    <li *ngFor="#name of names">
      <hero [title]="name"></hero>
    </li>
  </ul>`,
directives: [HeroComponent]
```

Exercice 6

Créer une classe de héro

```
export class Hero {  
  constructor(public name: string){  
  }  
}
```

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



Exercice 7

Créer un service de héros

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

Exercice 8

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

```
@Component({  
  providers: [HeroService],  
  
  export class HeroesComponent {  
    heroes: Hero[];  
    constructor(heroService: HeroService) {  
      this.heroes = heroService.findHeroes();  
    }  
  }  
}
```



Ensuite c'est
du bonus



Exercice 9

Passer le service en Observable

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

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



Exercice 10

Faire un appel HTTP

```
import {Hero} from "./hero";
import {Observable} from "rxjs";
import {Http} from "angular2/http";
import {Injectable} from "angular2/core";

@Injectable()
export class HeroService {
  private _heroesUrl =
    'http://ggaulard.github.io/workshop/heroes';
  constructor(private http: Http) {}
  findHeroes(): Observable<Hero[]> {
    return this.http.get(this._heroesUrl)
      .map(res => <Hero[]> res.json());
  }
}
```



Exercice 11

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 12

Création d'un pipe

app/heroes/hero.pipe.ts

```
import {Pipe, PipeTransform} from 'angular2/core';
import {Hero} from './hero';

@Pipe({name: 'hero'})
export class HeroPipe implements PipeTransform {
  isNameMatching(hero: Hero, filter: string){
    return hero.name.toLowerCase()
      .indexOf(filter.toLowerCase()) != -1;
  }
  transform(heroes: Hero[], args): Hero[] {
    return heroes.filter(hero =>
      this.isNameMatching(hero, args[0]));
  }
}
```



Exercice 13

Utilisation du pipe

```
@Component({
  selector: 'heroes',
  template: `<input type="text" [(ngModel)]="heroFilter"/>
<ul>
  <li *ngFor="#hero of (heroes | hero:heroFilter)">
    <hero [hero]="hero"></hero>
  </li>
</ul>`,
  pipes: [HeroPipe],
```

Exercice 14

Afficher les images associées aux héros.



Merci à tous

